M.I.T. Laboratory for Computer Science

Request for Comments No. 253
September 14, 1983

Distributed Name Management

by Karen R. Sollins

Attached is my doctoral thesis proposal:

My committee is composed of:

David P. Reed, supervisor
David D. Clark, reader
J. C. R. Licklider, reader

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
Proposal for Thesis Research in Partial Fulfillment
of the Requirements for the Degree of
Doctor of Philosophy

Title: Distributed Name Management

Submitted by: Karen Rosin Sollins
9 Southwick Circle
Wellesley, Mass. 02181

Date of Submission: August 25, 1983

Expected Date of Completion: May 15, 1984

Laboratory where Thesis will be done: Laboratory for Computer Science

Brief Statement of the Problem:

This work addresses the problem of providing a unified naming framework in a decentralized, distributed computer system. The system designer needs a mechanism of great simplicity and power for creating and managing names used when communicating with other people through the computer system. A computer naming facility should reflect closely the patterns of name usage between humans in communicating directly with each other. We assume a collection of cooperating computing nodes connected by a communication medium. The underlying model we propose contains objects called *contexts* and *aggregates*. A context can map names into any type of object, i.e., a person, a data object, a process, a port, or any other type of entity accessible to the client. The aggregate will provide a mechanism for further exploration of different kinds of naming used in human communication, some of which are currently unavailable in operating systems. The utility of the naming model will be tested in part through an implementation of the mechanisms within an electronic message system.

Table of Contents

List of Figures

1. Introduction

Names form the basis of communication both among humans and between humans and computers. In order to communicate with another human, the human must be able to name objects and actions in such a way that both humans understand the names. Analogously, in order to communicate with a computer, the human must be able to name operations and objects in a way meaningful to both the human and the computer. Therefore, what can be named and how is a central issue in designing a computer system useful to humans.

This work is an investigation of a naming framework for a distributed computer system, using human communication patterns to provide a set of goals for the framework. The system model is one of a *federation* of loosely coupled computers connected by a communications network. The goals for the framework based on human communication, plus the constraints presented by the federated system model, will provide the basis for the technical problems to be addressed in the thesis. In addition, since the functions provided by this naming facility will not be identical to those functions provided in past naming facilities, the thesis must address how those additional functions will be provided for the users of such a computer system.

In the past names in computer systems have been restrictive. The space of file names was likely to be hierarchical and the name on each branch of the hierarchy might be limited in length. The space of names identifying users might be flat or hierarchical and might be limited to a small number characters. Processes, even subprocesses, often were only namable very awkwardly (perhaps by a number) if at all, even by a subprocess's parent. None of these has much in common with the way people name things, particularly when communicating with other people.

There are two reasons for naming entities, both having to do with communication. First, names may be used by an individual to organize and remember named entities; names provide a catalog facility. This sort of name is used by an individual or group to organize information. Second, names may be used among a group of people as the basis of communication. In order to communicate the group must agree on the meaning of the names used. Over time, they may expand the set of names on which they agree. Both to reach such an initial agreement and to expand further their basis of agreement, they will have certain protocols.

The federated system places limitations on anything built on it. For example, sharing of information, such as collections of names, across node boundaries is restricted by the fact that the only means of communicating across node boundaries is by passing messages. The thesis will explore both the constraints from above (the clients) and the limitations from below (the federation of nodes), and will provide a naming facility conforming to those restrictions.

Briefly, the mechanisms proposed are based on two new proposed types of objects, the *context* and the *aggregate*. A context translates names into entities. It can be given pairs of names and entities to remember and translate on demand. An aggregate is a structured set of contexts. Each aggregate has a *current context* reflecting that part of the aggregate that is being actively used by all the participants in the communication and an *environment* reflecting the private information that a participant carries to the aggregate. The current context is a single context. The environment is a collection of contexts, possibly ordered. An aggregate is an individual's view of the name resolution facility available while communicating with others.

In order to clarify the ideas presented thus far, Figure 1-1 depicts two aggregates, one being used by Fran and the other by Alex, representing together an interaction between the two participants. The shared current context represents that part of the focus of interest that the two have in common. Each participant also has an environment or set of contexts, some shared and some possibly private upon which to call when needed. The notation used in this figure will be used in later figures.

This proposal elaborates the ideas presented above. The basis for much of the discussion is a pair of examples involving communication by electronic messages and a joint development and writing project among several people. These scenarios are used as a vehicle for exploring and exemplifying both the problems that arise in communication and, from the ensuing model, an approach to building a solution. In order to arrive at the model, a set of conclusions are drawn about how humans communicate with each other. Section 2 explains the problem further by addressing in greater detail the constraints mentioned above and considering the two scenarios used to explore the problem further. In Section 3 we present a model for a solution, using the scenarios to elaborate on the model and a particular approach to an implementation. Section 4 contains a discussion of the work of others as that work relates to the proposed research. Section 5 summarizes the problems and the solutions presented. The plan for the research, including discussion of an implementation and further work to be included in the thesis can be found in Section 6. Finally specifications in the language Clu for contexts and aggregates can be found in the Appendix.

2. The Problem

As discussed in the preceding section, a naming mechanism must suit both the system on which it is built and the needs of the clients of the mechanism. Section 2.1 describes a federated system and presents a list of underlying ideas that seem to be common to many clients' needs. These ideas suggest a common set of functions that should be provided by a naming facility. Two scenarios will be used as the basis for further discussion of naming problems and a solution to those problems.

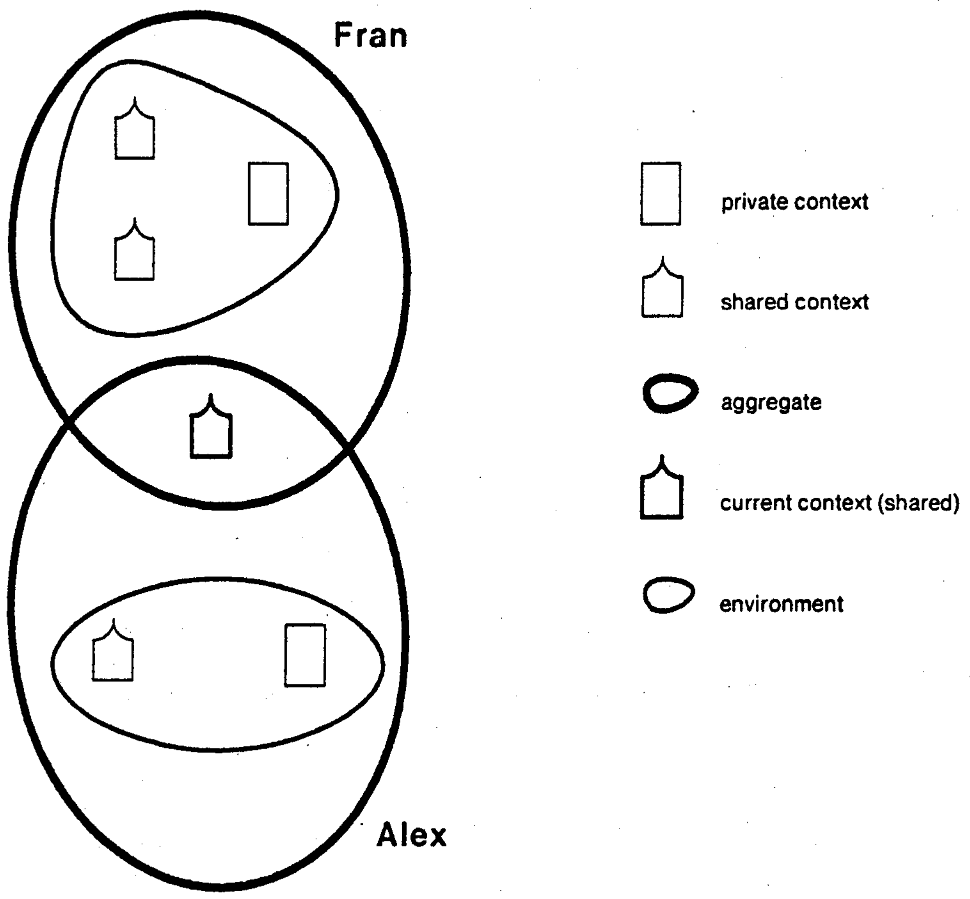

Figure 1-1: Two Aggregates Sharing a Current Context

2.1. Constraints on the Problem

Two sets of constraints on the designer of a naming facility arise from the needs and limitations of the clients and limitations of the underlying system. In this section, first the term federation will be discussed and then the issues in human naming relevant to the problem at hand will be considered.

The direction in which computer systems have been moving has been toward a multiplicity of machines interconnected by networks providing a communication medium. The concerns of privacy and independence from other users have always been issues among computer administrators and users, but the nature of those concerns have changed somewhat as smaller cheaper computers have become available. In many cases, administrators purchase such computers and put them into service in isolation. At some later time, the administrators decide to connect the computers under their management. From here, the collection may continue to grow with little control or consensus among the participants in such a "system". An *autonomous* computer is one for which all decisions are made independently of the decisions made for any other; all the activities on one computer are isolated from the activities of any other. Many administrators have pursued this option in order to escape large time-sharing systems. A *federation* is a loose coupling of computers to allow some degree of cooperation, while at the same time preserving a degree of autonomy. In a federation, there is some agreement on behavior and protocols to be utilized, but the barriers apparent in the isolated machine are still available to anyone who wants to enforce them. If the administrator or user wants to disconnect the computer from the network by simply not accepting messages, that is possible. If that computer provides a service to the participants in the network, they must understand that such a service will not always be available. On the other hand, federation provides the common ground for communication (such as agreement about protocols and services to be available) should it be desired. The loose coupling labelled federation is my underlying system model.

The human clients of a computer system have been trained since early childhood in using a naming framework for communicating with other humans. A move toward the mechanisms used among humans would be an improvement in the naming facilities provided by computer systems. The following seven observations about human use of names will clarify my basis for an improved computer naming facility.

1. **Communication:** *Names are the basis for communication. Therefore sets of names used by individuals should be sharable, reflecting common interests and communication patterns.*

2. **Multiplicity of names:**

   - *Different people use the same name for different things.*
   - *Different people use different names for the same thing.*
   - *A single user uses different names for the same thing.*
   - *A single user uses the same name for different things in different situations or at different times.*

3. **Locality of names:** *A person uses sets of names to reflect his or her focus of interest. A user also may use two or more sets of names to reflect a focus between or including several contexts.*

4. **Flexibility of usage of names:** *Humans use several sorts of names. For example, names are often descriptions. People use both full and partial descriptions. Humans also use generic names to label classes of objects. These generic names may be labels or descriptions. In fact, humans often use combinations of generic names and descriptive names in order to narrow the set of objects that are named.*

5. **Manifest meaning of names:** *The words used by humans for names have meanings*

constrained by human languages. These meanings are understood by other humans as well.

6. **Usability of names:** *Humans are able rapidly to define or redefine names and shift contexts on the basis of conversational cues. They also have mechanisms for disambiguating names, such as querying the source of a name for further information.*

7. **Unification:** *Humans use only one naming system for all kinds of things.*

These observations will be discussed further in light of the scenarios presented as examples later in this section.

A naming facility performs two basic operations. The first is to translate a name into an entity. This is the more common operation and is generally thought of as the function of a naming facility. The second is to answer the question of whether or not two names indicate the same entity or not, a question of equality. The former operation serves as the basis of much of the work presented in this proposal. The latter is frequently addressed in the literature by assuming that a mechanism for assigning unique identifiers or *uids* to entities solves the problem. Neither the problems of equality nor a solution to those problems is that simple, and therefore will be considered here briefly before returning to issues of name translation.

There are two issues that must be addressed in considering the question of equality. Of primary importance is the question of what is meant by equality. In fact, the issue can be moved back further to consider what is meant by an entity. What may be viewed as an entity by one client may not be viewed as an entity or may be viewed as a different entity by another client. Consider the situation in which one client creates a table of the results of an experiment. To that client that table is an entity. Unknown to the client, the storage facility for the statistical analysis package stores data as files. Is the collection of bytes comprising the file the same entity as the table? Should these two have the same uid or different ones? Are the file and the table the same entity even though very different sorts of operations are allowed on one than on the other? These questions will be answered differently for different scenarios and meanings of the word equality. No one answer will suffice in all situations.

The second issue of importance in addressing equality is whether or not uids can be created within the largest name space imaginable for the system model proposed, even if the lifetime of such a system is limited[1]. The model of a federated system precludes truly unique identifiers. If identifiers are to be unique. there must ultimately be a single source that is responsible for guaranteeing uniqueness (although this may be done as a cooperative venture). In order to avoid having to appeal

[1]Theoretically, in a system based on uids, the uids will never be reused. They are unique over all time, making the problem of creating them even more difficult.

to a single authority for every uid, a group of them might be distributed to potential clients for future use. Federation implies that an individual can dissociate itself from the federation for an arbitrary period of time. If the private pool of available uids is completely consumed during such a period, the temporarily isolated site may be unable to proceed with creating new entities locally, for a lack of uids. This is a contradiction of the assumption that nodes can operate in isolation if so desired. In contrast with such a system, in this work I will develop a mechanism that is not dependent on the provision of uids to support it.

The next two subsections present scenarios that will be used to consider in greater detail the issues raised by the underlying assumption of a federation of computers and the seven observations about naming. The discussion in Section 2.4 will also present a categorization of some further problems.

2.2. The mail scenario

Consider the following situation. A group has been formed at the State University (SU), chaired by Robin Rosenblum to discuss a naming service for SU (the group to be known as the Name Service Group or NSG). Alex Anderson is a particularly active member, and finds in discussing it with friends that Fran Fairweather of City University (CU) across town is also interested in naming and is a member of the National Protocol Committee (NPC). Alex reports Fran's interest to Robin, who invites Fran to join NSG at SU. Now the NSG consists of, among others, Robin, Alex, and Fran.

Because the computers at SU, CU, and NPC are connected to a network, communications among the NSG members is by electronic messages. Robin sends out a message to introduce Fran to the group (Fig. 2-1). To facilitate this and future messages, Robin creates a private mailing list, containing the names:

me
Kiddo
Fran from CU and NPC
...

It turns out that Alex and Robin attended the same elementary school and Alex acquired the nickname "Kiddo", which remains a private joke between the two of them. This message is followed by one from Fran recommending Terry Thorpe of the Subcommittee on Name Assignment (SNA) as a replacement (Fig. 2-2). In order to send this message, Fran creates a mailing list containing:

Robin
Alex of SU
me
...

The name Alex needed the additional descriptor because Fran also knows Alex Ashmont at CU.

```
From:    Robin Rosenblum, Chair, NSG
To:      Robin Rosenblum
         Alex Anderson
         Fran Fairweather, Rep from CU and NPC
         ...
Msg:     Fran Fairweather of City University and the National Protocol
Committee is joining our group...
```

Figure 2-1: Message announcing a new member of NSG

```
From:    Fran Fairweather, Rep from CU and NPC
To:      Robin Rosenblum
         Alex Anderson
         Fran Fairweather of CU and NPC
         ...
Msg:     I find I am unable to meet all my commitments and therefore
recommend to you Terry Thorpe of CU and the NPC, who currently chairs
the Subcommittee on Name Assignment of the NPC...
```

Figure 2-2: Message indicating Fran's replacement

2.3. The file scenario

Before moving into a discussion of how this exemplifies the observations discussed earlier, I will carry the example further. The NSG has been meeting regularly and decides that Alex and Terry will write a draft of a specification based on the discussion and work that has been done both at the two universities and at the NPC and SNA. In order to do this they collect the names of the files containing all the appropriate documents into one spot and then proceed to create a new file for the new document. In fact, in order to share the burden of writing, after private discussions, they decide that Alex will write the introduction and specification of the database, and Terry will write the specification of the client interfaces to the database. There will be a separate file for each and a small file called *name.spec* for creating the document as a whole. The file containing the sections are *intro.txt*, *database.txt*, and *interface.txt*. The SNA had produced a document earlier, a specification for name assignment. Its authors also created a file named *name.spec* containing *intro.txt*, *policy.txt*, and *implementation.txt*. While working on the NSG document, Alex and Terry develop the names *spec* for their own document and *SNA spec* for that of the SNA specification. Although, the story could be continued, enough of it has been told to provide two sets of examples of the issues raised in Section 2.1.

2.4. Review of the problem - human naming

This section will address each of the seven observations listed earlier. It will also highlight a set of problems that a name server or name resolution facility will face. These additional four problems will be discussed further in Section 3.

1. **Communication:** All the names used in the two scenarios were being used to communicate among people, to share the entities themselves, or to share information about them. For instance, the set of names of the sender and recipients of a message allows any recipient of it to know from whom the message came and who received it. The names for files allow Alex and Terry to share them, and later provide access to others.

2. **Multiplicity of names:** There are a number of examples of multiple names, both in the sense of using different names for the same thing and using the same name for different things. For instance, Robin uses *me* while Fran uses *Robin* to identify Robin Rosenblum. Meanwhile Robin uses *Kiddo* and Fran uses *Alex of SU* for Alex Anderson. The issue arises again in collecting files. Alex and Terry name their master file *name.spec*, but find for convenience that they are calling it *spec*. *SNA spec* is also an alternative name for the SNA file *name.spec*. These multiple names for a single item may involve one or more people. Thus, the principle of multiplicity arises frequently and naturally.

3. **Locality of names:** The issue of locality arises at several points in the two scenarios. Each of the universities is represented as a context. When Fran wants to include the name Alex in a list of recipients the specification of Alex by university is needed. The NPC and its subsidiary SNA also reflect contexts, one possibly nested within the other. The NSG reflects a confluence of these four contexts. It is possible that, for instance, there is yet another Alex at SU, but that within the context of the NSG the name Alex is unique. There are other names that have meaning within the context of NSG, *name.spec* and *intro.txt*. Many names have meaning in more than one context. In fact, the name *me* can be seen to have different meanings in different contexts. There are also names having meaning in only a single context, such as *database.txt* and *implementation.txt*. Locality is something that is used by humans all the time in order to avoid having to provide a unique name over all experiences for something being named.

4. **Flexibility of usage:** There are several different sorts of names that humans use in addition to unique, or relatively unique, names. The scenarios provide examples of three of these. In creating a list of recipients, Robin uses two, nicknames and descriptions. All three names, *me*, *Kiddo*, and *Fran* are nicknames or alternative or shortened forms of the people's names. Nicknames were also used for the two sets of specifications, *spec* and *SNA spec*. The phrase *from CU and NPC* in Robin's list can be considered either a description or a title. When the mail is created the title *Rep from CU and NPC* is added to Fran's name. This will be called here a **generic** name, a name for a classification of an entity. Later the same name *Rep from CU and NPC* will be applied to Terry. In fact, it is certainly possible that both Fran and Terry might be considered under that name at least for a while. It is also possible that Fran will forward any messages for the *Rep from CU and NPC* to Terry, recognizing them by that generic name. To Terry *from SNA* might also be added. Then, in the context of NPC, Terry might be thought of as the representative from any one of CU, NPC, or SNA, or as the representative from all. A human should be able to use logical combinations of names as well. For instance, one might want to talk about the person *from CU and from NPC or SNA*. The possibilities vary widely, and the

choices should allow the client of the naming facility as much flexibility as possible. The three kinds of names mentioned, nicknames, descriptions, and generic names often overlap. It may be difficult to determine whether a particular name is one or another of these types. The categories are useful only insofar as they are distinguishable.

A generic name or partial description often names a collection of entities. Selection from among that collection may vary with the use of such a name. At times the whole collection will be used. On other occasions only one or several will be used. This selection is determined by the use of the name and therefore should not be part of the function of name translation, but part of the function of the client.

5. **Manifest meaning of names:** Clearly, in both of the scenarios, the names discussed were meaningful to humans, and all could be passed among people outside the computer system. Certainly the names of people, and especially descriptive names of people must be manifest. But, the point can be made even more strongly in the file scenario. If the file system had determined that the master file for the SNA project should have the name *1100010011011011* several problems might arise. First of all, this would have little meaning to a human because it is not mnemonic. Second, it is unlikely that such a name is easily transported outside the system; somewhere in one of the transcriptions a human is likely to make a mistake in all those 0's and 1's, because their sequence is meaningless to the human. Certainly *name.spec* or *SNA spec* would be much more meaningful and therefore more likely to be usable to a human. Names should not be determined by the needs of the computer, but by the needs of the human users. Both computers and humans must create names meaningful to other humans, so that the names can be transported and used outside the computer system as well.

6. **Usability of names:** Usability is a result of making the meaning manifest, although several other factors make a naming system usable. Changing the name of the CU representative from Fran to Terry should be as easy an operation as it is for the human. In the same way, creating some means to name all the files that make up the SNA specification from within the context of the NPC project should be easy to do. It also should be easy for Terry to move from working on the NPC specification project, using all the names for that project, to the CU context, in which *Alex* means Alex Ashmont of CU. It certainly should not be the case that Alex Anderson need to know the names of the files in the SNA project, in order to create the first version of *name.spec* (and possibly avoid that name) for the NPC project. Finally, there needs to be some means for Fran to figure out, beyond a reasonable doubt, which Alex or which name.spec as been named in any given situation. Disambiguation is often a difficult problem for humans to solve and therefore is likely to be a difficult problem for naming systems designers to solve.

7. **Unification:** Finally, the previous six points should make it clear that the same observations hold true for naming people and files. This can be extended to all other sorts of naming. People do not have different mechanisms for naming different sorts of entities. A computer system that is supposed to be providing a friendly environment also should not require that the client use different mechanisms without good reasons. This is a strong argument for avoiding the standard system approach of including a naming system for a class of objects in the manager of that class.

2.5. Review of the problem - additional issues

Up to this point, I have discussed issues of naming related to sharing and communicating names. Once a client has received or chosen a name, there are four things that might be done with it: pass it to someone else, save it, ask the naming facility to translate it, or ask the naming facility whether it is the same as something else. There are variations on these. For example, the recipient might want to save a name for an entity with some additional names, such as further description or a nickname. The recipient might use the functions in combination, by discovering whether the name is really naming the same entity as another name, and then passing along the second name. For simplicity, I will address only those mentioned first. All four operations require knowing to what the name is relative. When passing a name, the recipient must know which namespace the sender intended for the name. When a name is saved, the namespace within which it is relevant must also be specified. Certainly, for the latter two operations a namespace must also be provided. Passing or saving a name othersie makes no demands on a name service. The last two, name translation and determining equality, comprise the basic functions of a name service and therefore are of great importance in this thesis proposal.

In addition to the seven issues surrounding human naming, there are a number of problems that on may encounter, especially using a federated computer system. One problem arises from locality of naming especially in electronic message systems when names cross boundaries of localities. Suppose Alex sends Terry a message, indicating Computation Center as the return address. To Terry at a different university Computation Center may have a different meaning. In fact, worse yet, *Alex at the Computation Center* may mean Alex Ashmont at CU or Alex Anderson at SU. It is possible that the content of the message will clear this up, but if it is simply "Meeting at 2pm tomorrow. Okay?" Terry may respond to the wrong person. This problem has been labelled the **reply-to** problem; it is a problem of not being able to distinguish two identical names that have different meanings in overlapping contexts. Often it is addressed in human communication by further questioning or exploring, if that is possible. In a message or mail situation, such questioning is not always possible.

The **name-equality** problem is a reverse problem. It may arise when Fran recommends Terry as a replacement in NSG. At CU, people often use initials as their identifiers; Robin has received messages from a TLT at CU and wonders whether Terry Thorpe is the same person. Because of the federated nature of the electronic message system, Robin may or may not be able to find out. This is a problem of authenticating both the names. If TLT and Terry Thorpe had some means of authenticating themselves, then the authenticators could be compared for equality to discover whether they were the same person or not. In a federation, this problem may not be solvable.

The **who-is** problem is similar to the **name-equality** problem, but goes beyond the computer system. When Robin receives the message from Fran suggesting Terry as an alternative, Robin may want to know whether that Terry is the Terrance who introduced himself at a party last week, or the Theresa with whom Fran was lunching at the faculty club yesterday. Both had been introduced as Terry. There may be no requirement that the computer contain people's full legal names, so there may be no guarantee that Robin could discover the answer to the question of identity in this case.

Fourth, as occurred in the scenario, a name or title may change hands from one entity to another, for example the title *representative of CU*. Initially, it was assigned to Fran. Later, it was assigned to Terry. At some other time *representative of CU* and *representative of NPC* may be assigned to separate individuals, rather than both to one person. Most naming facilities do not provide for names to be reassigned, especially across computer boundaries. This problem will be labelled the **mobile-name** problem. One reason that this problem arises is that, in general, the location of a foreign entity is often used as part of its name.

The fifth and final problem is one of **selection** of a translation. Descriptions and generic names may easily refer to more than one entity. Which translation should be chosen in one of these cases cannot be known by the naming facility. In one case, the client may not care, but wants exactly one, and will chose the first. In another case, the client may want all the translations, for instance if this is a mailing list. In yet another situation, the client may want a guarantee that at least one is contacted, but does not mind if more than one is contacted. There are likely to be other selection procedures needed by clients at other times. It is the job of the name facility only to translate or authenticate names, but not select among them. On the other hand, this problem is closely related to naming, and will be addressed in the thesis.

This last collection of five problems is often not fully resolved when people use names in conversation or prose. The thesis will address them, but since in our everyday lives we have not solved them completely, any naming facility that does, will be imposing something on its clients that they might not do to themselves.

The following section will present a model for a set of mechanisms that adhere to the seven observations listed above. This model for a solution will be applied as an example in the file management or document preparation scenario.

3. Model for a Solution

The solution to the problems of creating a more comfortable naming environment for people will have its basis in two ideas already presented as part of the problem, **context** and **aggregate**. These ideas will be extended and formalized, in order to make it possible to design an implementation of them. The basis for this proposal is a simple type of object called a **context**. A context translates names into entities from the point of view of the user of the context. A name is a label that allows the user of that label to refer to the entity by the label of his choice. In some cases, a name will be translated into another name less meaningful to or less easily used by the user of the original name. Further context translation may then be requested. In the remaining cases, the user or program will use the resulting translation as is. The result may be the full name and address of a recipient, the first hop in a route to the recipient of a piece of mail, or one of many other possibilities. Whether further translation is needed or not, the decision is not made within the context but by the client ,whether user or program, requesting the translation.

In addition to contexts, I propose another mechanism, **aggregates**. An aggregate has two parts, the **current context** and the **environment**; an aggregate is an individual's name resolution space. When two people communicate, there is a small set of names that they use regularly and to which they may add new names needed in that conversation; it is this current context that they share. They each also have a large pool of contexts from which to draw names into the current context. These pools may be different for each participant in the conversation. The pools, which are called their environments, consist of collections of contexts, which may or may not be partially ordered, but which are needed to translate names not in the current context. The current context is shared by the participants. Other contexts may also be shared. A context that is the current context of one conversation may be one of the contexts in the environment of another conversation. The reader should be aware that although the aggregate mechanism is based on the idea of human conversation, it will have a more general use. The concepts of current context and aggregate are extensions and modifications of the ideas of working directory and search rules used in may file systems.

3.1. An example: document preparation

In order to understand contexts and environments better, their application to the document preparation scenario is presented first. This will provide a basis and better understanding for further discussion. There are a number of methods by which the activities needed here can be achieved. One such set will be described here, that will be generalized later.

Consider Figure 3-1, in which Alex and Terry are discussing the name server specification on which they are jointly working. Each has a personal context. In addition, Alex can access the context of SU. On other other hand Terry has access to the contexts for CU, NPC, and SNA. They share two contexts, NSG, which is also shared with the other members of NSG, and the current context, which reflects the current topic of interest, the production of the specification. Note that, in Figure 3-2, the NSG context is the current context of the aggregates depicted. That figure represents the situation

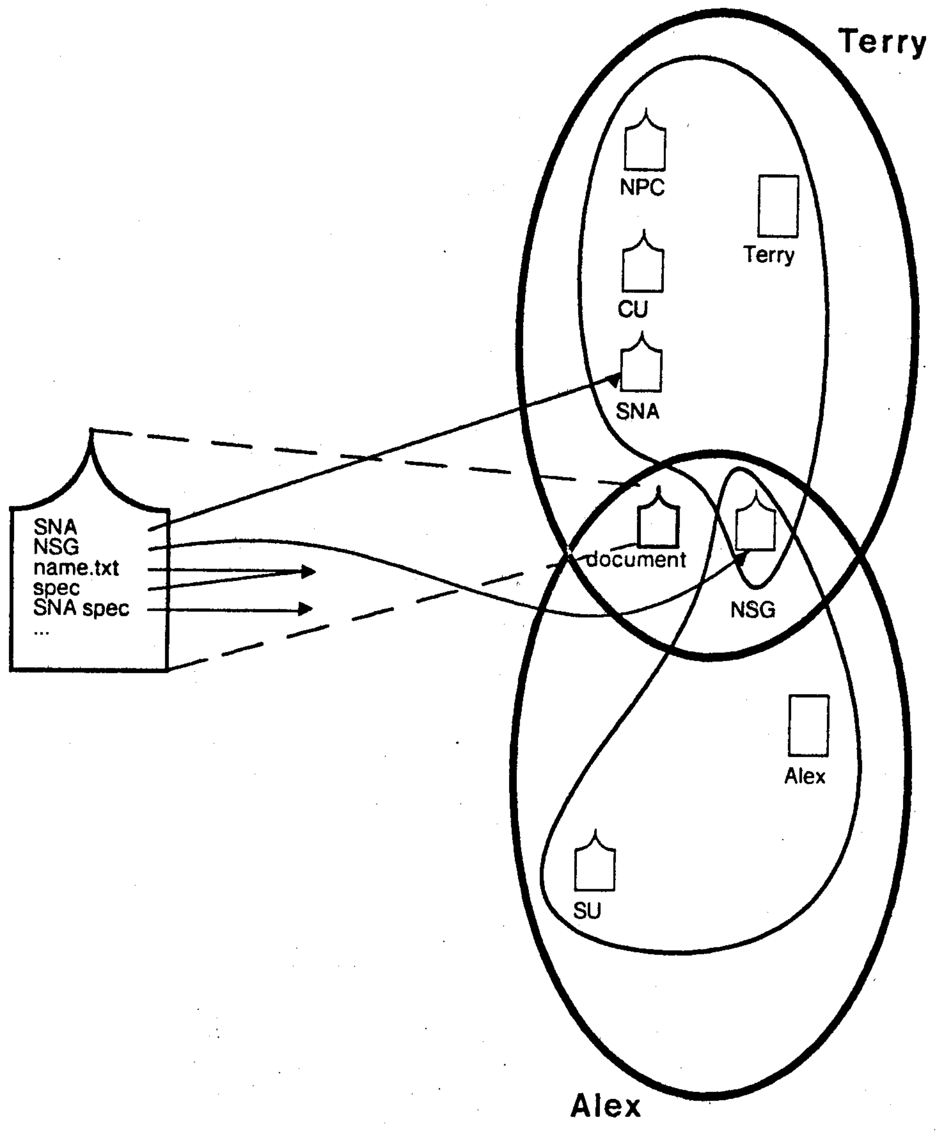

Figure 3-1: Alex's and Terry's *Aggregates*

when Fran has become a member of NSG and has not yet recommended Terry as a replacement. Terry has been included to exemplify additional sharing of contexts, although at this point Terry presumably is in an aggregate having a different current context. The entries in the *document* context, the current context in Figure 3-1 will contain among other names, *SNA* and *NSG* which point to those contexts, *name.txt* and *spec* which point to the same object. and *SNA spec* which point to the specification in the SNA context. There will be many other entries as well, defining all the names upon which they have agreed and for which they have found a use. The *document* context reflects

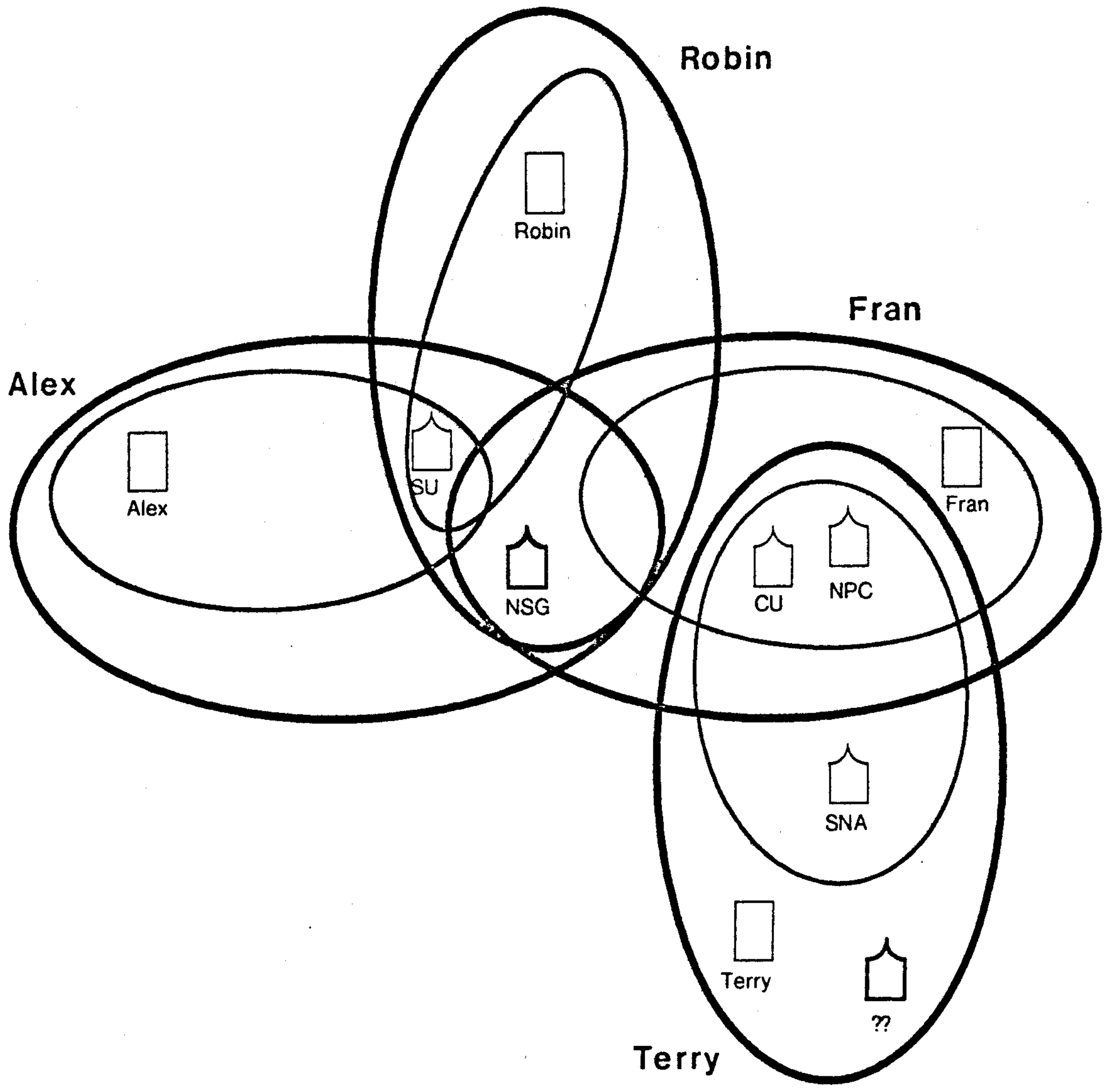

Figure 3-2: Aggregates in the NSG

Alex and Terry's continuing discussions and progress on the names server specification.

Given this arrangement of contexts and aggregates, there are a number of operations that Alex and Terry need. Initially they must create the *document* context and their own aggregates incorporating *document* as the current context. By doing this, they will be creating their own personal views of the namespace for the discussion and writing of the name server specification; each will need to invoke an operation creating a new aggregate specifying that the current context is the one being created by the other. They will then be able to name their aggregates or views of the namespace by the current context i order to move into that namespace at any time. For example, if Terry walks into Alex's office and says, "Is this a good time to continue what we were doing yesterday?", Terry will have some aggregate (current context) in mind. Alex may then ask, "Do you mean writing the document?" Once the subject matter has been agreed upon, the two will have moved themselves into the aggregates having *document* as the current context. Both creation of a new aggregate and moving into an existing aggregate must be simply expressed in a natural human interface.

Now, once the aggregates are created and can be accessed, several further issues arise, falling into two major categories. The first is the question of how multiple views of the current context can co-exists. The second is how the aggregates help the client make use of names.

The fact that Alex and Terry can be on different nodes using a shared current context called *document* highlights a problem; where should *document* be located? Since both Alex and Terry share *document*, both should be able to include it in the environments of other aggregates. For example, *document* may be incorporated into Terry's aggregate for CU, a general aggregate for discussing activities at CU. If *document* is located on Terry's node, then Terry can make use of it whenever necessary. But Alex cannot be guaranteed access. The reverse is true if it is located on Alex's node. Finally, if a neutral site is chosen, both may be denied access. The conclusion is that they both should have copies. Maintaining copies is a more accurate model of what happens in face to face conversations; each participant has a private copy of the information inside his or her head.

This leads to a closely related question of how the multiple copies are kept in synchrony. Synchronization is a particularly difficult problem because the desirable behavior pattern can very so much depending on the application. In conversation between two people, the general pattern is that after a small amount of negotiation a name is agreed upon by both. Ideally, the computer system would mirror this. Perhaps after at least one reference to an entity not in the current context by at least two participants a name can be added to the current context. This would mean that the participants would not need to add new names explicitly, although tacit agreement to some degree would be required. Explicit additions should also be possible. Part of this issue is how the copies of other participants are updated either if they are "listening" although not presently active and if they are unavailable at the time. Synchronization needs further research.

Now, consider how aggregates can improve Alex and Terry's usage of names over what they might have had previously. Consider that Alex and Terry are writing and assume that they share a log of activities so each knows what the other has done. Terry decides to print what has been accumulated and says:

Send spec to printer. (1)

At a later time Terry says:

Send SNA spec to same printer. (2)

And finally after more work, Alex says:

Send spec to Terry's printer. (3)

This example highlights many of the functions that the name service provides for them. When Terry use the name *printer* initially, there may be been no name *printer* in *document*, but one each in the SNA and NPC context. Suppose in Terry's environment SNA and NPC are at the same priority. When (1) occurs, the initial name resolution will produce two printers. A sophisticated system will understand that one must be chosen, notifying Terry of the location of the selected printer. A simpler system will leave the choice to Terry, allowing for further querying to learn more about the choices. In either case the name *printer* with the chosen one may be put into the context *Terry* temporarily. (During another writing session, Terry might opt for a different printer.) It should be noted that every person carries a personal context. Names may be offered from one of those to be part of the current context, but personal contexts will probably never be shared in the way contexts that reflect a focus of interest will be shared. When Terry makes statement (2), the system must verify equality. Printers can move to different rooms, change network addresses, etc., so some non-modifiable authenticator must be used. It is very difficult to guarantee uniqueness over the whole world, but Terry only really needs uniqueness in a small part of it. Such a reasonably unique authenticator can then be used to compare against the authenticators of other printers. Finally when Alex says (3), an indirect name is being used and from then on, either participant may mention *printer* and hopefully documents will be sent to the same printer, for later collection. At this point Terry's copy of the *document* context may have a direct pointer to that printer and Alex's may have an indirection through Terry's copy of the context or may provide direct access to printer as well. Both will also include the authenticator. They have used names for communicating with themselves over time (Terry using the previously defined name *printer*), passed names in order to increase the span of the *document* context (Alex accepting the name *printer* into *document* by using it after seeing it in the log), and have performed both the functions of name translation and name equality. In addition they have unwittingly touched on some other issues such as selection of a single entity when a name is resolved into a set of more than one.

3.2. Mechanics of using aggregates

The preceding example touched on three general kinds of actions: (1) creating and moving between aggregates, (2) managing shared current contexts, and (3) naming and otherwise making use of names. The three will be discussed further in order to elaborate on and generalize the ideas already presented.

Contexts and aggregates in isolation do not solve the problem. How the client can use them alone and in cooperation with another user must be clarified. This involves the operations available for them, the behavior or state transitions visible to the users, and how they, especially aggregates are created and named. If one considers the functions provided by a naming mechanism to each of many services in a system, some of those functions will overlap with each other and some will be application or specific to a single service. It is that set of shared and more basic functions that will provide the basis for the shared naming mechanism. Following is a list of issues related to functions and behavior patterns for a naming mechanism.

1. Creating and moving between aggregates

   - Creating a new aggregate: Creating a new aggregate involves providing some information for the aggregate. If the current context of the aggregate is to be shared and the system is to provide of help with managing the sharing, the other participants in the sharing must be identified to the system. In addition, many topics of interest are based on other topics. If this is the case, the new current context in the new aggregate may be initialized based on another context. Finally, the environment of a new aggregate needs some initializing. It is possible that all three of these, the list of participants, the current context, and environment will be modified later.

     How this information is provided may vary from one application to another. Single user applications may not allow for shared contexts. As a result no names of other participants will be needed in creating a new aggregate. Some applications may provide the user with a predetermined set of names to which the user can later add new names, in which case the initial state of the current context will be application specific. Yet other applications may assume such things as a particular arrangement of contexts in the environment. In this last case, the user may have little or no choice in the initial environment.

   - Choosing an aggregate: When an aggregate is created it must be given at least one name. It is assumed that a user will have more than one aggregate in which to manage names. In order to select an aggregate, it must have a name. The name will probably reflect the focus of the aggregate. The name of the current context of that aggregate will also reflect that focus. Since there appears to be no reason to have several aggregates with the same current context and different environments for a single person, the names that an individual will use for his or her aggregates will be the names of the current contexts of those aggregates. Contexts must be namable separately, since they are members of environments. In order to reflect the human situation, all contexts and aggregates must be namable from all other aggregates. One approach to handling this is to create a context for each user that contains all his or her contexts and aggregates and then assume that that special context will automatically be a permanent member of every environment created on that user's behalf. When it comes to the point of using on of the names that applies to both a context and an aggregate, the originator of the name will know which is needed and can specify the type of the object as well as its name to distinguish. Further research needs to be done on this issue.

2. Managing shared current contexts

   - One current context or many: as presented thus far, the current context of an aggregate is a single object. In fact, the name space shared in a conversation between several people does not behave like a single object. Consider a conversation among three people. To begin with, new names are added only upon some sort of tacit agreement of the participants. If one participant leaves and later rejoins, that person's image of the name space does not include anything that was added during the absence. Only after some updating might the third person be brought into synchrony. Thus, although the concept behind a current context is a single conversation, in order to reflect human communication more accurately a shared current context should be implemented as a collection of context with synchronizing mechanisms provided.

There is a second more practical reason for providing multiple copies of a shared context. At least some of the information in a context may be only of use locally, such as relative addresses or local addresses. For example, if two communicants are discussing an object that resides on the machine of one of them, for one communicant such a reference will be local, and for the other it will not. The fact that the two translations may or may not be visible to the two communicants, but in designing a naming mechanism, one must be aware of this issue.

A third reason is availability as was discussed in the example. One of our assumptions was that an individual node could operate in isolation. For example, in the case of mail, if a node is isolated from other nodes, the user should be able to prepare mail to be sent upon later reconnection using contexts and aggregates representing ongoing conversations. This means that each participant in such a conversation must have a private copy of the shared information, available at all times. Having private copies does not imply that an individual can necessarily make updates independently; that is a separate issue.

- Synchronization: There are two aspects of synchronization that must be considered. First, there is the question of whether the client plays a role in the synchronization; is the client's approval needed for an update? The issue of client approval must be addressed at the level of the application. Some applications will need the approval of all the clients, some will need some clients, and some will assume an individual using a name will will be enough to enter it into the current context.

The other question which is the degree to which the multiple versions are kept in synchrony. At one extreme, updating duplicates can be done as an atomic transaction. At the other, versions would never be guaranteed to be in complete synchrony. Of course there are degrees between those extremes. Synchronization is an area where further exploration is needed. It is clearly infeasible to require at all times that all participants agree before an update can occur; one of the assumptions underlying the system is that individual machines and users may become unavailable, and that this should hinder other operation as little as possible. One possibility is that an assumption might be made that if the system were allowed to quiesce all copies of a "shared" context would become identical within the limitations of relative naming discussed above. There might be some assumption that an update occurs when a majority of the participants agrees, and that a log is kept in order to bring others into synchrony when needed. Of course, this in turn has problems, such as the need for a great deal of storage for the logs.

Due to humans' imperfect memory, the way that synchronization is done in human communication is that an absent participant may be brought up to date on missed information to some degree, but that absent individual may never be brought up to date fully. Only with perfect recall (such as filming an interaction) could this be done. One question here is whether providing such perfect recall is an improvement, and whether a computer system should provide improvements where possible. Certainly complete logging would provide for perfect updating. People operate quite well without perfect recall. but compensate with elaborate reasoning mechanisms to detect and correct miscommunications. For communications with and through computers, perfect recall may help compensate. This area of synchronization is a critical one for further research.

3. Naming and making use of names

- Resolving a name: There are two issues related to resolving a name, (1) into what it is resolved and (2) in what namespace the resolution occurs. The latter will be discussed separately. In this research a name is resolved into a set consisting of one or more names and/or one or more entities. For example, consider the resolution of the name of a person with whom one wants to communicate. This may simply be resolved into a mailbox for electronic mail. But it may be more complicated and be resolved into a list. One item on the list may be that same mailbox. Another may be the person telephone number. A third may be the name of the person's secretary. In the last case, the resolution will produce a name of the same form as the name being resolved. One problem that can arise from this situation is a cycle in the names, and if there is concern that cycles not occur then there must be a mechanism to avoid them.

  One implication of allowing a name to be resolved into such a list is that names are not typed as entities are. This is at odds with many programming languages where variable names are typed either implicitly or explicitly and once a name has a type within a namespace it can never change. In fact, most names, as humans use them, except proper nouns represent descriptions. Consider the set that comprises the translation from a person's name. That set may be labelled with the person's name because the set represents the set of ways to reach the person, in a sense means of accessing the person. So although the mechanisms presented here may not match the programming language approach to typed naming, all naming other than proper nouns is typed naming. This then leaves the problem of how to determine which resolution to use when a name is resolved into a set containing more than one entity. Selection will be discussed separately.

- What name space: Aggregates provide the basis for the namespace to be used for translation of a name. One way of viewing them is as a greatly enhanced version of working directories and search rules. The current context corresponds to the working directory, the first place in which to consider name resolution, and the environment corresponds to the search rules, the set of alternatives in which to search for possible name resolution. The major differences are twofold. The first is the lack of any particular organization for contexts or aggregates. There is nothing in contexts or aggregates to prevent cycles or any other structure based on names, allowing the user to create any organization he or she wishes. The second major difference is that contexts are not limited to naming just files. or just something else specific. A context can contain names for anything that is namable in the system including other contexts, aggregates, people, data objects (possibly files), procedures, types, processes, resources, services, etc.

When a name is to be resolved, an aggregate must be invoked either explicitly by naming it or implicitly by being the "current" aggregate. The current context will be searched first. If the name is not found there, the entry in the environment with the highest priority will be searched. The entry with the highest priority may be a set, in which case all members of the set wide search. When a translation is found, the translations from all contexts at that priority in the environment will be returned to the client.

- Selection: Since it is assumed that a correct response to a request for translation is a set of names or entities, in general some selection procedure will invoked. The decision about which of the set to use is application dependent, although there are some common approaches that should be provided, such as "any one" or "all". In fact, the application may go further than that. Suppose, for example, it is a mail system. Some messages should be delivered to exactly one member of a group, others to at least one member, and still others to all members. Each of these involves a selection procedure on the part of the application, in this case the mail system. Certainly, procedures for selecting one or all members of such a group should be provided as a service to the subsystem builder, although some subsystem builders may want other more sophisticated selection procedures.

  Perhaps special contexts should be provided that have the added feature that each name has no more than one possible resolution, at least within that context. This requirement may be common enough that rather than requiring constant checking each time a modification is made to a context or using a such a selection procedure for every name resolution, it should be provided in some contexts. Indirection would still allow an escape from such a restriction.

- Passing references to other communicants: As part of communicating, there are three ways in which names can be used. The first is to translate the name into an entity and pass the entity. Consider the situation in which one person is telling another about an interesting book. In one case the speaker might say, "This has some interesting ideas in it," and hand the listener a book. There is no guarantee that the two people share a name for the book. The second is to pass the recipient a name that the recipient can resolve. In this case the speaker might say, "The book by Quine has some interesting ideas in it," leaving the recipient to find the book. The assumption here is that the two people share a name for the book, so that either can resolve that name. The third situation is one in which the speaker passes a name and an object to the recipient on the assumption that that name translation pair is one that the recipient does not know. In this last case the speaker might say, "This book by Quine has some interesting ideas in it," while handing the book to the recipient. In the future, if both have agreed to include "book" and "written by Quine" in their current context pointing to the book that was just handed from one person to the other, then they can use the second approach of simply using a name on the assumption that both can translate it.

  In the third situation above, if the assumption is made that names are only added to the current context after some agreement has been reached, the original translation by the speaker must have been done in some other context that was part of that person's environment within which he or she was operating. It is by this means of passing names and translations from one environment to another sharer (or all sharers) of the current context that new entries will be generated for the current context. There is only one other source of new entries to the current context; consensus on a new name for an entity given a name and translation for it. It is this sort of name creation that Carroll discusses in his paper on name creation [2].

3.3. Review of the model

The seven basic principles provide a basis for discussing a naming mechanism. The preceding mechanism will, therefore, be discussed using the principles as a guideline.

1. **Communication:** The partitioning of an aggregate into an environment and a current context provides a balance between what the individual brings to the communication (the environment) and that part of communication that involves sharing and cooperation (the current context).

2. **Multiplicity of names:** As can be seen, Fran can name two Alexes. In fact, the mechanism easily provides for the four options listed on page 4. Contexts allow different names for the same entity and the same name for different entities. Aggregates allow several people to share some names and at the same time use some names of their own choice without interference from other people.

3. **Locality of names:** It is the locality of aggregates and within that contexts that make the usage of multiple names possible. Aggregates allow clients to reflect an issue or topic of interest in the naming mechanism, as humans do in discussion. In fact, aggregates as described thus far, reflect each individual's view of a conversation or topic.

4. **Flexibility of usage:** In the examples, names are not restricted in length, content or quantity. The design of an implementation requires careful thought about how such a facility might be built. There is no restriction that names be unique; the same name can be applied to more than one entity. Logical combinations of names are handled in the procedure that translates a name. Several possible implementation strategies exist; these strategies correspond to methods used for *joins* and *projections* in a relational database. Choice of strategy is part of the design of the name implementation

5. **Manifest meaning of names:** The facility allows for names to be manifest, because the client chooses the names in contexts. Therefore the names can be chosen in order to reflect whatever meaning the client wishes to impose. What is more, the client or other clients can add additional names, providing for another route to naming an entity, and allowing for private names to be used for entities.

6. **Usability of names:** Usage of such a naming facility will, in the end, fail if the facility is not easy to use. This will be investigated in the implementation. Various operations must occur quickly and efficiently. The operations that are used frequently should be easy to use, and not time consuming.

7. **Unification:** It is clear from the examples above that names have been united within a single facility. The client may find that for convenience he or she will create contexts for different types of entities, but the naming mechanism does not impose that restriction. In fact, the user has the possibility of using one name to label several different sorts of

things. It should be noted that this view of names conflicts with "strong typing" which associates types statically with names. Consider that one of the objects named in the NSG context might be a list of ways of accessing different members of the group. For each name, there may be an electronic mail address, a telephone number, and a U. S. postal service address. The electronic mail box and the telephone number have little in common from the point of view of the computer, but to the client, they have a great deal in common. In this case, a collection represents ways of accessing some one. At other times, it may be useful to organize this information as a list of electronic mailboxes, providing an electronic mailing list. In this case, the collection represents a single route to accessing each member of a group. Incorporating the goal of a unified set of mechanisms provides the client with greater functionality and flexibility.

The mechanism of aggregates also addresses the four problems identified Section 2.4 labelled **reply-to, name-equality, who-is, mobile-name.** The **selection** problem will be discussed further, among the problems to be addressed in the thesis. The naming facility cannot solve any of these problems completely, because humans have yet to learn themselves how to avoid them at all times. The problems can be ameliorated by a mechanism such as that suggested here, for instance, because contexts and aggregates are labelled, helping to avoid ambiguities and confusion. At the same time, because names need not be unique, a certain degree of ambiguity will survive. Both the name-equality problem and the who-is problem arise from a client not informing the computer of all the information that the client later would like the computer to have. In human to human situations, what occurs is that one participant provides additional information or asks more questions. That is not always possible, for instance, if the client is a program and a server needs to ask it questions. The client may or may not be prepared to deal with queries. The mechanism should shield the client from the mobile-name problem to the extent that location of an entity is invisible to the client. None of these problems is solved completely, but all are addressed.

The next section introduces the reader to some of the work related to this research. Some of those papers discussed had a strong influence on this work, others did not, but present closely related ideas. The proposal concludes with a section describing a plan for the thesis including: (1) a partial list of problems to be solved; (2) a brief discussion of an implementation of some of the ideas; (3) what might be gained from such an exercise; (4) what the limitations are of such an exercise; (5) a schedule for completing the thesis.

4. Related Work

Because this work has its roots in several areas of Computer Science research (systems, languages, databases, communications) as well as linguistics, a discussion of related works is interdisciplinary. The list has been restricted for the thesis proposal; for the thesis it will be more inclusive. The discussion will also be limited only to those aspects of each that are relevant to the subject at hand,

and the ways in which this work differs from those mentioned. The references will be discussed in three subsections: those that have had a particularly strong influence on the nature of names and objects as reflected in this thesis, those that reflect locality of reference and contexts, and, finally, various forms of names, in particular descriptive names, generic names, and aliases or nicknames. A number of the documents will fall into several of these subsections.

4.1. The Nature of Names and Objects

One issue regarding the nature and kinds of names is whether various names share some qualities. Is there justification for a unified naming mechanism? Both Quine [18] and Carroll [1, 2] suggest that the nature and use of names does not vary with the kind of object that is being named. The work of both of these authors is concerned with human communication and the nature of names. Both authors have studied underlying human name usage patterns and conclude that, although humans use names as labels, they also impose semantics on names. Therefore, I conclude that a computer system should allow humans the flexibility to create their own names suitable to the semantics desired.

Most operating systems have naming mechanisms that distinguish among different kinds of entities that are namable by providing different naming facilities. Both Shoch [23] and Saltzer [21] find differences in the nature of the names or labels for various kinds of namable entities in networking. They disagree slightly on the categories of namable entities, but both are concerned with categorizing the entities as a basis for differentiating categories of names.

In contrast, this thesis proposes to provide a common naming mechanism shared by several applications as discussed by Saltzer in [20] and provided in Swallow [28, 27], CAP [13, 14, 12], and the Clearinghouse [15]. Saltzer carefully analyzes a fully hierarchical naming scheme, modifying it to produce full locality, while keeping it general enough that the same mechanism can be applied to virtual memory management and a file system. A conclusion to be drawn from Saltzer's work is that such a mechanism could be applicable to any situation in which a purely hierarchical naming structure was desired.

The Swallow project [28, 27] carries the idea of utilizing a shared need for a mechanism beyond naming and into storage as well. Swallow will store anything that anyone wants to store, as a collection of bits with no assumptions made about typing or internal structure of what is stored. It is this concept of a common need for a mechanism, albeit for a naming mechanism, that is one of the driving forces in this thesis.

The CAP File System [14] has a common maning mechanism for a large number of kinds of objects.

This file system is built on a capability system and provides translation from any kind of name into a capability. Therefore, anything for which a capability exists can be named through the file system. Unfortunately, this excludes users. although once the user has been authenticated properly, he or she is issued a capability for his or her own directory. The file system only allows for translation from a name into a capability, not into another name. This removes the possibility of delayed bindings.

The Xerox Network Systems Clearinghouse [15] is built on the assumption that anything that is namable across a distributed system should be namable through the Clearinghouse. Although the Clearinghouse sounds much closer to the work of this thesis, there are some underlying differences. Because it provides contexts, descriptive names, aliases, and generic names, all to some degree, it will be discussed again. Suffice it to say for now that, unlike this thesis, the Clearinghouse is not designed for frequent and efficient local use. Although Oppen and Dalal make a strong statement about combining pre-existing Clearinghouses so as to cause the least disruption by allowing local naming, in order to cooperate, Clearinghouses must recognize a central authority that dispenses unique names.

One of the conclusions of our initial work is that unique identification is not only impossible in a distributed environment, but also unnecessary. There are a number of situations in which this has been a basic assumption. For example, because the Cap File System is not strictly hierarchical, it allows for non-unique naming. In fact, no particular structure is imposed on the collection of directories and other objects named in it. Halstead [5] uses only relative names (addresses) for his lowest level entities in the Mu-net. As processing is passed to a neighboring processor that might be available to help with the work to be done. the names that allow objects to be located are maintained so that messages to the objects follow the paths along which objects were relocated. Source routing [4, 26, 21] also provides an example of a situation in which the same name can have completely different meanings in different situations, especially at different locations. In this case, the route used in a network is determined by the source of whatever is to be sent through the network, and is done relative to the source. Therefore, beginning at different sources, the same route will lead to different destinations. Saltzer [20] discusses reasons and a mechanism for achieving such local naming. to provide modularity at the operating system or even user level, in a centralized system. This local naming has been provided to some degree in systems such as Multics [16] and Unix [19], each of which provides a different kind of linking, resulting in different semantics for sharing. Saltzer's work and the Multics and Unix operating systems assume that the basic naming mechanism is hierarchical, and that being able to bind a name to different objects is the exception, not the rule. Only the CAP file system does not impose a hierarchy on the client while freeing the client from a requirement of unique names, as this research intends to do.

One finds, when reading the literature, that various people have done work that relates to parts of the underlying assumptions and philosophy of names proposed here, but none has pulled all the ideas together in a single work.

4.2. Locality and Contexts

Almost every system provides its clients with some degree of locality of naming, based on one or a small set of directories. Often this is only thinly veiled, and as soon as the client tries to use a name that is not defined uniquely within what the system considers to be the client's local name space, the client comes face to face with a message such as "Illegal access: no such name as *users.students.class.dbms.sollins.flmult.*" The routine *flmult* may only have been invoked on the client's behalf, yet somehow the client, who may not have a clear understanding of the hierarchical naming structure, is expected to fix the problem. We will not address the majority of work done in this area because much of it is repetitive and has not made much progress. Only those that have had a strong influence on this research or that have made progress in areas where there had not been any progress previously will be mentioned.

The works of Saltzer [20] and Carroll [2] have had the strongest influence on the concepts of contexts and aggregates. Saltzer analyzes how one might provide for sharing while simultaneously allowing for modularity and isolation from having to know about the names used by others with whom one might be cooperating. He proposes contexts and closures as a solution to his problem. His work assumes that the system is centralized. As a result, physical copies of names can be shared. Although this cannot be assumed in a federation of computers, Saltzer's work as discussed earlier has strongly influenced the work of this thesis.

Carroll's work suggests that mechanisms in a federated situation need more flexibility. Carroll studied the mutation of descriptive names into nicknames or abbreviations, in small groups of people. He led me to a better understanding of how people cooperate in creating and sharing a name space, and thence to my proposing **aggregates** as part of my solution to the problems.

Another strong influence on my work has been the Argus project [11], which proposes guardians, entities to manage resources, composed of one or more processes. Processes within a guardian share a name and address space. But across guardian boundaries communication is very restricted, and therefore naming is done at arms length. The guardian provides a strong boundary for locality. In fact, the work by Saltzer and Liskov were strong influences on previous work by this author [24, 25] based on an assumption of contexts similar to Saltzer's, but across the boundaries of which the only means of sharing was through messages. It became clear from the work of Carroll and further thought on my part that such a mechanism did not provide enough flexibility.

Four other works bear mentioning here as having made progress in this direction in related areas: the treatise by Pouzin and Zimmermann [17] on networks, the Clearinghouse project [15] at Xerox by Oppen and Dalal, Lindsay's work on the catalog for System R* [9], and the Flavors mechanism in the language Zetalisp for the Lisp machines [29] at M.I.T. Pouzin and Zimmermann describe briefly a distributed system, in which contexts play an important role. They hypothesize that the user connecting to a system will always be in some context that will be modified with time. This is a first step in the direction that this thesis is moving, but the thesis intends to provide much more flexibility and begins from an assumption of providing a basis for communication.

Clearinghouse provides layers of naming contexts, within each of which all naming is independent of naming anywhere else. Naming within one of these contexts need not specify the context, and, in fact, one of the underlying assumptions in designing Clearinghouse was that pre-existing networks and systems would be able to join without having to revise all their names to fit another scheme. Clearinghouse names are composed of three parts, two of which can default to the local names. There is no facility for changing the default context, although client software accessing the Clearinghouse could be written to do this.

Lindsay, in the area of database management systems, also approaches the problem of creating names in a distributed environment without having to resort to a central service. His approach is to assign each entity a unique, hierarchically structured **system-name** at the time of creation, consisting of the user's (creator's) name and site, and the object's name and birth site. When a client refers to an object, at least the object's name must be used. By not specifying one or another of the other components of the system-name, name resolution may vary from one context to another, using default values for those undesignated fields. The sites are assigned globally unique names by a central authority. This mechanism meets the goals specified by Lindsay, but would not suffice to meet the goals of this thesis, for example in the areas of flexibility and federation of sites.

Finally, the Lisp machine project has produced an interesting approach to contexts called flavors. Flavors have more to do with object type than name, but issues of scoping are addressed. A flavor is composed of components of other flavors possibly with additions and modifications. A flavor specifies among other things a set of operations that can be performed. A flavor based on another flavor can include modifications to occur both before and after any particular operation inherited from another flavor, in addition to new operations. In fact, a flavor can modify an operation provided by a flavor included only indirectly, by having been included in something else that was included. The flexibility provided by flavors is similar to the relationship between contexts and aggregates as discussed in Section 3.

4.3. Kinds of Names

In addition to basic assumptions about names and patterns of use, this thesis proposes to allow different sorts of names, descriptions, nicknames, generic names. Other researchers have taken steps to provide some of these features. This section presents a sample of this work.

Some of the earliest attempts to provide descriptive names began with allowing character string names, and then allowing multiple part names. For example, Multics [16] and Unix [19] allow extensions to file names. The guidelines set for these extensions include that they can provide additional descriptive information about the file. For example, the client might create four files with the names *testprog.pl1*, *testprog.run*, *testprog.documentation*, and *testprog.data*. In this way, the name *testprog* allows another client to find the four related files while the extension identifies the purpose and content of each. Two papers from the COCOS project [3, 7] discuss the approach in that project to allowing some attributes to be assigned to mail recipients in their mail system, COCOS. There is a fixed set of possible attributes, and the attributes can be used to help distinguish recipients. Clearinghouse carries this much further. A name has associated with it a property list, composed of an unordered set of triples. Each triple or property consists of a *propertyname*, a *propertytype*, and a *propertyvalue*. The propertyname must be registered with the Clearinghouse. The propertytype is either individual or group. If the propertytype is individual, Clearinghouse assumes no meaning for the propertyvalue. If the propertytype is group, then the propertyvalue is considered to be a list of names. This mechanism allows the client to name an entity using a property as well as the name. The Clearinghouse does not provide the full flexibility nor assume federation that this thesis proposes, although it has taken a step in the same direction.

Many systems have provided some form of aliasing and/or nicknaming. For purposes of distinction here, we will define aliases as alternative names chosen by the owner or manager of an entity, while nicknames are names chosen by the client of the naming mechanism. Multics allows users to have aliases chosen by the user being named and installed by a system administrator. It also provides a form of nicknaming in links in the client's directory. A link allows the client to equate the name of his choice to a full path name for a segment. Clearinghouse is more explicit about allowing a set of secondary names, but they must be registered explicitly with the Clearinghouse. Carroll [2] provides a scenario for the development of names or nicknames. When two people converse, one will introduce a new entity by a description. This description or part of it will be used only a small number of times (generally under three) before it is transformed into a name or nickname. There will be agreement on a name that probably is closely related to the original description. Such a transformation arriving at a nickname has not been provided in a computer system before.

Although generic names may, in fact, be closely related to descriptive names and possibly simply partial descriptions, the work in this area has only begun to move in that direction. The earliest generic naming was probably for mathematical operations. "+" and "-" were and continue to be valid for both floating and fixed point numbers in most programming languages. The National Software Works [8] attempts to allow generic naming of services to be provided on a heterogeneous group of timesharing systems. The systems are different enough that the services provided under a single name are quite diverse. As a result, in order to be useful to a client, the client needs to specify the host in addition to the generic service name thus negating the effect of the flexibility that generic naming might have provided. The ISO Reference Model [6] provides for generic naming in the Service layer. The intention here is to allow multiple applications to respond to a request for service. How this generic naming is handled is not made specific and will have to await an example in a protocol specification. The Clearinghouse attempts to provide generic names for its namable entities by allowing the client using a pair consisting of ⟨*name*, *propertyname*⟩ to indicate the first name as anything by using "*". This is very stylized and still not very flexible.

In accordance with the all-purpose mathematical operation notation, this author, in earlier work [24, 25] proposed generic copy operations to provide support for type specific copying operations in a strongly typed language, in which, generally, generic operations do not exist. The flavors mechanism in Zetalisp [29] carries the idea of generic naming of operations much further. In this case, the attempt was to share not only names, but as much supporting code as possible by including and modifying type information and operations from one type to another. The idea is that if several types share many features, the programmer should be able to make use of that knowledge of shared features to modularize along those lines. This means that a simple type may provide generic operations for a number of more sophisticated, more specific types. In all the cases mentioned except Clearinghouse, which has other limitations, generic names are restricted to a single kind of entity.

We can see in all these kinds of naming as with the concepts of contexts, aggregates, federation, and uniformity of mechanism, that the research of this thesis is bringing together ideas that may have been considered by others at least to some degree, but not simultaneously.

5. Summary

A summary of what has been discussed thus far will provide the basis for a discussion of the plan for the thesis itself.

The problem being addressed is that the naming mechanisms available in computer systems, in general, are awkward, inflexible, and overburdened with functions that are not part of naming. The thesis will address the issue of providing a more human oriented naming facility for communicating in a distributed system. In order to understand the problem better, it can be partitioned into two sets of issues. First, a model of the system on which the naming facility will reside must be considered; the model consists of a **federation** of cooperating computers. Federation has several implications. One is that a member of the federation may have had an existence prior to joining the federation. A second is that a member may leave the federation temporarily after joining. A third is that even while participating in the federation, a member may want to maintain a certain degree of autonomy.

The other part of the larger problem is an analysis of what is needed in the client interface. To this end, seven points about human communication were enumerated and are summarized here.

1. **Communication:** *Since names are the basis for all communication, sets of names should be sharable.*

2. **Multiplicity of names:** *The use of a name should not be restricted by other uses of a name. Nor should the use of a name for an entity restrict the use of other names for that entity.*

3. **Locality of names:** *Local naming should be possible.*

4. **Flexibility of usage of names:** *Humans use different kinds of names, not bound to the nature of the entity being named, but the result of choices made by the namer. Therefore various forms of naming should be available for naming any sort of entity.*

5. **Manifest meaning of names:** *Names must have meaning to users communicating with a computer as well as outside the computer between communicating users.*

6. **Unification:** *Any naming facility available for naming one sort of entity should be available for any others.*

7. **Usability of names:** *The naming facilities that humans use in human communication are easy to use. Any human to computer naming facility should also be easy to use in the same sorts of ways.*

The mechanism proposed to address these issues is based on a simple structure called a **context.** A context provides a service; it translates names into entities. There are very few limitations on contexts. To as great an extent as possible, a name is anything the client of the context wants to use as a name. Contexts need not be organized hierarchically unless the client wishes to do so. A name can be mapped by a context into several entities, and several names can be mapped into a single entity. From contexts, **aggregates** can be built. An aggregate has two components, a **current context** and an **environment.** An aggregate can be imagined to be one person's view of the name resolution support mechanism present during communication with another person. Each person brings his or her own set of contexts from previous experiences; these comprise the environment.

One context represents the shared names common to all participants; this context is the current context and reflects the focus of interest. Most of the contexts in an environment will be shared with others, not necessarily those people involved in communication at hand. With these simple mechanisms, the issues above can be addressed, as was described in presenting the two scenarios of electronic messages and file management.

6. Plan for the Thesis

The work for this thesis falls into two parts, as reflected in the next two subsections. The first stage will be to use the ideas presented here to implement a user environment for mail management. This work will be done in conjunction with an on-going project to create a new mail system. The second stage will be to continue the study of name management, both in the direction of higher level issues of functionality that must be addressed and also lower level, more technical issues of implementation. The final subsection of the thesis proposal will present a schedule and brief outline of resources needed to complete the work.

6.1. An Experiment

A mail system is currently being designed and built under the guidance of Dr. David Clark. It will provide a testbed for contexts and aggregates. This project was chosen for two reasons. First, one of the underlying assumptions is that the distributed environment for which it is being designed matches my assumptions of a federation among cooperating computers. The second reason is that the project is being done in Clu [10] at M. I. T. A similar mail system Grapevine [22] was developed in Mesa at Xerox PARC, but is not available at M. I. T. as a testbed.

There are several goals of the project itself. The primary goal is to build a mail system to serve a community using, possibly slow, small, personal computers communicating over a network and not always available. For this reason, the storage function and user interface functions are separated. One or more repositories send, receive, and store all mail. When a user wants to read mail, the personal computer will update its information about the user's mail and the user will interact only with the personal computer. When the user decides on the disposition of a particular piece of mail, the personal computer notifies the repository. If the user "sends" some mail, it first is moved from the personal computer to the repository, and from there is sent.

The second goal is to create a flexible yet helpful interface for the user. For example, the user will have access to several windows. One will be used to manipulate headers of received messages. A second will be used to read and possibly edit the text of a received message, coordinated from the header window. A third window will provide message creation facilities, for replying , forwarding, or creating new messages. Editing in this window will have two forms. The user will be provided with guidance in creating header information; for example,certain fields are required, while others are not required but permissible. At the same time the user should be free to include anything he wants in the text. The third goal is to create a mail system in Clu, so that it can be used on the VAX'es and other machines at M. I. T. while taking advantage of work already done in Clu, such as parts of the editor TED.

My work will involve certain parts of the user interface. I will write clusters for contexts and aggregates.[2] They will be utilized in all three windows, the header window, the reading window, and the sending window. A partial list of the issues that must be addressed in this implementation are:

1. Management of shared contexts such as current contexts.

2. Implicit versus explicit modification of context.

3. Implicit versus explicit management of aggregates such as adding, deleting, ordering contexts in the environment of an aggregates or switching from one aggregate to another.

4. Binding - when and to what extent names are translated.

5. Authentication:

    a. Are two names identifying the same entity?

    b. Is this the entity that I thought it was? (E.g., is this the same one I had yesterday?)

6. Selection algorithms - which ones are needed in a mail system?

7. Representation of descriptions.

This implementation will provide a useful experiment because the mail system is being built with the intention that it will be used. Since the user interface is separate from mail delivery and receipt, it should be usable for mail delivery and receipt on the ARPANET as well, thereby making a large amount of mail management a likelihood.

6.2. Further Work

The implementation will be in a single domain, electronic messages; therefore there are a number of issues that it will not address or will not address fully. Further work must be done to investigate these issues. One important issue in all decisions in the implementation must be whether or not they are specific to a particular application or whether they are generalizable. The additional work will include

[2] See the Appendix for Clu specifications of the context and aggregate clusters.

the following issues:

- **Binding:** Since names may be mapped into other names, whenever a binding occurs, there is a question of to what degree binding is done. In mail, there are several distinct sorts of bindings. For nicknames used for a recipient, a name must be translated into an address. For nicknames used within the body of the message, nicknames may be translated into full names, or simply left as is. Whether these translations occur while the message is being created so that the sender can see them or whether they are only done upon sending is another part of the question. For naming other entities in other applications, there may be other requirements. If the translation is simply to provide another name, we may need some facility for deciding whether a name is of the right form and, if not, manage further translation until the right form is found.

- **Indirect naming:** Indirect naming has not been addressed in this proposal, but must be provided in the thesis work itself. Indirect naming would, for example, allow the name *SNA spec* to be translated into a pair of names {SNA, name.spec} naming another context and a name to be translated further within that context. This would provide delayed binding, allowing the translation of *name.spec* to change without needing to change the translation in the document context. The meaning of that entry in the document context would be "that entity which is named name.spec in the SNA context." This problem is closely related to binding.

- **User interface:** As described in the example in Section 3.1 a user interface for a particular application is an implementation of the underlying model. Different applications will have different interfaces implying different expressions of the concepts. The thesis must explore some of these various views of the underlying facility and how any particular interface limits and recasts the supporting mechanisms.

- **Authentication:** Authentication may not be a serious problem in the electronic message system, although the related problem of disambiguation is. In general there is a more difficult problem of proving that a particular entity is the one it is thought to be.

- **Efficiency versus distribution:** The electronic message system will have to implement contexts that are replicated at several sites, but because of the slow speed of mail systems, updates to the multiple copies probably will not need to be reflected in immediate updates to a current context. On the other hand there may be situations in which more immediate updating is required. At the same time, using contexts and aggregates must be easy and efficient. The question of implicit versus explicit aggregate switching needs careful thought.

- **Sharing contexts:** The policies and mechanisms needed to provide sharing of contexts must be explored. The problems of sharing become more severe when the participants in sharing are on different machines and may not be available at all times. Policies and mechanisms for sharing and cooperating on updates to shared contexts must be provided as part of the protocols of managing distributed contexts.

- **Multiple implementations of contexts and aggregates:** It is possible that multiple implementations of contexts and aggregates will be needed to provide different behavior patterns under various conditions. One might want one implementation that allowed for very efficient translation, and another that provided efficient modification of contexts or environments. If more than one implementation were provided they would all have to meet the same specifications. Research is being done in this area in the Argus project, but I may also need to do some more practical work in this area in order to implement my ideas within the mail project, especially in light of the sharing mentioned above. If different implementations are allowed, the issue of whether copies of a single shared context can be implemented differently becomes even more complicated than otherwise.

- **Yellow Pages and generic naming:** Clearinghouse requires that all names and property names be registered. One reason for this is that the list of used property names is then available in order that the same property name can be used for different entities when they have the same property. This same concern must be addressed in the thesis. It might be done in conjunction with a "yellow pages" service that would allow public look-up of descriptions or generic names. It is in this area that the distinction between descriptions and generic names becomes especially blurred. Such a service might be provided by creating a context that is shared by anyone who wishes to share it. In this case, careful thought must be given to managing such a context. Should or need it be distributed? Is there any control over who can make entries or what entries can be made? Is there any validity checking to catch "false" advertising?

- **Implications of a federation of computers:** One question that must be addressed is what it means to have the name of an entity or the translation of a name. Does that mean that the entity exists? Does it have any implication for permission to access the entity? What happens to a name when the entity being named is temporarily unavailable, as happens when a user logs out of the system? Is there some way to discover whether an entity's unavailability is temporary or permanent? How is this information related to naming?

- **Selection among multiple names:** A context can map a name into more than one entity. Therefore, when a translation occurs, the client must be prepared for the response to be a collection of entities, and must be able to select among them. Selection algorithms are application dependent, but a careful study must be done of the sorts of selection algorithms needed, in order to identify common ones. There is no reason to require that each application create its own selection algorithm, if there are common ones that can be provided.

- **What to do with information that was traditionally managed by a directory system, but will not be managed by the proposed naming facility:** There is information that is not part of the name, but rather the state of an entity that has historically been managed by the naming facility. Examples of this are date and time of creation of a file, access permission for an object or service, and type of an object. A study must be done of these categories of information in order to determine how they should be handled. There are tradeoffs. For example, the type of an object is an immutable part of its state. Therefore, perhaps it should be managed in the same way that its value is. On the other hand, if one is compiling a program should one need to have access to the object itself in order to do type checking, in light of our federated policy? Might it not be more efficient if type information were kept with the name somehow, except that names are not typed? The flexibility of a generic name, representing a class of entities that may not all be of the same type, should not be lost. This is an area that needs careful thought.

This is only a partial list. It should certainly also contain all the issues raised with respect to the implementation as well, because they will be larger problems in the larger arena of a general system naming facility.

6.3. Schedule and Resources

The resources needed to complete this thesis are computing time. Since the implementation will be part of the mail system, the computing time will need to be on a VAX accessible from the Arpanet with Clu supported on top of Unix. In addition, text editing and formatting facilities will be needed to prepare the thesis.

Schedule:

August 26, 1983	Complete thesis proposal
September 30, 1983	Complete design and coding of my part of the mail project
October 31, 1983	Complete testing of code
February 1, 1984	First draft of thesis
April 15, 1984	Thesis exam
May 15, 1984	Complete thesis

Bibliography

1. J. M. Carroll. Creating Names for Personal Files in an Interactive Computer Environment. IBM Research Report RC 8356, IBM, July, 1980.

2. J. M. Carroll. Naming and Describing in Social Communications. *Language and Speech 23*, 4 (1980), 307-322.

3. N. W. Dawes, et al. The Design and Service Impact of Cocos, an Electronic Office System. International Symposium on Computer Message Systems, IFIP TC-6, Ottawa, Canada, April, 1981.

4. D. Farber and J. J. Vittal. Extendability Considerations in the Design of the Distributed Computer System. Proceedings of the National Telecommunications Conference, IEEE A & E S Society, IEEE Communications Society, and the IEEE G & E Group, Atlanta, Georgia, November, 1973, pp. 15E1 - 15E6.

5. R. Halstead. Reference Tree Networks: Virtual Machine and Implementation. MIT/LCS/TR 222, Laboratory for Computer Science, July, 1979. Also Ph. D. Thesis for the Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

6. International Organization for Standardization. Reference Model of Open Systems Architecture. Tech. Rep. ISO/TC97/SC16 N, Internation Organization for Standardization, November, 1978. Version 3

7. I. H. Kerr. Interconnection of Electronic Mail Systems - a Proposal of Naming, Addressing and Routing. International Symposium on Computer Message Systems, IFIP TC-6, Ottawa, Canada, April, 1981.

8. H. O. Lind, ed. *NSW User Reference Manual.* Bolt, Beranak, and Newman, Cambridge, Mass., 1982. Contributors: J. Ata, P. M. Cashman, H. O. Lind, N. Ludlam, S. A. Swernovsky, S. G. Toner. Prepared for Rome Air Development Center, Griffiss Air Force Base, Rome, NY 13440.

9. B. Lindsay. Object Naming and Catalog Management for a Distributed Database Manager. Proc. 2nd International Conference on Distributed Computing Systems, Paris, France, April, 1981. Also Available as IBM Research Report RJ2914, San Jose, Calif., August, 1980.

10. B. Liskov et al. Clu Reference Manual. MIT/LCS/TR 225, Massachusetts Institute Technology, October, 1979.

11. B. H. Liskov and R. W. Schiefler. Guardians and Actions: Linguistic Support for Robust, Distributed Programs. Conference Record of the Nonth Anual ACM Symposium on Principles of Programming Languages, ACM, Albuquerque, New Mexico, January, 1982, pp. 7-19.

12. wilkes. *The Computer Science Library: Operating and Programming Systems. Vol. 6: The Cambirdge CAP Computer and Its Operating System.* North Holland, New York, 1979.

13. R. M. Needham and R. D. H. Walker. The Cambridge CAP Computer and its protection system. Sixth Symposium on Operating Systems Principles, Special Interest Group on Operating Systems of the ACM, ACM, November, 1977, pp. 1-10.

14. R. M. Needham and A. D. Birrell. The CAP Filing System. Sixth Symposium on Operating Systems Principles, Special Interest Group on Operating Systems of the ACM, ACM, November, 1977, pp. 11-16.

15. D. C. Oppen and Y. K. Dalal. The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment. Tech. Rep. OPD-T8103, Xerox, October, 1981.

16. E. I. Organick. *The Multics Experience: An Examination of Its Structure.* M.I.T. Press, Cambridge, Mass, 1972.

17. L. Pouzin and H. Zimmermann. A Tutorial on Protocols. *Proc. of the IEEE 66*, 11 (November 1978), 1346-1370.

18. W. V. O. Quine. *Word and Object.* Technology Press of Massachusetts Institute Technology and John Wiley & Sons, New York, 1960.

19. D. M. Ritchie and K. Thompson. The UNIX Time-Sharing System. *Communications Of The ACM 17*, 7 (July 1974), 365-374.

20. J. H. Saltzer. Naming and Binding of Objects. In *Lecture Notes in Computer Science, Vol. 60*, Springer Verlag, New York, 1978, ch. 3, pp. 99-208.

21. J. H. Saltzer. On the Naming and Binding of Network Destinations. International Symposium on Local Computer Networks, IFIF/T.C.6, April, 1982.

22. A. D. Birrell, R. Levin, R. M. Needham, and M. D. Schroeder. Grapevine: An Exercise in Distributed Computing. *Communications Of The ACM* (April 1982).

23. J. F. Shoch. Internetwork Naming Addressing, and Routing. Proc. 17th IEEE Computer Society International Conference, IEEE, , 1978, pp. 72-79.

24. K. R. Sollins. Copying Complex Structures in a Distributed System. MIT/LCS/TR 219, Laboratory for Computer Science, Massachusetts Institute Technology, May, 1979. Also M.S. Thesis for the Dept. of Electrical Engineering and Computer Science, Massachusetts Institute Technology

25. K. R. Sollins. Copying Structured Objects in a Distributed System. *Computer Networks 5* (1981), 351-359. Also presented at the Fifth Berkeley Workshop on Distributed Data Management and Computer Networks.

26. C. Sunshine. Source Routing in Computer Networks. *Computer Communications Review 1*, 7 (January 1977), 29-33.

27. L. Svobodova. A Reliable Object-Oriented Repository for a Distributed Computer System. Proceedings of the 8th Symposium on Operating Systems Principles, Special Interest Group on Operating Systems of the ACM, December, 1981, pp. 47-58. Also published as Operating Systems Review, Vol. 15, No. 5

28. D. P. Reed and L. Svobodova. Swallow: A Distributed Data Storage System for a Local Network. Proc. of the International Workshop on Local Networks, IFIP Working Group 6.4, Zurich, Switzerland, August, 1980.

29. D. Weinreb and D. Moon. *Lisp Machine Manual.* 4th edition, Aritificial Intelligence Laboratory, Massachusetts Institute Technology, Cambridge, Mass., 1981.

I. APPENDIX:Clu Cluster Specifications of Contexts and Aggregates

```
context = cluster is create, delete, equal, copy, add - name, delete - name,
  how - many, translate, names

  rep = array[record[name: string,translation: any]]

  create = proc () returns (cvt)
                  % create creates a new empty context
  end create

  delete = proc (context1: cvt)
                  % delete deletes context1, invalidating any
                  % remaining references to it.  This
                  % operation is idempotent.
  end delete

  equal = proc (context1, context2: rep) returns (bool) signals
    (no - such - context)
                  % equal tests whether two contexts are the
                  % same.
  end equal

  copy = proc (old - context: rep) returns (cvt) signals
    (no - such - context)
                  % copy creates a new context that is a copy
                  % of the old one
  end copy

  add - name = proc (context1: rep, new - name: string, transformation: any)
    signals (no - such - context)
                  % add - name adds a name and translation pair
                  % to the context specified.
  end add - name

  delete - name = proc (context1: rep, delname: string) returns
    signals (no - such - context, not - found)
                  % delete - name deletes the name specified from
                  % the context specified.
  end delete - name

  how - many = proc (context1: rep, label: string) returns (int)
    signals (no - such - context)
                  % how - many indicates how many entities are
                  % named by the given label in the context
                  % specified.  The label can be a logical
                  % combination of names.
  end how - many
  translate = iter (context1: rep, label: string, t1: type) yields
    (force[t1] (any)) signals (no - such - context, no - such - name)
                    % translate returns all the entities named
                    % by the logical combination of names in
                    % label in the specified context of the
                    % specified type.
  end translate

  names = iter (context1: rep) yields (label: string)
                    % names yields all the names in context1.
  end names

end context

aggregate = cluster is create, delete, equal, copy,
  set - current - context, copy - environment, append - to - current - context,
  append - to - environment, current - context - is, current - aggregate - is,
  add - name, delete - name, how - many, translate, list - environment,
  add - to - rule, delete - from - rule, add - rule, delete - rule, move - rule

                    % The only context that is modified through
                    % the aggregate cluster is the current
                    % context.  Names are added to the current
                    % context by two operations, add - name,
                    % and translate.  The operation add - name
                    % is self explanatory.
                    % Translate will add the translation it finds
                    % through search rules to the current
                    % context unless it is already in the
                    % current context.  The current context is
                    % always assumed to be at the head of the
                    % list of search rules.  Contexts are
                    % searched in the order in which they
                    % appear in the list of search rules.
                    % Whenever a search rule is added it must
                    % also appear somewhere in the search
                    % domain by the name used in the search
                    % rules.  This means that when the current
                    % context is changed, each context must
                    % continue to be named somewhere in the
                    % search domain.  This is not a problem
                    % when search rules are deleted as long as
                    % each context named in the search rules
                    % list can be resolved in the current
                    % context, as will be the case when they
                    % are originally added, in order to add them.

 sas = sequence[array[string]]
```

```
rep = record[current - context: context, environment: sas]

create = proc () returns (cvt)
                  % create creates a new aggregate with an empty
                  % current context and environment.
end create

delete = proc (aggregate1: cvt)
                  % delete deletes aggregate1 invalidating
                  % any references to it.  This operation is
                  % idempotent.
end delete

equal = proc (aggregate1, aggregate2: context) returns (bool)
  signals (no - such - aggregate)
                  % equal returns true if the two
                  % aggregates are the same object.
end equal

copy = proc (aggregate1:rep) returns (cvt) signals
  (no - such - aggregate)
                  % copy creates a new aggregate that has
                  % the same values as aggregate1.
end copy

set - current - context = proc (current - context: context) returns signals
  (no - such - context)
                  % changes current context
end set - current - context

copy - environment = proc (aggregate1, aggregate2: rep) signal
  (no - such - aggregate)
                  % copy - environment copies the environment
                  % of aggregate1 into aggregate2.
end copy - environment

append - to - current - context = proc (context1: context) signals
  (no - such - context)
                  % append - to - current - context appends the
                  % contents of context1 to the
                  % current - context eliminating duplicates.
end append - to - current - context

append - to - environment = proc (aggregate1: rep) signals
  (no - such - aggregate)
                  % append - to - environment incorporates the
                  % environment in aggregate1 into the
                  % environment of the current aggregate
end append - to - environment
```

```
current - context - is = proc () returns (context)
                  % current - context - is returns the current
                  % context
end current - context - is

current - aggregate - is = proc () returns (cvt)
                  % current - aggregate - is returns the
                  % current aggregate
end current - aggregate - is

add - name = proc (new - name: name, transformation: any)
                  % adds a name and translation pair to the
                  % current context
end add - name

delete - name = proc (delname: name) returns signals (not - found)
                  % deletes the name specified from the
                  % current context
end delete - name

how - many = proc (context1: context) returns (int) signals
  (uncountable)
                  % how - many returns the count of the number
                  % of entries in context1.
end how - many

translate = iter (label: string, t1: type) yields (force[t1] (any))
  signals (no - such - name)
                  % translate returns all the entities named
                  % by the logical combination of names in
                  % the label in the aggregate by
                  % searching contexts in the aggregate in
                  % the order specified in the environment,
                  % if it has been specified.  Where ordering
                  % has not been specified, the search order
                  % is random.  If translation uses entries
                  % from contexts other than the current
                  % context, those entries will be made in
                  % the current context.
end translate

list - environment = proc (aggregate1: rep) returns (sas)
                  % list - environment returns a sequence of
                  % arrays of strings.  The sequence
                  % represents the partial ordering of
                  % contexts within an environment.  Each
                  % array represents those context which are
                  % not ordered with respect to each other.
                  % The strings are the names of the
                  % contexts.
```

```
    end list - environment

    add - to - rule = proc (i: int, label: string) signals (no - such - rule)
                % add - to - rule adds a label for a context to
                % the ith rule in the environment. It is

                % idempotent. If the name is already
                % there, nothing happens.
    end add - to - rule

    delete - from - rule = proc (i: int, label: string)
                % delete - from - rule deletes the label from
                % the ith rule in the environment. The
                % operation is idempotent.
    end delete - from - rule

    add - rule = proc (i: int, rule: array[string])
                % add - rule adds the rule in the ith
                % position. This will move everything from
                % that point on into the next position. In
                % other words what was in the ith position
                % will be in the (i + 1)th position.
    end add - rule

    delete - rule = proc (i: int) signals (no - such - rule)
                % delete - rule will delete the ith rule, if
                % there are at least i rules.
    end delete - rule

    move - rule = proc (i,j: int) signals (out - of - bounds)
                % move - rule moves the ith rule into the jth
                % position, and maintains the relative
                % positions of all other rules. If either
                % i or j is out of bounds this is signalled.
    end move - rule

end aggregate
```