Collaborative naming in distributed systems

To cite this article: S Benford and Ok-Ki Lee 1993 Distrib. Syst. Engng. 1 67

View the article online for updates and enhancements.

Related content
- Evaluation of attributed names
  Y Ni and A Goscinski
- The distributed computing environment naming architecture
  N Leser
- The evolution towards flexible workflow systems
  Gary J Nutt

Recent citations
- Simone Lupetti et al
- Locating mobile agents in a wide distributed environment
  A. Di Stefano and C. Santoro
Collaborative naming in distributed systems

Steve Benford and Ok-Ki Lee
Communications Research Group, The University of Nottingham, University Park, Nottingham, UK

Received 10 December 1992

Abstract. The process by which humans name objects is highly flexible and dynamic. This is particularly true of group work where common names for objects may be gradually established through the processes of proposal, translation and adoption. This paper introduces a naming model, for use mostly by humans, which supports this process of group naming and specifically allows names to be ambiguous and changeable. The model defines techniques for resolving names, translating names between different group members and proposing common names for objects within a group. The paper also discusses how individual naming contexts which support these facilities could be federated into a structured namespace and how the process of flexible name resolution can therefore be extended to large scale distributed environments. The resulting functionality contrasts strongly with existing, often hierarchical, naming models which deliberately attempt to reduce ambiguity of names and which provide minimal support for managing name changes. The intention is that the proposed naming model could be layered on top of existing naming services in order to provide additional human and group oriented support for naming.

1. Introduction

Names allow us to identify objects, to talk about them and to access them. Naming is therefore an important issue for large scale distributed systems. It becomes a critical issue when those systems are intended to support collaboration between humans.

A large volume of research has already been published on the subject of naming, particularly within the context of nameservers and directories. However, it can be argued that the hierarchical nature of many of the naming mechanisms so far proposed is too constraining to fully support the great flexibility of human naming practice, particularly where group work is concerned. More specifically, attempts to reduce ambiguity and to ensure the uniqueness of names (even in non-hierarchical systems) are in opposition to the often ambiguous and highly contextual human use of names. These problems will become particularly limiting as large scale collaborative systems come into being and the need to support communication between group members from different backgrounds becomes increasingly important.

This paper proposes an alternative naming model for distributed systems which is specifically oriented towards support for naming in group work (for example, in applications such as joint editing and computer conferencing). The goal of the model is to allow a great deal of flexibility in naming and to support both individual and group use of names. A particular focus of the work is on mechanisms for evolving from the initial use of individual names towards common names for objects in a given group context as well as for translating names between different group members and gradually propagating changes in names. The overall aim of the paper is that this model could influence the design of future distributed CSCW systems and also suggest extensions to current nameservers and directories.

Section two of the paper discusses the requirements of a group oriented naming mechanism. Section 3 then focuses on a flexible model of naming within a single naming context. Once this is established, section four then looks at how naming contexts could be federated in order to provide a more structured namespace suitable for large scale distributed systems.

2. Requirements of naming in collaborative systems

The goal of this paper is to propose a group oriented model for naming objects in distributed environments. The first step is to consider the way in which humans name objects. In so doing, we will define the requirements of our model and at the same time indicate some of the deficiencies of existing naming models. Many researchers have considered the issue of naming and its relationship to distributed systems (Saltzer 1978, Shoch 1978, White 1984). For those interested in further details, (Bhargava 92) provides an
interesting collection of recent work in this area. The work
presented in this paper has been particularly influenced by
the direction of Sollins’ work on distributed name
management and the idea of negotiating names between
different people’s contexts (Sollins and Clark 1987). It has
also been motivated by the following observations about
the human use of names.

- Names are not unique! Different people may use
  the same name for different objects. Furthermore, one
  person may use the same name for different objects in
  different contexts or at different times.
- An object may have many names used by different
  people or even by the same person.
- Names are not static. Instead, they are negotiated,
  proposed, gradually accepted and may then fall into disuse.
Names may change to reflect a change in the status of an
object or a change in preference, taste or policy of their
users. The process of a name being defined and then
gradually accepted within a context through its use is
particularly important.
- People may often be aware of the ambiguous nature
  of a name (i.e. a name that could refer to more than one
  object) and are either happy to live with this or may seek
  further clarification.
- People rarely impose arbitrary distinctions between
  names (e.g. talk about John1 and John2).

As well as considering the human use of names, it is
important to consider some more system oriented
requirements of naming.

- Names ought to be distinct from identifiers, routes or
  addresses. A name enables a human being to identify an
  object; an identifier is used by a computer system; an
  address tells you where an object is and a route tells you
  how to get there. These are distinct properties of an object
  (Schoch 1978).
- Management of large numbers of names may require
  a structured namespace.
- It should be possible to resolve names to arrive at a
  set of objects.
- Conversely, it should be possible to take an object
  and find out what its names are.
- The management of names needs to be a highly
decentralized process.

A range of naming mechanisms have been proposed for
different kinds of computer systems. However, most
current implementations centre around the idea of a loosely
hierarchical system. Examples include filestores in
operating systems (UNIX and DOS), nameservers (e.g.
BIND (Mockapetris 1983), The Clearinghouse (Birrell et al
1981)) and lately Directory Services (e.g. X.500 (ISO DS
1988, Kille 1988)). The reasons for adopting a hierarchical
design are most likely that hierarchies are easy to
understand and to implement (give a computer scientist any
problem and they give you a tree as the solution!).
However, the hierarchical approach suffers from two major
problems. First, universal agreement on the structure and
management of a global namespace is likely to be fraught
with political difficulties (Van der Linden 1992). Second,
hierarchy aims to ensure the uniqueness of names, to
reduce ambiguity and to impose common names for all
objects. This is contrary to the way in which humans use
names. The current trend towards federated namespaces
seems likely to address the first of these problems.
However, this paper argues that human oriented systems,
particularly collaborative systems, will require additional
naming support to address the second. The paper therefore
proposes a group oriented naming mechanism which is
intended to meet the following goals:

- Allowing ambiguity of names and multiple names
  for objects.
- Helping resolve the use of different names by
different people (e.g. translation of names).
- Being able to propose new names for objects without
  imposing them on people.
- Aiding groups in gradually establishing common
  names for objects.
- Supporting the migration and changing of names
  (where migration means the gradual spread and adoption of
  a new name throughout a community of users).
- Enabling the decentralized management of names.
- Resolving names into objects and vice versa.
- Scaling to large distributed environments through
  support for structured names.

It should be stressed that the model is not intended to
replace existing naming mechanisms. Instead, it aims to
complement them by providing an additional layer of
naming support between groups of humans and the system.
This relates back to the separation between human names,
identifiers, addresses and routes. In the case of existing
human oriented naming services such as the X.500
Directory Service, the model could either be interpreted
as a set of service extensions or could be layered on top of
existing services (see section 5 for a fuller discussion). The
relationship between the model and existing naming
models is summarized in figure 1.

![Figure 1. The relationship between the proposed model
and existing naming models.](image-url)
We start in section 3 by considering the limited case of naming within a single naming context and then progress to more complex environments.

### 3. A group oriented naming context

Naming contexts provide the cornerstone of the naming model. Each naming context defines a namespace for a particular individual, group, application or domain. As people move between different tasks and applications they therefore use different naming contexts. Naming contexts may be personal or may be shared and several naming contexts may be relevant to a particular task at a given moment.

The fundamental role of a naming context is to manage a local namespace and to try to resolve purported names into identifiers for objects. Management implies maintaining a record of all names in the namespace and providing facilities for creating, deleting and changing names. Name resolution means mapping from human oriented names onto system identifiers. For the time being, the term *name* will imply a meaningful string defined and used by humans in contrast to the term *identifier* (or ID) which will imply some kind of system reference to an object (e.g. a pathname, number or pointer). Identifiers are assumed to be unambiguous within a particular administrative domain. Thus, the system will be able to locate exactly one object corresponding to each identifier.

In order to scale for large systems, identifiers may also be negotiated across domains through the process of federation (Van der Linden 1992). Names are not assumed to be unambiguous!

A local namespace can now be defined as a set of tuples of the form:

\[
\text{<object's identifier> <object's name> <user's identifier>}
\]

The object's identifier indicates the object being named; the object's name defines a new name used for this object; and the user's identifier refers to the object which has defined this name. This third component is particularly important as it relates this tuple to a particular individual user of the naming context. Each tuple in the namespace is unique in its entirety. However, the object's name component need not be. In fact, there are no restrictions placed on how many times a particular name may appear in relation to a given named object or user. In other words, I can use the same name for many objects and an object may be called by many names. Fundamentally, a naming context contains a user's own mapping (the user's name to ID) as well as mapping created by the user for other identifiers. The following table shows an example namespace (and assumes that object identifiers are numerical).

Several observations can be made even about this simple example. First, each object defines at least one name for itself (object 1 actually defines two such names, Fred and Fredrick). We call these preferred names and they indicate the names by which the objects prefer to be known. Second, note that two of the objects prefer to be known by the same name (Jane). To avoid confusion, they have both defined alternative names for each other. Thus, object 2 names object 3 as Jane Evans whereas object 3 names object 2 as Jane Smith. Notice that in this case object 1 chooses to recognize both object 2 and 3 as Jane and also object 3 as Jane Evans.

The key point arising from this example is that the namespace places no restriction on the multiple binding of names to objects. In particular we see the same name used to name different objects and also different names used for the same object (in one case even by one object — the use of the name Jane by object 1).

Figure 2 shows a diagrammatic representation of this name space which may be easier to read. Each object is shown as a node and is labelled with its ID. Names are shown as labelled arcs where the label shows both the name and the ID of the object which defined the name. For example, Jane (1) means that the name Jane has been defined by the object with identifier 1. Notice that Fred (1) and Fredrick (1) are preferred names of object 1.

A further note is that the definition of names as strings may be rather simplistic. In particular, a great deal of useful work has been carried out in the context of the X.500 Directory standardization effort into the detailed syntax of names and the design of different matching rules to apply to these syntaxes (e.g. supporting both exact and approximate matching across a range of syntaxes) (ISO DS 1988). Although this paper does not consider these issues in detail, integration with this work would be a desirable goal.

### 3.1. Basic name resolution

Name resolution is the process of mapping names onto identifiers and starts with an object presenting a purported name to the naming context (an as yet unresolved name). Resolution then involves two successive attempts to map the name onto object identifiers:

- **Step one:** first, inspect the namespace to see if this name is actually defined by this user. If it is, return the relevant object identifiers.
- **Step two:** if the name is not defined for this user, find the most common definition of this name by other users in this context. This involves searching the context to find all

### Table 1. An example naming context.

<table>
<thead>
<tr>
<th>Object's identifier</th>
<th>Object's name</th>
<th>User's identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fred</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Jane</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Jane</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>Fredrick</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Freddy</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>Fred</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Jane Smith</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Jane Evans</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Jane</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Jane</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Jane Evans</td>
<td>1</td>
</tr>
</tbody>
</table>
uses of the purported name and then extracting the most common definition by counting the number of references to each object. This step therefore returns the identifiers of those objects which other people most frequently associate with this name.

From the perspective of groups and collaborative systems, step two is of critical importance as it enables people to use names which are known by other group members but which they have not yet defined themselves. Thus, if someone refers to a new object during a conversation, I may also be able to adopt their name for the object.

Let us consider a few simple examples with the naming context defined above. The purported name Fred presented by object 3 would map onto object 1 according to the first step. The purported name Jane presented by object 1 would map onto both objects 2 and 3 under step one. On the other hand, the purported name Freddy presented by object 3 would fail step one but would map onto object 1 by step two. In this last case, the naming context is recognizing that, even though object 3 does not know the name Freddy, it is used by object 2.

It seems sensible to associate different levels of confidence with the two stages of resolution. Thus, we can have a high degree of confidence that an identifier found under step one represents an object the user wants. On the other hand, we can have less confidence in a mapping under step two. It would be sensible to return this confidence level as an additional result of the resolution process thus enabling users and their applications to decide on the most appropriate course of action (e.g. to use the identifier or to investigate further). To go one step further, a user might specify a confidence level to be met when they request name resolution this governing whether rule two would come into force at all.

Notice also that, unlike other naming mechanisms where ambiguity is disallowed, name resolution may return multiple object identifiers. It is then up to the user or application to decide how to proceed. There are several approaches in the case of a human user. First, the system may present them with all of the named objects and ask them which ones they wanted to access. Second, the user might enter into further dialogue with the system by supplying more information (e.g. identifying other attributes of the desired object) and requesting further searching or name resolution. In either case, the user needs to decide how to proceed and the user interface needs to be sufficiently flexible so as to present alternatives and enter into dialogue. It could be argued that this approach passes the buck to the user and in some sense this is true. However, the rationale behind this paper is that ambiguity does not occur in the real world and is typically resolved through dialogue (this is the way we deal with ambiguity when interacting with other humans). The alternative which is adopted by most naming mechanisms (disallowing ambiguity), effectively runs away from the problem by constraining the flexibility afforded to human users. Where the user is an application (not the majority of intended cases), ambiguity is likely to cause greater problems. First, ambiguity might simply be refused and flagged as a name error (being dealt with through normal error handling mechanisms). Second, more intelligent applications may also be developed which enter into further dialogue with the system in order to resolve the ambiguity.

Under the proposed model, step two searches for the most common definition of the name within the context. A refinement of this approach might be to maintain a record of the number of actual dynamic uses of names and then to select the name most commonly used (as opposed to defined). In this case, the naming context might best be drawn as a weighted graph with objects as nodes, names as arcs and weights reflecting usage.

It is also interesting to reflect on the relationship between the proposed name resolution algorithm and the kinds of fuzzy matching techniques used by many name services. Fuzzy matching provides users with more flexibility in matching names that are similar to the ones they purport. In particular, syntactic, structural and phonetic similarities can be detected. However, fuzzy matching will have more difficulty coping with larger discrepancies (such as the English naming conventions that Elizabeth can be Betty or James can be Jim). Fuzzy matching certainly will not cope with cases where objects are known by quite different names to different people. On the other hand, the proposed resolution technique could be extended to additionally include fuzzy matching techniques to provide yet more flexibility in matching names.
3.2. Proposing and translating names and merging namespaces

Now that we have established the basis of a naming mechanism, we can consider how this might be used to enhance collaboration across groups of people. The use of common names for objects is likely to be a key factor in group cohesion. As a result, this section introduces two techniques aimed at encouraging the gradual migration from individual names for objects towards common group names. An important goal of this work is that such a migration should be both voluntary and evolutionary and that the system should never impose the use of particular names.

The first technique is a facility for the naming context to propose alternative names for an object to those currently in use where the alternatives represent more commonly used names among group members. This facility is also useful for proposing appropriate names for new objects which have not been encountered before and would therefore be particularly important when integrating new group members. To see how this might work, consider the extension of our current scenario through the introduction of a new user, object identifier 4, who has just started work in this naming context. They are initially registered in the context with a preferred name (see table 2). Throughout their work they will now come across many objects for which they do not have names (e.g. if they ask the system to list the names of all objects available to them). The naming context can propose names for these previously unseen objects by finding out what they are commonly called by other people. Thus for each object, the naming context would count up the occurrences of all its names and would then choose the most popular. In this case, the naming context would propose the names shown in table 3. Notice that two names are proposed for object 3; as they are both equally popular, object 4 can now choose whether to accept these names or not. In some cases it might like to define its own alternative names. In this way the naming context is encouraging, although not enforcing, the adoption of common group wide names for objects.

The ability to suggest names need not be restricted to previously unseen objects. The naming context could also support a facility to propose alternative names for an object at any time. This might be used by applications to periodically suggest new names for existing objects and so encourage convergence towards a common group name space. For example, when displaying current object names, a user interface might also show alternative commonly used names (perhaps greyed out to distinguish them).

Closely related to the issue of proposing names is that of translating them. Imagine a scenario where you receive a message or document from another group member who uses a name which you do not understand. The naming context might be able to translate between their use of the name and your currently defined name for the object. This would involve two steps:

1. Resolve the name from their perspective (not from yours) to get a set of object identifiers.
2. Carry out a reverse mapping from these object identifiers back to names that you have defined for the object (i.e. reverse map from your perspective).

For example, if object 2 sent a message to object 3 in which they used the name Freddy, object 1 could then request translation of the name via the naming context. This would involve mapping the name Freddy as used by object 2 resulting in the reference to object 1 and then the reverse mapping into the name Fred as known by object 3.

Translation such as this within a single context is relatively straightforward. However, in many situations the initial use of the name may have been in a different context to that of subsequent use. We will therefore need to return to the issue of translation in section 4 when we discuss naming in multiple contexts.

3.3. Managing names

So far, we have considered the role of a naming context in resolving, proposing and translating names within a group. Now we turn our attention to the management of names. More specifically, this means the problems inherent in creating, deleting and changing names.

Creation and deletion of names are relatively straightforward. In contrast to other naming mechanisms where guarding against ambiguity and checking for uniqueness makes name creation a difficult operation, the relaxed approach to naming actually makes name creation very simple. It suffices to check that the identifiers of the named and naming objects are valid. Deletion is also relatively simple.

Changing names, on the other hand, presents a more difficult problem. Given the fact that a naming context allows multiple names for an object, there is no technical need to propagate name changes by one individual to everyone else. However, this completely ignores the original motivation for the name change and the fact that a name change, particularly in the preferred name of an object, most likely reflects a change in status and ought to be propagated throughout an environment. The solution to this problem is to extend the facility for a naming context automatically proposing names. If a preferred name of an object is changed, the naming context should note the change and continue to remember the old name as well the new one for a period of time. The propose facility described above should now look for such changes as well
as for commonly used names. In this way, other people will be informed of the name change when they next access the object and may be asked if they wish to update their own name due to its increasingly common usage. This mechanism provides a natural and flexible way of managing name changes and of gradually propagating them throughout the context.

### 3.4. Definition of a name server object

The previous sections have outlined the functionality of a group oriented naming context for distributed systems including facilities for the flexible resolution of names; the ability to automatically propose names for new and existing objects, the ability to translate names between different group members and also to manage changes to the namespace. This section ties these ideas together by defining a naming context as an abstract object and specifying these facilities as operations supported by this object. In effect, the definition of such an object constitutes an outline specification of the interface to a nameserver. Figure 3 shows a naming context object and its interface.

The following paragraphs describe each of the operations in turn.

- **Add_name** adds a new tuple to the name space.
- **Delete_name** deletes a tuple from the name space.
- **Change_name** updates a name and maintains a record of the change where preferred names are concerned.
- **Resolve_name** maps from a name to a set of object IDs by considering both valid names for this user and also commonly used names in this context.
- **Resolve_id** performs the reverse mapping between a name and a set of object IDs.
- **Translate_name** attempts to map a name as given by one user into a set of names as understood by a second.
- **List_users_of_name** returns the IDs of all objects which know the specified object by the specified name.
- **List_names_by_user** returns all of the names valid for this user. This concludes our discussion of the facilities offered by a single naming context. The following section turns its attention to supporting structured naming in large-scale distributed namespaces by federating naming contexts.

### 4. Federated naming contexts and structured names

Nearly all computer systems will require multiple naming contexts (for example, each directory in a file store could be considered as a separate context). This will be particularly true of large distributed systems where naming contexts might be required for different groups, domains, applications and tasks. Within existing large scale systems, flat (i.e. single-level) naming structures have come to be regarded as too unwieldy both to perceive and to manage. Consequently, some kind of structured naming mechanism which involves names being resolved through multiple contexts has usually been introduced. There are many examples of such structured namespaces including filestores (UNIX and DOS), nameservers (BIND (Mockapetris 1983)) and The Clearinghouse (Birrell et al 1981), Context Space in the ODP Trader (ANSA 1989, ISO ODP 1990) and Directories (e.g. X.500 (ISO DS 1988, Kille 1988)). This section considers the implications of multiple contexts and structured name spaces for our group oriented model. In particular, it shows how contexts may be structured into a non-hierarchical space with support for cross context name resolution. It also considers the problems inherent in translating names across different contexts.

We can extend our current model to support multiple contexts by recognising that a naming context is itself an object and can therefore be named in other naming contexts. Continuing in our earlier spirit, we treat naming contexts just as we would any other object and allow them to have multiple names valid for different users. The result is a graph of connected naming contexts with possibly many arcs between any two contexts representing the different names used by different users. It should be noted that no particular context is given priority or control over any other context resulting in a federation of naming contexts managing a structured namespace. Figure 4 shows an example namespace built from four naming contexts.

Each context is shown as a node and is labelled with its ID and also its preferred name (the example is based on our current project structure at Nottingham with COMIC and MOCCA being two ongoing projects). Additional names are shown as labelled arcs where the label shows both the name and the ID of the object which defined the name (the latter in parentheses). Thus, the label Steve (1) means that...
the name Steve has been defined by the object with identifier 1. In addition to the four naming contexts, the diagram also shows an example report object which is named within two different contexts.

As a person works within the system, they will navigate between different contexts with the names they use for objects always being resolved starting at their current context. At this stage we can extend the structure of a name to be an ordered sequence of name parts where each name part is a single string. Name resolution now proceeds from context to context considering each name part in succession until either the complete name is matched, resolving to one or more object IDs, or it is determined that the name does not resolve to any object (a name error). Within each context en route, resolution proceeds as described in section three by first looking for names known by this user and then considering commonly used names. For example, when working in the COMIC context (ID = 1), the example report could be named as either MOCCAISteve-#3 or SteveNaming Paper or even CRGIS BenfordNaming Paper (we are adopting the convention of a '/' character to separate name parts). The use of commonly used names in the process of name resolution is likely to be an important factor in structured naming as it allows a common group framework to be established just from the names that contexts use for each other. Other people may then tailor this structure for more personal use.

The proposed naming model supports 'relative naming'. In other words, names are always interpreted relative to the current working context. This also means that the same structured name might resolve to different objects when used in different contexts (a more detailed discussion of relative naming is given in [Brownbridge 1982]). Many existing naming mechanisms support absolute naming where a particular context is identified as the root of the name space (the resulting name space is frequently hierarchical) and absolute names are then interpreted starting at this root context. As an example, UNIX allows both absolute and relative names for files and directories. Absolute naming might best be introduced into our model through the applications themselves. It is not sensible to define a root naming context in the naming scheme itself (particularly in large federated systems). Instead, each application might specify its own root and perform some preprocessing of names before resolving them. This might include making sure that absolute names are always resolved from the root context. Notice that in this case each application is free to define its own root naming context. In this way, the proposed model can scale up not as a whole, but in the sense of collection, without resorting to an hierarchical approach.

Although we have spent some time discussing the resolution of structured names, it is likely that users will find such names unwieldy to use. Instead, we anticipate that people will move to different contexts according to the task at hand and will use single part names within each context (in contrast to remaining in one context and using multi-part names). Contexts may represent different work structures (e.g. projects) and will probably be closely associated with the notion of groups. Thus, we would expect a person to spend much of their time moving between a few well known naming contexts. It should be noted that this requires the introduction of a new move_context operation for moving between naming contexts (an extension to figure 3).

The notion of movement between contexts complicates the process of name translation. The problem is that the original context in which a name was used may be different from that in which it is being resolved. Consider the example where a message is created and sent in one context and received and read in a different one. The translation process should now be modified as follows:

1. Resolve the name from the first person's perspective in the initial context.
2. Carry out a reverse mapping from the second person's perspective in the final context.

The problem is knowing the initial context. As names are always relative, it will not always be possible to identify the initial context in which a name was used. However, several heuristic approaches are possible:

- Indicate the name of the initial context whenever a name is used (e.g. include it in the message somewhere). However, the name must be expressed relative to the final context and this will not always be known in advance.
- Include the identifier of the initial context with the name. This will work fine but it is unrealistic to assume that this information can be provided every time a name is used.
- Assume that the initial context is the same as the final one or at least that the initial user of the name may define the same names in both. This is clearly not always going to be true although it may work in some cases.
- Identify all contexts which both people use and try to translate the name across combinations of these.

To summarize, translation of names may not always be possible as both initial and final naming contexts may not always be known. However, the above strategies may be tried to increase the chances of successful translation.
5. Summary and future work

This paper has been concerned with the definition of a naming model oriented towards support for group work in large scale distributed environments. The motivation for this work has stemmed from the observation that, on the one hand, human naming practice exhibits a great deal of flexibility in terms of dealing with ambiguity, negotiation and management of names, and on the other, the naming models adopted as the basis of nearly all existing nameservers deliberately aim to reduce ambiguity and provide little support for negotiation of names.

The resulting model has therefore specifically allowed names to be multiply bound to objects and has then focused on the problems of translating between different names used by different people, proposing names for new and existing objects and gradually propagating name changes throughout an environment.

The cornerstone of the model is the naming context, an object that manages a local namespace. Each individual defines their own names for objects in a given context. However, the context may then additionally use other people’s names during the process of name resolution. The naming context also includes a facility to propose new names for objects so that users may be offered the choice of adopting commonly used names for objects in preference to their own individual names as they work as part of a group. This encourages the gradual establishment of a group-wide namespace for objects. Translation of names between individuals is made possible through the two stage process of first resolving a name from the perspective of one individual and then reverse mapping the resulting object identifier back to a set of names from the perspective of the second. Finally, a context supports the gradual propagation of name changes by the process of proposing the new name for an object each time an out-dated name is used.

Once the basis of a single naming context had been established, the paper examined how contexts could federate in order to provide a structured namespace. Given the observation that one naming context could name others just as it would any other object, the paper showed how a distributed naming graph could be constructed and how the name resolution process could be extended to navigate this graph.

Of course, much work remains to be done, particularly implementation and evaluation of the model. A key issue for future work concerns the relationship between the proposed model and existing human oriented naming services such as the X.500 Directory Service. At one extreme, the model could be seen as a set of service extensions to X.500 which provide enhanced support for CSCW applications. However, extending a complex and well developed standard such as X.500 may not be a viable or sensible option. A more feasible approach might be to support the model with existing X.500 facilities and then construct specialised Directory User Agents (DUAs) to provide the functionality shown in figure 3. These DUAs would accept the proposed group naming operations and would map them onto standard X.500 operations. This approach may be possible due to the presence of the aliasing mechanism in X.500 which allows the addition of cross-links to the Directory Information Tree, resulting in a more graphical structure (in effect, aliases are analogues to UNIX soft links). Aliases, may therefore allow the mapping of the naming graphs proposed in this paper onto the hierarchical X.500 model, with the specialized DUAs being responsible for maintaining consistency between the two. Working out the details of this approach and then implementing the model on top of X.500 are tasks for future work.

In summary then, the paper has proposed a general purpose naming model that focuses on many of the human aspects of naming objects in collaborative systems. We hope that this model proves useful for informing the design of both future collaborative and naming systems.

References


Bharat Bhargava B (ed) Naming Facilities in Internet Environments and Distributed Systems (Special Issue of the Distributed Processing Technical Committee Newsletter) Vol 14, No 1 (IEEE Technical Committee on Distributed Processing)


Brownbridge D R, Marshall L F and Randell B 1982 The Newcastle Connection—or UNIXes of the world unite!, Software Practice and Experience 12 1147–62


Mockapetris P 1983 The Domain Name System Computer Based Message Services (Proc. IFIP 6.5 Conf.) (Nottingham) ed H T Smith (Amsterdam: North-Holland)

Saltzer J H 1978 Naming and Binding Objects (Lecture Notes in Computer Science 60) (Berlin: Springer) pp 98–208

Shoch J P 1987 Internetwork Naming, Addressing and Routing Report Xerox Palo-Alto Research Center), Palo Alto, USA

Sollins K R and Clark D D 1987 Distributed Name Management, Proc. IFIP 6.5 Int. Working Conf. on Message Handling Systems (Munich) (Amsterdam: North-Holland)

Van der Linden R J Naming and Federation Naming Facilities in Internet Environments and Distributed Systems (Special Issue of the Distributed Processing Technical Committee Newsletter) ed B Bhargava, Vol 14, No 1, (IEEE Technical Committee on Distributed Processing) pp 21–7