Naming and Addressing of Objects without Unique Identifiers

Nobuhisa Fujinami Yasuhiko Yokote
Sony Computer Science Laboratory Inc.
Takanawa Muse Building
3-14-13 Higashi-gotanda, Shinagawa-ku, Tokyo, 141 JAPAN

Abstract

This paper proposes the hierarchical naming scheme, *which is a way of naming and addressing suitable for large-scale distributed systems. FIrst, assumptions of the systems are provided, and features required for naming in the systems are discussed. Then, the methods for giving location-independent IDs and addresses representing objects' location are proposed. Our scheme constructs global IDs and addresses from locally unique ones. They have relative representations, which are translated if they are transmitted among naming contexts. This ensures uniqueness of IDs while efficiency and availability are preserved. Next the paper introduces methods which make our scheme suitable for mobile naming contexts and dynamic reconfiguration of networks and systems. Implementation issues of our method and their solutions are also presented.*

1 Introduction

For sharing objects in computer systems, naming them is one of the most convenient ways. In object-oriented systems, object IDs which are unique within systems are commonly used for sharing. Distributed systems generally have IDs that include the addresses of the hosts where the objects are instantiated and that ensure the uniqueness within systems. However, recent distributed systems have become much larger in scale, and quite a large size is required for unique IDs. For instance, the Amoeba distributed system [5] uses 72 bits within 128-bit capabilities to specify objects. Further, the possibility of running short of IDs still remains, considering the evolution of computer systems.

Large-scale distributed systems also have problems of mobile hosts, object migration for load balancing and communication efficiency, and reconfiguration of networks. An IP-address, for example, varies if the host moves to another network, and the user must be conscious of the host's location for communication. It is preferable that the locations of objects or hosts have nothing to do with specifying them, i.e, location transparency.

This paper proposes the *hierarchical naming scheme*, which can give short representations to logically-nearby objects' IDs and which has no possibility of running short of IDs even if the system evolves. Its addressing scheme is an extension of the *propagating cache method* in the *virtual network* [8] and realizes object/host/gateway migration transparency.

The virtual network considers migration-transparent communication between mobile hosts. This paper applies the propagating cache method to objects, and hierarchical application enables host migration and network configuration transparency. The virtual network assumes that host IDs are unique within the network system. The *hierarchical naming scheme* does not assume unique host IDs and constructs them from locally unique ones. It translates the representations of IDs and addresses each time they are transmitted. This ensures the uniqueness of IDs in every naming context while short representations of IDs and addresses are given to logically-nearby objects.

Section 2 gives our assumptions for large-scale distributed systems and the goals of this paper. Section 3 briefly describes our naming scheme. Section 4 explains the method of constructing object IDs and the virtual concept of *interpretation*. Section 5 discusses object migration transparency, and Section 6 discusses host migration and network reconfiguration transparency. Section 7 provides the method of connecting systems for expansion. Implementation problems and their solutions are in Section 8. Section 9 overviews related research, and Section 10 concludes the paper.

2 Assumptions and goals

We assume that the distributed systems discussed in this paper have the following properties:

Ultra-large-scale: The scale of the system is so large that broadcast is impossible, it is hard to assign unique fixed size IDs to each host, and objects and hosts migrate independently of each other.

Locality: Communication with or migration to distant objects or hosts, e.g. those in different local networks, are rare compared to those to nearby ones.

Hierarchy: The system has local naming contexts with hierarchical structure.

The last assumption is not a restriction since most practical systems are hierarchically managed. For example, wide area networks, local area networks, hosts, operating systems and processes can be used as the local naming contexts. This is not necessarily related to the physical structure[1] or social structure[2]. Representing the hierarchy as a tree structure, we assume that every pair of objects connected by parent-child branches can directly communicate with each other. The system may have other connections, as most networks do.

We design the global names with the following features for the distributed systems:

Uniqueness: Each object has just one name, and each name represents just one object. The name never changes until the object is deleted.

Location transparency: The name is independent of object/host migration and network/system reconfiguration. Objects' locations are separately represented.

Scalability: The naming scheme allows easy addition of hosts and easy reconfiguration of the system.

Efficiency: The overhead of using global names is small.

Availability and fault tolerance: Distant faults have little influence on the availability of operations on names.

3 Overview

Each naming context gives local IDs (**LID**s) to the objects instantiated in it and local addresses (**LAD**s) to the objects in it. LIDs and LADs are unique within the naming context. At instantiation time, the LID and the LAD of one object are equal. Since the manager of the naming context is also an object, it has an LAD that is unique within its parent naming context. There is only one **root naming context**, which is the root of the naming context hierarchy, in one system. The root can change as the system expands.

Each object in the system has the notion of the **original naming context**, which the object was instantiated in, and the **current naming context**, which the object is currently in. The object ID (**OID**) of the object represents its original naming context and the LID in it, and the object address (**OAD**) of the object represents its current naming context and the LAD in it. OIDs and OADs have relative representation seen from the object that memorizes the OIDs or OADs. The interpretation of OIDs and OADs is the meaning of them which is unique within the system in the conventional sense. The interpretation of the OID of one object never changes until it is deleted. If a message or an object is transmitted, the OIDs and OADs in them are translated to keep the interpretation of the OIDs and OADs invariant.

The unit of communication is a message. The receiver of a message is specified by its OID. The managers of the naming contexts take care of the relationship between the OID and the OAD. Users do not have to know about OADs.

Each object remembers its OID. It is "0:", which means "myself", at instantiation time, and varies as the object migrates. The naming context manager, which is also an object, has more information: the **reverse OID** of the manager seen from the original place, all pairs of LIDs and OADs of objects instantiated in it, and all pairs of OIDs and LADs of objects currently in the naming context. Further, the manager may cache the pairs of OIDs and OADs of the senders of the messages going through the naming context. If the naming context has a connection other than parent-child ones, the destination OID seen from the source and the source OID seen from the destination are needed.

4 Object IDs and their interpretation

An OID is for identifying an object. The format of an OID is

$$m : l_1.l_2.\cdots.l_n$$

where m, n are non-negative integers[3] and each l_i is an LID. The interpretation of an OID is defined as follows:

Definition 1 *Interpretation of OID*
The interpretation of OID $m : l_1.l_2.\cdots.l_n$ in naming context S is

$$L_1.L_2.\cdots.L_{N-m}.l_1.l_2.\cdots.l_n$$

where $L_1.L_2.\cdots.L_N$ is the sequence of LIDs from the current root naming context to S. □

Examples: An OID seen from a naming context manager which represents the object instantiated in it is $0 : l$, where l is the LID of the object. Without object migration, the OID seen from an object which represents the object in the same naming context is $1 : l$, and the OID of itself is 0:. Notice that the OID seen from a naming context manager and the OID seen from the object in it are different.

[1] For instance, all hosts in one local network may share the same naming context. In this case, uniqueness is ensured by a conventional technique of object IDs including host IDs.

[2] E.g. companies and countries.

[3] m can be seen as the number of "../"s of a relative pathname.

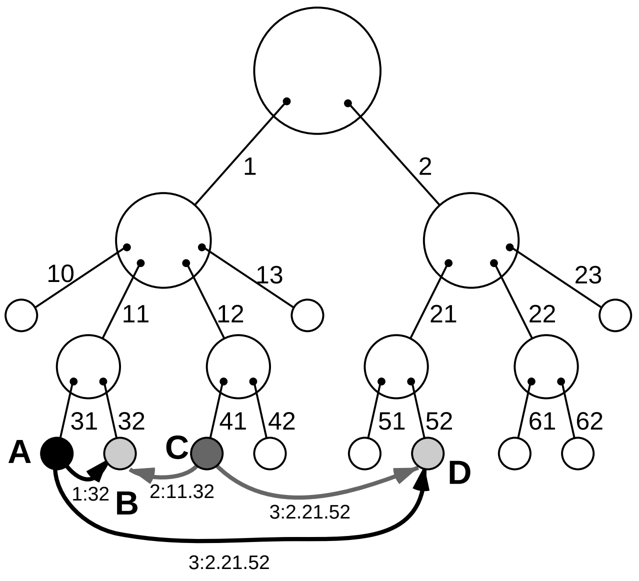

Figure 1: Hierarchical Naming Contexts
Circles represent objects or naming contexts. The numbers represent LIDs.

Figure 1 shows an example of local naming contexts with hierarchical structure. LIDs are all different from each other for simplicity. The connections other than tree branches are omitted. The OID of object B from A is 1 : 32, and that of D is 3 : 2.21.52. The OIDs of B and D from C are 2 : 11.32 and 3 : 2.21.52 respectively. The interpretation of B's OID is 1.11.32 in both A and C, and D's is 2.21.52.

When OIDs are transmitted to another naming context, e.g. the OIDs of the sender, the receiver and the arguments of a message, they are translated to keep their interpretation invariant. If, in Figure 1, a message is sent from object C to D, the receiver's OID 3 : 2.21.52 is translated into 2 : 2.21.52 when the naming context manager with LID=12 receives the message. The translation follows as 1 : 2.21.52, 0 : 2.21.52, 0 : 21.52, 0 : 52, and 0:. When it arrives at D, the receiver's OID is 0:, the OID for itself. The following is the rule for translation of OIDs. Let the transmitted OID be $m : l_1.l_2.\cdots.l_n$.

Rule 1 OID translation

If transmitted to parent naming context: If m is positive, decrement m by one. If m is zero, insert the LID of the naming context manager or the object after ":"[4].

If transmitted to child naming context or child object with LID=l: If m is positive, increment m by one. If m is zero, compare l with l_1. If equal, delete it. Otherwise, increment m by one.

If transmitted to a naming context that is not listed: Perform above translations one by one as going through the branches of the naming context tree until reaching the destination naming context. The collected expression as follows. Let the OID of the destination naming context manager be $M : L_1.L_2.\cdots.L_N$ and the OID of the source naming context seen from the destination be $N : L'_1.L'_2.\cdots.L'_M$.

If $m > M$: Replace m with $m - M + N$.

If $m < M$: Replace m with N and insert $L'_1.L'_2.\cdots.L'_{M-m}$. after ":".

If $m = M$: Let k be the maximum number that satisfies $l_1 = L_1, \cdots, l_k = L_k$. If $l_1 \neq L_1$, let $k = 0$. Delete k LIDs after ":" and replace m with $N - k$. □

This rule does not use the interpretation of OIDs. Especially, if the communication is done only through parent-child connection, naming context managers have only to know the children's LIDs, not global information such as the LIDs of other naming context managers.

5 Object addresses and object migration

An OAD is a hint of the current location of the object[5]. An OAD consists of three parts, a body, a timestamp and an uncertainty value. The timestamp is for indicating the time when the location is correct. The uncertainty value denotes the number of LIDs that are inferred from the OID. The value is a non-negative integer or ∞. The body of an OAD has the same format of OIDs, except that the LIDs are replaced by the LADs. The interpretation of an OAD is defined in the same way as that of an OID. The translation rule for transmission is also the same. Without migration, the OID and the body of the OAD have the same value.

A message includes the sender's OID and OAD. The timestamp of the OAD is the time of message construction, and the uncertainty value of it is 0. A message also includes the receiver's OID and OAD. The body of the OAD is the same as the OID, and the uncertainty value of it is ∞, if the current location is unknown.

Object migration is performed by the following procedure.

Procedure 1 *Object migration*

Begin migration: *The migrating object sends a disconnection message to its original naming context*

[4]This translation is performed in the parent naming context. Memorizing the LID of the naming context manager or the object is not necessary.

[5]The optimum path to the indicated location is the routing problem and not discussed in this paper.

manager. The managers through which the message goes invalidate the pair LID-LAD, LID-OAD, OID-LAD, or OID-OAD of the object.

Do migration: *The object moves. The OIDs belonging to the object are translated including its own OID.*

End migration: *The manager of the new current naming context assigns an LAD to the object. The manager keeps the pair of the object's OID and the LAD. Then the object sends the OID and the new OAD to the manager of the original naming context*[6]*. The manager updates the OAD and returns acknowledgment.* □

The assigned LAD is the same as the LID of the object, if the new current naming context is the original naming context. Otherwise, the LAD must be different from the LIDs of all objects instantiated in the new naming context. If the object does not receive the acknowledgment, it repeats sending the OID and the new OAD with a certain interval[7].

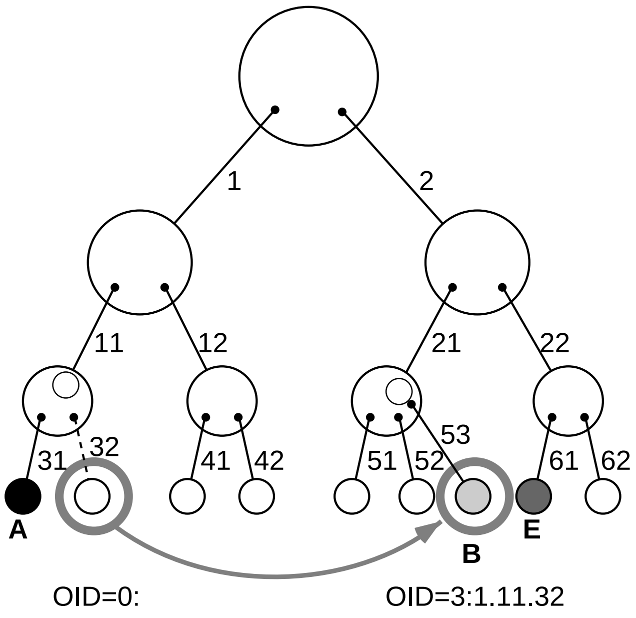

Figure 2: Object Migration

Circles represent objects or naming contexts. The numbers represent LADs.

Examples: In Figure 2, object B has migrated and got 53 as an LAD from the naming context manager with LID=LAD=21. B's OID seen from B has been translated from 0: into 3 : 1.11.32. B sends its OID 3 : 1.11.32 and OAD 0: to the manager of its original naming context (3 : 1.11). This information is cached by the naming contexts with $*$ marks. The managers of the current and original naming context, indicated by ◯ marks, always keep this information. As the OID and the OAD are translated by rule 1, the pair in, for example, the instantiated context is the pair of 0 : 32 and 2 : 2.21.53.

In each naming context, the receiver's OAD of a message is rewritten according to its OID. This is done when the OID and the OAD is in the form of the naming context, i.e., before or after the translation of rule 1 depending on the direction of the transmission.

Rule 2 Rewriting OAD of object A in naming context S.

If A is in S or instantiated in S: Replace the OAD with the correct OAD that S knows.

If A is not in S while OAD indicates that A is in S: Return an error message[8]. If the pair of the OID and the OAD is cached by the naming context that the error message goes through, the pair is invalidated.

If S caches the pair of A's OID and OAD and the timestamp of cache's OAD is newer: Replace the OAD with the cache's OAD.

Otherwise: Do nothing. □

After rewriting, the message is transmitted according to the address information. If a message is sent from object A to B in Figure 2, both B's OID and OAD are 1 : 32 at message construction time. The current context manager rewrites the OAD 0 : 32 as 2 : 2.21.53 which is the correct location. If a message is sent from object E to B, the naming context manager with LID=LAD=2 rewrites the OAD 1 : 1.11.32 as 0 : 21.53 according to the cached information, and the message arrives at object B without going through the original naming context.

6 Mobile naming contexts and dynamic reconfiguration of networks

This section discusses the cases of logical migration of naming contexts and physical migration of hosts or local networks. Here we assume each host or local network corresponds to one naming context[9]. The objects in the naming context migrate together in these cases.

It is not efficient to translate all the OIDs and OADs in all the objects in the migrating naming context. We

[6]This can be implemented as the manager of the current context does.

[7]This is for network partitioning.

[8]There is an alternative way to rewrite the body of the OAD as the same value as the OID. We adopt the way of returning an error because this also disposes the messages to non-existent objects.

[9]If not, e.g. a host has only a part of a naming context, host migration is treated as migration of all objects in the host.

do not touch the OIDs other than that of the naming context manager. The manager translates the OIDs and OADs of the messages to and from inside and outside of the naming context. The manager's OID is translated according to rule 1 in the same way as in Section 5. Procedure 1 in Section 5 are updated as follows:

Procedure 2 *Migration of naming context S*

Begin migration: *The manager of S sends a disconnection message to its original naming context manager. The managers through which the message goes invalidate all the pairs LID-LAD, LID-OAD, OID-LAD, or OID-OAD of S and its descendants.*

Do migration: *S moves. The OIDs belonging to the manager of S are translated including its own OID.*

End migration: *The manager of the new current naming context assigns an LAD to S. The manager keeps the pair of the object's OID and the LAD. Then S sends the OID, the new OAD, and a virtual OID 0: to the manager of the original naming context. The manager updates the OAD and returns acknowledgment. It includes the current value of the virtual OID 0: called the* **reverse OID**. *It must be quoted not to be translated.*

End migration 2: *The manager of S sends a prompt message to its descendants which came from outside of S to send their OIDs and the OADs to the manager of their original naming context.* □

The last operation can be implemented as remembering all the OIDs of the objects migrating from outside of S, or as sending prompt messages to the child naming contexts and the objects which came from outside, recursively. Since migration of a large naming context is rare, the cost of prompt messages is low.

Rule 2 in Section 5 are updated as follows:

Rule 3 Rewriting OAD of object A in naming context S.

If A is an object currently in S or its descendant: That is, if A's OID is $m : l_1.l_2.\cdots.l_n$, and there exists $1 \leq k \leq n$ such that object B with OID=$m : l_1.l_2.\cdots.l_k$ is in S, then replace A's OAD with $0 : L.l_{k+1}.\cdots.l_n$ where B's LAD is L. Let the timestamp be the current time and the uncertainty value be 0.

If A is an object instantiated in S or its descendant: That is, if A's OID is $0 : l_1.l_2.\cdots.l_n$ and the object B with LID=l_1 is instantiated in S, then replace A's OAD with $M : L_1.L_2.\cdots.L_N.l_2.\cdots.l_n$ where B's OAD is $M : L_1.L_2.\cdots.L_N$. Let the timestamp be that of B's OAD and the uncertainty value be 0.

If neither A nor its ancestor is in S while OAD indicates that A or its ancestor is in S: That is, if A's OAD is $0 : L_1.L_2.\cdots.L_N$, and there is no object with LAD=L_1 in S, then return an error message.

If A is an object or its descendant whose OID and OAD are cached by S and if the timestamp of the cache's OAD is newer: That is, if A's OID is $m : l_1.l_2.\cdots.l_n$, and there exists $1 \leq k \leq n$ such that the pair of the OID $m : l_1.l_2.\cdots.l_k$ and the OAD $M : L_1.L_2.\cdots.L_N$ of object B is cached and the timestamp is newer than that of A's OAD (if there are multiple ks, the one with the smallest uncertainty value is used), then replace A's OAD with $M : L_1.L_2.\cdots.L_N.l_{k+1}.\cdots.l_n$. Let the timestamp be that of B's OAD and the uncertainty value be $n - k$.

Otherwise: Do nothing. □

Further, the following translation of OIDs and OADs is performed in the migrating naming context to maintain the consistency between inside and outside of the naming context.

Rule 4 Translation of OIDs and OADs in naming context S.

Let S's OID be $M : L_1.L_2.\cdots.L_N$ and the reverse OID be $N - 1 : L'_1.L'_2.\cdots.L'_M$.

If transmitted to the parent naming context:

After rewriting the OAD according to rule 3,

Translation of OID: Let the OID be $m : l_1.l_2.\cdots.l_n$.

If $m > N$: Replace m with $m - N + M$.

If $m < N$: Replace m with M and insert $L_1.L_2.\cdots.L_{M-m}.$ after ":".

If $m = N$: Let k be the maximum number that satisfies $l_1 = L'_1, \cdots, l_k = L'_k$. If $l_1 \neq L'_1$, let $k = 0$. Delete k LIDs after ":" and replace m with $M - k$.

Translation of OAD: Translate the OAD in the same way as the OID only if the OAD is not a descendant of S.

Transmitting the OID and the OAD to the parent naming context, they are translated according to rule 1 in Section 4.

If transmitted from the parent naming context:

Translation of OID: Let the OID be $m : l_1.l_2.\cdots.l_n$.

If $m > M$: Replace m with $m - M + N$.

If $m < M$: Replace m with N and insert $L'_1.L'_2.\cdots.L'_{M-m}.$ after ":".

If $m = M$: Let k be the maximum number that satisfies $l_1 = L_1, \cdots, l_k = L_k$. If $l_1 \neq L_1$, let $k = 0$. Delete k LIDs after ":" and replace m with $N - k$.

Translation of OAD: Translate the OAD in the same way as the OID only if the OAD is not a descendant of S.

After this translation, rewrite OADs according to rule 3. □

Examples: In Figure 3, the naming context S with LID=LAD=12 has migrated and got 23 as its LAD. The OID and the OAD of the naming context are cached in the same way as in Section 5. Marks $*$ and $\bigcirc$ indicate caching. If a message is sent from object A to F, the receiver's OID and OAD are both 2 : 12.42 in the first place. Then the naming context manager with LID=LAD=1 rewrites the OAD 0 : 12.42 as 1 : 2.23.42. When arriving at S, the OID is 2 : 1.12.42 and the OAD is 0 : 42. The OID is translated into 0 : 42 according to rule 4, and the message is transmitted to F. If a message is sent from object C to B, the receiver's OID is 2 : 11.32 as B's OID from C is independent of the migration of S. S translates the OID 1 : 11.32 into 2 : 1.11.32.

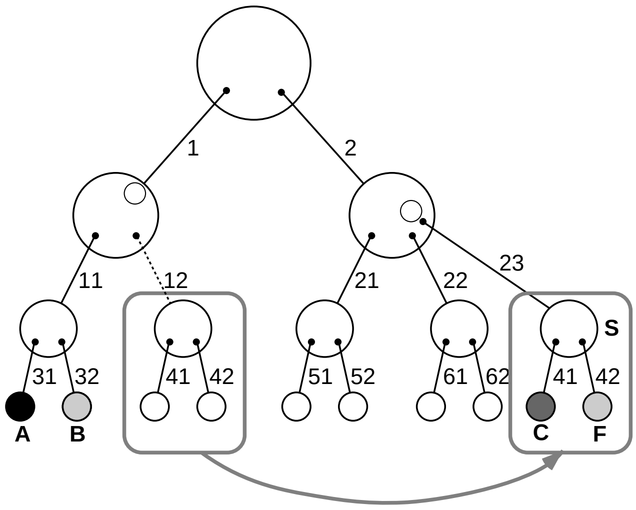

Figure 3: Migration of Naming Context
Circles represent objects or naming contexts. The numbers represent LADs.

In the case of physical migration, there is a problem of constructing the OID of the migrated naming context. It is not trivial because of the relative representation of OIDs. In the case of logical object migration, the OID of the object is translated as the object moves. Physical migration does not allow this method. The destination cannot be determined at disconnection time in most cases.

Considering the format of OIDs, it is enough for OID construction if the number of steps to the common ancestor and the sequence of LIDs to the original location are known. Thus the host that begins to migrate gets some candidates of OIDs from the ancestors at disconnection time. They are used for the notification of the OID and the OAD. For ensuring that the acknowledgment is from the manager of the instantiated naming context, an appropriate authentication method, e.g. public key authentication, is used. If it is successful, the OID is determined. Authenticating by keys is similar to having global unique IDs, however, the situation that needs authentication is limited and one can use keys long enough for ignoring the chance of incorrect construction of the OID. The chance is essentially small as authentication is used with candidates of OIDs.

7 Connecting system for expansion

This section describes the method for connecting two systems that adopt the *hierarchical naming scheme*. If the tree structure is preserved, connecting causes no problems. Two types of connection methods are allowed. The first is letting the root of one system be a child of the naming context of the other system. The second is making a new root naming context, which becomes the parent of the two old roots. At connection time, the interpretations of OIDs and OADs are changed at once. These correctly represent the original locations and the hints of current locations. Translating existing OIDs and OADs is not necessary. Thus the connection operation is local.

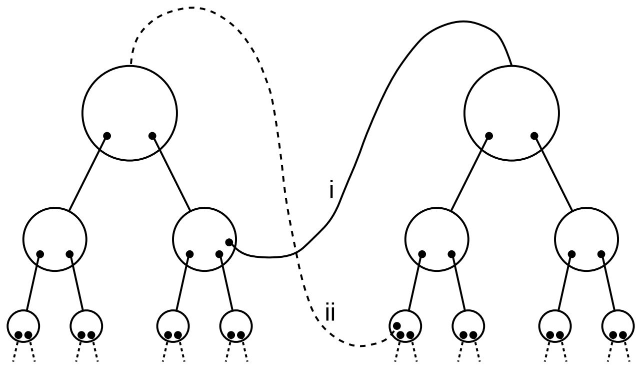

Figure 4: The Case of Causing a Problem when Connecting Systems
Circles represent objects or naming contexts.

It is possible to attempt connection of the same pair of systems at different places if the systems are large enough. Figure 4 shows an example. While one is connecting the left system and the right system by link i, some other one tries to connect the systems by link

ii. Both of the connections destroy the uniqueness of OIDs. This may happen if the connection is made by the first method of letting the root of one system, say A, be a child of the naming context of the other system, say B. Assuming an appropriate authentication method, the following procedure is used.

Procedure 3 *Connection protocol*

Confirming the root: *Get the authentication of B's root, and compare it with that of A's root. If different, go to the next step. If equal, they must already be connected with each other. Send a message to B's root and confirm that A's root receives.*

Lock: *Lock A's root and confirm that B's root is not locked. If locked, unlock it to avoid deadlocks, wait for releasing the lock of B's root and try the protocol again.*

Connect: *Assign A's LID in the naming context of B and connect A's root to B.*

Unlock: *Release the lock of A's root.* □

8 Implementation issues

The implementation of the *hierarchical naming scheme* has issues of variable length OIDs and OADs and the translation of OIDs and OADs when transmitted between naming contexts. This may lead to inefficient implementation because the naming context manager has to seek messages for all the OIDs and the OADs and to translate into, maybe different length, new ones.

Variable Length OIDs and OADs OIDs and OADs are used only when referring objects between different address spaces. For referring objects in a program, short and fixed length IDs (SIDs) are used. SID is unique within the address space of the program. When an object is instantiated in the address space, only an SID is assigned. If it is known by the objects outside the address space, e.g. by sending as a message argument, an LID is also assigned and the OID is constructed. If the program refers to an object outside the address space, an SID is assigned for referring to it in the program. The SID of an object is reused if there are no references to it in the address space and the LID is not assigned to it. The size of SIDs is enough if different SIDs can be assigned to the objects used in one address space.

An SID is implemented as an index of an object table. The table has the addresses of the objects if they are in the same address space. For external objects or objects with external references, the table has their OIDs. Each table entry of an SID has at least one of these. For communication, the program executes a "send" kernel call with SIDs as their destinations and arguments. In the kernel call, the SIDs are translated into OIDs and sent to the receiver's kernel. The kernel translates the OIDs into SIDs for the receiver's addresses space. If the sender and the receiver are in the same address space, the translation is omitted. If they are in the same host, a more efficient way of translation can be used.

Translation of OIDs and OADs The OIDs in a message are not used on the way to the receiver except those of the sender and the receiver. It improves the performance to translate the OIDs in message arguments at the receiver side and not in each naming context the message goes through. For the receiver side translation, the path of the message is recorded. The OIDs are translated as the third case of rule 1 in Section 4. This technique also solves the problem of encryption because the encrypted arguments can be decrypted and translated on the receiver side.

9 Related work

There is another method of giving global names without global management, the Domain Port Model [2] (DPM). It does not assume hierarchical structure. It gives global names by specifying the sequences of ports to the next domain. Thus the structure of naming contexts (domains) is flexible. For ensuring uniqueness, however, the path between each pair of domains must be uniquely decided independently of the actual path of message passing. This is realized by the translation rules kept in each domain. The rules can be incomplete and uniqueness is not always ensured. Though our method has the assumption of hierarchical structure of naming contexts, it does not restrict the physical structure. The merit of the unique IDs of our method is greater than the restriction from this assumption.

Other ways of hierarchical naming are GALAXY [7] [6], the DEC global name service [3], and the Stanford design [1]. GALAXY adopts hierarchical symbolic names using distributed directory management and IDs for identifying objects. The ID consists of the host ID and the local ID, each 64-bit long. The host IDs are assumed to be unique in the world. This is not suitable for our goal.

The DEC global name service adopts absolute paths. Each naming context is called a directory and has unique directory identifier (DI). Directory translation entries (translation rules) are added for changing the root directory. This is not suitable for our goal for two reasons: assuming DIs, and the fact that changing the root causes global addition of translation rules.

In the Stanford design, the structure of naming has nothing to do with the location of objects. The name is directly bound to the object. The existence of name "A/B/C" does not imply the existence of a directory "A/B". Objects are managed by managers, and the addressing of the object managers is by caching. Using multicast or broadcast for cache misses and assuming one root for one system do not suit our goal.

10 Conclusion

This paper has proposed the *hierarchical naming scheme*, which is a way of naming and addressing suitable for large-scale distributed systems. The scheme has the following characteristics:

- It gives unique IDs to objects.
- Short IDs are used for nearby objects.
- It allows to implement scalable systems.
- Communication is performed using only local information.
- It allows object migration.
- It allows mobile hosts and dynamic reconfiguration of networks.

Our scheme separates the representation and the interpretation of object IDs (OIDs) and object addresses (OADs). Relative expression is used for the representation. Using different representations of OIDs and OADs in different naming contexts improves **efficiency**. Local communication can be performed without translation rules, just like the communication in a conventional naming scheme. The interpretation relates the OIDs and OADs to the conventional ones and ensures the **uniqueness** of OIDs. The interpretation is a virtual concept. The global information used for interpretation is never used for implementation. When connecting the systems, the interpretations of OIDs and OADs are changed at once while the representations of OIDs and OADs remain unchanged. The connecting operation is local. This leads to **scalability**. The locality of the operations on OIDs and OADs leads to high **availability** and **fault tolerance**. The hierarchical application of the propagating cache method realizes the **location transparency** with respect to object/host migration and network reconfiguration.

Our scheme requires an appropriate method of authentication for reliable communication to mobile hosts and connected subsystems. Since security and authentication are indispensable to practical distributed systems, the requirement is not a restriction. Our scheme allows the application of large scale authentication [4]. We are investigating the problem of security and authentication using this approach. This research is based on the hierarchical naming of objects in the Muse Operating System [9]. The correctness of our scheme has been confirmed by simulation. Our scheme is being implemented on the Muse Operating System.

Acknowledgments

We would like to express our gratitude to Dr Mario Tokoro (Keio University and Sony CSL) for supervising our research. We would also like to thank Mr Atsushi Mitsuzawa (Keio University) and the other members of Sony CSL for their valuable advice.

References

[1] David R. Cheriton and P. Mann. Decentralizing a Global Naming Service for Efficient Fault-tolerant Access. *ACM Transactions on Computer Systems*, 7(2):147–183, May 1989.

[2] Yasunori Harada and Eiichi Miyamoto. An Open Distributed Language Kamui-C — Introduced Open Systems Naming Model. In *Proceedings of 8th Conference of Japan Society for Software Science and Technology*, September 1991. (in Japanese).

[3] Butler Lampson. Designing a Global Name Service. In *Proceedings of 5th ACM Symposium on Principles of Distributed Computing*, pages 1–10, August 1986.

[4] Butler Lampson, Martin Abadi, Michael Burrows, and Edward Wobber. Authentication in Distributed Systems: Theory and Practice. In *Proceedings of the 13th ACM Symposium on Operating System Principles*, October 1991.

[5] Sape J. Mullender, Guido van Rossum, Andrew S. Tanenbaum, Robbert van Renesse, and Hans van Staveren. Amoeba, A Distributed Operating System for the 1990s. *Computer*, 23(5):44–53, May 1990.

[6] Pradeep K. Sinha, Hirohiko Nakano Kantaro Shimizu, Naoki Utsunomiya, and Mamoru Maekawa. *Network-Transparent Object Naming and Locating in Distributed Operating Systems.* Technical Report 89-033, Department of Information Science Faculty of Science, University of Tokyo, November 1989.

[7] Pradeep K. Sinha, Mamoru Maekawa, Kentaro Shimizu, Xiaohua Jia, Hyo Ashihara, Naoki Utsunomiya, Kyu S. Park, and Hirohiko Nakano. The Galaxy Distributed Operating System. *Computer*, 24(8):34–41, August 1991.

[8] Fumio Teraoka, Yasuhiko Yokote, and Mario Tokoro. A Network Architecture Providing Host Migration Transparency. In *Proceedings of ACM SIGCOMM'91*, September 1991. also appeared in SCSL-TR-91-004 of Sony Computer Science Laboratory Inc.

[9] Yasuhiko Yokote, Atsushi Mitsuzawa, Fumio Teraoka, Nobuhisa Fujinami, and Mario Tokoro. *Continuous-Grained Objects in the Muse Operating System.* Technical Report SCSL-TM-91-008, Sony Computer Science Laboratory Inc., February 1991.