A Location Service for Worldwide Distributed Objects

Franz J. Hauck[1], Maarten van Steen, Andrew S. Tanenbaum

Dept. of Math. and Computer Science, Vrije Universiteit, Amsterdam, The Netherlands

Abstract

This position paper introduces the Globe object model for wide-area distributed systems and its location service. The location service provides transparency of location, migration, distribution, and replication of distributed objects. We present the architecture of the service and briefly discuss scalability.

1 Introduction

Since the early days of the Internet worldwide distributed services are available to users. One problem with this is that their inherent distributed architecture is not transparent to users. While some services in the Internet provide moderate transparency, such as electronic mail and the WWW, programming models for building worldwide distributed applications that scale with respect to large geographical areas and the number of users are generally not available.

One form of transparency is location transparency. For many applications and services, users do not care where the application or a server is located as long as it does its job and has a reasonable response time. An example is the World Wide Web. For object-based systems, location transparency means that objects can be named independently of their physical location. A location is usually expressed in terms of a network address of an underlying network protocol.

Another form is replication transparency. Users need not know whether and how an object is replicated. This is closely related to the naming of objects. A single name for a replicated object or service should be mapped to multiple (sub-)objects or to multiple network addresses without the user being aware of this.

The DNS name service addresses location transparency of Internet hosts and mail addresses [4]. However, the only reason it scales is, because the DNS name-to-address bindings hardly change. This cannot be assumed in general. For example, the World Wide Web suffers from a lack of location transparency. WWW names (URLs) depend on DNS domain names and usually on a filename on the server's file systems [1]. Thus, URLs are coupled to hosts. If one WWW document, such as a user's home page, has to move to another WWW server or to another directory it has to get a new URL.

Author's current address is IMMD-IV, University Erlangen-Nürnberg, Germany.

Even worse is replication transparency in today's systems. There are mirrors for heavily used WWW and ftp sites, but the same file or document has multiple names, one for each replica. In CORBA, replication services have not yet been incorporated at all. It remains to be seen if a general replication mechanism can be devised that is applicable in all cases, and in a transparent way [6]. In Spring, replication can be achieved by special *Subcontracts* which handle replication at the client side by multiple invocations in different objects [2]. It is unclear to us how this will scale to wide-area systems.

In Section 2, we introduce the Globe object model which is a uniform model for building distributed applications. We show how state is replicated and how naming is done. Section 3 focuses on the location service of Globe which is the second step in a two phase naming system. We discuss the architecture of the service and its scalability. Finally, we give our conclusions in Section 4.

2 Globe Object Model

The Globe architecture introduces *distributed shared objects* [9]. Objects provide methods made available through interfaces. Objects are passive. Activity is provided by processes that can share objects and can invoke their methods concurrently. An object's state can be physically distributed through *local objects*. A local object resides in exactly one address spaces and communicates with other local objects to form a distributed object. This is completely transparent to clients.

If a process wants to access an object, it first has to bind to it. To that end, the process has to create a local object in its own address space. This local object connects to the distributed object, and thus becomes part of it. This is different to most other object-based models, which adopt a general client/server approach, such as DCE [7], CORBA [6], and Spring [3]. A local object can play the role of a client stub as in these models, but it can also be an important part of the distributed object by fully or even partially replicating its state.

We use a two-level naming scheme for objects. The *name service* forms the first level of the naming system. It maps user-defined verbose names to *object handles*. An object handle is a pure name [5]: it is a fixed size bit pattern that is uniquely assigned to every object.

The *location service* handles the second stage of naming by mapping object handles to one or more so called *contact addresses.* A contact address contains the network address of a *contact point* located in one of the local objects of the distributed object. Additionally, it contains a protocol identifier for a initial binding protocol. For the binding, a process creates a new local object which knows the requested binding protocol and initializes the local object with the contact address.

Each of the local objects may provide a contact point for the distributed object. Therefore, the local object registers or unregisters the corresponding contact address at the location service. Migration of objects is expressed in terms of newly created contact points and deletion of old ones. Rather than using the term migration, we say that a distributed object expands and shrinks.

The first part of the naming system can provide one or more user defined names for an object. Each name is resolved to a single object handle. The second part of the naming system, implemented by the location service, achieves transparency of location, migration, distribution, and replication. Whether the object is distributed or mobile, whether it is replicated, and how replicas are kept consistent, is completely hidden to the client. Having the object handle at hand, a process may bind to the object without knowing the physical location of its local objects.

3 Location Service

The first part of the naming system is subject of ongoing research. It can be designed without looking at replication or location of objects. As mentioned above, the second part, the location service, maps object handles to one or more contact addresses. We have devised a location service that scales worldwide and that can support trillions of objects.

The location service is structured using a search tree of *directory nodes*. Each node represents a geographical, topographical, or administrative region of the worldwide communication system. The region of an intermediate node is the union of all the regions in its subtree. Finally, the root node of the tree represents the entire world. Fig. 3–1 illustrates a search tree and its regions.

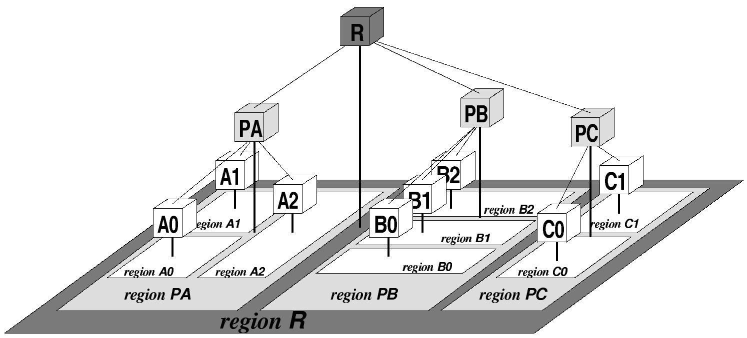

Fig. 3–1 An example of a search tree of the location service and its regions.

1.1 Operations

When an object registers a contact address at the location service it is usually stored in the directory node of the smallest enclosing region in which the address is located, a leaf node of the search tree. For each object, a path of forwarding pointers is established starting from the root node of the tree to each place where a contact address is stored. Fig. 3–2 shows a search tree and the stored data for one object with two contact points in different regions.

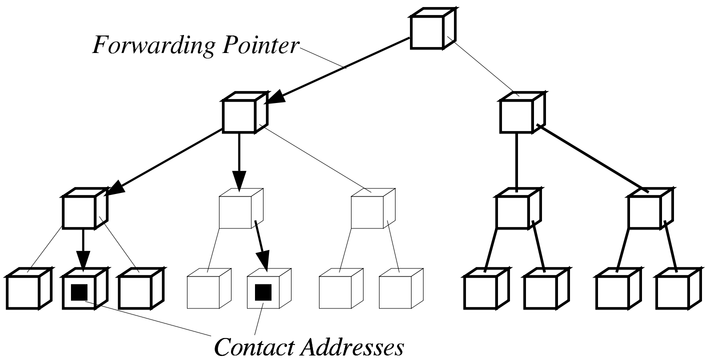

Fig. 3–2 A search tree with two contact addresses for one object.

A client starts searching in the region where it is located. If it does not find contact addresses or a forwarding pointer, it successively goes up the tree expanding the search area. In the worst case it will find a forwarding pointer in the root node. Once a forwarding pointer is found it is followed until a contact address is reached. This scheme prefers contact addresses that are located near a client and thus reduces the search path.

To reduce the length of a search path as much as possible, we apply a number of optimizations. First, we cache pointers to nodes storing contact addresses. Whenever we find a contact address, all nodes on the return path of a lookup request update their caches. Cached pointers are preferred on search operations and reduce the search path to a length of two in the best case. Cache entries are invalidated on time-outs and when an attempt to follow a cache pointer fails.

Second, we collect stability information on contact addresses. A directory node does not hand out cachable pointers if the stored contact addresses are likely to be withdrawn by the object. This instability is derived from the insert and delete history in the corresponding region. Third, instable addresses are stored in higher level directory nodes to get better stability measurements and to increase the probability of stableness.

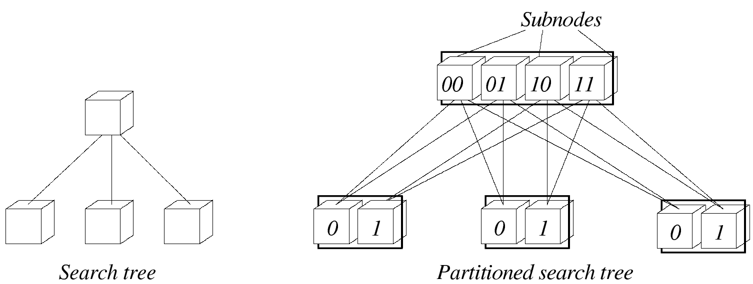

Fig. 3–3 The partitioning of directory nodes into subnodes.

2.2 Implementation and Scalability

The search tree is implemented by partitioning each directory node into a number of servers called *subnodes*. Each subnode serves a dif ferent part of the object handle space essentially by using hashing techniques. In Fig. 3–3 we use the first two bits of an object handle for selecting the root subnode, and only the first bit for selecting a child subnode[1]. The partitioning introduces scalability. The root node of the tree has to know about every object in the system, but we can show, that each subnode of the root will have to store only about 10 gigabytes of data if the root is partitioned in one subnode per 10^8 objects. More details about scalability can be found in [8].

4 Conclusion

We introduced the Globe object model and its location service which provides transparency of location, migration, distribution, and replication. We can show that the service scales to a huge number of objects. Finally, we developed sophisticated insert, delete, and lookup algorithms which can handle concurrent invocations and fault tolerance. Without going into the details, the algorithm can handle network partitions and node failures without loss of data by exploiting RPC semantics. Current work is being done on prototype implementations and simulations. Further research is considering failure resilience and security.

References

1. T. Berners-Lee, L. Masinter, M. McCahill. “Uniform Resource Locators (URL).” *RFC 1738*, Dec. 1994.
2. G. Hamilton, M. Powell, J. Mitchell. “Subcontract: A flexible base for distributed programming.” In *Proc. 14th Symp. on Operating Sys. Principles*, Asheville, NC, Dec. 1993. ACM.
3. J. Mitchell et al. “An overview of the Spring system.” In *Proc. Compcon Spring 1994*. IEEE, Feb. 1994
4. P. Mockapetris. “Domain names – concepts and facilities.” *RFC 1034*, Nov. 1987
5. R. M. Needham. “Names.” In S. Mullender, (ed.), *Distributed Systems*, pp. 315–327. Addison-Wesley, Wokingham, 2nd edition, 1993.
6. Object Management Group. “The Common Object Request Broker: Architecture and specification, Revision 2.0.” Techn. Report 96-03-04, OMG, July 1995.
7. W. Rosenberry, D. Kenney, G. Fisher. *Understanding DCE*. O’Reilly, Sebastopol, Calif., 1992.
8. M. van Steen, F. J. Hauck, A. S. Tanenbaum. “A model for worldwide tracking of distributed objects.” In *Proc. TINA ‘96 Conference*, VDE, Berlin, 1996.
9. M. van Steen et. al. “Towards object-based wide area distributed systems.” In L.-F. Cabrera and M. Theimer, (eds.), *Proc. 4th IWOOOS*, pp. 224–227, Lund Sweden, Aug. 1995. IEEE.