Proceedings of FREENIX Track:
2000 USENIX Annual Technical Conference

San Diego, California, USA, June 18–23, 2000

THE GLOBE DISTRIBUTION NETWORK

A. Bakker, E. Amade, G. Ballintijn, I. Kuz, P. Verkaik, I. van der Wijk,
M. van Steen, and A.S. Tanenbaum

The Globe Distribution Network

A. Bakker, E. Amade, G. Ballintijn
Department of Mathematics and Computer Science
Vrije Universiteit
Amsterdam, The Netherlands
I. Kuz
Delft University of Technology
P. Verkaik, I. van der Wijk, M. van Steen, A.S. Tanenbaum
Vrije Universiteit Amsterdam
arno@cs.vu.nl, http://www.cs.vu.nl/globe/

Abstract

The goal of the Globe project is to design and build a middleware platform that facilitates the development of large-scale distributed applications, such as those found on the Internet. To demonstrate the feasibility of our design and to test our ideas, we are currently building a new Internet application: *The Globe Distribution Network.* The Globe Distribution Network, or GDN, is an application for the efficient, worldwide distribution of free software and other free data. The GDN can be seen as an improvement to anonymous FTP and the World Wide Web due to its flexibility and extensive support for replication. This paper describes the design of the GDN. We start by explaining how the replication facilities of the Globe middleware are used to make the GDN efficient, and how these facilities are implemented. Next, we present the architecture of the GDN and discuss how the Domain Name System can be used as a first approach towards a worldwide service for naming software packages and other entities. This is followed by an analysis of the security requirements for the GDN and measures taken to satisfy these requirements. We hope to make Globe and GDN itself available for free under the BSD license by 2001.

1 Introduction

Developing a large Internet application is a difficult task due to the complex nonfunctional aspects that have to be taken into account. A developer has to deal with a potentially very large number of users, high communication delays, security threats, and machine and network failures. The goal of the Globe project is to design and build a middleware platform that facilitates the development of worldwide distributed applications by providing extensive support for handling all of these complex nonfunctional aspects [van Steen *et al.*, 1999].

As replication is a powerful technique for dealing with many of these aspects, support for flexible replication plays an important role in the Globe middleware. In Globe, processes communicate by invoking methods on a special kind of distributed object, called a *distributed shared object* (DSO). What makes a distributed shared object special is that we can vary the replication strategy on a per-object basis, allowing the way the object is replicated to be governed completely by object- and application-specific requirements with respect to consistency and nonfunctional aspects, such as security and fault tolerance. Replication and application code are separated, which means that we can reuse replication protocols developed for one distributed shared object to build other DSOs.

The first version of our middleware platform is nearly complete. To demonstrate the feasibility of our ideas and the design of our middleware we are currently building a prototype of a new Internet application using the Globe middleware. This paper describes the design of this application, called the *Globe Distribution Network.* The Globe Distribution Network, or GDN for short, is an application for the efficient, worldwide distribution of data. In the beginning, it will be used for the distribution of publicly redistributable software packages, such as the GNU C compiler, Linux distributions and shareware. We intend, however, to extend this to other types of data, such as free digital music, in the future.

Because it deals with the worldwide distribution of data, the GDN is similar in function to the World Wide Web.

They do differ in one important aspect, however. Although the architecture of the World Wide Web has been shown to be quite scalable, the WWW does suffer from performance problems. We think that these problems are mainly caused by the Web's limited and inflexible support for replication. The GDN therefore needs extensive and flexible replication support. This will be provided by the Globe middleware, via its distributed shared object concept. It is therefore better to compare the GDN with commercial content delivery networks, such as Digital Island's *Footprint* [Digital Island, Ltd., 2000], or wide-area file systems such as AFS [Howard *et al.*, 1998] or CODA [Satyanarayanan *et al.*, 1990].

To avoid any confusion: the purpose of the GDN application is not to replace the Web or become the world's leading distributed file system. It is a research vehicle which should demonstrate the feasibility of our ideas about middleware for large Internet applications. The GDN is a prototype of a system that could one day replace the Web as the Internet application for distributing information, just as the Web has essentially replaced FTP. Nonetheless, it is a serious application that will be running and publicly accessible on the Internet.

Throughout this paper we refer to two versions of the GDN. The first version of the GDN is a limited prototype that will run entirely on machines at our university in June 2000. The version intended to be used by the general public is scheduled to be ready by the end of 2000 and is referred to as the second version of the GDN. The source code of Globe and the Globe Distribution Network will be made available under the BSD license by the end of 2000.

The rest of this paper is organized as follows. Section 2 describes the functionality of the Globe Distribution Network. Section 3 explains how we intend to make the distribution of software packages efficient by using the Globe middleware. After this explanation we present the architecture of the GDN in Section 4. In Section 5 we describe the naming of software packages and our prototype worldwide name service for distributed shared objects. An analysis of the security requirements and the concrete measures we take to meet these requirements are described in Section 6. Section 7 discusses availability of the various Globe and GDN components and gives an overview of their current status. A summary of the paper and our future plans for the GDN can be found in Section 8.

2 The Globe Distribution Network

The Globe Distribution Network is to be a worldwide distributed application for the efficient dissemination of free software packages (e.g. Gimp, teTeX and Linux distributions) and other free data. We assume, for the first versions of the application, that a software package has the following basic properties:

1. It consists of one or more files
2. It has a unique name
3. The collection of files or the individual files that are part of the package can be very large

The functionality of the GDN is initially simple: it should be possible to add software packages to the GDN, retrieve copies of software packages, update them and remove packages that are no longer of interest.

We currently divide the user community into three groups: the *GDN users*, the *GDN moderators* and the *GDN administrators*. GDN moderators are allowed to create, update and remove software packages. GDN users are allowed to retrieve packages only. To add a package to the GDN, GDN users must contact a GDN moderator. GDN administrators have complete control over the GDN application and hand out moderator privileges. In the future we intend to introduce a fourth group, the *GDN maintainers*. A GDN maintainer is allowed to manage just the contents of a package. He or she would typically be the person that also maintains the software package (i.e., fixes bugs, etc.). In the first versions we, the Globe team, will play the role of GDN administrators, and together with a number of volunteers act as GDN moderators.

3 Distributing Packages Efficiently

3.1 Flexible Replication

To make software packages available to a worldwide audience they will need to be replicated, for two reasons. First, there are a potentially very large number of people interested in a particular software package and multiple machines are needed to handle such a load. Second, wide-area bandwidth is a scarce resource and with interested people distributed all over the world replicas must be created close to where the clients are (e.g. in each country) to avoid wasting bandwidth. This is, in fact, a trade-off between server capacity (disk space) and bandwidth. Another reason for replicating packages close to clients is the resulting low response time (i.e., a down-load starts quickly), which is an important usability aspect.

However, replication does not come for free. Each replica of a software package, or, in general, a piece of information, requires a certain amount of disk space and also computing resources while it is begin transmitted. Moreover, there is the management aspect: when replicated data is changed, the different replicas have to be made consistent again, and adding or removing replicas to adapt to changes in access patterns is often not fully automated.

For software packages the cost of replication is not a problem. Most countries probably have their own replicas of the complete collection of freely redistributable software packages, distributed over a number of machines throughout the country. The cost of replication does become a problem if we start looking at using the GDN for distributing other types of information. The amount of data that people want to make available to the world is enormous (cf. the Web). Furthermore, the change rates of this data can be much higher.

From this we conclude that for the Globe Distribution Network to be efficient, we should selectively replicate the information we are distributing, based on popularity and update patterns and that the information's *replication scenario* should adapt to changes in its popularity and rate of change. We use the term replication scenario to denote a specification of *how* (using what replication protocol) and *where* (which machines should host replicas) information or objects should be replicated.

We have found evidence to support this conclusion. We analyzed the retrieval and update patterns of our department's Web pages and found that, if we assign a replication scenario to each Web page that reflects that page's individual usage and update patterns, we get significant improvements in a number of areas compared to situations in which a single replication scenario is used for the whole site. In particular, we found that less wide-area network traffic was generated and the response time for the end-user improved [Pierre *et al.*, 1999]. Although this is just one case study, it does suggest that performance problems for large-scale data distribution systems such as the Web and the GDN can be alleviated by introducing more flexible replication capabilities.

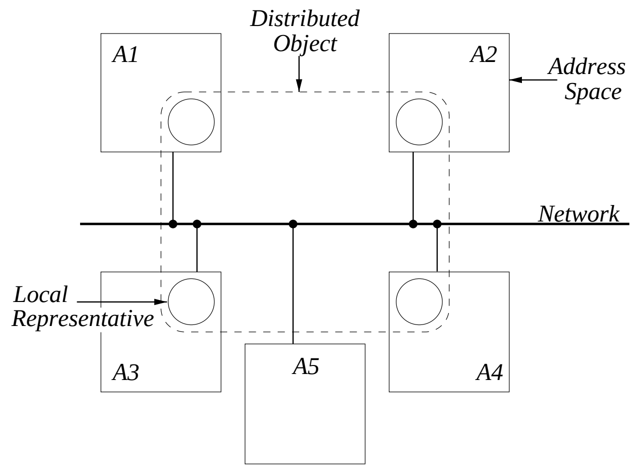
We believe that the ability to selectively replicate data is something that is required by all large Internet applications. Therefore, this is an important part of the Globe middleware. Globe is based on the concept of a distributed shared object. The most important aspect of the distributed shared object for the purposes of this paper is that it allows different replication scenarios to be assigned to each object. The distributed shared object concept is discussed in detail in the next section.

All data stored in the GDN is stored in distributed shared objects. For example, every software package is contained in a *package DSO*. By assigning the right replication scenario, we can make efficient use of the available servers and bandwidth.

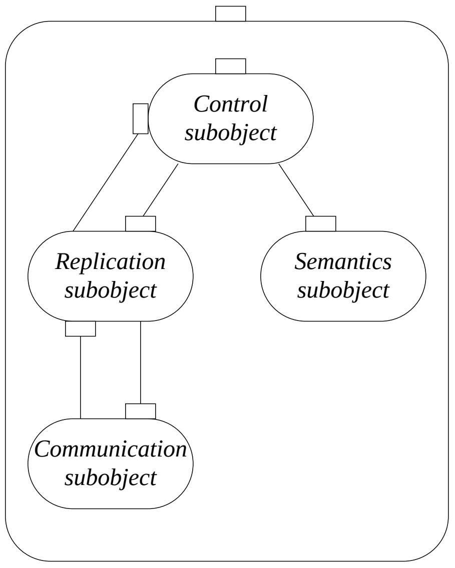
3.2 Globe's Distributed Shared Objects

The distributed shared object is the unifying concept in the Globe system [van Steen *et al.*, 1999]. It provides a uniform representation of both information and services and implementation flexibility by decoupling interface and implementation. The fundamental idea behind the design of the distributed shared object is that it is *physically distributed*. Most current middleware, such as CORBA [Object Management Group, 1999] and DCOM [Eddon and Eddon, 1998], view a distributed object as an object running on a single machine, possibly with copies on other machines. This object (group) is presented to remote clients as a local object by means of proxies. In contrast, we view a distributed shared object as a distributed entity, a conceptual object distributed over multiple machines with its *local representatives* (proxies and replicas) cooperating to make the object's functionality available to local clients. In other words, a distributed shared object is a wrapper encompassing all the object's proxies and replicas, rather than a remotely accessible object implementation. This view is illustrated in Figure 1(a).

Our view of what a distributed object is gives us flexibility with respect to replication, caching and distribution of the object's state. A distributed shared object encapsulates its own replication and distribution strategy. The local representatives of an object take care of the replication and distribution of the DSO's state and all necessary communication. Only minimal (protocol independent) support is required from the run-time system. This means that the way the state of the object is replicated can now be governed completely by object- and application-specific requirements with respect to consistency and nonfunctional aspects, such as security, and is under no restriction from the supporting middleware platform. However, we do not leave everything to the application programmer. The structure of local representatives, described below, separates replication and communication code. This means that a programmer can write his or her own replication protocol based on existing communication protocols. Furthermore, we pro-



(a)



(b)

Figure 1: (a) A distributed shared object (DSO) distributed over four address spaces (A1-A4). In each address space the DSO is represented by a local representative. Address space A5 does not currently contribute to the distributed shared object. (b) A local representative is composed of a number of subobjects. The exact composition depends on the role the local representative plays in the distributed shared object.

vide the application programmer with implementations of frequently used replication protocols.

3.3 Implementation of the Globe Object Model

In this section we describe how this object model can actually be implemented. Logically, a DSO consists of multiple local representatives. A local representative resides in a single address space and communicates with local representatives in other address spaces. Each local representative is composed of several subobjects as shown in Figure 1(b). A typical composition consists of the following four subobjects.

Semantics subobject: This is a local object that implements (part of) the actual semantics of the distributed object. As such, it encapsulates the functionality of the distributed object. The semantics subobject consists of user-defined primitive objects written in programming languages such as Java, C, or C++. These primitive objects can be developed independent of any distribution or replication issues. In the case of a package DSO this subobject would implement all the DSO's methods, such as methods for adding files to a package, for listing the files currently in a package and for retrieving the contents of a file.

Communication subobject: This is generally a system-provided subobject (i.e., taken from a library). It is responsible for handling communication between parts of the distributed object that reside in different address spaces, usually on different machines. Depending on what is needed from the other components, a communication subobject may offer primitives for point–to–point communication, multicast facilities, or both.

Replication subobject: The global state of the distributed object is made up of the state of the semantics subobjects in its local representatives. A DSO may have semantics subobjects in multiple local representatives for reasons of fault tolerance or performance. In particular, the replication subobject is responsible for keeping the state of these replicas consistent according to some (per-object) coherence strategy. Different distributed objects may have different replication subobjects, using different replication algorithms. For example, one object may actively replicate all the state at all the local representatives while another may use lazy replication. An important observation is that the replication subobject has standard interfaces.

Control subobject: The control subobject takes care of invocations from client processes, and controls the interaction between the semantics subobject and the replication subobject. This subobject is needed to bridge the gap between the user-defined interfaces of the semantics subobject, and the standard interfaces of the replication subobject.

A key role, of course, is reserved for the replication subobject. Replication (and communication) subobjects are unaware of the methods and state of the semantics subobject. Instead, both the replication subobject and the communication subobject operate only on opaque invocation messages in which method identifiers and parameters have been encoded. This independence allows us to define standard interfaces for all replication and communication subobjects. This approach is comparable to techniques applied in reflective object-oriented programming [Kiczales *et al.*, 1991].

3.4 Binding to a Distributed Shared Object

To access a distributed shared object (i.e., to invoke its methods), a client first needs to install a local representative of the object in its address space. The process of installing a local representative in an address space is called *binding*. Before we explain binding, however, we first have to describe how naming is done in the Globe middleware.

Each DSO in Globe is identified by a worldwide unique object identifier (OID). This object identifier, or *object handle*, never changes during the lifetime of the object and, most importantly, is location independent. The actual locations of the DSO, that is, *where* (network address, port number) its local representatives are located, and *how* (which replication and communication protocol) they can be contacted is maintained by a special service, the *Globe Location Service* (GLS) [van Steen *et al.*, 1998]. Typically only local representatives acting as replicas are registered in the GLS. The information that identifies the location of a local representative and how to talk to it is called a *contact address*. The set of contact addresses stored in the GLS for a specific DSO describes that object's replication scenario.

Object identifiers are long strings of bits and thus unusable for humans. We therefore have an additional name service which maps symbolic names to object identifiers. This results in two-level naming scheme: symbolic object names are mapped to object identifiers by the *Globe Name Service* (GNS) which are, in turn, mapped to one or more contact addresses for the object by the Globe Location Service. The inner workings of the Globe Location Service are described in the next section. Our prototype of the Globe Name Service is discussed in Section 5.

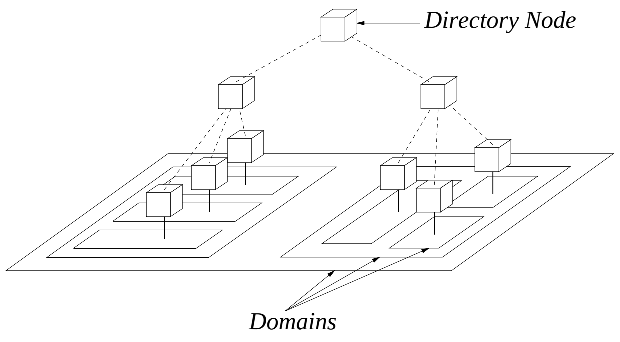


Figure 2: The Globe Location Service divides the Internet into a hierarchy of domains, represented by rectangles in the figure. Associated with each domain is a directory node, represented by a little box.

Binding to a DSO now works as follows. For brevity we assume the client has already acquired an object identifier of the DSO whose methods it wants to invoke. The client calls a special function in the run-time system, named bind, and passes it the object identifier. The run-time system takes the OID and asks the Globe Location Service to map this OID to one or more contact addresses. In general, the returned contact addresses will identify the nearest replica of the DSO. Using the information in the contact addresses, the local run-time system then creates a new local representative in the client's address space and integrates this new representative into the DSO. This involves loading the implementation of the local representative (i.e., the appropriate set of subobjects) from a nearby implementation repository in a way similar to remote class loading in Java.

3.5 The Globe Location Service

To efficiently map object identifiers to contact addresses on a worldwide scale, we organize the Internet into a hierarchy of *domains*. The domains at the bottom of the hierarchy represent moderately-sized networks, such as a university's campus network or the office network of a corporation's branch in a certain city. The next level in the hierarchy is formed by combining these leaf domains into larger domains (e.g. representing the city's MAN). This procedure is applied recursively until the root domain which encompasses the whole Internet. Note that domains in this hierarchy do not necessarily correspond to DNS domains.

With each domain in the hierarchy we associate a *directory node*, as shown in Figure 2. Each directory node keeps track of the locations of the distributed shared objects in its associated domain, as follows. For each DSO that has local representatives in the node's domain, a directory node stores either the actual contact address (network address and protocol information for contacting the representative) or a set of *forwarding pointers*. A forwarding pointer points to a child directory node and indicates that a contact address can be found somewhere in the subtree rooted at that child node. Because a DSO may consist of multiple replicas located in different child domains, a directory node may store more than one forwarding pointer per DSO. Normally, the contact addresses are stored in the leaf directory nodes. However, storing the addresses at intermediate nodes may, in the case of highly mobile objects, leads to considerably more efficient look-up operations, as we explained in [van Steen *et al.*, 1998]. This design has some (apparently) radical consequences. For each DSO on the Internet, there is a tree of forwarding pointers from the root node to the directory nodes that contain the actual contact addresses. Before we explain that this, in fact, does not create a single point of failure or bottleneck, we first look at how object identifiers are resolved.

During binding, the (run-time system of a) client sends a look-up request to the directory node of the leaf domain the client is located in. The leaf node checks if it has a contact address for that DSO in its tables (i.e., it checks if the DSO has a representative in this (leaf) domain). If not, it forwards the request to its parent node, which, in turn, checks its tables. This process is repeated until either a contact address for the object is found or a forward pointer is discovered. In the latter case, the look-up operation continues down into the subtree pointed to by the forwarding pointer and follows the tree of forwarding pointers to the node in that subtree that stores the actual contact address. If multiple forwarding pointers are found, one is chosen at random.

The advantage of this design is, that if a distributed shared object has a representative near to the client, the Globe Location Service will find that representative using only "local" communication. In other words, the cost of a look up increases proportional to the distance between client and nearest representative.

The apparent problem with this design is that the root node, or in general, the higher-level nodes in the hierarchy have to store a lot of forwarding pointers and handle a lot of requests (if representatives of the DSO are not located near their prospective clients). Our solution to this problem is to partition a directory node into one or more *directory subnodes*. Each subnode is made responsible for a specific part of the object-identifier space via a special hashing technique and can run on a separate machine. For further details we refer to [Ballintijn and van Steen, 1999a].

4 The GDN Architecture

Having explained the distributed shared object concept, we can now describe the basic architecture of the GDN application. The core of the application is a set of *Globe Object Servers* (GOSs), running on machines all over the world. A Globe Object Server is an application-independent daemon for hosting replicas of any kind of distributed shared object. Globe Object Servers allow replicas to save their state during a reboot and reconstruct themselves afterwards. The set of GOSs hosts the replicas of the DSOs containing the software packages.

To access the contents of a package DSO, a user would normally have to start up a tool that binds to the distributed shared object and allows the user to invoke methods on that package DSO. The disadvantage of this approach is that users have to run a dedicated client to access the GDN. We want to make the threshold for users to access the GDN as low as possible, and have therefore decided to make the GDN accessible through standard Web browsers. Furthermore, being able to access the GDN via a Web browser allows use to easily integrate it with the World Wide Web.

As such the GDN also consists of a number of modified HTTPDs running on machines all around the world. In our first versions they will be colocated with the Globe Object Servers. These modified, or *GDN-enabled* HTTPDs work as follows. We use URLs that have embedded in them the name of a package DSO. The GDN-HTTPD extracts this object name and binds to the DSO. The HTTPD then invokes the appropriate method(s) on the package DSO's newly created local representative. For example, it could call listContents() to obtain the list of files contained in the package, which is subsequently reformatted into HTML and sent back to the requesting browser. If the URL designates a particular file in the package, the HTTPD calls the getFileContents() method and sends back the returned content. The local representative that is installed in the GDN-HTTPD during binding may act as a replica for the DSO, in which case downloading a software package is fast.

Users communicate with only one GDN-HTTPD, in particular, with the one nearest to them. This HTTPD is the user's access point to the GDN. We currently require users to manually select this HTTPD, using a list published on a central web site. Once connected to the

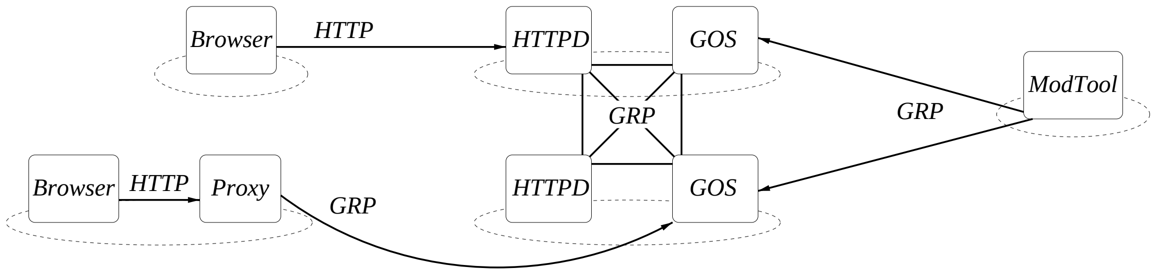


Figure 3: The architecture of the GDN application. Ovals represent sites, rounded boxes represent programs running at those sites, arrows and thick lines represent communication. GRP stands for Globe Replication Protocol. ModTool is the moderator tool. Omitted from this figure are the programs belonging to the Globe Location and Name Services.

GDN, however, the storage location of software packages becomes transparent. The GDN will transparently find the nearest replicas using the Globe Location Service.

Using standard Web browsers is fine, but we would also like people to use the GDN directly. To this extent we allow people to run GDN-HTTPDs on their own machines. We refer to these GDN-HTTPDs as *GDN-enabled proxy servers*, or GDN-proxy servers for short. The last element in the architecture are the moderator tools. A GDN moderator (see Section 2) can add, update and delete package DSOs from the GDN, using a special tool. Figure 3 shows the complete architecture of the GDN.

5 Naming Packages

Most software packages have unique names and people should, of course, be able to retrieve and update packages from the GDN using that name as a key. In addition, we would like the GDN to support some form of attribute-based search, such that people can look for a software package with some specific functionality.

We introduce a hierarchical name space in which the first part of the name gives some information about what a software package does. For example, the *Gimp* graphics package would be named something like /apps/graphics/Gimp to indicate that it is a package for manipulating graphics. A package is allowed to have more than one name so we can have multiple classifications. Having a hierarchical name space also allows us to name DSOs other than packages in a separate name subspace in the future. The exact structure of the name space (/apps, /os, /middleware, ...) is outside the scope of this paper.

As described above, the assignment of human-readable names to distributed shared objects is handled by the Globe Name Service (GNS). The GNS found in the current Globe middleware is a prototype version based on the Domain Name System [Mockapetris, 1987]. The reason for using DNS is that we wanted to build a reasonable name service in a short period of time, so we took an existing system that was suitable for our purposes.

DNS maps symbolic names to other types of data and scales to large numbers of users. DNS works under the assumption that the mapping of names to addresses does not change very frequently. This allows the DNS to cache entries at client-side resolvers and to replicate parts of the database on multiple machines. Combined with distributing the mapping of names to addresses across hosts this results in a scalable system. We can make that same assumption: we expect our name-to-object-identifier mappings to be stable, because of the two-level naming scheme of Globe.

The DNS-based version of the Globe Name Service works as follows. Globe object names have a one-to-one mapping to valid DNS names. These DNS names point to a TXT DNS Resource Record that contains the encoded object identifier for the DSO. To map a Globe object name, say /nl/vu/cs/globe/somePackage, to a Globe object identifier, the object name is first translated to a DNS name, in this case somePackage.globe.cs.vu.nl. This DNS name is then resolved using the normal DNS name resolution mechanism and returns a TXT record from which the object identifier is extracted.

An advantage of this approach is that there is a global name space for objects (the DNS name space) and anyone in control of a DNS domain can create their own subspace in this name space which is immediately accessible to anyone in the world. There are also disadvantages. Firstly, DNS places restrictions on name syntax (i.e., which characters can be used in a name and how long the individual parts of a name can be) which have been lifted in modern name systems. Secondly, DNS domain names are always part of object names, which is not always desirable. Thirdly, the current DNS is insecure because it is vulnerable to spoofing attacks [Vixie, 1995]. We come back to this issue in Section 6.

For the Globe Distribution Network, we intend to work around the second disadvantage. We do not want users to see the DNS domain, we want them to be able to use names such as /apps/graphics/Gimp. To achieve this we use only a single DNS leaf domain to register the names of package DSOs. This means that we can omit the DNS domain name part from the package DSO's name, given that we also modify the GDN software to always prefix this DNS domain name before it is passed to the Globe Name Service. We refer to this DNS leaf domain as the *GDN Zone*.

We expect that this will not cause problems for the first two versions of the GDN. For these versions, we control the addition and naming of package DSOs to the GDN and we can distribute the load by creating multiple authoritative name servers. The number of updates to our zone can be kept low by batching them. For later versions we hope to replace the DNS-based prototype with a GNS based on distributed shared objects [Ballintijn and van Steen, 1999b].

6 Security

6.1 Security Requirements

An important aspect of any new Internet application is security. We discuss security of the Globe Distribution Network in three parts. First, we identify the security requirements for the GDN. We start by identifying the requirements at a high level of abstraction and then translate them into more specific requirements. Second, we describe the security situation, that is, the assumptions we make about the machines on which the GDN will run and their network environment. Finally, we describe the concrete measures we take to satisfy the identified security requirements given the environment in which the GDN will operate.

An important security requirement is that the GDN application is protected against unauthorized use. It should not be possible to use the GDN for the unlawful distribution of commercial software, copyrighted music and such. In the beginning, the GDN will be used primarily for distributing software packages which results in two additional security requirements: attackers should not be able to violate the integrity of the software being distributed and users of the GDN should be assured of the origin of the software. Another requirement is availability. Like the Web and FTP, the GDN application should be highly available and measures should be taken to fend off attacks intended to stop the application from operating. Other factors threatening availability are host and network failures. How these failures are handled in the GDN, and more general, in the Globe middleware is still an open research issue, but replication is, of course, one technique. These high-level requirements can be translated into more specific requirements, as follows.

Unauthorized Use

Only a GDN moderator should be able to add packages, names, etc. to the GDN. This (a) prevents people from filling the GDN with junk packages (i.e., denial of service through resource allocation) and (b) it prevents the GDN from being used for the illegal distribution of copyrighted data. We can further split this up into a number of subrequirements.

Adding and Removing Packages Only a moderator should be able to add and removing packages to the GDN. Adding a package DSO consists of a number of steps, which are executed by the moderator tool (see Section 4). The creation of a new package DSO starts with the definition, by the moderator, of the package's replication scenario. Recall that the replication scenario of a DSO describes how (using what replication protocol) and where (which machine(s) should host replicas) a DSO should be replicated. The moderator tool will present the moderator with the choice of available replication protocols and the set of available Globe Object Servers.

When the replication scenario has been defined, the moderator tool starts sending commands to the chosen Globe Object Servers. It starts by sending a "create first replica" command to one (randomly chosen) GOS in the scenario. This Globe Object Server constructs a local representative for that DSO in its address space, and registers a contact address for this local representative in the Globe Location Service. As part of the registration, an object identifier is allocated for the DSO by the GLS. This object identifier is returned to the moderator tool. The other GOSs are then sent "bind to DSO $<$OID$>$, create replica" commands. The replicas they create are also registered with the GLS.

The final step in creating a package DSO is registering a name for it in the Globe Name Service. To this extent the moderator tool calls a library routine which communicates with the GNS. In particular, this library routine contacts the so-called *GNS Naming Authority* for the GDN Zone. This is the daemon that sends DNS UPDATE messages [Vixie *et al.*, 1997] to the name servers responsible for the GDN Zone, in response to add and remove requests from clients.

Looking at this procedure, we can derive three security subrequirements:

1. A Globe Object Server should accept only commands sent by a GDN moderator.

2. The Globe Location Service should accept only object registrations (and deregistrations) from Globe Object Servers which are officially part of the GDN.

3. A GDN Naming Authority should accept only updates from moderator tools operated by official GDN moderators.

Modifying Packages Without loss of generality, we can say that to ensure the integrity of the data inside a package DSO, Globe Object Servers and GDN-enabled HTTPDs (i.e., the processes potentially hosting replicas of the DSO) should not accept state-modifying method invocations and state update messages from unauthorized senders. Authorized senders are: (1) a moderator tool operated by an official GDN moderator and (2) Globe Object Servers that are part of the GDN (e.g. a Globe Object Server acting as master replica in a master/slave replication protocol). This is, of course, not a sufficient condition. We should also protect servers from direct tampering through break ins on the machines they are hosted on.

Availability

People should not be able to crash our critical servers, nor render them inoperable using bogus protocol messages. The critical servers in the GDN are:

- Location Service directory nodes and auxiliary GLS daemons

- Object Servers

- GDN-enabled HTTPDs

- DNS servers and auxiliary daemons used by the DNS-based GNS

6.2 Operating Environment

We assume the following security situation. The different parts of the GDN application run on machines distributed all over the Internet, however, the critical parts of the application, such as the Globe Object Servers, the Location Service's nodes and moderator tools run only on secure machines. By secure we mean that only authorized personnel can install software on them, log in, etc. We call these machines the *GDN hosts*. For the first versions of the GDN we assume that the networks connecting the GDN hosts cannot be tapped by attackers. These networks are not, however, firewalled, so anyone on the Internet can send network packets to these hosts.

We consider the parts of the application that are running on users' machines to be insecure. These are the GDN-enabled proxy servers and the users' browsers (see Section 4). Furthermore, we assume the connections between the GDN hosts and the users' machines are not secure. The last aspect of the security situation is that the source code of both the GDN application and Globe are publicly available, which makes staging an attack simpler.

6.3 Security Measures

As the security framework for Globe is still under development and will not be incorporated into Globe before the end of 2000, we will not be able to use it in the first versions of the GDN. Instead we will develop a more limited, GDN-specific security model for these versions.

Because the first (June 2000) version will run in a controlled environment we will not actually implement any security measures until the second version. To secure this version we replace all communication between GDN parties by integrity-protected and authenticated communication. In particular, all TCP connections between GDN parties are replaced by connections secured via the TLS protocol [Dierks and Allen, 1999] and its predecessor, the Secure Sockets Layer (SSL) [Freier *et al.*, 1996]).

TLS offers one-way or two-way authenticated communication channels which are encrypted and protected against content modification. The idea is that GDN hosts use two-way authenticated channels for internal communication, and server-side authentication for all communication with software running on users' machines (i.e., browsers or GDN-proxy servers). This situation is illus-

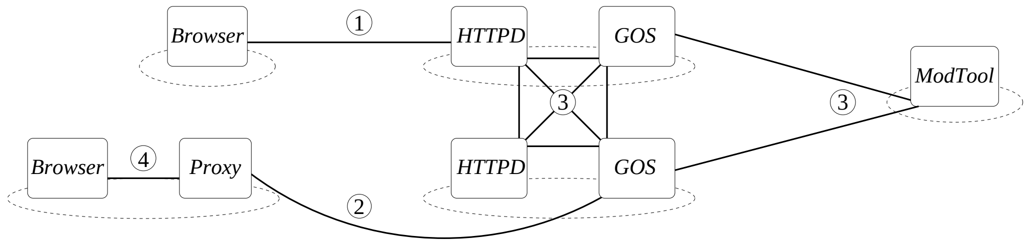


Figure 4: We secure the GDN using a transport-layer security library. Communication between GDN hosts, marked (3) in the figure, is fully authenticated. Communication between GDN hosts and browsers running on users' machines (marked (1) in the figure) is authenticated one way: the GDN host authenticates itself to the users' machines. This is also the case for communication between GDN hosts and GDN-enabled proxy servers on users' machines ((2) in the figure). If desired, a user could configure a GDN-enabled proxy server to also authenticate itself to the local browsers ((4) in the figure), but we consider this a local administrative matter.

trated in Figure 4.

By making sure that sensitive requests, such as state-modifying method invocations, are executed only when sent from authenticated hosts (i.e., GDN hosts) attackers are not able to compromise the integrity of the data contained in the GDN. This also protects software packages downloaded via browsers from malicious modifications.

For the GDN to work we will, however, still have to accept network traffic from unauthenticated hosts (in particular, from users' machines). This means that attackers are potentially able to crash the GDN, that is, compromise availability by sending malformed packets which cause the GDN to crash. We intend to counter these type of attacks by good programming, avoiding buffer flows, etc. We will not take extra measures against denial of service through flooding.

A disadvantage of this scheme is, of course, that we are paying for something we do not need: confidentiality. TLS and SSL provide confidentiality as well as authentication and integrity protection. We are interested only in the latter two. If performance is affected too negatively by the superfluous encryption and decryption we will have to rethink our security scheme.

This solution seems to be quite feasible in practical terms. To implement this security scheme we need to rewrite our communication layers to use TLS or SSL. This should not require too much effort since we have cleanly separated communication from functional layers in all our software (e.g. see Section 3.3) and TLS/SSL builds on the BSD socket interface. Availability of a TLS/SSL library for Java, the language in which all our software is written is a potential issue. Fortunately, Sun has recently published the Java Secure Sockets Extension (JSSE) [Sun Microsystems, Inc., 1999] that implements TLS and SSL.

The JSSE package can be legally exported from the US. The only potential problem is usage restrictions in the countries where the GDN hosts are located. At this point in time we have no knowledge about what machines will be available to us, therefore we cannot asses how serious a problem this is.

The one case where the TLS scheme cannot be used is the Globe Location Service. For efficiency reasons this is based on UDP. We have yet to determine if it is acceptable to temporarily replace it with TCP, or that we should implement a specific security scheme for the GLS.

Another special case is the DNS-based Globe Name Service. The problem is that this TLS/SSL scheme can be used to secure only the connection between the GDN Naming Authority, that is, the daemon which sends DNS UPDATE messages and the moderator tools that request these changes. We cannot protect the DNS itself using this method, for obvious reasons.

The effects of DNS spoofing on the GNS, and the GDN in general, are limited, however. Basically, attackers can only prevent resolution of object names to object identifiers or cause an object name to resolve to an invalid object identifier or to one belonging to another object. Denial of service attacks on other parts of the GDN can be prevented if we use IP-addresses instead of DNS names for internal GDN configuration. Our use of TLS and BIND's TSIG security feature (the GNS is build on

BIND8 [Internet Software Consortium, 2000]) will prevent abuse or modification of the GDN's contents.

7 Availability and Current Status

The source code of the GDN and Globe will be made available through the Globe WWW site located at

http://www.cs.vu.nl/globe/

The current (March 2000) status of the GDN is as follows. We are writing the control and semantics object for the package DSOs and will then start working on the moderator tool and the GDN-HTTPD. For the latter we can build heavily on an earlier prototype. The current status of the Globe middleware can be described best by looking at what steps are necessary to create a new kind of distributed shared object and to get an application using an instance of that kind of DSO up and running.

The application programmer starts by defining the interfaces of the DSO[1] in Globe's interface definition language (IDL). Using our IDL compiler these interfaces are translated into Java. Using these translated definitions the application programmer writes two subobjects: the semantics subobject that implements the actual functionality of this kind of DSO and the control subobject (see Section 3.3). Control subobjects should be generated automatically in the future.

These implementations are copied to all machines that need to run local representatives of DSOs of this kind and placed in the local implementation repository (currently a directory in the local file system). The last step in writing a Globe application is to write the clients that use the DSO. The Globe part of these clients is easy to implement. The programmer should initialize the runtime system and ask it to bind to a given object identifier, after which the client can access the DSO via its local representative.

To actually run the application the application programmer first has to start and configure the name and location services. Our current Java implementation of the Globe Location Service supports the basic look-up, insert and delete operations and, in addition, persistent storage of the state of a directory node (location information and forwarding pointers). We are in the process of adding a simple crash recovery mechanism to this implementation. The source code of the Java-based GLS cannot be released due to contractual agreements until January 2001. Releases of Globe prior to that time will contain the GLS in byte-code form.

[1]Globe uses a model in which an object can have multiple interfaces, as in Microsoft's COM.

The DNS-based prototype of the Globe Name Service is implemented on top of BIND8 [Internet Software Consortium, 2000]. It is fully functional, meaning that a user can add, resolve, change, and delete object names and directories via routines in the Globe run-time system. These routines communicate with the GNS Naming Authorities and through it, with the name servers for the domain the updates are made in.

Once the application programmer has the services up and running, he or she starts up a number of Globe Object Servers and instructs them to create an instance of the DSO with a particular replication scenario. The application is now ready to be used. There are currently two replication protocols an application programmer can choose from: client/(single) server and master/slave. The Globe Object Server and supporting tools are currently being implemented and should be part of the first public Globe release.

8 Conclusions

The goal of the Globe project is to design and build a middleware platform that facilitates the development of Internet applications. These application are characterized by the complex nonfunctional aspects their programmers have to take into account: potentially huge numbers of users, high communication delays, host and network failures. An important technique for dealing with these aspects is replication of data and functionality, making it an important topic in the Globe middleware. We think that Globe's distributed shared object concept, combined with a worldwide location service for tracking the whereabouts of these distributed objects can offer the flexibility with respect to replication that Internet applications require.

The first version of our middleware platform is nearly complete. To demonstrate the validity of our design and ideas we are building a prototype application. This application, the Globe Distribution Network, or GDN, is a distributed application for the efficient, world-wide, distribution of data. This data initially consists of publicly redistributable software packages. Although comparable in function to the World Wide Web, the GDN has one important advantage, namely the builtin support for replication that it inherits from the Globe middleware.

Current GDN functionality is simple: software packages can be added, retrieved and removed from the Network. Two possible functional additions we are considering are a more powerful mechanism for attribute-based search and version-management facilities. In the nonfunctional arena, fault tolerance is a topic that needs to be addressed. The naming of software packages is currently done by a prototype object name service that builds on the Domain Name System. Furthermore, we currently use the Transport Layer Security (TLS) protocol to satisfy the GDN's security requirements. We intend to replace these two parts with properly designed name and security services in the future.

The GDN will be usable as an Internet application in December 2000. The source code of the GDN and of the Globe middleware will be released under the BSD license during the course of 2000.

Acknowledgments

The Globe team would like to thank Stichting NLnet and Océ for their support in the development of Globe and the Globe Distribution Network.

References

[Ballintijn and van Steen, 1999a] G. Ballintijn and M. van Steen. Exploiting Location Awareness for Scalable Location-Independent Object IDs. In *Proceedings Fifth Annual ASCI Conference*, pages 321–328, Heijen, The Netherlands, 1999. Advanced School for Computing and Imaging.

[Ballintijn and van Steen, 1999b] G. Ballintijn and M. van Steen. Scalable Naming in Global Middleware. Technical Report IR-464, Department of Mathematics and Computer Science, Vrije Universiteit Amsterdam, October 1999.

[Dierks and Allen, 1999] T. Dierks and C. Allen. RFC 2246: The TLS Protocol Version 1.0, January 1999.

[Digital Island, Ltd., 2000] Digital Island, Ltd. Footprint. http://www.digisle.net/services/cd/footprint.shtml, March 2000.

[Eddon and Eddon, 1998] Guy Eddon and Henry Eddon. *Inside Distributed COM*. Microsoft Press, Redmond, WA, 1998.

[Freier *et al.*, 1996] A. Freier, P. Karlton, and P. Kocher. The SSL Protocol Version 3.0, November 1996. Netscape Communications, Inc., Mountain View, CA.

[Howard *et al.*, 1998] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J West. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1998.

[Internet Software Consortium, 2000] Internet Software Consortium. BIND. http://www.isc.org/products/BIND/, March 2000.

[Kiczales *et al.*, 1991] G. Kiczales, J. Rivières, and D. Bobrow. *The Art of the Metaobject Protocol*. MIT Press, Cambridge, MA, 1991.

[Mockapetris, 1987] P. Mockapetris. RFC 1034: Domain Names – Concepts and Facilities, November 1987.

[Object Management Group, 1999] Object Management Group. The Common Object Request Broker: Architecture and Specification. Revision 2.3.1. OMG Document formal/99-10-07, Object Management Group, Framingham, MA, October 1999.

[Pierre *et al.*, 1999] G. Pierre, I. Kuz, M. van Steen, and A.S. Tanenbaum. Differentiated Strategies for Replicating Web Documents. Technical Report IR-467, Department of Mathematics and Computer Science, Vrije Universiteit Amsterdam, November 1999.

[Satyanarayanan *et al.*, 1990] M. Satyanarayanan, J.J. Kistler, P. Kumar, M.E. Okasaki, E.H. Siegel, and D.C. Steere. Coda: A Highly Available File System for a Distributed Workstation Environment. *IEEE Transactions on Computers*, 39(4), April 1990.

[Sun Microsystems, Inc., 1999] Sun Microsystems, Inc. Java Secure Socket Extension. http://java.sun.com/products/jsse/, August 1999.

[van Steen *et al.*, 1998] M. van Steen, F.J. Hauck, and A.S. Tanenbaum. Locating Objects in Wide-Area Systems. *IEEE Communications Magazine*, pages 104–109, January 1998.

[van Steen *et al.*, 1999] M. van Steen, P. Homburg, and A.S. Tanenbaum. Globe: A Wide-Area Distributed System. *IEEE Concurrency*, pages 70–78, January 1999.

[Vixie *et al.*, 1997] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound. RFC 2136: Dynamic Updates in the Domain Name System (DNS UPDATE), April 1997.

[Vixie, 1995] Paul A. Vixie. DNS and BIND Security Issues. In *Proceedings of the Fifth USENIX UNIX Security Symposium*, Salt Lake City, Utah, June 1995.