A Scalable Middleware Solution for Advanced Wide-Area Web Services

Maarten van Steen†, Andrew S Tanenbaum†, Ihor Kuz‡and Henk J Sips‡

†Vrije Universiteit, Department of Mathematics and Computer Science

De Boelelaan 1081a, 1081 HV Amsterdam, {steen,ast}@cs.vu.nl

‡Delft University of Technology, Department of Computer Science

Zuiderplantsoen 4, 2628 BZ Delft, {ikuz,h.j.sips}@cs.tudelft.nl

Abstract.

To alleviate scalability problems in the Web, many researchers concentrate on how to incorporate advanced caching and replication techniques. Many solutions incorporate object-based techniques. In particular, Web resources are considered as distributed objects offering a well-defined interface.

We argue that most proposals ignore two important aspects. First, there is little discussion on what kind of coherence should be provided. Proposing specific caching or replication solutions makes sense only if we know what coherence model they should implement. Second, most proposals treat all Web resources alike. Such a one-size-fits-all approach will never work in a wide-area system. We propose a solution in which Web resources are encapsulated in physically distributed shared objects. Each object should encapsulate not only state and operations, but also the policy by which its state is distributed, cached, replicated, migrated, etc.

1. Introduction

As the Web continues to gain popularity, we are increasingly confronted with its limited scalability. Web servers are often unreachable due to an overload of requests for pages. Likewise, we are faced with long downloading times caused by bandwidth limitations and unreliable links. Many of these problems are caused by the growing number of users and the steadily increasing size of resources such as images, audio, and video.

Traditional scaling techniques, such as caching and replication [20], have been applied as solutions. Unfortunately, inherent to these techniques are *consistency problems*: modifications to one copy of a cached or replicated Web page makes that copy different from the other replicas. Also, most proposals assume that a single consistency model is required and appropriate for all resources. With the large variety of Web pages already existing, and the increasing alternative applications of Web technology, it is clear that such a one-size-fits-all approach will eventually fail. Instead, different consistency models based on the content and semantics of Web resources will need to coexist if we are to solve scalability problems.

Consider, for example, a seldom-accessed personal home page. Caching such a page is hardly effective and doing so simply wastes storage capacity. On the other hand, it could make sense to actively push updates of popular home pages to areas with many clients to reduce bandwidth and latency problems. Other examples easily come to mind.

Another problem faced by the Web is its limited flexibility with regards to the introduction of new resources and services. Although nonstandard resources, such as Java applets, have been integrated into the Web, the means by which this is done usually requires a unique solution for each new type of resource. Creating such solutions is not always an easy task, and they are rarely elegant.

It is clear that a different approach is needed to overcome the limited scalability of the current Web. Our starting point is that caching and replication are crucial to scalability, but that effective solutions can be constructed only if we take application-level requirements into account. In this light, we propose an object-based middleware solution called Globe. Key to our approach are physically distributed objects that encapsulate not only state and methods, but also complete distribution policies. In other words, each object in our approach carries its own solution to the distribution of its state, including how that state is partitioned, replicated, migrated, etc. Consequently, all implementation aspects are hidden from clients, who see only the interfaces offered by the object.

By offering a framework that allows us to apply scaling techniques on a per-object basis, we will be able to develop worldwide scalable components from which the next generation of networked applications can be built. To demonstrate the feasibility of our approach, we are developing a large-scale, wide-area distributed Web service. The service is transparently distributed across a (potentially large) number of servers in a global network. In this paper we describe Globe and its application to the Web service.

This paper makes two main contributions. First, we show how scalability problems in wide-area systems can be alleviated by a middleware solution in which objects are physically distributed and fully encapsulate their own distribution policy. Second, we describe an alternative organization of Web-based applications that allows us to deal with distributed Web resources in an elegant and scalable way. We also show how our service can be fully integrated into the current Web.

The paper is organized as follows. In Section 2 we describe the basic approach followed in Globe. How Globe can be used to build a wide-area distributed Web service is described in Section 3, which is partly based on our experience with a Java prototype. Related work is described in Section 4; we conclude in Section 5.

2. Scalable Distributed Objects

2.1. Distributed-Object Technology

An important goal of distributed systems is *distribution transparency*: providing a single-system view despite the distribution of data, processes, and control across multiple machines. There are different kinds of distribution transparency as shown in Table 1. Object technology came into vogue some years ago as the means for realizing transparency in distributed systems. For example, access transparency can be achieved by following an interface-based approach as is in CORBA [22] and ILU [13]. Likewise, location and migration transparency can be supported by means of forwarding pointers as in the Emerald system [14] and more recently in the Voyager toolkit [21]. Finally, seamless integration of object persistence has been investigated for distributed systems such as Spring [24].

However, when we take a closer look at the way distribution is actually supported in object-based systems, it appears that objects are used only in a restricted way to address transparency problems. For example, all well-known systems today adopt the remote-object model. In this model, an object is located at a single location only, whereas the client is offered access transparency through a proxy interface. At best, the object is allowed to move to other locations without having to explicitly inform the client.

There are a number of serious drawbacks to the remote-object model, most notably its lack of scalability. To alleviate scalability problems it is necessary to apply techniques such as caching and replication. This means that multiple copies of the object reside at different locations. Having only a remote-invocation mechanism available, we now have to solve the problem how an invocation is to be propagated between the object replicas. Unfortunately, there is no standard solution. For *active replication*, an invocation or the results could be shipped to every replica. In addition, we generally have to implement a total ordering on

Table 1. Different kinds of distribution transparency relevant for distributed systems [12]

Transparency	Description
Access transparency	Hides differences in data representation and invocation mechanisms
Failure transparency	Hides failure and possible recovery of objects
Location transparency	Hides where an object resides
Migration transparency	Hides from an object the ability of a system to change that object's location
Relocation transparency	Hides from a client the ability of a system to change the location of an object to which the client is bound
Replication transparency	Hides the fact that an object or its state may be replicated and that replicas reside at different locations
Persistence transparency	Hides the fact that an object may be (partly) passivated by the system
Transaction transparency	Hides the coordination of activities between objects to achieve consistency at a higher level

concurrent invocations [25]. In the case of *passive replication*, update invocations are to be propagated to a master copy only, whereas read invocations can often be performed at backup copies [3]. There are numerous variations on this theme.

The remote-object model itself provides no mechanisms that support a developer in designing and implementing different invocation schemes, which is necessary if we are to apply scaling techniques such as caching, replication, and distribution.

2.2. Globe: An Alternative Approach

As an alternative to the remote-object model, we have developed a model in which processes interact and communicate through **distributed shared objects**[30]. Like distributed objects in other models, an object offers one or more **interfaces**, each consisting of a set of methods. Objects are passive, but multiple processes may simultaneously access the same object. Changes to the object's state made by one process are visible to the others. However, unlike any other model, a distributed object in Globe is *physically distributed*, meaning that its state may be partitioned and replicated across multiple machines at the same time. Clients of an object are unaware of such a distribution: they see only the interface(s) made available to them by the object.

Besides being physically distributed, each object fully encapsulates its own **distribution policy**. In other words, there is no systemwide policy imposing how an object's state should be distributed and kept consistent. For example, we may have a distributed object whose state is replicated at each client, and where method invocations are forwarded to all clients. Another object may have adopted an approach in which state updates always occur at a master copy and are subsequently shipped to the replicas. Likewise, there may be objects that move their state between locations, have their state highly secured against malicious clients, or keep state at highly fault tolerant servers only. The important thing is that clients need not be aware of such details as they are hidden behind an object's interface.

In order for a process to invoke an object's method, it must first **bind** to that object by contacting it at one of the object's contact points. A **contact address** describes such a contact point, specifying a network address and a protocol through which the binding can take place. Binding results in an interface belonging to the object being placed in the client's address space, along with an implementation of that interface. Such an implementation is called a **local object**. This model is illustrated in Figure 1.

2.2.1. Architecture of a Distributed Shared Object A local object resides in a single address space and communicates with local objects in other address spaces. Each local object is

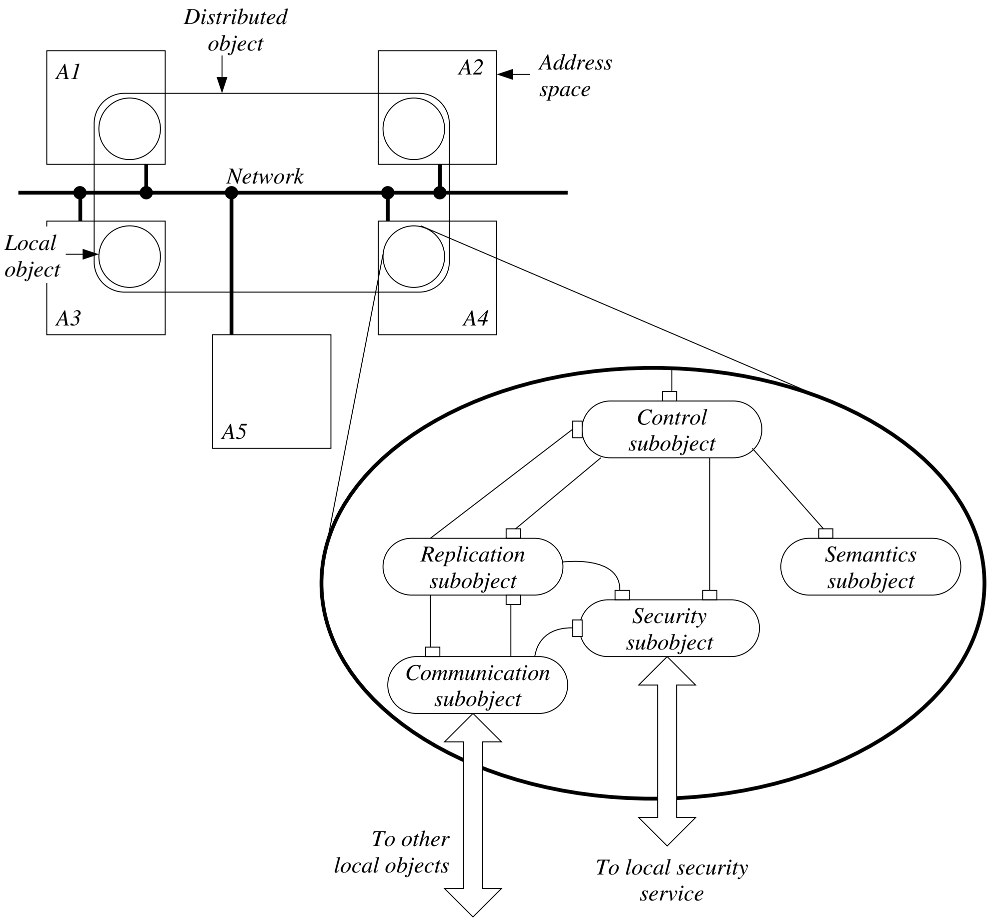

Figure 1. Example of an object distributed across four address spaces.

composed of several subobjects, and is itself again fully self-contained as also shown in Figure 1. A minimal composition consists of the following five subobjects.

Semantics subobject. This is a local subobject 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 scalability issues.

Communication subobject. This is generally a system-provided subobject. It is responsible for handling communication between parts of the distributed object that reside in different address spaces. 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 its various semantics subobjects. Semantics subobjects may be replicated for reasons of fault tolerance or performance. In particular, the replication subobject is responsible for keeping these replicas consistent according to some (per-object) coherence strategy. Different distributed objects may have different replication subobjects, using different replication algorithms.

An important observation is that the replication subobject has a standard interface. However, implementations of that interface will generally differ between replication subobjects. In a sense, this subobject behaves as a meta-level object comparable to techniques applied in reflective object-oriented programming [16].

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.

Security subobject. The security subobject represents the internal protection of the distributed object against intruders. The subobject checks whether incoming invocation requests are valid, checks whether invocations are actually allowed, and assists the control subobject in verifying local invocations. Finally, it can communicate with local security services. Like the interfaces of the communication and replication subobject, the interfaces of the security subobject are also standardized.

A key role, of course, is reserved for the replication subobject. An important observation is that communication and replication subobjects are unaware of the methods and state of the semantics subobject. Instead, both the communication subobject and the replication subobject operate only on invocation messages in which method identifiers and parameters have been encoded. This independence allows us to define standard interfaces for all replication subobjects and communication subobjects.

2.2.2. Client-to-Object Binding To communicate with a distributed object, it is necessary for a process to first **bind** to that object. Binding consists roughly of two phases: finding the object, and installing the interface. This process is illustrated in Figure 2.

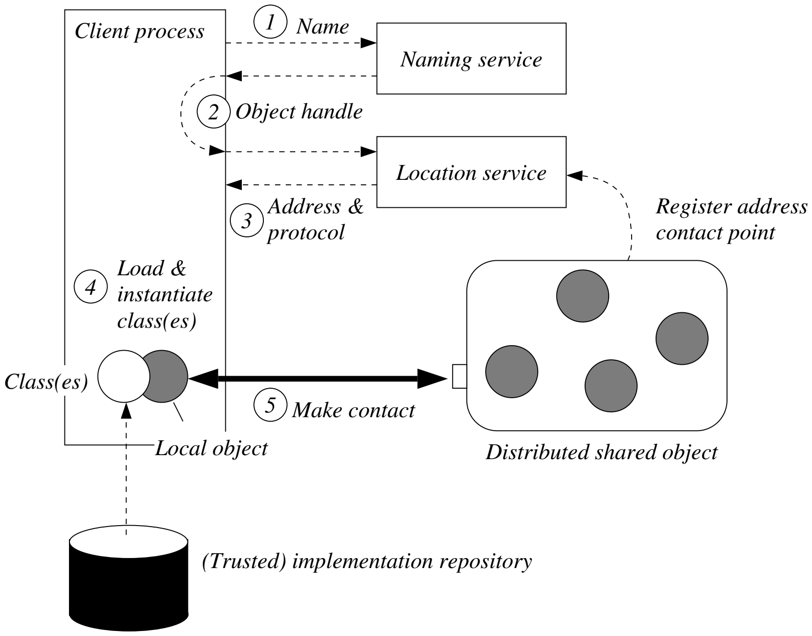

Figure 2. Binding a process to a distributed shared object.

To find an object, a process must pass a name of that object to a naming service that can resolve that name (step 1 in Figure 2). The naming service returns an **object handle** (step 2), which is a location-independent and universally unique object identifier, such as a 128-bit number, which is used to locate objects. It can be passed freely between processes as an object reference. The object handle is given to a location service, which returns one or several contact addresses (step 3).

This organization of a naming and a location service allows us to separate issues related to naming objects from those related to contacting objects. In particular, it is now easy to support multiple and independent (human-readable) names for an object, analogous to multiple links to a file name in UNIX. Because an object handle does not change once it has been assigned to an object, a user can easily bind a private, or locally shared name to an object without ever having to worry that the name–to–object binding changes without notice. On the other hand, an object can update its contact addresses at the location service without having to consider under which name it can be reached by its clients. However, we do require a scalable location service that can handle frequent updates of contact addresses in an efficient manner. We have designed such a service [29, 31] and have implemented an initial prototype version for tested on the Internet.

Once a process knows where it can contact the distributed object, it needs to select a suitable address from the ones returned by the location service. A contact address may be selected for its locality, but there may also be other criteria for preferring one address over another.

A contact address describes *where* and *how* the requested object can be reached. The latter is contained as protocol information in the contact address. The protocol information is used to load classes from a (trusted) implementation repository, and to subsequently instantiate those classes (step 4 in Figure 2). Finally, the client needs to contact the distributed shared object (step 5). The local object implements the interface(s) offered by the distributed shared object.

3. Scalable Distributed Web Services

To illustrate how our approach can be applied to solve scalability problems of the World-Wide Web, we discuss the design of a Globe-based distributed Web service.

3.1. Overview of the Globe Web Service

3.1.1. Globe Web Documents The essence of a Globe-based Web service is that it allows clients access to Globe Web documents, referred to as GlobeDocs. Conceptually, a **GlobeDoc** is a distributed shared object containing a collection of logically related Web pages. Each Globe Web document may consist of text, icons, images, sounds, animations, etc., as well as applets, scripts, and other forms of executable code. We refer to these parts as **elements**. The hyperlinked structure as normally provided by Web pages is maintained in a GlobeDoc. An **internal hyperlink** that is part of some GlobeDoc, refers to an element in that same document. An **external hyperlink** refers to an element of another GlobeDoc.

For simplicity, all elements and hyperlinks of a GlobeDoc are collected into a single archive, which is subsequently wrapped into a (nondistributed) semantics subobject. This semantics subobject offers several interfaces as shown in Table 2. In principle, these interfaces are available to each client that is bound to the GlobeDoc. Details on how these interfaces are implemented are described in Section 3.2.

3.1.2. Document Coherence What makes our approach unique compared to existing Web services, is that each GlobeDoc has its own associated distribution policy. For example, a document containing personal information as in the case of ordinary personal home pages, may support a policy by which updates are always done at a master copy and clients are offered only remote access to that copy. On the other hand, a document consisting of a shared whiteboard may adopt a policy by which each client has local access to a full replica of the

Table 2. Interfaces offered by the semantics object of GlobeDocs

Interface	Description
Document interface	Contains methods for listing, adding, and removing elements to a GlobeDoc
Content interface	Contains methods for reading and writing the content of an element
Attribute interface	Contains methods for attributes of elements, such as type, last modification date, etc.

whiteboard, and by which updates are immediately propagated to all other clients. Other distribution policies can easily be associated with a document and will generally depend on what, how, and where the document offers functionality to its clients.

For our distributed Web service, we concentrate primarily on scalability. Instead of tackling scalability problems by focusing directly on caching and replication, we advocate that it is necessary to concentrate first on coherence issues. Coherence deals with the effect of read and write operations by different clients on a possibly replicated distributed object, as viewed by clients of that object. Caching and replication are part of *coherence protocols*, which implement a specific *coherence model*. In Globe, we distinguish two types of coherence models:

Object-centric coherence models describe the coherence a distributed shared object offers to concurrently operating clients. The models are based on those developed for distributed shared memory systems, and include sequential consistency [17], PRAM consistency [18], causal consistency [1, 10], and eventual consistency.

Client-centric coherence models allow a client to express its own coherence requirements. Our approach here is similar to work done in the Bayou project [28]. Bayou provides mobile users weak consistency support in a replicated database. We have basically retained their models, which include scenarios for monotonic writes, monotonic reads, writes follow reads, and read your writes.

Details on our support for coherence models are described elsewhere [15]. Important for our present discussion, is that each GlobeDoc has an associated object-centric coherence model, which is implemented by means of the replication and communication objects described in Section 2.2.1. In addition, implementations are provided to support client-centric coherence models as well.

3.1.3. System Architecture It is necessary to offer storage facilities for the various components that comprise a document. In particular, being a distributed shared object, a GlobeDoc will generally consist of a number of replicas, each replica located at a different machine. Ignoring security issues for now, a replica is organized as a local object, consisting of a semantics subobject, a replication subobject, a communication subobject, and a control subobject, as explained in Section 2.2.1. In our model, each replica is kept at a **store**. In principle, clients may perform read and write operations at any store where the document resides, that is where a replica is located. We distinguish three different types of stores:

Permanent stores implement persistence of a GlobeDoc. This means that if there is currently no client bound to the document, the document will be kept only at its associated permanent stores. The permanent stores keep replicas consistent according to the object-centric coherence model that the document offers to its clients. A Web server is an example of a permanent store.

Object-initiated stores are installed as the result of the document's global replication policy. Replicas are kept consistent independent of clients although these stores may, for performance reasons, support a weaker coherence model than the one guaranteed by the permanent stores. A typical example of an object-initiated store is a mirrored Web site.

Client-initiated stores are comparable to caches. They are installed independent of the replication policy of the document and fall under the regime of the client processes that read and update the document. A sitewide cache at a Web proxy is an example of a client-initiated store.

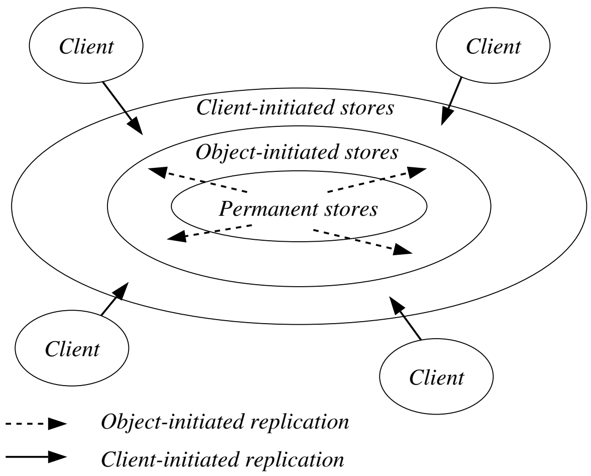

Figure 3. A system model for replicated Globe Web documents (GlobeDocs).

Stores are organized in a layered fashion as shown in Figure 3. This architecture allows us to separate replicas managed by servers (permanent and object-initiated stores) from those managed by clients (client-initiated stores). Whereas permanent stores must implement a document's coherence model, object-initiated and client-initiated stores may offer weaker coherence, but perhaps offering the benefit of higher performance. Effectively, for some applications, some delay in propagating a change is often acceptable. It is generally up to the client to decide to which replica it will bind.

3.1.4. Integration with the Current Web It is important that GlobeDocs are integrated into the current Web infrastructure such that they can be accessed and manipulated by existing tools such as browsers. Our approach is to use a filtering gateway that communicates with standard Web clients (e.g. browsers), as shown in Figure 4.

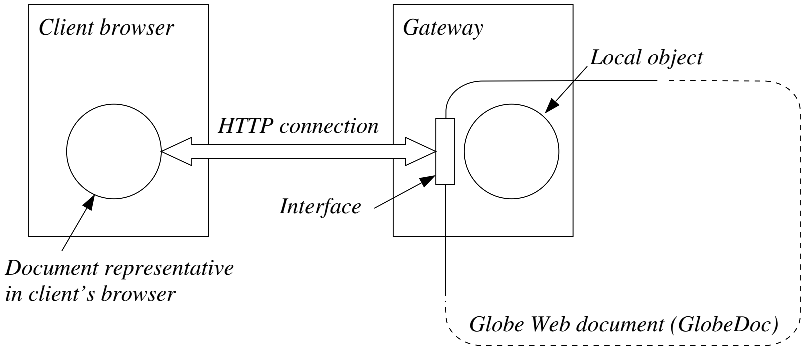

Figure 4. The general organization for integrating Globe Web services into the current Web.

The main purpose of the gateway is to allow standard Web clients that communicate through HTTP, to access GlobeDocs. The gateway is a process that runs on a local server machine and accepts regular HTTP requests for a document. In our model, GlobeDocs are distinguished from other Web resources through naming. A Globe name is written as a **Globe URN**, that is a URN (or URL) with globe as scheme identifier. So, for example, globe://cs.vu.nl/∼steen/globe/ could be the name of our project's home document, constructed as a distributed shared object.

The gateway accepts all URLs and Globe URNs. Normal URLs are simply passed to existing (proxy) servers, whereas Globe URNs are used to actually bind to the named distributed shared object. Because most browsers cannot handle extensions to the URL name space, we are forced to build a front end that translates Globe URNs to a form that is embedded in an HTTP URL. For example, globe://cs.vu.nl/∼steen/globe/ is embedded into the HTTP URL http://globe.cs.vu.nl/∼steen/globe/. When a Globe URN is passed to the gateway, the gateway binds to the GlobeDoc named by that URN, and passes the document's state in HTML form to the browser. In this way clients are unaware of the fact that they have actually accessed a distributed shared object.

The drawback of this approach is that we are constrained to the functionality of Web clients. In particular, this means that it may be hard to support GlobeDocs containing interactive parts. Ideally, we can make use of extensible browsers that can dynamically download the necessary support code for actually binding to distributed shared objects and subsequently presenting the object's interfaces to the user. As an alternative, we may assume that Web clients support Java. In that case, a GlobeDoc having interactive content provides a Java applet that is downloaded into the client's browser, and which subsequently presents the object's interfaces in any way that is felt appropriate by the developer of the document. Effectively, we are extending the distributed shared object to the Web client by means of a simple Java applet instead of using a Globe local object. This situation is shown in Figure 5, and is the approach followed in our prototype.

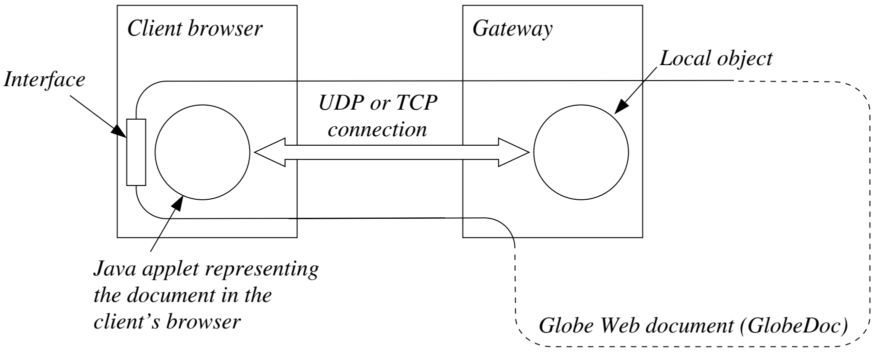

Figure 5. Using Java-enabled browsers to interface to interactive GlobeDocs.

3.2. Constructing a GlobeDoc

There are many ways to actually construct a GlobeDoc and make it available as a distributed shared object. In the following, we outline one such solution.

3.2.1. Constructing the First Replica Completely analogous to the construction of Web pages, a GlobeDoc is constructed by first providing all the necessary content. This includes HTML files containing hyperlinks, files containing executable code, files for images, audio, etc. All these content files are then collected into a **state archive**. Effectively, a state archive is a structured representation of the information offered by a document. In our initial set-up, a state archive is transferred as a whole to clients, although it will also be possible to transfer only those parts that a client needs.

The state archive forms the actual content, that is, state of a semantics object. Besides providing the state archive, a developer will also construct definitions of the interfaces containing the methods that give access to a document's content. In the case that the GlobeDoc consists of only noninteractive data, such as HTML text, animations, etc., all interfaces and their implementations are generated automatically from the archive. For interactive parts, such as editors, spreadsheets, whiteboards, and calculators, a developer explicitly specifies interfaces in the Globe Interface Definition Language (Globe IDL). Our IDL resembles those of CORBA and ILU, but has been tailored to describe local as well as remote interfaces.

The implementation of IDL interfaces is described by means of the Globe Object Definition Language (Globe ODL). We support implementations written in C and Java. Note that a developer may provide several implementations of the same interface. For example, clients of a document containing a calculator, may be offered a choice between an interpreted and a compiled version.

A state archive combined with the appropriate interfaces and their implementations, is in fact a semantics object. We separate the interfaces and implementations from the actual state, by collecting the former in a **class archive**. A class archive not only contains implementations, but also identifies how those implementations are to be (down)loaded by a client. For example, it may identify a specific class loader that first needs to be installed in the client's address space.

Taking the interface definitions of the semantics subobject, we then generate one or more implementations for the control subobject, and add those to the class archive.

The next important step is to select an object-centric coherence model for the GlobeDoc, and add implementations for the replication and communication subobject of that model to the class archive. In addition, implementations of the client-centric coherence models that will be supported also are added to the class archive. We envisage that a developer will generally choose default implementations provided as part of the development kit for documents, and possibly fine-tune those to specific requirements. However, there is nothing that prevents a developer from providing his own implementation of a coherence model.

As we have described so far, a Web document consists of a separate state and class archive. Of course, it is also possible to construct more than one state or class archive, or alternatively to combine them into a single archive. For our present discussion we ignore such alternatives.

3.2.2. Making a GlobeDoc Worldwide Available Having state and class archives allows us to actually construct a distributed shared object to which clients can bind. First, we make the class archive available by storing it in one or more **implementation repositories**. Such a repository can be as simple as an ftp-able file system, or as sophisticated as a worldwide distributed database. We assume that when a class archive is stored, the repository returns an **implementation handle** that can be uniquely resolved to the archive. We return to this aspect below.

The state and class archives are initially combined at one permanent store, where the first replica is subsequently instantiated. The store returns a network address that can be used to contact the replica. If the store is willing to make the class archive available as well, that is it willing to act also as an implementation repository, it will additionally return an implementation handle. At this point, we have actually created a distributed shared object. More replicas can be registered at other permanent stores, provided those stores cooperate in keeping the replicas consistent. In principle, this requires the stores to run the implementation of the coherence model as contained in the class archive forming part of the replica.

The distributed shared object is registered at the Globe location service, which subsequently returns an object handle. A network address that has been returned by a permanent store, is taken together with one or more implementation handles as returned by the repositories, to form a contact address. Note that the implementation handles implicitly describe the protocol by which the object can be contacted. These contact addresses are subsequently inserted into the location service so that they can be looked up by clients. The final step consists of registering the object handle at one or more (worldwide) naming services.

3.3. Client-to-Document Binding

Binding a client to a GlobeDoc is now fairly straightforward. We first describe the simple binding process in which a client contacts a document at one of its permanent stores. We then proceed by explaining how client-initiated stores, such as caches, can be used.

3.3.1. Simple Binding through Permanent Stores A contact address generally consists of a network address and protocol information that allows a client to contact an object. In the case of GlobeDocs, the protocol information consists of one or more implementation handles. After looking up a contact address for a document through the naming and location service, a client passes the implementation handles contained in that contact address to a local **implementation service**. This service is responsible for selecting and downloading an appropriate implementation. An implementation may not be appropriate for several reasons. For example, the client or the local implementation service may require that an implementation has been certified by a specific authority. Another possible reason is that an implementation does not match the architecture of the client machine, or that specific libraries are not available.

An implementation handle implicitly refers to the repository where the class archive is stored. In the case of simple repositories, such as an ftp-able file system, the implementation handle may consist of an IP address and a pathname identifying the class archive. More sophisticated solutions exist as well. For example, an object-oriented database may offer a front end to its clients in the form of a distributed shared object. In that case, an implementation handle may contain an object handle that is to be resolved to a contact address for that front end. The local implementation service must then first bind to the front end following the complete binding procedure as described in Section 2.2.2.

After an implementation has been selected and the client has loaded the class archive into its address space, the implementations (i.e., classes and objects) are instantiated, followed by a preliminary initialization by means of the network address that was part of the contact address. The client has now set up a connection to the replica through the permanent store. The store, in turn, activates the replica, after which the necessary state as contained in the state archive is shipped to the client. At that point, the client has the interfaces of the GlobeDoc at its disposal and can invoke the document's methods.

3.3.2. Advanced Binding: Selecting a Store A client should also be allowed to cache GlobeDocs independently of the object-centric coherence model offered by that document. In case caching is to be done at the client only, we can basically follow the approach for binding through a permanent store. The client need only provide an implementation for locally storing its copy of the document's semantics object.

Making use of a proxy cache, as is common for many client Web sites, is somewhat more intricate. We have adopted the following model. A process, called a **cache manager** that is prepared to offer caching facilities registers itself as a **cache manager object** at the Globe location service. A cache manager object is just a distributed shared object whose contact address is made only locally available by the location service. A client process wishing to bind to a GlobeDoc using local caching facilities, simply passes the document's object handle to the location service, indicating that it is also prepared to accept contact addresses of local, sitewide cache manager objects.

When a contact address is returned, the client binds to the object associated with the contact address, as usual. The contact address indicates whether the client is binding to a cache manager object, or to the GlobeDoc. In the former case, the client passes the document's object handle to the cache manager object. The cache manager, in turn, will bind to the GlobeDoc at one of the document's contact addresses.

When the cache manager is bound to the GlobeDoc, it inserts one or more local contact addresses for the document at the location service. The client that originally initiated the binding process is now instructed to bind to the document at an address offered by the cache manager, and to unbind from the cache manager object.

Note that after the cache manager is bound to the GlobeDoc, subsequent clients can bind directly to the document through its local contact address(es) as inserted into the location service by the cache manager. There is no need to bind to the cache manager object as before.

4. Related Work

To alleviate scalability problems in the Web, research has mainly concentrated on traditional caching techniques. Replication has been applied in the form of mirroring popular Web sites. Recently, it has been recognized that more advanced forms of caching and replication are needed. Wessels [32] proposes to allow servers to grant or deny a client permission to cache a resource. Push-caching [9] allows popular resources to be optimally distributed to other servers based on knowledge of the resource's access patterns. In a similar fashion, Baentsch et al. [2] propose a replication scheme in which replicas are pushed to a collection of replication servers, and in which clients locate the nearest server for downloading a Web page. Harvest caches [6] provide a hierarchically organized solution, and are currently gaining popularity in the Web. An interesting approach is to keep client caches up to date by have servers invalidate entries on updates [4]. This approach is also followed in AFS, of which the designers claim it can be used as the basis for building strongly-consistent Web applications [26].

Research has also concentrated on replication schemes for specific classes of Web resources. For example, the distribution point model [7] is tailored to active replication of relatively static sets of bulk, non real-time data. It is mainly applicable to magazine-like Web documents such as those that appear as electronic periodical publications.

Hardly any proposals exist that allow each resource to have its own replication scheme. In the Bayou system a mobile client can specify coherence requirements for data that is replicated and distributed across multiple servers [28, 23]. We have adopted some of the results of the Bayou project in our own work. In the W3Objects system, Web resources are encapsulated into distributed objects that can have their own replication scheme [11]. Their model is strongly based on the notion of remote objects, which we argue is less flexible than a model in which objects can be truly physically distributed. Also, where we strive for distribution transparency, the developers of the W3Objects system aim at a highly visible caching mechanism [5].

In general, much work is currently being done to incorporate CORBA and similar distributed object technologies into the Web. It is especially the combination of Java and CORBA that is receiving much attention [8]. These approaches hardly tackle the problem of scalability, and do not provide solutions for caching, replication and consistency. In this respect, a perhaps more interesting development is the proposed HTTP-ng protocol [27] the goal of which is to present a new object-based protocol for the Web. In principle, HTTP-ng will allow clients and servers to specify options for caching individual Web pages.

A solution that comes close to ours is the work based on fragmented objects [19]. Fragmented objects, like Globe's distributed shared objects, are physically distributed across multiple machines, encapsulating their own distribution policy. However, fragmented objects have not been designed for worldwide scalability and do not address caching and replication as we do.

5. Future Research

We have presented Globe's distributed shared objects, in the form of GlobeDocs, as a solution to a number of the Web's scalability problems. A GlobeDoc is a physically distributed object encapsulating one or more Web resources. Each document takes care of its own distribution issues such as caching, replication, consistency, and communication. In addition, our approach provides a flexible and extensible approach for implementing future Web resources.

To assess our research, we have developed a simple prototype implementation of a Globe distributed Web service in Java. The main purpose of this prototype was to obtain feedback on the feasibility of our approach, and also to gain insight in possible implementations. Currently, we are developing a toolkit in Java that will allow us to more easily construct the GlobeDocs as described in this paper.

There are still a number of open issues that we need to address. We are investigating how we can incorporate security into our framework such that security policies can be attached to individual GlobeDocs in a similar fashion as distribution policies. Also, more research is needed with respect to different caching and replication policies, and how policies can be implemented efficiently in a worldwide system. With respect to Globe-based distributed Web services, we also need support for partitioning and distributing state archives, as well as user-oriented tools that replace much of the manual construction of GlobeDocs.

References

[1] M. Ahamad, R. Bazzi, R. John, P. Kohli, and G. Neiger. "The Power of Processor Consistency." Technical Report GIT-CC-92/34, College of Computing, Georgia Institute of Technology, Dec. 1992.

[2] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm. "Enhancing the Web's Infrastructure: From Caching to Replication." *IEEE Internet Computing*, 1(2):18–27, Mar. 1997.

[3] N. Budhijara, K. Marzullo, F. Schneider, and S. Toueg. "The Primary-Backup Approach." In S. Mullender, (ed.), *Distributed Systems*, pp. 199–216. Addison-Wesley, Wokingham, 2nd edition, 1993.

[4] P. Cao and C. Liu. "Maintaining Strong Cache Consistency in the World Wide Web." *IEEE Transactions on Computers*, 47(4):445–457, Apr. 1998.

[5] S. Caughey, D. Ingham, and M. Little. "Flexible Open Caching for the Web." *Computer Networks and ISDN Systems*, 29(8-13):1007–1017, 1997.

[6] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell. "A Hierarchical Internet Object Cache." Technical Report CU-CS-766-95, Department of Computer Science, University of Colorado – Boulder, Mar. 1995.

[7] J. Donnelley. "WWW Media Distribution via Hopwise Reliable Multicast." *Computer Networks and ISDN Systems*, 27(6):781–788, 1995.

[8] E. Evans and D. Rogers. "Using Java Applets and CORBA for Multi-User Distributed Applications." *IEEE Internet Computing*, 1(3):43–55, May 1997.

[9] J. Gwertzman and M. Seltzer. "The Case for Geographical Push-Caching." In *Proceedings 5th Hot Topics in Operating Systems*, Orcas Island, WA, May 1996. IEEE.

[10] P. Hutto and M. Ahamad. "Slow Memory: Weakening Consistency to Enhance Concurrency in Distributed Shared Memories." In *Proceedings 10th International Conference on Distributed Computing Systems*, pp. 302–311. IEEE, 1990.

[11] D. Ingham, M. Little, S. Caughey, and S. Shrivastava. "W3Objects: Bringing Object-Oriented Technology To The Web." *The Web Journal*, 1(1):89–105, 1995.

[12] ISO. "Open Distributed Processing Reference Model - Part 3: Architecture." International Standard ISO/IEC IS 10746-3, 1995.

[13] B. Janssen and M. Spreitzer. *ILU Reference Manual*. Xerox Corporation, May 1996.

[14] E. Jul, H. Levy, N. Hutchinson, and A. Black. "Fine-Grained Mobility in the Emerald System." *ACM Transactions on Computer Systems*, 6(1):109–133, Feb. 1988.

[15] A. Kermarrec, I. Kuz, M. van Steen, and A. Tanenbaum. "A Framework for Consistent, Replicated Web Objects." In *Proceedings 18th International Conference on Distributed Computing Systems*, pp. 276–284, Amsterdam, The Netherlands, May 1998. IEEE.

[16] G. Kiczales. "Towards a New Model of Abstraction in the Engineering of Software." In *Proceedings International Workshop on New Models for Software Architecture (IMSA): Reflection and Meta-Level Architecture*, Tokyo, Nov. 1992.

[17] L. Lamport. "How to Make a Multiprocessor Computer that Correctly Executes Multiprocessor Programs." *IEEE Transactions on Computers*, C-29(9), Sept. 1979.

[18] R. Lipton and J. Sandberg. "PRAM : A Scalable Shared Memory." Technical Report CS-TR-180-88, Princeton University, Sept. 1988.

[19] M. Makpangou, Y. Gourhant, J.-P. Le Narzul, and M. Shapiro. "Fragmented Objects for Distributed Abstractions." In T. Casavant and M. Singhal, (eds.), *Readings in Distributed Computing Systems*, pp. 170–186. IEEE Computer Society Press, Los Alamitos, CA., 1994.

[20] B. Neuman. "Scale in Distributed Systems." In T. Casavant and M. Singhal, (eds.), *Readings in Distributed Computing Systems*, pp. 463–489. IEEE Computer Society Press, Los Alamitos, CA., 1994.

[21] ObjectSpace Inc. *Voyager 2.0 User Guide*, 1998.

[22] OMG. "The Common Object Request Broker: Architecture and Specification, revision 2.2." OMG Document Technical Report 98-07-01, Object Management Group, Feb. 1998.

[23] K. Petersen, M. Spreitzer, D. Terry, and M. Theimer. "Bayou: Replicated Database Services for World-wide Applications." In *Proceedings 7th SIGOPS European Workshop*, pp. 275–280, Connemara, Ireland, Sept. 1996. ACM.

[24] S. Radia, P. Madnay, and M. Powell. "Persistence in the Spring System." In *Proceedings 3rd International Workshop on Object Orientation in Operating Systems*, Asheville, North Carolina, Dec. 1993. IEEE.

[25] F. Schneider. "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial." *ACM Computing Surveys*, 22(4):299–320, Dec. 1990.

[26] M. Spasojevic, M. Bowman, and A. Spector. "Using a Wide-Area File System Within the World-Wide Web." *Computer Networks and ISDN Systems*, 26, 1994.

[27] S. Spero. "HTTP-NG Architectural Overview." http://www.w3.org/Protocols/HTTP-NG/http-ng-

arch.html.

[28] D. B. Terry, A. J. Demers, K. Petersen, M. J. Spreitzer, M. M. Theimer, and B. B. Welsh. "Session Guarantees for Weakly Consistent Replicated Data." In *Proceedings 3rd International Conference on Parallel and Distributed Information Systems*, pp. 140–149, Austin, TX, Sept. 1994. IEEE.

[29] M. van Steen, F. Hauck, P. Homburg, and A. Tanenbaum. "Locating Objects in Wide-Area Systems." *IEEE Communications Magazine*, 36(1):104–109, Jan. 1998.

[30] M. van Steen, P. Homburg, and A. Tanenbaum. "The Architectural Design of Globe: A Wide-Area Distributed System." *IEEE Concurrency*, 7(1), Jan. 1999. Scheduled for publication.

[31] M. van Steen, F. J. Hauck, G. Ballintijn, and A. S. Tanenbaum. "Algorithmic Design of the Globe Wide-Area Location Service." *The Computer Journal*, 41(5):297–310, 1998.

[32] D. Wessels. "Intelligent Caching for World-Wide Web Objects." In *Proceedings INET '95*, Honolulu, Hawaii, June 1995. Internet Society.

Biography

Maarten van Steen is assistant professor at the Vrije Universiteit in Amsterdam since 1994. He received an M.Sc. in Applied Mathematics from Twente University (1983) and a Ph.D. in Computer Science from Leiden University (1988). He has worked at an industrial research laboratory for several years in the field of parallel programming environments. His research interests include distributed-software engineering, operating systems, computer networks, and distributed systems. Van Steen is a member of IEEE Computer Society and ACM.

Andrew S. Tanenbaum has an S.B. from M.I.T. and a Ph.D. from the University of California at Berkeley. He is currently a Professor of Computer Science at the Vrije Universiteit in Amsterdam and Dean of the interuniversity computer science graduate school, ASCI. Prof. Tanenbaum is the principal designer of three operating systems: TSS-11, Amoeba, and MINIX. He was also the chief designer of the Amsterdam Compiler Kit. In addition, Tanenbaum is the author of five books and over 80 refereed papers. He is a Fellow of ACM, a

Fellow of IEEE, and a member of the Royal Dutch Academy of Sciences. In 1994 he was the recipient of the ACM Karl V. Karlstrom Outstanding Educator Award and in 1997 he won the SIGCSE award for contributions to computer science.

Ihor Kuz has an M.Sc. in Computer Science (1996) from the Vrije Universiteit in Amsterdam. He is currently a Ph.D. student at the Delft University of Technology, doing research in the field of worldwide scalable distributed Web services. His research interests include operating systems, scalable distributed systems, and Web-based technologies.

Henk J. Sips received his M.Sc. degree in 1976 in electrical engineering and his Ph.D. in 1984 from Delft University of Technology. Currently, he is Professor in Parallel and Distributed Systems at Delft University. His research interest include computer architecture, parallel programming, parallel algorithms, and distributed systems. He is member of IEEE and ACM.