Replicated Web Objects: Design and Implementation

Ihor Kuz†, Anne-Marie Kermarrec‡, Maarten van Steen‡‡, Henk J. Sips†

† Delft University of Technology, Delft, the Netherlands
‡ IRISA, Rennes Cedex, France
‡‡ Vrije Universiteit, Amsterdam, the Netherlands
i.kuz@cs.tudelft.nl, Anne-Marie.Kermarrec@irisa.fr, steen@cs.vu.nl, h.j.sips@cs.tudelft.nl

Keywords: distributed systems, object replication, object coherence, World-Wide Web, worldwide scalable systems

Abstract

The Web currently suffers from a number of scalability problems. While general caching techniques can be used to help some, the vast number of different resources now available requires an alternative approach. In particular, we need to develop resource-specific strategies. Such an approach is not supported in the current Web infrastructure. Also, the current Web is limited in flexibility with regards to integration of new resources and services. This is partly caused by the rigid nature of the HTTP protocol.

We introduce Web objects as a way to address these problems. Web objects are worldwide scalable distributed shared objects that represent Web resources. Each such object provides at least a common interface and encapsulates and implements its own distribution policy. In this paper we concentrate on the design of these Web objects as well as their implementation in Java.

1 Introduction

As the Web gains popularity, scalability problems are becoming more and more apparent, the largest of these being the problem of performance. 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 combined with a growing number of users and steadily increasing size of resources such as images, audio, and video.

Traditional scaling techniques, such as caching and replication [13], have been applied as solutions to these problems. Unfortunately, inherent to these techniques are *consistency problems*; modifications to one copy of a cached or replicated Web page makes that copy different from all the others. Currently, a simple consistency protocol is used for caches and there is no standardized protocol for replicas. Neither of these are optimal situations so new schemes are being developed and researched.

The drawback to most proposed solutions is that they generally assume that a single consistency model is required and that this model is 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 policies that take Web usage into account will need to coexist in order for caching and replication to be effective. As an example, caching or replicating a seldom-accessed home page will hardly be effective, whereas for a popular page it might be. As far as different consistency models are concerned, consider a popular Web page that is maintained concurrently by several people at different sites. Such a page would best be physically distr ibuted across those sites. At the same time, its readers may want updates to be immediately pushed to their respective local copies. As a last example, an online magazine, would most likely benefit from a master/slave strategy by which a master copy is pushed to mirror sites, but where clients pull in copies from those mirrors on demand.

Another problem faced by the Web is its limited flexibility with regards to the introduction of new resources and services. Although nonstandard resources 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 it is rarely elegant.

In order to address these problems we introduce *Web objects*: an object-based infrastructure for the Web. Web objects are distributed objects that represent Web resources. A Web object encapsulates not only the functionality of the Web resource but also the implementations of policies for replicating and distributing the resource. By encapsulating all Web resources in Web objects we can provide a standard interface to these resources, allowing new resources to easily be integrated into the Web. Because each Web object implements its own distribution policy, the caching and replication policies can be devised on a per-object basis. This approach opens the way towards worldwide scalability as we can now take distribution requirements and possibilities of individual resources into account.

In this paper we concentrate on the design of Web objects, as well as its prototype implementation in Java. Our main contribution is that we show how existing Web technology can be deployed to construct better solutions to scalability problems. In addition, we introduce a novel distributed-object model that can be applied to more than just the World-Wide Web.

The paper is organized as follows. In Section 2 we present our notion of Web objects, concentrating on functional issues rather than distribution aspects. The latter are discussed in Section 3. In Section 4 we present a Java implementation of Web objects and its interface to existing Web browsers. Related work is discussed in Section 5.

2 Web Objects

As mentioned above, scalability of the World-Wide Web can be solved only by taking the distribution requirements of individual pages into account. Instead of considering only Web pages, our starting point is formed by what we call **Web objects**. In our model, a Web object represents any kind of resource that can be accessed through the World-Wide Web. Examples of such resources include HTML files, files containing images, CGI scripts, applets, etc. At present, these resources can usually be natively handled by browsers and servers. However, we expect new resources to evolve as well. For example, applications are emerging that allow users to make reservations for theaters, hotels, etc. Also, collaborative applications such as shared whiteboards are becoming increasingly available through the Web. Most browsers and servers do not have the capabilities to directly deal with these new resources.

Web objects can encapsulate any kind of resource. However, it is necessary that applications can access every kind of Web object in a standard way. The approach we follow is to define **interfaces** for Web objects. An interface consists of a set of methods offered by a Web object. In this section, we concentrate on what interfaces Web objects offer to their clients, ignoring distribution aspects and concentrating only on functionality.

2.1 Common Web Object Interface

The success of the Web is partly due to the fact that many different kinds of resources can be transferred and presented to clients in a uniform way. Web objects are no different in this respect. In particular, there is a **common (Web object) interface** that is offered by each Web object, as shown in Figure 1.

Method	Meaning
Put	Change the content of the Web object by writing data in the Web object's native format to it
Get	Get the content of the Web object in its native format
Size	Return the size (in number of bytes) of the Web object's content
Get_HTML	Get the content of the Web object in HTML format that can be interpreted by a browser
HTML_Size	Return the size (in number of bytes) of the Web object's content when it is viewed as an HTML file
Clear	Clear the content of the Web object

Figure 1: The common interface that is implemented by each Web object.

The common Web object interface includes methods for reading, writing, and clearing the content of the Web object. Each type of Web resource has a specific format for representing its content. For example, an image will usually be represented as binary data (a pixmap, GIF file, etc.), whereas a text file or a Postscript file will be represented as plain ASCII text. We call this format a resource's native format. The Get and Put methods allow a Web object's content to be retrieved or set in this format. We also offer a method (Get_HTML) to acquire an HTML representation of the Web object. For an image or Postscript Web object this might be a simple description of the actual contents, whereas for plain text this might be the text enhanced with some HTML tags. The Web object itself is ultimately responsible for deciding how to represent its content in HTML.

2.2 Common Management Interface

Each Web object has at least three attributes associated with it: one that tells when the Web object's state was last modified, a resource type, and a name. The **common (Web object) management interface**, shown in Figure 2, contains the methods that are used to read and write these attributes.

The management interface can be extended for different classes of Web objects. For example, Web objects providing persistent state offer additional methods for storing and retrieving state.

Method	Meaning
LastMod	Returns the time the Web object was last modified
SetType	Sets the type of resource this Web object represents
GetType	Returns the type of resource the Web object represents
SetName	Associates a name (string) with the Web object
GetName	Returns the Web object's associated name

Figure 2: Methods in the common management interface of each Web object.

2.3 Object-specific Interfaces

Besides the common interfaces, Web objects can have one or more specific interfaces depending on the type of resource they represent. To give an impression of what an object-specific interface would look like, consider a Web object representing a general Web page. Such a **Web page object** includes the HTML code, any inline images, sounds, films, applets, etc.

When using a Web page object a client may want to access the Web page object's entire state (i.e., all the resources contained in it), or it might want to access only certain parts of it. For example, a Web browser needs to access each of the resources individually, whereas a client copying the Web page object to another location would prefer to access the whole state at once.

As such, the interface provided by a Web page object has methods which allow each of the resources to be accessed separately. There are, for example, methods which allow clients to access the HTML text of the page, to add images, to get a list of all images, etc. Reading or modifying the entire state of the Web page object can be done through its common Web object interface, which we showed in Figure 1

A Web page object provides persistent state, leading to an extension of the management interface with a Store and ReadFrom method. In our prototype implementation, the entire state of a Web page object is stored in an archive file local to the client.

3 Web Objects and Distribution

So far, we have concentrated only on the functionality of Web objects and have ignored distribution issues. Ideally, a client is not aware of the distributed nature of a Web object. For example, it should be transparent where the Web object resides, whether it is replicated, or that it can migrate between locations.

3.1 Remote Method Invocation

An approach currently followed by most object-based distributed systems is to allow clients transparent access to an object through *remote method invocations*. In such cases, a proxy is installed at the client that offers the same interface as the actual object. When a client invokes a method, the proxy sends a message to the object's current location containing the identification of the method and its marshalled parameters. A server process is responsible for unpacking the message, invoking the method, and sending a reply message containing the method's return values back to the proxy. The proxy subsequently unpacks the reply message and passes the results of the method invocation to the client.

Remote method invocation basically deploys the same technique as RPCs [2], and has recently been introduced in Java [20]. Combined with object serialization, it forms a powerful technique for transparently invoking remote objects, even in the face of object migration and persistence, as demonstrated by the Voyager toolkit [14].

However, remote method invocations alone are not enough to handle complex distribution issues such as caching and replication. Moreover, the requirements for distribution are highly dependent on the nature of the object as we discussed in the introduction. In other words, distribution policies for Web objects are object specific. However, we should take care not to mix the implementation of the functionality of a Web object, with the implementation of its distribution policy. In particular, it should be possible to change a Web object's distribution policy without affecting what the Web object does.

3.2 Separating Concerns

We adopt an approach by which a Web object is partitioned into several subobjects, as shown in Figure 3.

In this approach, the semantics of the Web object is represented by a **semantics object**. A semantics object is a user-defined object that contains an implementation of the functionality of the Web object. In practice, the semantics object is constructed from primitive objects and facilities offered by a programming language such as Java or C++. For example, the semantics object of a Web page object may consist of several Java objects, each one implementing part of the actual Web page. As such, we could have a Java object encapsulating the page's text body as HTML code, another Java object implementing an applet, and yet another one offering facilities for displaying an image. The important issue is that the semantics object is constructed without considering distribution aspects. The latter are handled by the replication and communication object.

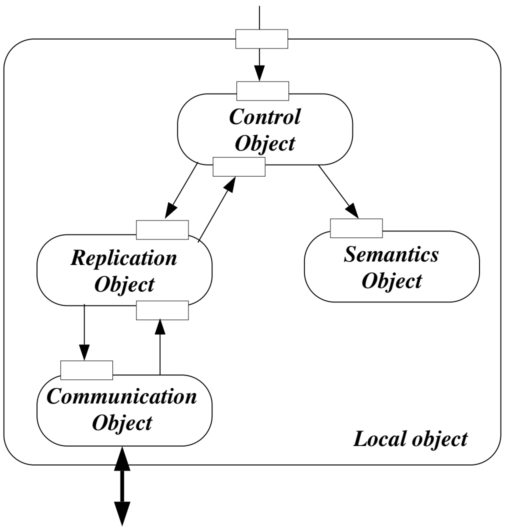

Figure 3: The organization of a (local) Web object.

The function of the **replication object** is to handle replication and caching of the state of the Web object, as represented by the semantics object. In our model, the semantics object may be copied to multiple machines. It is the responsibility of the replication object to keep these copies consistent. Exactly which consistency protocol is used depends on the Web object. We do not enforce a general distribution policy for all Web objects. Although distribution policies may differ between Web objects, replication objects all have the same interface, which is shown in Figure 4.

Method	Meaning
Start	Called to start an invocation on the semantics object
Send	Send marshalled parameters to the other replicas
Finish	Called when the invocation has finished

Figure 4: The basic methods provided by the replication object to support active replication.

Basically, each replication object need merely provide three methods. When a client wants to invoke a method at the semantics object, it first requires permission from the replication object. The Start method is used for this purpose. Invocation of Start allows the replication object to globally order the invocation request, and if necessary, acquire locks and such. After Start has returned, the client is, in principle, allowed to do the local invocation at the semantics object. The invocation requested by the client is forwarded to the other replicas by invoking Send. The parameters of the semantics object's method are marshalled, and together with an identifier of that method are passed to Send. This is similar to dynamic invocation mechanisms as supported by, for example, CORBA [15]. Once all is done, the caller invokes Finish allowing the replication object to synchronize the replicas, and possibly releasing locks as well. Further details can be found in [9]

This approach strongly resembles meta-level protocols as used in reflective object-oriented programming [11]. The replication object provides a standard interface to handle active replication, but policy details are entirely hidden behind that interface.

The communication object is used by the replication object for all network communication needs. It offers a standard interface and implements point-to-point as well as group communication.

The control object manages invocations, calling the semantic and replication objects at the right times. This control object isolates the semantics object from the replication object. It is responsible for marshalling and unmarshalling invocation requests, and invoking methods at the semantics object when instructed to do so by the replication object.

3.3 Towards Truly Distributed Web Objects

Having just split a Web object into separate subobjects, it is important that clients still view this collection of subobjects as one single entity. This is achieved by ensuring that clients can access a Web object only through its control object. In this way clients have only one access point to the Web object and therefore see it as a single object.

This approach is sufficient for nondistributed objects. However, we also want this single-object view to persist even when parts of the Web object are replicated or distributed. For example, the fact that the state of the Web object is replicated by having copies of the semantics object reside at multiple locations, should be hidden from clients. How those copies are kept consistent, for example when an invocation request is actually executed, should also be hidden. In other words, the Web object must offer the client **distribution transparency**.

Achieving such transparency is relatively simple: the Web object is no longer considered to consist of only the local copy at a specific client, but is taken to be the collection of all those local copies. In other words, the complete management of replicas is now also part of a Web object. When a client binds to a Web object, the Web object can decide to install a replica of the semantics object at the client. Alternatively, the client may get only a proxy implementation of the Web object's interfaces. Such differences are hidden from the client: they are part of the overall implementation of the Web object.

This approach is, in fact, the one adopted in the Globe architecture [8, 18]. Central to Globe are **dis-**

tributed shared objects which fully encapsulate their own distribution policy, and which can be physically distributed across multiple machines. Web objects as we have described here, are a special instance of Globe's distributed shared objects.

4 Implementing Web Objects

This section presents the Java 1.1 implementation of Web objects as described in the previous sections. The reason for choosing Java as the programming language was the possibility of integrating Web objects with current Web browsers that offer Java support. Another reason was the portability of the resulting binaries, which will allow us to experiment with dynamically loading Web objects and subobjects.

Web objects are based on the Globe architecture, thus their implementation relies heavily on that of Globe. For this reason, we first describe the Globe Java run-time and development environment before paying specific attention to the implementation of Web objects.

4.1 Basic Globe Implementation

Globe, like CORBA and many other distributed object architectures, is based on the use of interfaces. One of its main development components is an Interface Definition Language (IDL). The Globe IDL is used by developers to define the interfaces that will be exported and used by Globe objects. We have defined a mapping from the Globe IDL to Java.[1]

A developer starts with defining, in Globe's IDL, the interfaces for a class of Globe's distributed shared objects. After that, an Object Definition Language (ODL) is used to describe how a Globe distributed shared object will actually implement and export its interfaces. This may be done in a general programming language such as Java, or a special-purpose ODL. Since we have not yet developed such an ODL, Globe distributed shared objects are currently defined directly in Java.

The run-time environment itself needs basic services such as memory management, thread management, I/O, persistence, etc. In Globe, these are to be implemented as library services or basic objects (e.g. stream objects for I/O or thread objects). For our Java implementation, we have used native Java objects and services whenever possible.

Finally, to implement Web objects, the Globe environment must provide implementations of the replication, communication and control subobjects. These are currently available in the form of simple Java objects contained in libraries.

[1]The Globe IDL is different enough from other IDLs to warrant a different mapping than that used by existing systems.

Because the interaction of Globe distributed shared objects is based on interfaces, the use and mapping of these interfaces is important. In Globe, interfaces are represented by actual structures that clients can store or pass around (as parameters or return values). These representations of interfaces (which we will refer to as **Globe interfaces**) are used to perform the actual invocation of methods. A Globe interface provides clients with access to (part of) the methods exported by a Globe distributed shared object. Each Globe interface is mapped to a Java class representing that interface. In other words, we use a separate Java class for each Globe interface. Such a Java class contains the methods belonging to the Globe interface, and implements them by forwarding method invocations to Globe objects containing the actual implementations. Clients can thus invoke a Globe distributed shared object directly through a Java object.

Figure 5 shows a situation where a Java client has received a Globe interface for a Web object and calls a method from this interface. The Java object representing the Globe interface calls the same method on the control object (also a Java object) of the local implementation of the Web object. The control object then calls methods on the replication and semantics objects (which are also Java objects) as necessary.

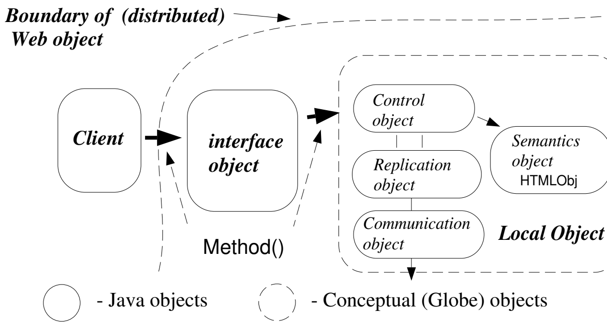

Figure 5: A Java client accessing a Web object through an interface object

4.2 Web Objects

The main activity in creating a Web object is implementing a semantics object. Once the semantics object has been written, a control object is generated from the description of the semantics object. This control object accepts all the method calls from the client and uses the replication object's knowledge of the distribution policy to decide whether to call the semantics object directly, or to marshal parameters and to let the replication object take care of the call.

The replication object uses the communication object to send and receive messages. The communication object implements point-to-point communication using TCP/IP from Java's networking API. It can also do blocking sends, or nonblocking sends and explicit receives. The replication object uses the control object to call methods of the semantics object. The replication object need not know which method is to be called, it just passes information on to the control object. The control object uses Java's reflection API [16] to unmarshal any parameters and to subsequently call the appropriate method of the semantics object.

We have implemented several replication objects, based on different replication coherence requirements of Web objects. Details on the algorithms and policies are described in [10].

It is important that Web objects are integrated into the current Web infrastructure in such a way that they can be accessed using currently available Web tools such as Web browsers, spiders, etc. There are three approaches to integrating Web objects into the current Web. The first approach involves a gateway, the goal of which is to allow standard Web clients (e.g., browsers) to access Web objects. The gateway is a process that runs on a server machine and accepts regular HTTP requests for a Web object. When a request comes in, the gateway locates the desired object, binds to it (i.e., acquires a Globe interface for it), and retrieves data from it. If necessary, this data is converted into an appropriate format (e.g. to HTML) and an HTTP reply containing the data is sent back to the client. With this approach, clients are completely unaware of the actual implementation of Web objects.

The second approach involves a custom extensible browser that is aware of Web objects and knows how to find and communicate with them. The browser is directly responsible for locating the appropriate Web object, binding to it, and receiving data from it. The browser will also have to be able to deal with the Web object's data, possibly using plug-in extensions when necessary.

The third approach is a hybrid one that combines elements from the previous two approaches using the Java support present in most major Web browsers. As in the first approach, a gateway is used to access Web objects. However, instead of returning the data directly to the browser, the gateway returns a custom Java applet. This applet will contact and bind to the Web object via the gateway, and represent the Web object in the client's browser. Such an applet can be omitted in certain cases, such as for HTML Web objects. This is the approach we have chosen for our prototype implementation.

5 Related Work

We have presented Web objects as a solution to the Web's scalability problems. Web objects are however not the only solution devised to solve this problem. Currently a number of replication and caching techniques are in standard use in the Web. Popular Web sites (or parts thereof) are often replicated by creating mirror sites. Consistency is usually achieved through manually updating the mirror sites based on changes to the master site. Sometimes these changes are automated by the Web site maintainers. However, the techniques used are ad-hoc and have not been standardized.

Three forms of caching are currently in use. Client caching is done by browsers with a separate local cache for each user. The use af a separate cache for each user results in inefficient use of cache space as there are often many redundant entries among the users. Caching proxies [12] offer a solution to this by keeping site-wide caches. For large sites (companies, universities, etc.) only single copies of resources need to be cached allowing more efficient use of cache space. This solution still suffers from the fact that *every* accessed document is added to the cache, and that coherence is not tuned to individual documents. Finally, server caches are pr ovided as a solution to the problem of degrading server performance. By caching recently accessed pages in main memory, servers can speed up access to popular resources, thus increasing their performance. However, at very popular sites the problem of server performance is often coupled with that of bandwidth, so this offers only a partial solution.

A number of Web problems are partially addressed in the new HTTP version (1.1) [6]. It contains improved use of network resources and a large and extendible set of parameters to facilitate the sending of caching information between client and server. This allows current caching techniques to make wiser decisions, but the basic drawbacks remain.

Due to the limitations of the current caching and replication approaches much work has been done to change current Web caching. Wessels [19] proposes to allow servers to grant or deny a client permission to cache a resource. Push-caching [7] allows popular resources to be optimally distributed to other servers based on knowledge of the resource's access patterns. Harvest caches [4], which provide a hierarchical cache, are currently gaining popularity in the Web. The Bayou [17] project, though not directly related to the Web, has looked at the problem of replica and cache coherence, and we have based a number of our coherence models on some of their work [10]. Caching goes Replication [1] looks at caching as replication where updates are pushed to the client rather than the current situation where the client decides when to update the cache.

Due to the limited and noninteractive nature of HTTP, much attention has been focused on the combination of object technology and the Web. Currently much work is being done in integrating CORBA [15] and similar distributed object technologies and the Web. Especially the combination of Java and CORBA is receiving much attention [5]. These approaches do not tackle the problem of scalability in the Web, and do not provide solutions for caching, replication and consistency. An approach with similar aims as ours is that adopted by the W3Objects system [3]. A number of fundamental differences are that this system aims at providing a highly visible caching mechanism, whereas we aim for maximum transparency. Also, W3Objects do not address coherence models for their objects as we do. An interesting development in the combination of object-based techniques and the Web is the proposed HTTP-NG [21] protocol, the goal of which is to present a new object-based protocol for the Web.

There are still a number of open issues concerning Web objects and their development. First of all, we must come up with an acceptable naming scheme for Web objects. Dynamic loading of (sub) objects is also a desirable feature. The current implementation of Web objects is in Java. However, our goal is to allow Web objects to be implemented in other languages (e.g. C++, C, etc.) and allow these Web objects to be able to interact with each other. Other issues such as security must also be looked into.

References

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

[2] A. Birrell and B. Nelson. "Implementing Remote Procedure Calls." *ACM Trans. Comp. Syst.*, 2(1):39–59, Feb. 1984.

[3] S. Caughey, D. Ingham, and M. Little. "Flexible Open Caching for the Web." In *Proc. Sixth Int'l WWW Conf.*, Santa Clara, CA, Apr. 1997.

[4] 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.

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

[6] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. "Hypertext Transfer Protocol – HTTP/1.1." RFC 2068, Jan. 1997.

[7] J. Gwertzman and M. Seltzer. "The Case for Geographical Push-Caching." In *Proc. Fifth HOTOS*, Orcas Island, WA, May 1996. IEEE.

[8] P. Homburg, M. van Steen, and A. Tanenbaum. "An Architecture for A Scalable Wide Area Distributed System." In *Proc. Seventh SIGOPS European Workshop*, pp. 75–82, Connemara, Ireland, Sept. 1996. ACM.

[9] P. Homburg, M. van Steen, and A. Tanenbaum. "Unifying Internet Services using Distributed Shared Objects." Technical Report IR-409, Vrije Universiteit, Department of Mathematics and Computer Science, Oct. 1996.

[10] A. Kermarrec, I. Kuz, M. van Steen, and A. Tanenbaum. "A Framework for Consistent, Replicated Web Objects." Technical Report IR-431, Vrije Universiteit, Department of Mathematics and Computer Science, Sept. 1997.

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

[12] A. Luotonen and K. Altis. "World-Wide Web Proxies." *Comp. Netw. ISDN Syst.*, 27(2):1845–1855, 1994.

[13] 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.

[14] ObjectSpace Inc. *Voyager User Guide*, July 1997.

[15] OMG. "The Common Object Request Broker: Architecture and Specification, revision 2.1." OMG Document Technical Report 97.09.01, Object Management Group, Aug. 1997.

[16] Sun Microsystems, Mountain View, Calif. *Java Core Reflection API and Specification*, Feb. 1997.

[17] 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 *Proc. Third Int'l Conf. on Parallel and Distributed Information Systems*, pp. 140–149, Austin, TX, Sept. 1994. IEEE.

[18] M. van Steen, P. Homburg, and A. Tanenbaum. "The Architectural Design of Globe: A Wide-Area Distributed System." Technical Report IR-422, Vrije Universiteit, Department of Mathematics and Computer Science, Mar. 1997.

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

[20] A. Wollrath, R. Riggs, and J. Waldo. "A Distributed Object Model for the Java System." *Computing Systems*, 9(4):265–290, Fall 1996.

[21] WWW Consortium. "HTTP-NG Architectural Overview." http://www.w3.org/Protocols/HTTP-NG/http-ng-arch.html.