W3Objects: Bringing Object-Oriented Technology to the Web

D. B. Ingham, M. C. Little, S. J. Caughey and S. K. Shrivastava
Department of Computing Science, University of Newcastle upon Tyne,
Newcastle upon Tyne, NE1 7RU, United Kingdom
{dave.ingham, m.c.little, s.j.caughey, santosh.shrivastava}@ncl.ac.uk

Abstract

In this paper we discuss some of the problems of the current Web and show how the introduction of object-orientation provides flexible and extensible solutions. Web resources become encapsulated as objects, with well-defined interfaces through which all interactions occur. The interfaces and their implementations can be inherited by builders of objects, and methods (operations) can be redefined to better suit the object. New characteristics, such as concurrency control and persistence, can be obtained by inheriting from suitable base classes, without necessarily requiring any changes to users of these resources. We describe the W3Object model which we have developed based upon these ideas, and show, through a prototype implementation, how we have used the model to address the problems of referential integrity and transparent object (resource) migration. We also give indications of future work.

Keywords: World Wide Web, Object-oriented, Referential integrity, Mobility, Distributed systems

1. Introduction

Electronic publishing in the large has grown massively since the advent of the World Wide Web. Many factors have contributed to its success, in particular, the simplicity of the system has played a significant role in its acceptance. Accessing information via the hypertext interface is relatively easy to learn and use. Becoming a publisher on the Web is also straightforward; users from all professions have been able to participate, resulting in the huge diversity of the information currently available. These factors coupled with free Web software with cross-platform support have helped the Web gain critical mass.

Within the current Web environment, hypertext links are used to glue together resources which are primarily static, read-only, and file-based (henceforth referred to as *standard resources*). Although, the Web has been very successful in serving such resources, it suffers from a number of shortcomings and it is questionable whether the current system will be able to scale indefinitely given the massive growth rates currently being witnessed. In particular, one highly visible problem is that of broken links between documents, caused by the lack of referential integrity within the current hypertext implementation. With the predicted growth in the number of resources, this problem will increase in significance, and the Web will become more difficult to use and manage.

The future is also likely to bring new demands on the Web in terms of the types of resources that it will be required to serve. The Web has proved its usefulness in the organization of distributed documentation, and users will wish to access other kinds of resource, with richer interfaces, within the same environment. Furthermore, service providers will wish to take advantage of the Web in order to tap the potentially huge customer base.

In this paper, we will show how the current Web implementation is object-based, with a single interface, comprising the set of HTTP [1] operations. Although extensions have been implemented to allow the incorporation of *nonstandard resources*, it is the opinion of the authors that the system does not exhibit the necessary extensibility characteristics that are required if the Web is to cope with the introduction of more advanced resources and services. We will show how making the change to an object-oriented system can yield an extensible infrastructure that is capable of supporting existing functionality and allows the seamless integration of more complex resources and services. We aim to use proven technical solutions from the distributed object-oriented community to show how many of the current problems with the Web can be addressed within the proposed model.

In the next section, a critique of the current Web is presented, highlighting existing problems in serving standard resources and the current approach for incorporating nonstandard resources. The section entitled "W3Objects" describes the W3Object design, its aims, object model, and system architecture. The "Illustrations" section gives an example, describing how particular Web shortcomings can be addressed within the proposed architecture. The remaining sections describe our implementation progress, plans for further work and concluding remarks.

2. Critique of the Current Web

This section explores some of the problems with the current World Wide Web, which we have classified into three categories. Firstly, we consider the problems associated with the primary function of the Web, that is, in serving standard resources. The current techniques for incorporating nonstandard resources are then discussed. Finally, the recent proposals for extending the existing Web model are analyzed and reviewed.

2.1 Serving Standard Resources

Currently the Web is primarily populated with standard resources including Hypertext Markup Language (HTML) [2] documents, GIF images, postscript files, audio files, etc., with the common characteristics of being passive, read-only, and static. All resources are currently named using Uniform Resource Locators (URLs) [3], a locational naming scheme which describes the protocol used to access the resource, the Internet address of the server containing the resource, and the location of the resource within that server. HTML documents contain hypertext links which are either intraresource, for providing multiple navigational paths through a document, or interresource, providing the *webglue*, allowing related resources to be connected.

One of the major advantages of the Web is the speed and ease by which information can be made available. In this environment Web resources are frequently created, moved, and destroyed in an ad-hoc manner, suiting the owner of the information or publishing site. These changes are usually made autonomously, with little regard for users of the information. There are three distinct problems tied together here, namely, referential integrity, migration transparency, and resource management. Quality of service, which is also affected by these issues, is also discussed.

2.1.1 Referential integrity

A system supports referential integrity if it guarantees that resources will continue to exist as long as there are outstanding references to the resources. The Web does not support this property and cannot do so since the system is unaware of the number of references that exist to a particular resource.

It is impractical to maintain every resource that has ever been published on a particular server forever, this simply does not scale. Resources that are no longer of value, for whatever reason, become garbage and need to be collected. This may involve moving the resources to backing storage, or in some cases, deleting the resources entirely. Access pattern information, which is currently available through examination of server logs, is not a sufficient basis to decide whether an object is safe to garbage collect as important though rarely used references to a resource may exist. Safe garbage collection can only be performed if referencing information is available.

The consequences of deleting resources that are still referenced affects both the user and the information provider. In the Web environment deleting a resource that is referenced by another resource results in a broken hypertext link. Such broken links are the single most annoying problem faced by browsing users in the current Web. Broken links result in a tarnished reputation for the provider of the document containing the link, annoyance for the document user, and possible lost opportunity for the owner of the resource pointed to by the link.

2.1.2 Migration transparency

In addition to the problems associated with deleting Web resources, migrating resources (either intra- or interserver) also has the potential to break hypertext links. Using the URL naming scheme, when a resource moves its identity also changes. Therefore, hypertext links to the old name will now break, with the same consequences as stated previously.

A partial solution to this problem is provided by the use of the HTTP redirect directive, which provides a forwarding pointer to the new location, allowing clients to rebind to the resource (automatically in the case of redirection-aware browsers). However, this is only a partial solution for the following reasons: firstly, documents containing references to the old location of the resource are not automatically updated, and so future requests will continue to access the old location first. Secondly, even if there were an automatic update mechanism, the lack of referential integrity means that the redirector can never be safely removed since it is impossible to determine whether all of the links have been updated. There is also the possibility that the URL may be reallocated following the migration.

The IETF Uniform Resource Identifiers (URI) Working Group's [4] work on Uniform Resource Names (URNs) [5], an alternative naming scheme, based upon logical rather than locational naming attempts to address the problem of migration transparency. The mapping from logical name (URN) to locational name (URL) is stored within a name-server which is updated with the new URL when the resource moves [6]. The disadvantage of this approach is the performance penalty associated with name-server lookups, updates, and access bottlenecks. Furthermore, this scheme does not address the issue of referential integrity.

2.1.3 Resource and service management

As previously mentioned, most resource accesses through the Web are read-only operations. Updates to the resources by the information provider are performed using mechanisms orthogonal to the Web; that is, using the native commands and editing tools of the server machine. In effect Web resources reside in two distinct domains in parallel: the traditional structure of the filesystem and the complex interlinking Web of hypertext. Within these two environments, the interfaces to the resources as well as the relationships between the resources are fundamentally different, as illustrated in Figure 1.

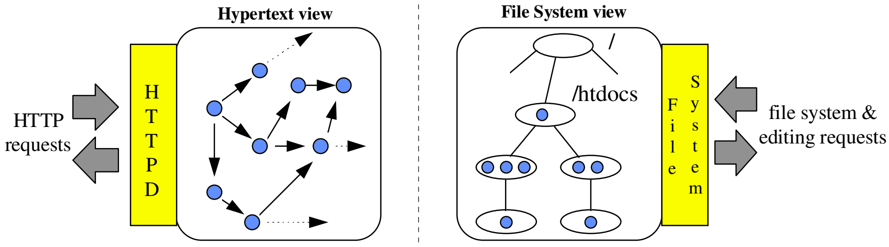

Figure 1: Disjoint interfaces to web resources

Maintenance operations carried out by the information provider or site maintainer typically require manipulation of the resources within both domains. As described earlier in the section entitled "Migration Transparency," moving a resource within the file system has the side effect of changing its name in the Web domain. To reflect such changes, other Web resources must also be modified. Further, the internal state of the moved resource may also require modification, due to the use of relative naming [7]. At present such changes are performed manually and are prone to mistakes and inconsistencies (although tools, for example, *MOMSpider* [8] exist to help eventually detect some of these). This can be seen as a result of having two parallel interfaces without any support for maintaining consistency.

2.1.4 Quality of service

The perceived quality of service (QoS) of the Web is influenced by many factors, including the broken link problems already mentioned. Even if a user holds a correct reference to an existing Web resource, it may still be unavailable due to a number of reasons, including unavailability of the machine serving the resource, and partitions in the network between the client and server. Partitions may either be real, caused by breaks in the physical network, or virtual, due to excessive network or server load making communications between the client and server impossible. Even if communication is possible, very poor response characteristics may effectively make the resource unusable. QoS will become more of an issue as the Web continues its transformation into a commercially oriented system

Technical solutions for improving QoS are fairly well understood, including caching for responsiveness, replication for availability, and migration for load balancing. Caching in the Web is reasonably common, both through the use of browser memory and disc caches, and also through the use of caching servers [9]. Effective caching is not a trivial task since there are many subtle problems that need to be addressed, including cache consistency, accounting, etc. Current caching servers use a heuristic approach for consistency management, where resources can only apply coarse-grained tuning based on expiry dates. The IETF URI working group is implementing a framework for resource replication, but appear to only be addressing the problem within the realm of read-only, static resources, where a read-from-any policy is appropriate [6]. Replicating more complex read/write resources is much more difficult due to the problems associated with maintaining consistency between concurrent users and in the presence of failures.

2.2 Incorporating Nonstandard Resources

Those Web resources which possess a richer interface than standard resources are manipulated through the Common Gateway Interface (CGI) protocol [10], which sits above HTTP. Through this protocol it is possible to perform arbitrary operations, not supported directly by HTTP, on these resources. Such resources are still identified by URLs, but the HTTP daemon is able to distinguish CGI URLs from standard resources. Upon receiving a request containing a CGI URL, rather than dealing with the request itself the daemon invokes a CGI *script* identified in the URL. Additional parameters can be included in the URL, which the daemon passes directly to the script, to be used in an application specific manner in the servicing of the request. When the script finishes the request it returns a reply message to the client and terminates.

Unlike publishing standard resources, writing CGI scripts is not simple; the resources manipulated need no longer be read-only and the issues involved in interacting with them are much more complex. There is no support for controlled sharing of resources, and because HTTP is still the underlying communication protocol, connections between client and resource last only for each request. This forces users to adopt ad-hoc solutions to problems such as persistence, concurrency control, and sharing of resources between requests.

2.3 Extending the Model

The HTTP protocol specifies the operations a client can attempt to perform on a Web resource; for example GET, POST, etc. There is also support for an *extension method*, which allows user-specific operations to be added to the protocol. Resource providers who wish to use this feature, must liaise with the Internet Assigned Numbers Authority (IANA) to register these operations. The IETF have also proposed new standard methods to be incorporated into HTTP in response to user requests [11]. Irrespective of the approach used to add new methods to the protocol, changes will be required to both the daemon and browsers. Further, the new methods may be inappropriate for the majority of Web resources and there will still be requirements which they do not address. This will result in a further round of protocol negotiations and changes to Web software. There is a long lead time involved between initial method proposals and final implementations.

The result of this model and the constraints it imposes are that people may change the daemons in an ad-hoc manner. This makes migration of Web resources from modified daemons difficult because modifications need to be made to every daemon which services them. There is also no easy route for code reuse, leading to individuals implementing similar methods in different ways.

2.4 Conclusions

From an object perspective, the HTTP daemon provides a single interface through which all objects it manages are accessed. In effect, the current Web may be viewed as an object-based system with a single class of object. What is desired is the ability to modify this interface on a per-resource basis. Within the object-oriented paradigm such specializations are achieved through the use of inheritance, and it is the absence of this which distinguishes an object-based from an object-oriented system [12].

We believe that the model and implementation we shall present in this paper represent a cleaner way of addressing the issues of flexibility and extensibility in a uniform manner, and can be used to provide solutions to the problems highlighted earlier. They can also be applied to the current Web with little or no modifications to users of these resources, and minor requirements from their providers. In the following sections we will describe our model and illustrate it with an implementation which supports referential integrity and transparent object migration.

3. W3Objects

The primary objective of our research is to develop an extensible Web infrastructure which is able to support a wide range of resources and services. Our model makes extensive use of the concepts of object-orientation to achieve the necessary extensibility characteristics. Within this object-oriented framework, proven concepts from the distributed object-oriented community will be applied to the problems currently facing the Web.

The next section introduces our object model, describing how the principles of object-orientation are applied to the Web domain. The interactions between the system components are described in the section entitled "System Architecture," which is followed by a section entitled "W3Object Properties" which classifies and describes a collection of properties applicable to different classes of W3Object.

3.1 Object Model

In the proposed model, Web resources are transformed from file-based resources into objects, *W3Objects*. W3Objects are *encapsulated* resources possessing internal state and a well-defined behavior. The objects themselves are responsible for managing their own state transitions and properties, in response to method invocations. This model supports *abstraction*since clients only interact with W3Objects through the published interfaces; the implementation of a particular operation is not externally visible.

Different classes of W3Object support different operational interfaces, which are obtained through the use of *interface inheritance*. Abstract classes are used to define an interface to a particular object abstraction, without specifying any particular implementation of the operations. Different classes of W3Objects may share conformance to a particular abstract interface, but may implement the operations differently, in a manner appropriate to the particular class. For example, consider a `Manageable` interface, including a `migrate` operation, for moving objects from one location to another. While the same interface is appropriate for many classes of W3Objects, the implementations may differ; for example, migration of a hypertext object may require some internal link manipulation operations in addition to the operations required by, say, a text file.

The use of interface inheritance provides *polymorphism*; that is, all derived classes that conform to an interface provided by some base class may be treated as instances of that base class, without regard for any other aspects of that class' behavior. Continuing with the previous example, consider a dedicated GUI-based Website management tool, providing a graphical interface for performing management-style operations on the objects; one such operation may be object migration. The management tool is able to abstract away from other features of the different objects (supported through various other interfaces) and simply address all of the different objects as instances of the `Manageable` interface. In addition to inheritance of interface, the model also supports *behavioral inheritance*, thereby supporting code reuse. For example, object properties such as persistence and concurrency control may be provided by *mixin*base classes to be inherited as required. Mix-in classes are not designed to be instantiated themselves. They are used to augment the functionality of the derived class, by providing some particular behavior, usually orthogonal to the primary function of the class.

The diagram in Figure 2 illustrates the key points of our object model by showing how two example W3Object classes, `Spreadsheet` and Play, are composed using both interface and behavioral inheritance. The abstract class, `Manageable` provides the interface description for management-style operations (only a single operation, `migrate`, is shown). Both of the derived classes inherit this interface, providing their own implementations. In addition, both of the derived classes provide other interfaces, describing the primary operations of the classes. The `Play` class, representing a theatrical performance, provides a `bookSeat` operation, and `Spreadsheet` provides `get` and `set` operations for manipulating the contents of the spreadsheet cells. Further, the `Spreadsheet` class is concurrency controlled, having derived this property from the mixin class, `ConcControllable`. The operations `setLock` and `releaseLock` are used in the implementation of the `get` and `set` operations to ensure the integrity of the spreadsheet in the event of concurrent access.

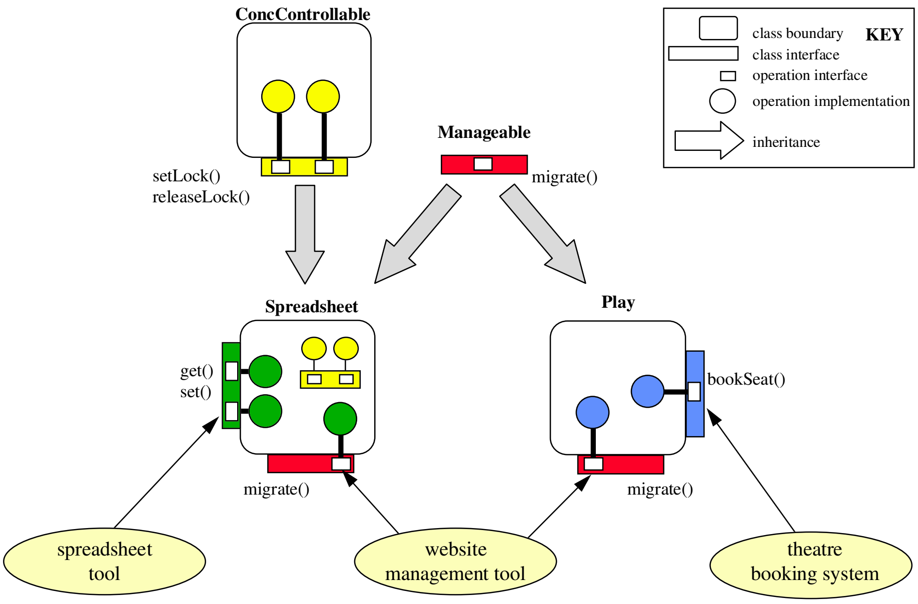

Figure 2: Illustration of using the object model

Also shown in the diagram are three different clients which manipulate instances of `Play` and `Spreadsheet`. A Website management tool, previously mentioned, is solely concerned with the operations provided through the `Manageable` interface. The tool is able to invoke the `migrate` operation on instances of either derived class without knowledge of the nature of the class. Two further clients are shown, a theatre booking application and a spreadsheet tool, which manipulate instances of `Play` and `Spreadsheet` respectively. The fact that these classes also conform to the Manageable interface is of no consequence to the clients who only interact with the objects via the interfaces supporting the classes' primary function.

3.2 System Architecture

In common with the current Web, the proposed W3Object architecture consists of three basic entity types, namely, clients, servers, and published objects. In the current Web environment, these three types correspond to Web browsers (e.g., mosaic), Web daemons (e.g., CERN HTTPD), and documentation resources (e.g., HTML documents) respectively. Our architecture supports both client-object (client-server), and interobject (peer-to-peer) communication.

3.2.1 Client-object interactions

Figure 3 illustrates, the logical view of client-object interactions within the W3Object architecture. A single server process is shown, managing a single W3Object (although servers are capable of managing multiple objects of different types), which is being accessed via two different clients, a standard Web browser, and a dedicated bespoke application. This diagram highlights *interoperability*as one of the key concepts of the architecture, that is, the support for object accessibility via different applications using multiple protocols.

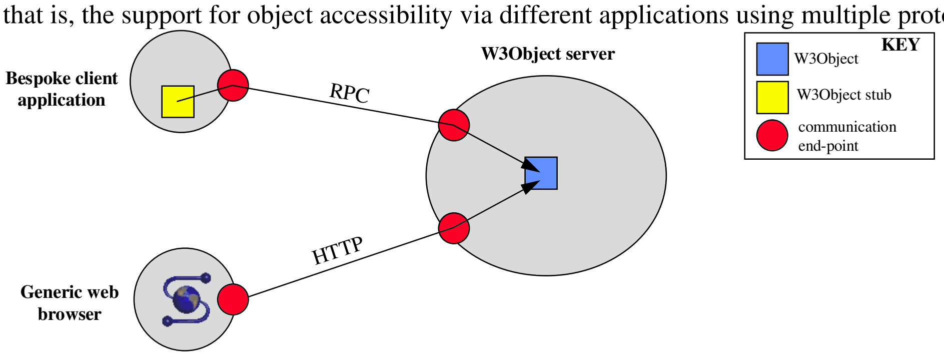

Figure 3: Client-object interactions

As stated earlier, W3Objects are encapsulated, meaning that they are responsible for managing their own properties (e.g., security, persistence, concurrency control etc.) rather than the application accessing the object. For example, in the case of concurrency, the object manages its own access control, based upon its internal policy, irrespective of which application method invocations originate from.

The local representation of an object, together with the available operations, may vary depending upon the particular type of client accessing it. The Web browser uses a URL to bind to the particular object in the server. The operations that are permitted on the object, via the URL, are defined by the HTTP protocol. The HTTP communication end-point of the server may perform name mapping between URL and the internal name for the object and may also map HTTP requests to appropriate method invocations on the object. Within the bespoke application, a client-stub object acts as the local representation of the remote object. From the point of view of the application, this stub object presents the illusion that the remote object is actually within the address space of the client. Like any other object, the stub presents a well-defined interface describing the supported operations. This interface has the potential to be much richer than that provided through HTTP, including application specific operations. Operation invocations on the stub are passed to the object using the remote procedure call (RPC) protocol. Client-stub objects may be automatically generated from a description of an object interface. Our implementation uses C++ as the definition language and we provide stub-generation support for creating client and server side stubs which handle operation invocation and parameter marshalling [13]. Other possible interface definition languages are possible, including CORBA IDL [14] and ILU ISL [15].

Continuing with the previous example of a Web-based theatre booking application; to support access via conventional browsers using HTTP, the theatre object would conform to an `HTTP` interface, supporting an operation, say, `htmlGet`. An HTTP GET request from the browser would be mapped to an invocation of the `htmlGet` method of the object. Through this limited interface, the programme of events and availability information may be available. A client-stub object for the theatre may support more advanced operations such as seat booking using *atomic actions* (*transactions*).

Although not illustrated, a number of other client implementations are possible. The common gateway interface (CGI) could be used to provide a richer client-side interface than is readily available through HTTP. Although, it has been already stated that we believe CGI to be too low-level for direct programming, CGI interfaces to remote objects can be automatically created using stub-generation tools. We have implemented a basic stub-generator, which uses an abstract definition of the remote object, and ANSA have recently released a more complete tool [16] based on CORBA IDL. Recent developments using interpreted languages within the Web, including Java [17] and SafeTcl [18] are potentially very useful for developing client-side interfaces to W3Objects. Using such languages, complex, architecture-neutral, front-ends dedicated to a particular W3Object class can be developed, supporting rich sets of operations.

3.2.2 Inter-object interactions

In addition to client-object communication, our architecture also supports inter-object communication, regardless of the objects' location. In effect, the architecture may be viewed as a single distributed service, partitioned over different hosts as illustrated in Figure 4. Inter-object communication is used for a variety of purposes, including referencing, migration, caching, and replication.

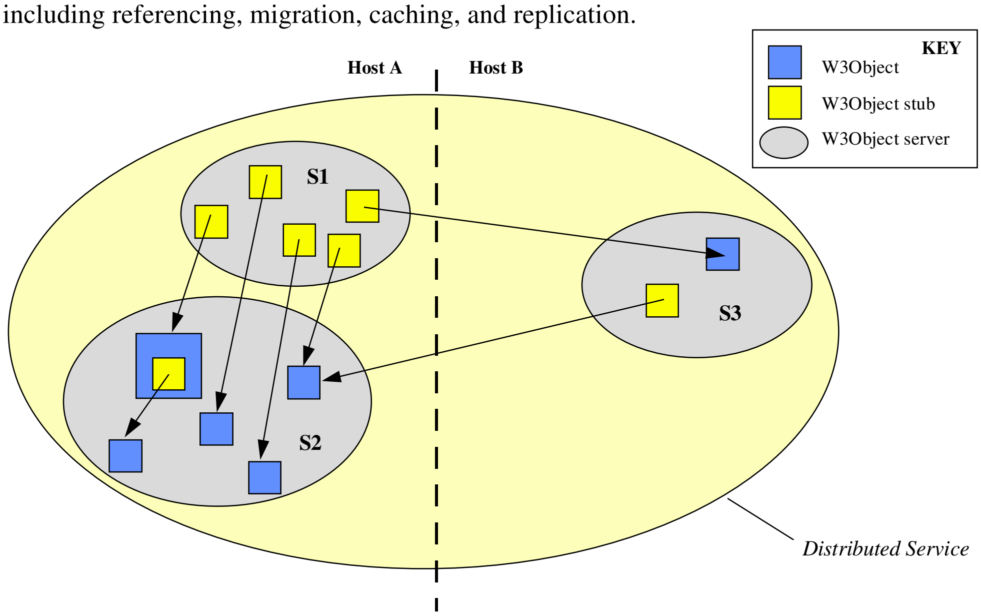

Figure 4: Inter-object interactions

In addition to W3Objects, servers may contain W3Object stubs, or aliases, which are named objects that simply forward operation invocations to another object, transparently to clients. One particular use of aliases is in implementation of name-servers, since a name-server may be viewed simply as a collection of named objects which alias other objects with alternative names (server S1 in diagram). Objects may also contain stubs to other objects (shown in S2 in diagram). This feature is used in our implementation of referencing, which is described further in the "Illustrations" section.

3.2.3 Implementation considerations

As previously mentioned, a particular W3Object server is capable of holding W3Objects belonging to a set of specific types. At the time it is created, it is unlikely that one can predict all of the object types that a particular server will be required to serve over its lifetime. This raises the question of how new types are introduced to the system. The usual approach is to bring down a server, rebuild it with the code which supports the new object types, and then restart it. This approach is less than ideal for an application such as the Web where there is a requirement for keeping downtime to a minimum. An alternative technique relies on *dynamic loading*, which allows an active process to load new code as necessary. Although this approach is clearly more elegant and we do not preclude its use, dynamic loaders currently suffer from lack of standardization and therefore portability. A third, more pragmatic, approach is to create another server to support the new object types. This approach may be desirable even if suitable dynamic loading support is available due to requirements for fault tolerance and security because, as a single server for all object types is a single point of failure, multiple servers can be used to protect users against such failures.

One method of interfacing with multiple servers is to make use of an HTTP Gateway, which uses stub objects to forward object invocations through to the appropriate server. The gateway is transparent to clients accessing the objects; incoming requests are simply forwarded to the destination object, which parses the request and replies accordingly. This is illustrated in Figure 5, in which server S1 manages a number of different types of object (illustrated by different shapes) and server S2 manages objects of a single type. As the processing of operations is entirely the responsibility of the individual object, the introduction of new object types is transparent to the gateway.

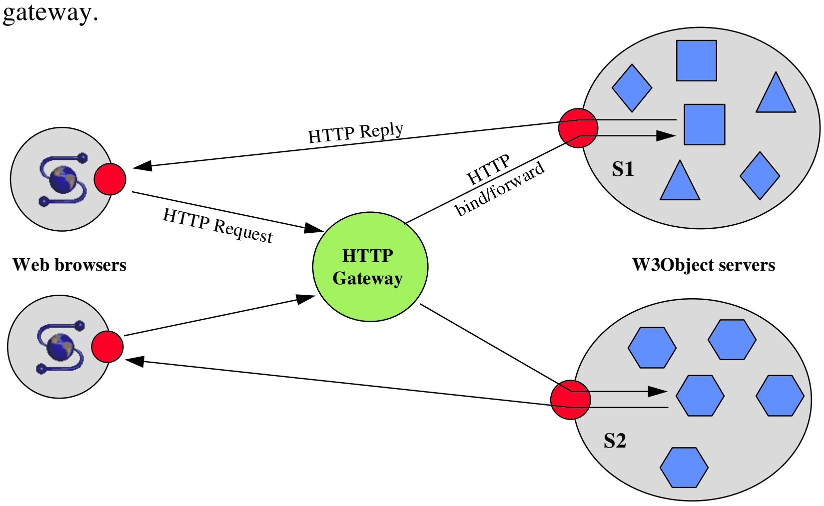

Figure 5: Client-object communication through gateway

3.3 W3Object Properties

Based on critiques of the current Web by ourselves and others [19], and also our experience with distributed systems in general, we have attempted to identify the set of properties that are required by W3Objects. We have classified these properties into three categories: core properties, common properties, and class-specific properties. In this section we shall present what we believe to be the core properties required by all W3Objects and give examples of some common properties.

3.3.1 Core properties

Four properties have been identified as being the core requirements for W3Objects: Naming, Sharing, Mobility, and Referencing. The implementation of these properties is divided between the objects themselves and the supporting infrastructure which manages the objects. Each property will be considered in turn.

Naming: One of the fundamental concepts of the object-oriented paradigm is identity. The ability to name an object is required in order to unambiguously communicate with and about it. Context-relative naming is an essential feature of our environment so as to support interoperability and scalability. As mentioned previously, different clients may use different local representations of a remote object (URLs, client-stub objects, etc.). Since it is impractical to impose new naming conventions on existing systems, we require the ability to translate names between system-boundaries. Furthermore, for extensibility, we need to be able to incorporate new naming systems. Within our design, naming is provided via the object infrastructure. Context-relative naming is supported via the use of name-servers, implemented as collections of object aliases, as described previously.

Sharing: Implicit within the Web domain is the requirement that objects can be shared. Although the basic function of allowing multiple users to interact with objects is simple to achieve, there are a number of other associated mechanisms that require interaction with the base sharing functionality. Access control, either user and group based, or access restriction based on the location of the client, are both likely requirements. Additionally, with objects supporting a rich set of interfaces, the granularity of the control must be configurable.

Mobility: One of the lessons learned from the current Web is that support for object mobility is a necessary requirement for W3Objects. At object creation time, migration of the object may not be envisaged, but it is virtually impossible to predict the future requirements of a particular object. Mobility may be required for many reasons, including load balancing, caching, and improved performance through locality etc., with different forms of migration, including intra- or inter-host.

Referencing: In order to address what may be viewed as the primary problem with the current Web, namely referential integrity, we believe that low-level referencing support is required by all objects. A range of schemes is possible, including forward referencing [20], call-backs, and redirection through a location server (as in the URN approach). Referencing is closely related to mobility, since referencing schemes may be used to locate objects even in the event of object migration. Our approach to referencing is described in detail in the "Illustrations" section.

3.3.2 Common properties

There are a potentially large number of common properties for W3Objects which can be encapsulated within appropriate base classes. In the rest of this section we shall highlight four of these properties which we have selected based upon our experience of building reliable distributed systems [21]

Replication: There is a range of replication protocols from active to passive, and strong consistency to weak consistency. There is no single replication protocol which is suitable for every object which may need to be replicated and at the same time can satisfy a user's required quality of service [22]. As such, it is our intention to implement a suitable base class for object providers, which will enable them to select the appropriate replication protocol on a per object basis. In addition object providers will also be able to select the optimum number and location of these replicas, and modify this as required [23].

Concurrency control: By enabling users to share arbitrary objects it may be necessary for these object state transitions to be managed through an appropriate concurrency control mechanism. Consider the theatre booking example earlier: if user A wishes to examine the seats which are available while user B is in the process of booking a seat, it would be desirable for B to lock the seat in order to prevent conflicting updates. There are a number of concurrency control mechanisms available, but our initial implementation will be based upon the familiar multiple reader, single writer policy.

Caching: The caching of object states, either at or close to users, can help alleviate problems of network congestion and latency. However, as with replication, there is a need for a range of caching policies based upon user requirements and object properties.

Fault tolerance: In a large-scale distributed system, fault tolerance is an important property. One way of addressing the issues of fault tolerance is by using atomic actions to control method invocations. Objects inherit necessary persistence and concurrency control characteristics, and application programmers then manipulate these objects within atomic actions which guarantee the usual ACID properties.

4. Illustrations

Having described our model in the previous sections, we shall now illustrate how two of the core properties, referencing and mobility, are implemented within the model. Our aim is to address the current problem of broken links and provide transparent object migration.

4.1 Referential Integrity

In our model Web resources are represented as W3Objects and may be referenced from some *root*, either directly, via W3Object stubs, or by being contained within another W3Object (note that W3Object stubs are themselves W3Objects). This is illustrated in Figure 6, which shows a number of objects, all of which are reachable from some roots.

Our service maintains the distributed referencing graph and uses reference counting to detect unreferenced objects. Stubs, when created, perform an explicit *bind* operation on the object they refer to (thereby incrementing the object's reference count) and perform an *unbind* operation whenever the stub is deleted (thereby decrementing the count).

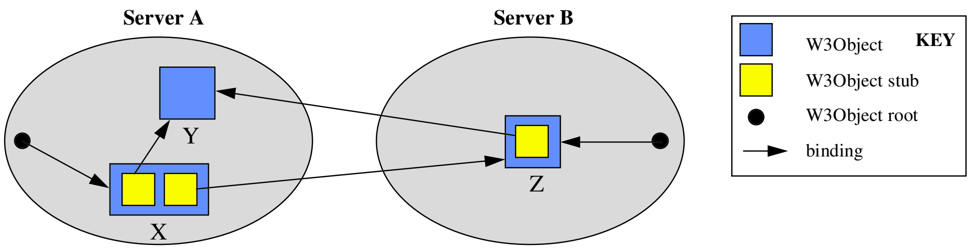

Figure 6: Object referencing

Reference from a root represents the service’s continued interest in the object while stubs represent hypertext links from one object to another. Therefore an object is not available for deletion until (1) the service has expressed its lack of further interest in the object (by no longer referencing it from a root) and (2) there are no hypertext links from other objects. Once these conditions are met the object is marked as unreferenced and is available for deletion.

4.2 Migration Transparency

An object may be moved from one location to another (potentially to some remote site). Our service guarantees that object moves are transparent to all other objects i.e., references to the moved object continue to be valid. Whenever an object is moved a stub representing the object is left behind at the old location and this automatically forwards invocations (without involving the invoker) to the new location. Further moves of the object may cause a chain of such stubs to be created (in a similar manner to that used by SSP chains [20], leading ultimately to the real object. For example, in Figure 7, object Y has been moved, first to server B, and then subsequently to server C, leaving behind the forwarding chain shown. Whenever an invocation follows such a chain these indirections are short-cut i.e., the invoker's stub communicates directly with the W3Object at the new location. Following a short cut intermediate stubs which are no longer referenced are automatically garbage collected.

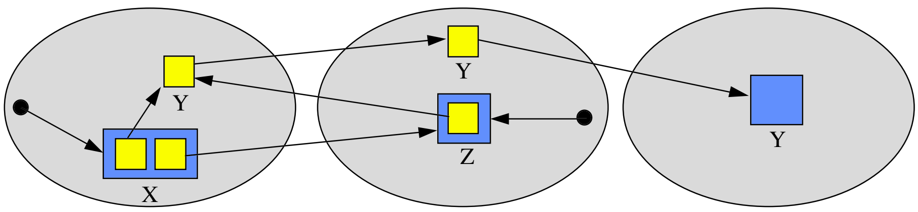

Figure 7: Referencing chains

Using the short-cutting mechanism stubs will in general point directly at their W3Object; however, there may be circumstances where the chain becomes longer than is desirable, i.e., the Object moves a number of times between invocations from a particular stub. Long chains have more points of failure and a performance cost in the necessary indirections at invocation time, and consequently there may be situations in which long chains are to be avoided. Therefore as an extension to our basic forwarding location mechanism we allow stubs to register callbacks with objects such that, whenever an object moves, the registered stubs are automatically informed of the move. These stubs may either represent hypertext links from other resources and/or belong to a location service which offers the current position of the object to any of its users. More details are given in [24]
.

Our use of forward references is an entirely distributed solution which scales well by not relying on centralized location services or broadcasts of moves, as is proposed by Hyper-G [25]. Hyper-G ensures that all references to all resources are (eventually) updated to reflect any movement, but this requires additional network traffic that may be appropriate for individual enterprises but does not scale to a worldwide solution. Our system also guarantees that any object movement is transparent to any reference holder and does not require the handling of object faults and subsequent rebinding. Our policy is to provide inexpensive forward referencing (which introduces no extra network traffic when an object is moved) as the default location service but to offer alternative, potentially more expensive solutions such as callback and the use of location servers for resources with specific performance or fault-tolerance requirements. Our model allows the choice of appropriate location mechanisms for an application and the ability to vary that choice at run-time as required.

The mechanisms described above offer a consistent view of referencing within the system so long as (1) neither the referenced objects or their stubs can be lost through failures and (2) the bind and unbind operations are totally reliable. We guarantee that W3Objects and stubs recover from failures by saving them on stable store i.e., a file system. The bind and unbind operations use an RPC to access the W3Object, which may be held within a remote server (optimized to a local procedure call whenever the object is held in the same server). Our remote object-invocation layer guarantees "at-most-once" semantics for the RPC; i.e., despite failures operations will eventually be performed.

5. Implementation Progress

Using Shadows, a lightweight object support system [26], we have begun the implementation of the core W3Object properties, concentrating initially on referential integrity and object mobility. During this prototyping stage, we have developed a system which operates in parallel with conventional HTTP daemons. Therefore, object states are stored in their native formats, (e.g., HTML files, etc.) and are available for direct manipulation through the standard file system and Web interfaces. The advantage of this approach is that we are able to develop and test our design without requiring site maintainers to alter any aspect of their existing Web configurations. The disadvantage is the inherent loose coupling between the two systems, requiring users to register entities with our service and be sufficiently disciplined so as not to delete entities directly through the file system without consulting the service. Following this prototyping stage we will introduce the HTTP Gateway and provide user-level tools to replace the direct manipulation of the object through the file system, without restricting the use of any existing resource manipulation tools, e.g., text editors.

In the prototype system, an additional daemon per host is used to provide the referencing service, which is administered using a GUI-based tool. This tool allows the site maintainer to browse through the entities currently being served by the HTTP daemon and to *include* them within the service. On being instructed to include a Web entity, the service creates a corresponding W3Object which is *registered*, causing it to be referenced from a root and thereby ensuring it is not immediately marked as unreferenced. The administration tool then parses the resource to identify any hypertext links it contains. If the links point to resources already included in the service, then references to their corresponding objects are created and stored within the newly created object, thereby extending the referencing graph. Links to resources not already within the service are reported back to the user who then has the option to explicitly include them if they are under his control. A *deregister* operation is used to express the site maintainer's lack of further interest in a resource which causes the reference count for the resource to be decremented. If the reference count reaches zero the site maintainer is informed that it is available for deletion. If the site maintainer confirms it is to be deleted then the W3Object is removed from the service and the corresponding resource removed from the file system. Deleting an object may result in the reference counts of other objects being decremented, i.e., those referenced from the deleted object.

The site maintainer may *move* a resource from one location to another (potentially to some remote site where the resource may come under new ownership). In addition to moving the object through the process described in the previous section, the service performs the following operations: (1) the resource is removed from the file system, (2) an HTTP redirector is inserted, (3) all relative URLs contained within the moving resource are transformed, and (4) the resource is placed in the file system at the new location.

The next stage of our development is to implement the full W3Object model as described in this paper, integrating the referencing work with other software subsystems developed locally as part of the Arjuna Project [27]. To provide the flexibility and extensibility we have described, we shall make use of the Gandiva system [28] which supports the separation of object interfaces from implementation, and enables reconfiguration of this relationship without requiring the rebuilding of objects or applications. For the common W3Object properties we will make use of the work developed for the Arjuna system [21], which provides the necessary fault-tolerance and replication support.

6. Conclusions

As a way of serving standard resources, the Web has proven extremely successful but still suffers from a number of shortcomings. Furthermore, in order to cope with new resource types the Web needs improved flexibility and extensibility characteristics. We have illustrated how the application of the concepts of object-orientation can achieve these extensibility requirements and how problems, such as the lack of referential integrity, can be addressed through the application of techniques developed by the distributed object research community.

The W3Object model, presented in this paper, is intended to provide a flexible and extensible way of building Web applications, where Web resources are encapsulated as objects with well-defined interfaces. Objects inherit desirable characteristics, redefining operations as is appropriate; users interact with these objects in a uniform manner. We have identified three categories of object properties: core, common, and specific, and have described an implementation using the core properties which addresses what we believe to be one of the most significant problems facing the current Web--that of referential integrity. A key feature of our design is support for interoperability; for example, in addition to sophisticated clients which may use the rich object interfaces that our model provides, our implementation will also allow W3Objects to continue to be accessed using existing Web browsers.

7. Acknowledgments

The work reported here has been partially funded by grants from the Engineering and Physical Sciences Research Council (EPSRC) and the UK Ministry of Defense (Grant Numbers GR/H81078 and GR/K34863) and GEC-Plessey Telecommunications.

8. References

1. Berners-Lee, T., Fielding, R. and Nielsen, H. F., *Hypertext Transfer Protocol--HTTP/1.0*, work in progress within the HTTP Working Group of the IETF.
2. Berners-Lee, T., and Connolly, D. W., *Hypertext Markup Language - 2.0*, *Internet RFC 1866*, November 1995.
URL: ftp://ds.internic.net/rfc/rfc1866.txt
3. Berners-Lee, T., et al., *Uniform Resource Locators (URL)*, *Internet RFC 1738*, December 1994.
URL: ftp://ds.internic.net/rfc/rfc1738.txt
4. Uniform Resource Identifiers (URI) Working Group of the IETF.
5. Sollins, K. and Masinter, L., *Functional Requirements for Uniform Resource Names*, *Internet RFC 1737*, 20 December 1994.
URL: ftp://ds.internic.net/rfc/rfc1737.txt
6. Hoffman, P. E., and Daniel, R. Jr., *URN Resolution Overview*, Internet Draft, April 1995.
URL: http://www.ics.uci.edu/pub/ietf/uri/draft-ietf-uri-urn-res-descript-00.txt
7. Fielding, R., *Relative Uniform Resource Locators*, *Internet RFC 1808*, June 1995.
URL: ftp://ds.internic.net/rfc/rfc1808.txt
8. Fielding, R., *Maintaining Distributed Hypertext Infostructures: Welcome to MOMspider's Web, First International Conference on the World Wide Web*, 1994.
URL: http://www.ics.uci.edu/WebSoft/MOMspider/WWW94/paper.html
9. Luotonen, A., *World Wide Web Proxies*, Computer Networks and ISDN Systems. Vol. 27, No.2, 1994.
URL: http://www1.cern.ch/PapersWWW94/luotonen.ps
10. Rob McCool, *The Common Gateway Interface*.
URL: http://hoohoo.ncsa.uiuc.edu/cgi/overview.html
11. Hypertext Transfer Protocol (HTTP) Working Group of the IETF.
12. Cardelli, L., and Wegner, P. *On Understanding Types, Data Abstraction, and Polymorphism*, ACM Computing Surveys Vol. 17, No. 4, p. 481, December 1985.
13. Parrington, G. D., *A Stub Generation System for C++*, USENIX Computing Systems Journal, Vol. 8, No. 2, Spring 1995, pp. 135-169.
URL: http://arjuna.ncl.ac.uk/arjuna/papers/stubgen-c++.ps
14. Object Management Group Inc, *The Common Object Request Broker: Architecture and Specification, Revision 1.2*, OMG Document Number 93.12.43, December 1993.
URL: ftp://ftp.omg.org/pub/docs/93-12-43.ps
15. Inter-Language Unification (ILU), Xerox Parc, 1991
URL: ftp://parcftp.parc.xerox.com/pub/ilu/ilu.html
16. Edwards, N., *Object Wrapping (for WWW)--The Key to Integrated Services?*, Document Identifier APM.1464, Architecture Projects Management Ltd, Cambridge, 1995.
URL: http://www.ansa.co.uk/ANSA/ISF/1464/1464prt1.html
17. Gosling, J., and McGilton, H., *The Java Language Environment: A White Paper*, Sun Microsystems, 1995.
URL: http://java.sun.com/whitePaper/javawhitepaper_1.html
18. Borenstein, N.S., *EMail With A Mind of Its Own: The Safe-Tcl Language for Enabled Mail*, ULPAA '94, Barcelona, 1994.
URL: http://minsky.med.virginia.edu/sdm7g/Projects/Python/safe-tcl/ulpaa-94.txt
19. Edwards, N., and Rees, O., *Distributed Objects and The World Wide Web*, ANSA Technical Report APM.1283.00.08, 1994.
URL: http://www.ansa.co.uk/phase3-doc-root/sponsors/APM.1283.00.08.html
20. Shapiro, M., Dickman, P., and Plainfosse, D., *Robust, Distributed References and Acyclic Garbage Collection*, Symposium on Principles of Distributed Computing, Vancouver, August 1992.
URL: ftp://ftp.inria.fr/INRIA/Projects/SOR/RDRAGC:podc92.ps.gz
21. Parrington, G. D. et al., *The Design and Implementation of Arjuna*, USENIX Computing Systems Journal, Vol. 8, No. 3, Summer 1995, pp. 253-306
URL: http://arjuna.ncl.ac.uk/arjuna/papers/designimplearjuna.ps
22. Little, M. C., *Object Replication in a Distributed System*, PhD Thesis (Newcastle University Computing Science Laboratory Technical Report 376), September 1991.
URL: ftp://arjuna.ncl.ac.uk/pub/Arjuna/Docs/Theses/TR-376-9-91_USLetter.tar.Z
23. Little, M. C., and McCue, D., *The Replica Management System: A Scheme for Flexible and Dynamic Replication*, in *The Proceedings of the 2nd International Workshop on Configuration, Pittsburgh*, March

1994.
URL: http://arjuna.ncl.ac.uk/arjuna/papers/replica_management_system.ps
24. Caughey, S. J., and Shrivastava, S. K., *Architectural Support for Mobile Objects in Large Scale Distributed Systems*, In *The Proceedings of the 4th IEEE International Workshop on Object Orientation in Operating Systems (IWOOOS)*, Lund, Sweden, August 1995.
URL: http://arjuna.ncl.ac.uk/arjuna/papers/shadows-iwooos95.ps
25. Kappe, F., *Hyper-G: A Distributed Hypermedia System*, in *The Proceedings of INET `93.*
URL: http://info.iicm.tu-graz.ac.at/
26. Caughey, S. J., et al., *SHADOWS: A Flexible Support System for Objects in a Distributed System*, *Proceedings of the 3rd IEEE International Workshop on Object Orientation in Operating Systems (IWOOOS)*, Ashville, North Carolina, USA, December 1993.
URL: http://arjuna.ncl.ac.uk/arjuna/papers/shadows-iwooos93.ps
27. The Arjuna Project Information Web Page.
URL: http://arjuna.ncl.ac.uk/
28. Wheater, S. M. and Little, M. C. *The Design and Implementation of a Framework for Extensible Software*, Broadcast Project Technical Report, University of Newcastle upon Tyne, 1995.
http://arjuna.ncl.ac.uk/arjuna/papers/framework-extensible-software.ps

About the Title

Since 1994, W3Objects was known as WebObjects. We have been forced to change this owing to a recent trademark application by the NeXT Computer Inc.

About the Authors

All of the authors share the common address:
Department of Computing Science,
University of Newcastle upon Tyne
Newcastle upon Tyne,
NE1 7RU,
United Kingdom
Tel: +44 191 222 7972
Fax: +44 191 222 8232
David Ingham [http://www.cs.ncl.ac.uk/~dave.ingham]
dave.ingham@ncl.ac.uk
David Ingham received his B.Eng. in Electrical and Electronic Engineering from Northumbria University in 1991 and an M.Sc. in Computing Software and Systems Design from Newcastle University in 1992. He is currently a Research Associate in the Department of Computing Science at Newcastle, where he is a member of the team developing the Arjuna reliable distributed programming system. His research interests include distributed computing, reliable systems, object-oriented computing and distributed-debugging tools.
Mark Little [http://www.cs.ncl.ac.uk/~m.c.little]
m.c.little@ncl.ac.uk
Mark C. Little received his B.Sc. in Physics and Computing Science from Newcastle University in 1987 and a Ph.D. in Computing Science in 1991. Since 1990 he has been on the research staff of the Department of Computing Science at Newcastle where he is currently a Research Associate. He is one of the principal designers and implementors of the Arjuna reliable distributed programming system. His research interests include reliable distributed computing, object-oriented programming languages, object replication, and operating systems.
Steve Caughey [http://www.cs.ncl.ac.uk/~s.j.caughey]
s.j.caughey@ncl.ac.uk
Steve Caughey obtained his B.Sc. in Physics from Queen's University, Belfast, in 1979 and was employed by a number of telecommunications companies working primarily in the areas of Operating System design, integration, and testing. In 1989 he obtained an M.Sc. in Computing Systems and Software Design from Newcastle University, and has since been employed by the University as a Research Associate. His research interests include large scale distribution, object migration, fault tolerance, and naming.
Santosh Shrivastava [http://www.cs.ncl.ac.uk/~santosh.shrivastava]
santosh.shrivastava@ncl.ac.uk

Santosh Shrivastava obtained his PhD in Computing Science from Cambridge in 1975. After several years in industry, he joined the Computing Science Department of the University of Newcastle in 1975 where his present position is Professor of Computing Science. His main area of research is in fault-tolerant distributed computing. He is currently leading Arjuna and Voltan research groups. The Arjuna group has developed the Arjuna object-oriented fault-tolerant distributed system. Arjuna is forming the basis for research on new network services in large scale distributed systems. The Voltan group is undertaking research into high integrity real-time systems, which involves investigation of agreement protocols, failure detection and reconfiguration, communication primitives, clock synchronization and real-time scheduling. He has over 70 publications in the areas of fault-tolerance and distributed computing.

```
ERROR: undefined
OFFENDING COMMAND: D

STACK:

(e )
0
-0.648
32
0
```