W3Objects:
A Distributed Object-Oriented Web Server

David B. Ingham
Department of Computing Science, University of Newcastle upon Tyne,
Newcastle upon Tyne, NE1 7RU, United Kingdom
dave.ingham@ncl.ac.uk

Position paper for the Object-oriented Web Servers and Data Modeling Workshop,
Sixth International World Wide Web Conference. 7th April 1997, Santa Clara, USA.

W3Objects overview

In our model, Web resources are represented as objects (W3Objects), which are encapsulated resources possessing internal state and a well defined behaviour, rather than the traditional file-based entities [Ingham95]. W3Objects may support a number of distinct interfaces, obtained via interface inheritance. Common interfaces may be shared thereby enabling polymorphic access, for example, all W3Objects conform to an HTTP interface, providing methods including httpGet() and httpPost(). The specific implementation of the methods may differ between the different classes of object. For example, a simple W3Object for holding HTML state may simply return this state in response to a httpGet() request. Common functional properties such as distribution, persistence and concurrency control are made available to application developers through base classes from which user classes can inherit.

W3Objects are organised and named within *contexts*, which may be nested. W3Object server processes (*W3OServers*) are simply active contexts. Objects are accessed using RPC and addressed by specifying the communication end-point of their containing server and the name of the object within that server. Inter-object communication is used thoughout W3Objects to support functionality such as referential integrity [Ingham96] (see later) and caching [Caughey97].

Web access to W3Objects

Web access to W3Objects is provided though a gateway, implemented as a plug-in module for an extensible Web server, such as Apache [Apache]. The gateway is fully compatible with the CGI interface allowing standard HTML forms to be used to create user interfaces to services. The Web server is configured to pass requests for part of the URL space (for example, URLs beginning `/w3o/`) to the gateway module. The remainder of the URL identifies the name of the required service and any parameters to be passed to it. The module then binds to the requested named object within the nameserver (a standard context) and invokes the appropriate method on it, e.g., `httpGet()` or `httpPost()`. Additional data associated with the request, including URL-encoded data from the client and environment data generated by the server is grouped together as a request object that is passed as a parameter to the HTTP interface operations. The request is passed on through the nameserver object to the destination object which performs the necessary computation and returns the results to the client, via the server. The diagram in Figure 1 illustrates the architecture through an example W3Objects site.

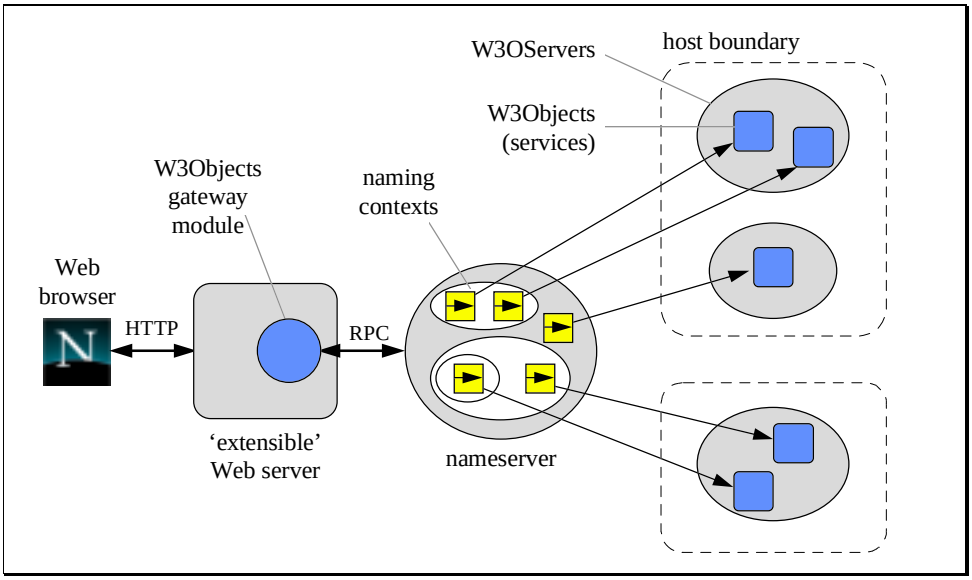

Figure 1: Architecture of a W3Objects site

W3Objects architectural properties

The W3Objects system provides basic low-level referencing mechanisms to address the broken-link problem, thereby improving quality of service. Inter-object references, such as those representing hypertext links, are maintained by an object in the form of a smart reference. Through the use of a forward referencing scheme, such references remain valid even in the event of the migration of the target object. This allows information providers to restructure sites without the fear of breaking links to their objects. Furthermore, the scheme provides an information provider with useful knowledge of which objects are currently being externally referenced and which are not. This knowledge ensures that referenced objects are not inadvertently deleted and garbage resources (i.e., those no longer referenced) are identified. See [Ingham96] for further details of the W3Objects referential integrity mechanisms.

The distributed object architecture of W3Objects also provides the following properties:

- **Scalability through transparent distribution:** The architecture supports arbitrary allocation of services to processes and processes to machines, in a manner which is completely transparent to users. This provides administrators with great freedom in selecting the optimum cost/performance configuration for their site.
- **Transparent service migration:** Services may be migrated between processes and machines as desired. Since services are encapsulated entities (discussed in more detail later), migrating a service is achieved simply by invoking a migrate operation on the service object. This is accessible via a programming-language API or via a Web-based management interface. The referential integrity support inherent in the W3Objects system will ensure that all intra- and inter-object references (for example, hypertext links) will remain valid after the migration and will be optimised over time to provide the most direct paths [Ingham96].
- **Introduction and removal of services:** Services are made available to Web clients by registering them in the nameserver. Installing a new application is simply a matter of starting the service and registering it. Similarly services can be removed by deregistering them. These operation can be performed while the system in on-line without disruption to users (naturally, service removal should be performed with care.) Again, these operations are available either through a programming-language API or a Web interface.
- **Support for stateful services:** Since W3Objects persist across requests, session-based state can be held internally, either held in memory, or optionally on secondary storage. To aid the construction of such services, application builders are able to use state persistence support provided by the W3Objects class library.

High-level object properties

As stated previously, a W3Objects application developer is able to derive application classes from system base classes to gain commonly required features. One set of classes provided assists in the construction of *highly manageable services* [Ingham97]. The support for manageability is based on the separation of presentation logic from the functional aspects of a service. Each of the pages of a service are created using *views*, which may either be passive or active entities. Examples of passive views include an HTML page or a page component, such as a navigation bar. An active view may interact with the functional aspects of the service to generate a dynamic representation. Support for scripted views is provided through a server-side scripting language, known as *W3OScript*. By encoding the presentation logic of a service in an interpreted language allows changes to be made without requiring the service to be taken off-line. Views can be nested, that is a view representing a complete page may be composed of a number of sub-views representing page components. As views are themselves W3Objects, they may be distributed and references between views support referential integrity.

Services based on this model are manageable by virtue of the ability to configure their Web-interface at run-time without requiring them to be brought down. New operations can be added to a service, existing ones modified, and redundant ones removed, through the creation, modification and removal of view objects respectively. These operations are provided through a dedicated management interface, that is accessible, either through a programming-language API or via a Web interface. Additional features of the model are summarised below:

- **Isolation of commonality:** Since common components, such as a navigation bar or author contact details are maintained in a single location the task of maintaining consistency of information is simplified. Furthermore, updates to such components are performed to a single object, with the changes automatically incorporated into views which use them.
- **Encapsulation:** Since the entire service is represented as a single object, it can be managed as such, for example, migrating the service is achieved by invoking a migrate operation on the service object. All enclosed view objects will automatically migrate with the service and links to external views will be preserved.
- **Service evolution:** View objects may be migrated between view managers, for example, a view that is initially created privately can later be exported to be shared with other services. Again, the underlying system ensures that all references to the migrated component will remain valid.
- **Accessible management interface:** All management operations are accessible via a Web interface, allowing views to be created, destroyed, and edited using a single familiar interface.

Implementation details

The W3Objects system has been implemented in C++ using Shadows, a lightweight ORB-like platform [Caughey93]. In addition to basic distributed object support, Shadows provides a number of useful facilities including support for migration, referential integrity and garbage collection of objects [Caughey95]. Although originally designed for fine-grain, programming-language level objects, many of Shadows' features have been successfully applied to the larger-grained Web-level entities within the W3Objects system.

W3Objects can be accessed and manipulated through a C++ API allowing bespoke client applications to be developed, or via the Apache gateway. In addition to providing access to the functional interface of the objects, the Web interface also provides a management interface allowing all management operations to be performed through familiar forms-based Web interfaces.

Data Modeling: Object wrapping using W3Objects

The ease-of-use of the Web browser interface coupled with cross-platform availability has meant that the Web is becoming the platform-of-choice for developing new Intranet client-server applications. However, many organisations have considerable investment locked into existing information systems. A common desire is to provide Web interfaces to these legacy information systems.

Incorporating legacy resources into a new system typically requires the construction of gateways that translate requests in the new environment to invocations on the legacy system, transforming the operation parameters and results as necessary. In the object-oriented model clients interact with objects through well-defined interfaces without concern for the object's implementation. This strong separation of interface and implementation makes object technology particularly suitable for constructing legacy system gateways. By wrapping legacy resources within objects that conform to existing interfaces of the new system, allows clients to interact seamlessly with legacy resources as if they were native objects.

One of the interesting issues to be considered when developing legacy system wrappers is the granularity at which the wrapping is applied. For example, a legacy database could be mapped to a single object or to multiple objects, representing individual records within the database.

Using this object wrapping technique within W3Objects allows legacy information systems to be seamlessly integrated into the Web. The distribution that is inherent in the W3Objects system further improves its suitability since object wrappers may be used to bring together data stored within localised systems that are physically distributed from the Web server. By using W3OScript to encode the presentation logic responsible for creating the Web representation of the legacy objects, system reconfigurations can be made without requiring system outages, thereby improving the manageability of the system.

References

[Caughey93] S. J. Caughey 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.

[Caughey95] S. J. Caughey and S. K. Shrivastava, "Architectural Support for Mobile Objects in Large Scale Distributed Systems," Proceedings of the 4th IEEE International Workshop on Object Orientation in Operating Systems (IWOOOS), Lund, Sweden, August 1995.

[Caughey97] S. J. Caughey, D. B. Ingham, and M. C. Little, "Flexible Open Caching for the Web," Proceedings of the 6th International World Wide Web Conference, Santa Clara, California, USA, April 1997. <URL:http://proceedings.www6conf.org/HyperNews/get/PAPER159.html>

[Ingham95] D. B. Ingham, M. C. Little, S. J. Caughey, and S. K. Shrivastava, "W3Objects: Bringing Object-Oriented Technology To The Web," The Web Journal, 1(1), pp. 89-105, Proceedings of the 4th International World Wide Web Conference, Boston, USA, December 1995.
Available at <URL:http://www.w3.org/pub/Conferences/WWW4/Papers2/141/>

[Ingham96] D. B. Ingham, S. J. Caughey, and M. C. Little, "Fixing the Broken-Link Problem: The W3Objects Approach," Computer Networks and ISDN Systems, 28(7-11), pp. 1255-1268, Proceedings of the 5th International World Wide Web Conference, Paris, France, May 1996.
Available at <URL:http://www5conf.inria.fr/fich_html/papers/P32/Overview.html>

[Ingham97] D. B. Ingham, S. J. Caughey, and M. C. Little, "Supporting Highly Manageable Web Services," Proceedings of the 6th International World Wide Web Conference, Santa Clara, California, USA, April 1997. <URL:http://proceedings.www6conf.org/HyperNews/get/PAPER27.html>

[Little97] M. C. Little, S. K. Shrivastava, S. J. Caughey, and D. B. Ingham, "Constructing Reliable Web Applications Using Atomic Actions," Proceedings of the 6th International World Wide Web Conference, Santa Clara, California, USA, April 1997.
<URL:http://proceedings.www6conf.org/HyperNews/get/PAPER12.html>