Supporting Highly Manageable Web Services

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

Abstract

This paper focuses on the management aspects of Web service provision. We argue that support for manageability has to be considered at the design stage if services are to be capable of delivering high levels of quality of service for their users. Examples of the problems caused by lack of manageability include maintenance operations that necessitate service downtime, or difficulties in ensuring consistency of information. We categorise management issues into those concerning a site as a whole and those pertaining to individual services. Our approach to site management supports the arbitrary distribution of services to machines, allowing the optimum cost/performance configuration to be selected. Services can be easily migrated between machines, resulting in sites that scale, both in terms of the number of services and the number of users. Service management issues may be generalised as supporting evolution, for example, supporting changes to the functionality, the presentation logic, and the overall look and feel of a service. Our approach, based on the separation of functionality and presentation, allows such changes to be performed on-line and ensures that updates are reflected consistency across the various pages of a service, or across services. This approach also facilitates the development of services that utilise dynamic content for service customisations, such as tailoring a service to match the profile of users. Furthermore, all management operations are available through Web-based interfaces, making them accessible to a broad range of users, not only specialist system administrators.

Keywords: web; management; object-oriented; dynamic content; Dublin core; metadata

1. Introduction

The Web continues to evolve from the initial role as a provider of read-only access to static documentation-based information, and is becoming a platform for supporting complex services. This evolution is being partially driven by commercial organisations that are beginning to use the Web for critical applications, such as customer support systems and electronic commerce, and not simply as a low-cost advertising medium.

These advanced services typically share a number of common functional requirements, such as support for session-based interactions, generation of dynamic content and controlled update of persistent system state. To facilitate their implementation a number of Web application toolkits have been produced, including our own W3Objects [Ingham95], which provide system support for these common features.

We believe, however, that providing support for the *construction* of Web services only addresses part of the problem. Of equal, if not more, importance is the *manageability* of such services. Manageability encompasses many issues, some relate to individual services while others are concerned with the collection of services that together constitute a site. Scalability is an important aspect of site management so as to allow a site to grow to support increased numbers of users and services. Service management is concerned with supporting change while preserving consistency, both in terms of information and its presentation. If services are to be able to deliver the high levels of quality of service demanded by users (customers), then manageability features have to be *designed in*. This is particularly important for commercial services, where the consequences of providing inconsistent information or suffering downtime for maintenance are often financial.

Another aspect of the changing Web is the increased use of dynamic content. Its use is becoming widespread, not only for creating responses to a client-driven applications, but throughout services, in a system-driven manner to support various customisation requirements. Examples include tailoring advertising to match the profile of the user, or customising a merchant's catalogue based upon current stock levels. Such systems require logic-based mechanisms to drive the customisations. The specification and maintenance of these logic components is another service management issue.

This paper describes an architecture to support manageable sites based on the use of distributed object technology. Our approach supports scalability by allowing arbitrary distribution of services to machines allowing sites to be configured for optimum performance. Furthermore, configurations can be modified at run-time without disruption.

To support manageability and customisation of services, we introduce a novel structuring technique, based on the separation of presentation and functional aspects. This technique supports on-line service maintenance, allowing changes to presentation and logic components to be performed. A particular feature is the ability to ensure the consistency of changes to replicated components, such as an author's contact details, within a service or across services.

A particular goal of this work is accessibility: traditionally service management was so complex a task that it was reserved for only specialist system administrators. By allowing all management operations to be performed via Web-based interfaces we aim to break down this *technology barrier*, making site and service management accessible to non-specialists.

The remainder of the paper begins with an overview of the W3Objects system that has been used as a testbed for our ideas. This is followed by a review of site management issues and how they are addressed by our architecture. The next section focuses on service management, describing the model, an implementation overview and several illustrations. An extended example is presented that demonstrates how the techniques can be used to support the incorporation and management of resource metadata. Finally, we present our concluding remarks and ideas for further work.

2. W3Objects Overview

Several research groups [Rees95, Merle96] and commercial organisations [ActiveX, WebObjects] have seen the benefits to be gained from the application of object-oriented techniques to various aspects of Web service provision. Our system, W3Objects, uses distributed object technology, to assist in the construction of advanced Web-based services [Ingham95, Ingham96]. In our model, Web resources are represented as objects, which are *encapsulated* resources possessing internal state and a well defined behavior, rather than the traditional file-based entities. The model supports abstraction since clients interact with W3Objects only through well-defined published interfaces. The objects themselves are responsible for managing their own state transitions and properties, in response to method invocations.

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.

Arbitrary classes of application object can be implemented using this framework, adding class specific operations. Desirable properties, such as persistence and concurrency control are obtained via the use of behavioural inheritance.

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] and caching [Caughey97].

2.1 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.

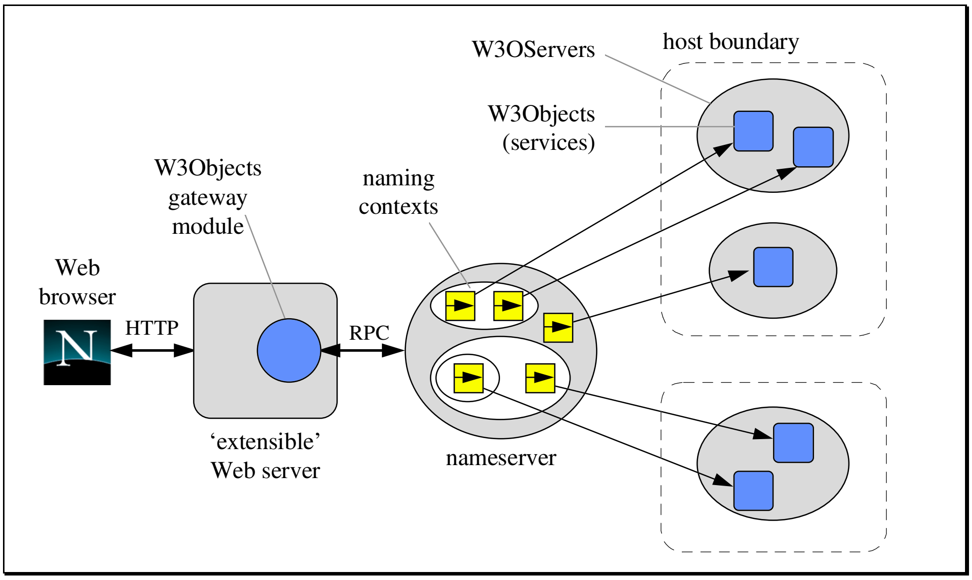

Figure 1: Architecture of a W3Objects site

3. Site Management

With the number of Web users increasing rapidly, site managers can expect the future to bring increased load on their services. Similarly, it is likely that new services will be added to a site over time. Furthermore, application-based services tend to be more computationally intensive than those that simply load and transmit data files. Together these factors raise the issue of scalability; in order to continue to provide the desired levels of quality of service, it is likely, that services will have to be partitioned across machines, with occasional reconfiguration of the partitioning as load patterns vary or new services are added.

Using traditional server technology, applications are implemented as a number of disparate components, including programs, HTML files, access control configuration, etc. Also, services typically share common components, such as images or access control information, thereby creating dependencies between services and between services and the server. In such conditions, migrating services can be a complex, error-prone operation. For example, hypertext links, contained either in HTML files or embedded in programs, may require updating, or changes to server configurations files may be required.

Due to the international nature of the Web, there is no appropriate time to bring down a service to perform maintenance operations, therefore, a service provider would ideally like to perform such tasks on-line, without the need for service downtime or major disruption.

The W3Objects architecture possesses a number of manageability features that help to address many of the above issues, namely:

- **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.

3.1 Comparison with Alternative Techniques

In addition to the manageability features mentioned, the W3Objects architecture has several advantages over both the traditional CGI-based mechanisms and the API-based schemes available with many of today's servers.

The mechanism of creating new processes to serve individual requests that is used in conventional CGI is highly inefficient and results in unnecessary load on the server machine, thereby reducing the observable performance of the server. It also makes supporting session-based services difficult, necessitating the use of bespoke disc-based mechanisms to store session state, typically incurring additional performance penalties. W3Objects services persist between requests and are accessed using RPC, resulting in high performance, and the inherent ability to support session-based services.

The API mechanisms supported by many modern Web servers allow arbitrary applications to be loaded directly into the Web server, thereby removing the need for CGI and its associated poor performance. Using this approach, a separate module is typically installed for each application. The installation of user-code within the Web server increases the probability of introducing bugs. A programming error in any of the installed modules can result in the whole server crashing thereby rendering all applications unavailable. The single gateway module used by W3Objects is a relatively simple piece of code that can be thoroughly tested. The effects of programming faults within a W3Object application are contained to the process supporting that particular application; other services will be unaffected.

Supporting session-based interaction using an API-based approach is complicated by the fact that most servers use a multi-process architecture, making it possible for several instances of an application to be executing in parallel. Since individual requests may be directed to any of the available server processes, external persistence mechanisms are required to synchronise session-state between application instances.

4. Service Management

A Web service may be thought of as consisting of both functional and presentation components. For example, the functional requirements of a search engine can be summarised as the ability to query a database based upon some search criteria. The presentation requirements include providing usage instructions, accepting search terms and returning results to the user. From a client's perspective, a service is simply seen as a collection of pages. The server creates these pages either by returning some static data (e.g., usage instructions) or by populating templates with data from programs to create dynamic content. In general, dynamic content may either be used as the response to a client-driven request (e.g., the results of a search request) or in a system-driven manner to support various presentation customisations, such as tailoring advertising to match user profiles. The presentational aspects of a service can therefore by sub-divided into static and dynamic components. Frequently services are implemented without a clear distinction between functionality and presentation. For example, templates may be embedded within programs or the computation concerned with the primary role of the service may be tied together with the computation concerned with presentation customisation.

Service management is primarily concerned with supporting changes to services as they evolve over time. Experience has shown that updates are often made to the look and feel, the logic that drives customised presentations, and the functionality. We argue that combining functional and presentation aspects of a service together complicates the task of performing change. Consider, as an example, the effects of modifying the contact details of a service administrator. It is likely that this information is replicated on every page of the service. Ensuring that all instances are consistency updated is a time-consuming and potentially error-prone activity. Furthermore, instances may exist in both static and dynamic components. Although a trivial change, it is likely that updating a presentation template within an application would require the skills of a programmer. There is also the danger of accidentally interfering with the functional aspects of the service. This is more of an issue when updates are required to the presentation logic of a service.

We aim to address these issues using a model for Web services that clearly separates the functional aspects of a service from its presentation. Our model supports the isolation of commonality, ensuring the consistency of updates to replicated components. The implementation allows changes to static and dynamic components to be performed on-line by non-programmers using familiar Web-based interfaces.

4.1 Manageable Web Service Model

Our model is based on a novel structuring technique, in which a service is logically represented as a single object which internally maintains a number of *view* objects that are assembled together to create the pages as seen by clients.

A view may correspond directly to a complete HTML page, or may represent a fragment, such as a navigation bar, that is used as a component within one or more pages. Views may either be static, as in a HTML fragment, or may interact with the environment in some fashion to generate a dynamic representation. Furthermore, views may either be private to a single resource or may be shared by multiple resources. Since views are themselves W3Objects, they may be distributed, making it possible for view objects to be remote from the services that utilise them. The example in Figure 2 illustrates the logical relationship between services, views and the pages seen by users.

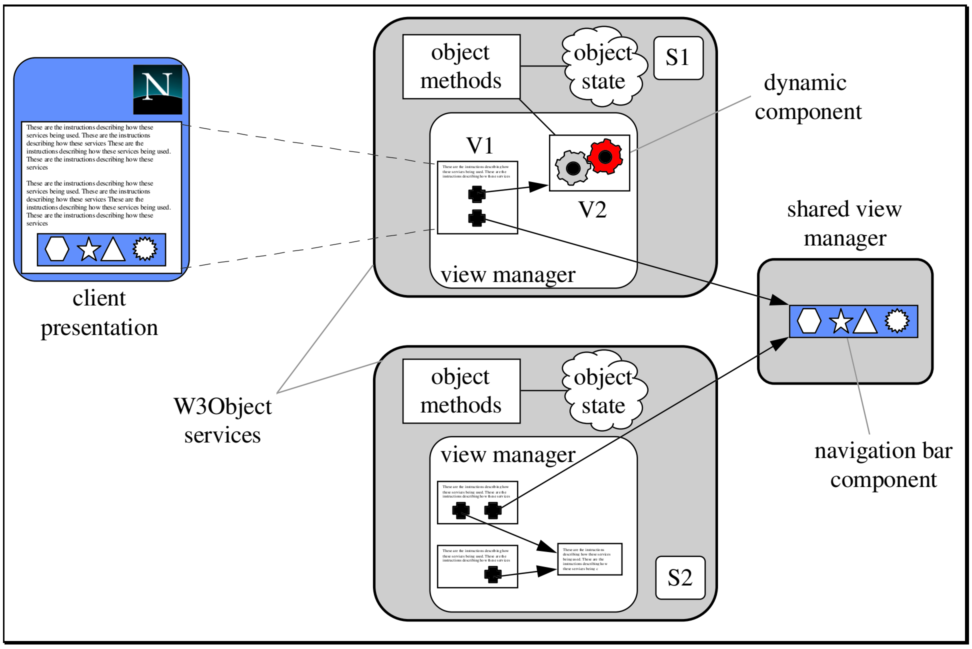

Figure 2: Relationship between client pages, services and views

Two services, S1 and S2, are shown that utilise both private and shared views. As a result of a request, a service makes an invocation on the appropriate view object. The object responds by creating a static representation of itself to be returned to the client. For example, a request on dynamic view, V2, may generate a fragment of HTML data based upon some interaction with the functional operations of the service. Views may also contain references to others, indicated by the arrows in the diagram. In response to a request, such views issue requests on the objects they include, assembling their responses as a static block of data. To illustrate this mechanism, the diagram shows the page as seen by the client as a result of a request on S1, V1. The client's representation is a complete HTML page, containing the results of V2's interaction with the service and the inclusion of the shared navigation bar object.

4.2 Manageability Features

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.

4.3 Implementation Details

The first step in creating a manageable service is the implementation of a user-defined class to support the functionality of the service. This stage is completed without consideration for how the service will by accessed or presented via the Web. Inheriting a user class from the Manageable class, provides an implementation of the HTTP interface, that is driven by the gateway module in the Web server. Furthermore, it provides a dedicated service-management interface that allows view objects to be created, edited, removed and migrated.

The Web interface to the class provides access to both service operations, i.e., those that are invoked by users of the service, as well as management operations, i.e., those that are typically accessed by service providers and managers. Internally, this interface is defined by two components, the *dispatcher*, that defines the set of accessible operations and the *view manager*, which holds the view components associated with the service. A logical representation of the internal structure of a manageable service is shown in Figure 3.

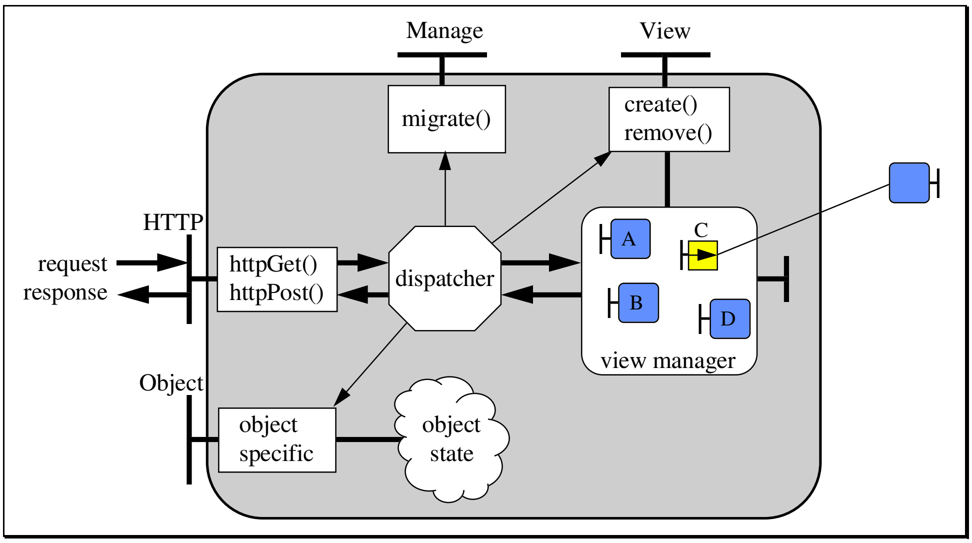

Figure 3: Internal structure of a manageable W3Object service

All Web-based requests arrive at the object via the HTTP interface and are forwarded to the dispatcher. Its role is to allow the arbitrary operations supported by a user class to be accessed via the generic HTTP interface. To act as a logical bootstrap mechanism, the dispatcher uses compiled *dispatch objects* to provide the Web interfaces for the programming level operations, e.g., management operations (illustrated by the light arrows in the diagram). In addition the dispatcher interacts with the view manager to map service requests into view object invocations. The diagram shows four view objects A, B, C, and D (C is a reference to a remote view.) In effect, each operation a service supports over its Web interface is implemented as a view object. As an example, consider the following HTTP request:

```
[HTTP headers]
GET /w3o/confReg?opcode=instructions HTTP/1.0
```

To serve the request, the dispatcher attempts to bind to the object whose name corresponds to the value of the opcode attribute in the request, "instructions", in this example. It then invokes a standard method on the view, passing the request object as a parameter. The view object is free to respond to the request as appropriate, eventually returning some data, for example, HTML code, to be passed back to the client; it may also modify the request object, inserting headers to be returned to the client, for example, to specify the MIME type of the data being returned.

4.4 Varieties of Views

View objects are standard W3Objects that inherit from the View class. Therefore, view classes may define new interfaces and can support arbitrary functionality. Typically, a view-specific management interface is provided that allows their functionality to be customised. View objects can be independently accessed via the Web and typically provide access to their management operations through this interface. Furthermore, views may also be manageable, in that their internal operation may too be constructed using views. The W3Objects library supports a number of generally useful views. One of the simplest is the *HTMLView*, which is used to hold HTML components.

4.4.1 HTMLView - an example view object

The HTMLView is a simple class that is capable of storing HTML data, complete pages or page fragments. In response to a request, such views simply return their internal HTML state. Their management interface supports three operations: read, write and load (allowing the internal HTML to be overwritten by new data from a file.) Their Web interface uses these operations to allow the object to be edited using a forms-based editor, as shown in Figure 4.

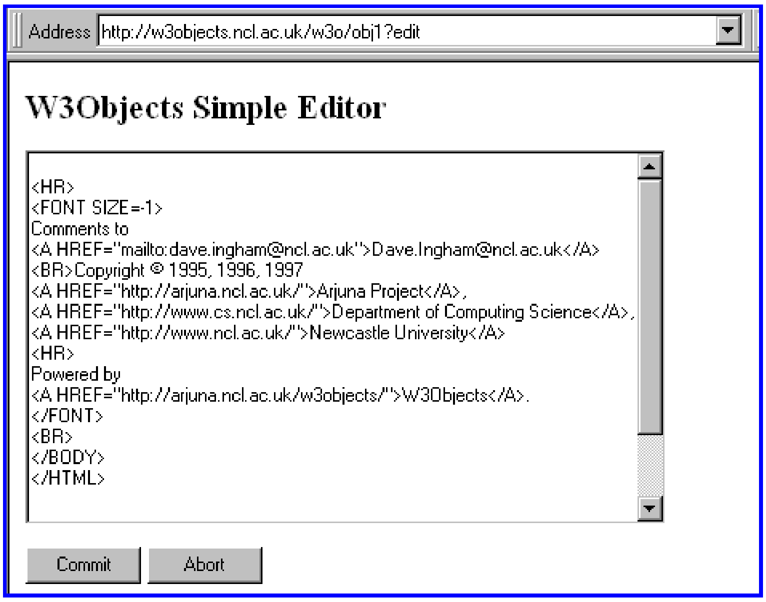

Figure 4: HTMLView Web-based management interface

Note that we also support object editing using the HTTP PUT method, that is supported by some browsers.

4.4.2 Providing maximum flexibility using scripted views

Scripted views allow the logic that drives dynamic components to be modified at run-time via a Web interface. Such objects are implemented using *W3OScript*, a server-side scripting language based on tcl [Ousterhout94]. W3OScript programs are executed by a safe tcl interpreter which only supports a sub-set of the standard tcl command set. W3OScript adds a number of additional W3Object-specific commands to the interpreter. The system is implemented using the Embedded Tk (ET) toolkit, which provides a convenient way of executing tcl applications within compiled C or C++ programs [ET]. W3OScript views are editable using the same Web-based interface as shown previously.

W3OScript views are able to access the functional interface of a service. In particular, this mechanism can be used to include other views as components. The native Tcl language provides a convenient way of expressing logic to support tailored dynamic views. Consider a simple example in which a service has a requirement to tailor a page, offering a different menu depending on whether a user is local or an outsider. Such a requirement can be implemented with a few lines of W3OScript as shown below:

```
<HTML>
<HEAD><TITLE>Welcome</TITLE></HEAD>
<BODY>
<H1>Welcome</H1>
<!--W3OSCRIPT>
switch -regexp [w3ovar REMOTE_HOST] {
ncl.ac.uk {set html [w3oinclude parent localMenu]}
default {set html [w3oinclude parent remoteMenu]}
}
return $html
</W3OSCRIPT-->
</BODY></HTML>
```

4.4.3 Supporting session-based interactions

Many applications require the ability to create session-based interactions between the client and the service. Implementing such services requires mechanisms to maintain session state between requests. This state can either be held by the client or at the server. A W3OScript application can use either or both techniques as required. Client-side state maintenance is achieved by encoding session-state in the forms returned to clients. Holding session state at the server is achieved by using client-held cookies to index into server-held data. W3OScript provides a convenient programming abstraction, known as the *cookie jar*, to shield users from the complexity of implementing such services. W3OScript objects place data in the cookie jar using attribute-value pairs and in subsequent requests can retrieve the data by specifying the attribute name.

4.5 Service Management Example - managing resource metadata

In common with many service providers, Netskills, a UK based project, providing network service training to academics, are seeking to incorporate Dublin core metadata [Dublin, Miller96] into their resources so as to facilitate indexing and searching. This process is a time-consuming and potentially error-prone activity requiring many resources to be edited.

Analysing the problem, it is observed that many resources share common metadata elements, for example, all of an author's resources contain the same author-based information and similarly authors belonging to the same organisation share the same organisation-based information. This raises the issue of maintaining consistency, for example, if an author's contact details change, we would ideally like to make the change in a single place and have it reflected in all resources. Another issue is the complex syntax of metadata; tool support is ideally required to facilitate its creation removing the need for hand-crafting. Furthermore, the metadata standards are in a state of flux, therefore, it is likely, that the syntax used to embed metadata in HTML resources will change.

To illustrate the flexibility of our approach for service management, a W3Objects-based implementation was designed to support this task. Firstly, a dedicated metadata view class was defined as a specialisation of HTMLView. The Web-based management interface provides a form-based metadata editor, allowing metadata to be created simply by filling in boxes, as shown in Figure 5.

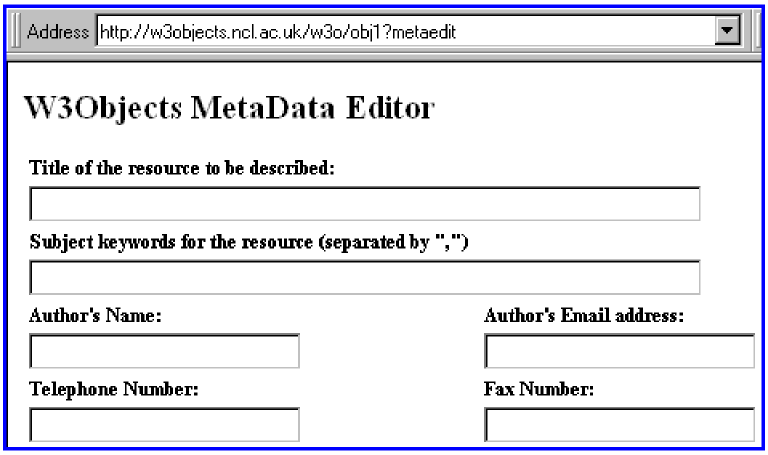

Figure 5: Web-based management interface for metadata views

The data is maintained internally within the view object in a structured fashion. W3OScript is used to specify the logic rules that map this structured information in to the desired format for embedding in a HTML resource. This approach has two advantages, firstly, if the standards for embedding metadata are changed, then a single change to the presentation logic is all that is required. Secondly, the metadata can also be presented in a user-friendly form if required, for example, through an additional operation on the object.

A hierarchy of view objects was then created, reflecting the commonality inherent in the metadata. A particular page object includes within its "HEAD" block a private metadata view, which contains the metadata elements unique to the resource. This view includes the shared metadata view of its author, which, in turn, includes the organisation metadata view. When a resource is accessed the various components are pulled together to create the page as seen by the client. This approach removes replicated information, allowing changes to be made at a single place which are reflected in all resources that use it.

4.6 Comparison with Alternative Techniques

Supporting changes to look and feel can be partially achieved using style sheets [Lie96], which allow HTML formatting commands, such as heading colours, to be stored separately from the resources that use them. Style sheets can therefore be shared by multiple resources, helping to ensure the consistent look and feel of a set of pages. Our approach does not conflict with the use of style sheets, in fact their use is encouraged (style sheets can be implemented as shared views objects.) However, style sheets do not assist in all look and feel changes, for example, the aforementioned problems of updating replicated items of information, such as an author's contact details cannot be addressed using style sheets.

An alternative approach for supporting dynamic content is through the use of server-side include (SSI) techniques. Special server-parsed HTML files are used to define templates that may contain embedded requests to CGI applications. When the server receives a request for such a resource, the template is loaded and processed, executing the embedded CGI requests and entering the results into the template before it is returned to the client. This approach provides greater flexibility than embedding templates within programs since they can be modified without editing application code. The disadvantage of SSI is its poor performance as the service must load and parse the template file for each request. Our approach improves on this by pre-loading objects and pre-parsing where possible. The resulting system performs well; overheads include executing interpreted W3OScript objects and the occasional RPCs used to communicate with remote objects (we are investigating ways of reducing this cost, as discussed in the conclusions.)

5. Conclusions and Further Work

Manageability is becoming an increasingly important aspect of Web-based systems, especially for commercial services where quality of service is understood to be an important issue. This paper has shown how distributed object technology can be used to build manageable Web sites that scale according to the increased number of users and services that they are required to support.

Orthogonal mechanisms to address the issues of service management illustrate the advantages to be gained from separating the presentation from the functional aspects of a service. Services constructed using the class library provided are managable by virtue of the ability to add, remove and modify the service's set of operations at run-time. The problem of preserving the consistency of replicated information in the face of updates is addressed by isolating changes to a single location, with the assurance that all occurrences will reflect the changes. The use of scripted resources has the advantage of allowing functional components to be modified on-line. This is especially useful for generating dynamic content for customisation purposes, for example, to tailor the presentation of a service to match user profiles.

A key feature of the work has been improving the accessibility of site and service management. Allowing all management operations to be performed via Web-based interfaces helps to make such tasks accessible to a broad-range of users, not just specialist system administrators.

One aspect of our planned future work is concerned with further improving performance, in particular, we aim to deploy internal, inter-object, caching mechanisms to minimise inter-object communication [Caughey97]. Additionally, we are aiming to further improve the flexibility offered to service managers. One specific aim is to provide the ability to group together multiple management operations, involving numerous resources, as a single operation, so as to prevent users from seeing intermediate stages. We are currently investigating the use of atomic actions to support this functionality [Little97].

6. Acknowledgments

The work reported here has been partially funded by grants from the Engineering and Physical Sciences Research Council (EPSRC) (Grant Number GR/K34863), Hewlett-Packard Laboratories and GEC-Plessey Telecommunications.

Thanks also go to the Netskills team, in particular to Brian Kelly (now at UKOLN as UK Web Focus Officer), whose experience in managing a large Web site provided valuable insight into current management issues, thereby contributing to the goals of this work.

7. References

[ActiveX] The ActiveX Home Page, Microsoft Inc.
See <URL:http://www.microsoft.com/activex/>

[Apache] The Apache Project Home Page.
See <URL:http://www.apache.org>

[Caughey97] S. J. Caughey, D. B. Ingham, and M. C. Little, "Flexible Cache Consistency for the Web," Proceedings of the 6th International World Wide Web Conference, Santa Clara, California, USA, April 1997.

[Dublin] The Dublin Core Metadata Element Set Home Page.
See <URL:http://purl.org/metadata/dublin_core>

[ET] The Embedded Tk Home Page.
See <URL:http://users.vnet.net/drh/ET.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/>
or <URL:http://arjuna.ncl.ac.uk/w3objects/papers/www4/Overview.html>

[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>
or <URL:http://arjuna.ncl.ac.uk/w3objects/papers/www5/Overview.html>

[Lie96] H. W. Lie and B. Bos, "Cascading Style Sheets, level 1," W3C Proposed Recommendation, November 1996.
Available at <URL:http://www.w3.org/pub/WWW/TR/PR-CSS1>

[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.

[Merle96] P. Merle, C. Gransart, and J. Geib, "CorbaWeb: A Generic Object Navigator," Computer Networks and ISDN Systems, 28(7-11), Proceedings of the 5th International World Wide Web Conference, Paris, France, May 1996.
Available at <URL:http://www5conf.inria.fr/fich_html/papers/P33/Overview.html>

[Miller96] P. Miller, "Metadata for the Masses," Ariadne (The Web Version), Issue 5, ISSN: 1361-3200, September 1996.
Available at <URL:http://www.ukoln.ac.uk/ariadne/issue5/metadata-masses/>

[Netskills] The Netskills Project Home Page.
See <URL:http://www.netskills.ac.uk/>

[Ousterhout94] J. K. Ousterhout, "Tcl and the Tk Toolkit," Addison-Wesley, 1994.

[Rees95] O. Rees et al., "A Web of Distributed Objects," The Web Journal, 1(1), Proceedings of the 4th International World Wide Web Conference, Boston, USA, December 1995.
Available at <URL:http://www.w3.org/pub/Conferences/WWW4/Papers/85/>

[WebObjects] The WebObjects Home Page, NeXT Computer Inc.
See <URL:http://www.next.com/WebObjects/>

ERROR: undefined
OFFENDING COMMAND: F0F

STACK: