**Poseidon House**
**Castle Park**
**Cambridge CB3 0RD**
**United Kingdom**

TELEPHONE: **Cambridge (01223) 515010**
INTERNATIONAL: **+44 1223 515010**
FAX: **+44 1223 359779**
E-MAIL: **apm@ansa.co.uk**

**ANSA Phase III**

# Object Wrapping (for WWW) - The Key to Integrated Services? (Full Paper)

**Nigel Edwards**

**Abstract**

In the world of information services heterogeneity is inevitable: not only because of the huge amount of legacy data which exists on a wide variety of different platforms in different formats; but also because there is unlikely ever to be a format, protocol or platform which is suitable for all applications.

This paper explores how heterogeneity is managed within WWW (the World-Wide Web) and CORBA (Common Object Request Broker Architecture). It shows how the concept of object wrapping is useful for incorporating external information services and providing an integrated view of that information. In doing this work we are developing a programming model for WWW which is very close to that of CORBA.

IDLs (interface definition languages) are useful notations for describing existing (or legacy) information services. Encapsulation and integration of these services is made much simpler if the middleware (or ORB) makes it easy to plug new protocols into it and can support multiple, parallel protocol stacks: then the ORB can use the information service's own protocol. In addition such an ORB could make servers accessible via multiple protocols.

The paper begins by comparing the CORBA and WWW models, and goes on to show how object wrapping can be applied to both.

APM.1464.01 **Approved** 25th April 1995
External Paper

**Distribution:**
**Supersedes**:
**Superseded by**:

# 1 Object Wrapping (for WWW) - The key to Integrated Services?

In the world of information services heterogeneity is inevitable: not only because of the huge amount of legacy data which exists on a wide variety of different platforms in different formats; but also because there is unlikely ever to be a format, protocol or platform which is suitable for all applications.

This paper explores how heterogeneity is managed within WWW (the World-Wide Web) and CORBA (Common Object Request Broker Architecture) [OMG 93]. It shows how the concept of object wrapping is useful for incorporating external information services and providing an integrated view of that information. In doing this work we are developing a programming model for WWW which is very close to CORBA. We conclude with some recommendations about the kind of facilities middleware needs to incorporate to support easy integration of information services.

We begin by comparing the CORBA and WWW models, and go on to show how object wrapping can be applied to both.

## 1.1 Comparing CORBA & WWW

Both CORBA and WWW (using HTTP) are distributed object oriented systems: clients invoke methods (or operations) on server objects.

In CORBA the methods a service supports are defined in an Interface Definition Language (IDL). Objects communicate with each other using Object Request Brokers (ORBs) which provide a transport protocol for passing invocation requests and replies between objects. ORBs can support multiple transport protocols and abstract the programmer from many aspects of distribution (e.g. object location). In the future all CORBA compliant ORBs will be required to support the Internet Inter-ORB Protocol (IIOP) [OMG 95], so that ORBs from different vendors can interoperate.

Many WWW services use the Hypertext Transfer Protocol (HTTP) [BERNERS-LEE 95]. This defines both a transport protocol and a set of standard methods; two of the standard methods are mandatory for all “general purpose” HTTP servers (GET and HEAD). Servers are also free to implement their own “extension” methods.

CORBA defines no standard methods which all application services are required to support. However, it does define an environment for building clients and servers which hides most of the complexity of the underlying platform from the programmer. From the IDL description of a service, client and server stubs are generated which hide the Application Programmer’s Interface (API) of the underlying ORB from the application programmer. Hence remote invocation looks very similar to local invocation. This makes it relatively easy to write new clients and servers as well as making it easy to

extend existing implementations —something which is important to assist integrating new information services into systems.

In contrast it is harder to write clients and servers which use HTTP, because of the need to drive the protocol stack directly — there is no equivalent to client or server stubs. Despite this a number of very sophisticated WWW clients and servers have been implemented, although I know of no server which implements any extension methods.

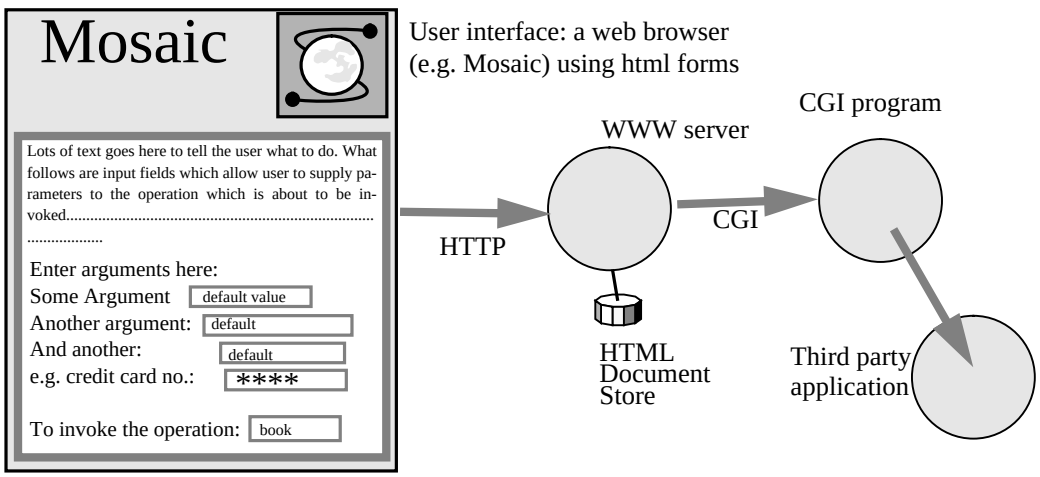
WWW uses Uniform Resource Locators (URLs) [BERNERS-LEE 94b] as handles to access resources. URLs are associated with a particular resource at a particular location. This can cause problems; for example if the object or resource has moved the access attempt will fail. In contrast CORBA uses opaque object references and relies on the underlying ORB to locate the remote object.

The WWW community have recognised the problems associated with URLs and are developing concepts which have much in common with CORBA's object references (e.g [SOLLINS 94]). The trick is to introduce a level of indirection between the client and server: instead of giving the client the location of the server, it is given a handle which can be resolved at runtime by the underlying infrastructure to locate the server. This is consistent with the principle that binding to a service should be done as late as possible.

## 1.2 Integrating third party information services

The distributed object, or CORBA, approach to integrating third party services (or legacy) applications is to write an object wrapper which encapsulates that service. The system designer writes a description of the service using IDL. The implementor must then write code which invokes the appropriate set of actions in the legacy application when one of the object wrapper's methods is invoked. This has proved extremely effective in a number of projects (e.g. [DRAHOTA 94]); a general model has been developed for the integration of databases [THOMAS 94].

**Figure 1.1: CGI in WWW[1]**




1. Mosaic is a trademark of the University of Illinois

Common Gateway Interface (CGI) [McCOOL] is a standard for external gateway programs to interface with HTTP servers. As shown in figure 1.1, this allows

WWW to encapsulate third party information services such as databases. CGI defines the format of the data stream between the WWW server and gateway, and also the environment variables available to the gateway.

Bespoke clients for these encapsulated information services are implemented by HTML (Hypertext Markup Language) “Forms” [CONNOLLY]. HTML Forms can be thought of as primitive agent technology: the client program (in the form of an active document or “fill-out” form) migrates to the WWW browser, asks the user for parameters and sends these back to the HTTP server.

CGI programs are not only used as gateways; they are also used without invoking external services, to extend the functionality of a server and to implement some HTTP methods (such as DELETE).

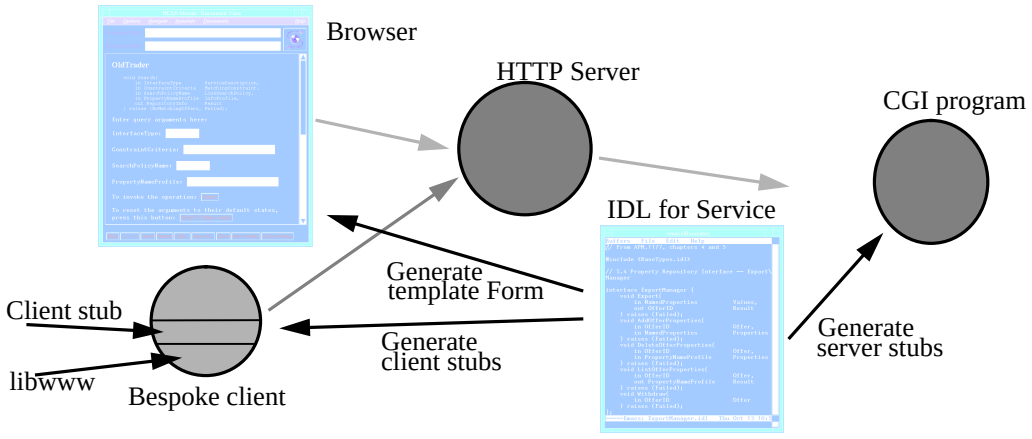
In the next two sections we explore two complementary uses of object wrapping: object wrapping for CGI and object wrapping for HTTP.

## 1.3 Object wrapping for CGI: a stub compiler

There are few tools to assist CGI programmers: they must explicitly parse the incoming data stream to extract parameters. This is analogous to CORBA programmers having to write their own unmarshaling routines for servers.

To alleviate this, we have developed a stub compiler for CGI. From a CORBA IDL description of the CGI program it generates a server stub, a client stub and a template HTML form (see figure 1.2). Currently all CORBA IDL primitives and data types are supported, except the “ANY” data type.

**Figure 1.2: CGI CORBA IDL Compiler**




Server stubs make the job of the CGI programmer easier by unmarshaling the incoming data stream.

Client stubs encapsulate libwww (the underlying library used by many WWW applications) in much the same way as stubs encapsulate the CORBA programmer from the API of the underlying ORB. Consequently, it becomes very easy to write bespoke client applications for CGI services: we have written two line C programs which are CGI clients.

Template HTML forms are supported for the simplest CORBA data types (e.g. strings and longs). Template forms generated by the stub compiler are guaranteed to produce a parameter stream consistent with the CGI server

stub generated from the same IDL definition. This helps to ease the problem of keeping an HTML form consistent with its corresponding CGI program. Template forms need to be annotated with text to explain the input fields to humans. As noted above, HTML forms can be thought of as primitive agents, so the stub compiler is generating a template agent.[1]

The programming model provided by the stub compiler is very close to the CORBA programming model. The main difference is that CGI programs are stateless: they are forked for each invocation by the HTTP server. This means that any state change made by the client has to be stored externally (e.g. on the local file system).

Further details of the stub compiler are given in our programmer's guide [EDWARDS 95].

## 1.4 Object wrapping for HTTP

Currently we are exploring object wrapping for HTTP. Specifically we are writing a mapping of HTTP to CORBA IDL.

This will allow us to apply our stub compiler technology to build an HTTP server which supports extension methods and WWW clients which can use these extension methods. The aim is to provide a distributed object programming model for WWW similar to that of CORBA, making it easy to encapsulate legacy information services within WWW.

This IDL mapping will also make it very easy to plug an implementation of the HTTP protocol into an ORB, giving CORBA programmers the ability to encapsulate WWW inside CORBA. This kind of encapsulation requires an ORB which supports multiple, parallel protocol stacks. Given such an ORB it will be possible to write CORBA clients which access HTTP servers and also to write CORBA HTTP servers (supporting at least the GET and HEAD methods). Such servers might be accessible via multiple protocols, one of which will be HTTP.

In principle it is possible write an IDL description for other application protocols such as FTP. This is slightly less interesting, because unlike HTTP, most of these protocols are not designed to be extensible.

## 1.5 An application scenario

Suppose the management of a theatre wish to make program information available on-line using WWW. As well as browsing program information, customers should also be able to make bookings on-line.

Specific requirements might be to allow customers to engage in some kind of conversation with the server in which they select preferences for dates and seats. Furthermore it should be possible to book seats for different events on different dates, making a single payment at the end of the interaction.

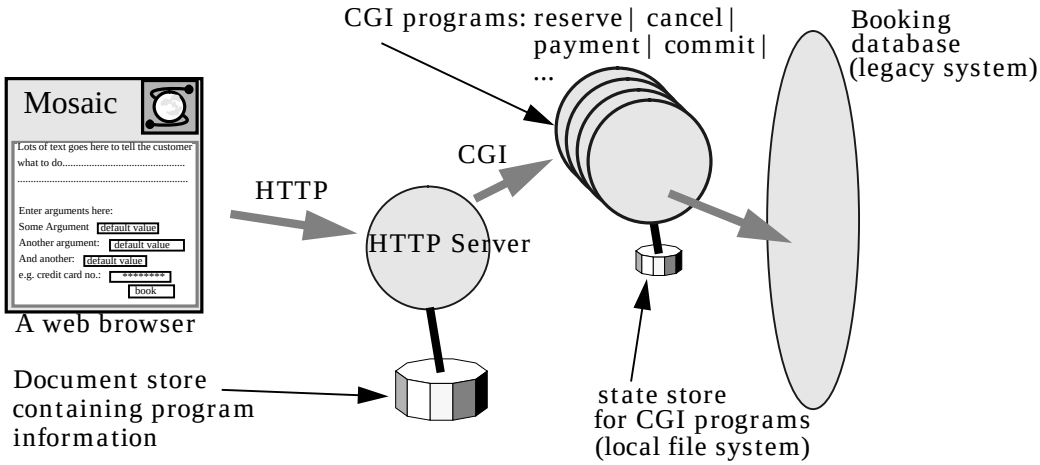
This requires the server to store state concerning the customer and to be able to undo any changes should the customer decide to abort the interaction. Furthermore any customer's booking needs to be atomic: either both booking and payment occur or neither do.

1. The question of tool support and abstractions for writing agents is a topic of ongoing research within the ANSA project.

Consider how this might be implemented today using WWW technology.[2] The technology to allow customers to browse program information is well understood and works well: it would just be a matter of building an (HTML) document store containing the latest program information.

Allowing customers to make on-line bookings is more difficult: typically it would be done by using CGI programs driving a legacy booking database (we are assuming that most theatres have such a system). These CGI programs would be accessed by using HTML form technology. The architecture of the system is shown in figure 1.3.

**Figure 1.3: A "commercial" WWW application**




CGI programs would implement operations allowing customers to make interactive seat and performance selections. To make a booking, customers would normally engage in a conversation with the server making multiple method invocations. This means that the CGI programs will have to store temporary state between method invocations; this state must be stored externally because CGI programs are re-forked each time they are invoked.

From an IDL description of the services offered by the CGI programs, the CGI stub compiler described in §1.3 will generate:

- A server stub which unmarshals parameters from the incoming CGI data stream (these are used by the CGI program)
- A template HTML form which acts as a client of the CGI program
- A client stub which can be used to write a bespoke client program for the CGI program (not shown in figure 1.3, shown in figure 1.2)

The IDL compiler described for the extensible web server proposed in §1.4 would do a similar job, but lead to a different implementation to the one shown in figure 1.3. The main difference is that it would generate server stubs for the HTTP server itself. This means that the facilities to make a booking would be provided by extension methods rather than CGI scripts. Any temporary state associated with the customer could be stored inside the HTTP server itself rather than on the local file system.

Client stubs make it easy to write bespoke client applications. So rather than driving the CGI programs or HTTP extension methods using HTML forms, the

2. This description ignores security issues like authentication and confidentiality (e.g of credit card numbers): in practice these issues would also need to be addressed.

theatre could provide a client program designed specifically for the purpose. How does the customer obtain this client? It could be done by prior agreement: for example the customer might ftp the viewer (or client program). More attractively, it could be done by making the client an agent which migrates to the customer's machine when it was needed. Thus the only software the customer needs is a browser and a compute engine to support the agent; everything else could be bootstrapped from this. This is already happening in WWW with the Java and Hot Java technology [JAVA 95].

It is probable that any theatre would already have a (legacy) database which stored program information and customer bookings. This database would still be used by booking clerks to record bookings made by other means (e.g. customers calling the box office). Hence the CGI programs or extension HTTP methods need to drive the database: the act of a customer "committing" a booking would involve updating the database.

If the legacy database contained detailed information about programs, it should be possible to apply the techniques of object wrapping described in this paper. Program information could be generated directly from the database itself, rather than storing it as a separate HTML file system.

A facility which may also prove attractive to customers is allowing them to view video clips of the performance, or a preview of the movie, before making a booking. Again this requires customers have the right client to view the video stream. Any agent technology used to solve this problem would need the ability to support (either stored or real time) multimedia streams. Currently it is possible to view video using WWW, but it relies on clients already having the right video viewer (e.g. Quicktime or MPEG players) on their machine.

## 1.6 Conclusions

Object wrapping is a powerful technique for integrating information services. It also enables extensions to existing functionality. This promotes innovation and competition by allowing competitors to offer different functionality.

By using (CORBA) stub compiler technology we have been able to make object wrapping technology available to WWW CGI applications and, in so doing, have a CGI programming model which is close to that of CORBA.

The same idea can be applied to HTTP itself: if a reasonable IDL description of HTTP can be obtained, a stub compiler for HTTP can be built. This will make it very easy for HTTP server writers to provide extension methods. These extension methods could be used as an alternative to CGI to encapsulate other information services in WWW.

A similar effect would be achieved by plugging an implementation of HTTP into an ORB: this requires obtaining a reasonably tractable mapping of HTTP to CORBA IDL.

IDLs have proved useful notations for describing existing (or legacy) information services. Object wrapping relies on encapsulating these information services inside an object which is either resident in the same process or acting as a gateway. The gateway approach is made much simpler if the middleware (or ORB) makes it easy to plug new protocols into it and can support multiple, parallel protocol stacks: the ORB can then use the information service's own protocol (e.g. HTTP).

A major factor in the success of WWW has been its browsers, such as Mosaic [NCSA 94], which successfully integrate a number of information services and protocols (e.g. HTTP, WAIS, FTP, Gopher and telnet). The above is arguing for middleware in which it is extremely easy to add new protocols. This will make it much easier to write integrating applications like Mosaic.

Enhancing the functionality of existing services and integrating new information services into a system is only useful if users can be provided with clients which make new features available to them. In a large information system such as the Internet, it is unlikely that users will have all possible clients for services they might use, so some kind of agent or migration technology would be useful for making new clients available. As explained in §1.2, HTML forms do have some of the characteristics of agents. Other technologies such as Java [JAVA 95], Tcl [OUSTERHOUT 94] or Telescript [RAYL 93] offer the promise of much richer client functionality. However, they introduce a number of issues, including security, discussion of which is beyond the scope of this paper. When these issues are solved, it is likely that these richer agent technologies will be extremely important in any kind of middleware for the network. The question of how these technologies relate to ORBs is worthy of further research.

## 1.7 Acknowledgements

The author would like to acknowledge the contributions of all members of the ANSA team, past and present, towards the ideas discussed in this report. Comments and discussions with Andrew Herbert, Rob van der Linden, Mark Madsen, Ashley McClenaghan and Owen Rees were particularly helpful.

# References


[BERNERS-LEE 95]

T. Berners-Lee, R.T. Fielding, H. Frystyk Nielsen, “Hypertext Transfer Protocol - HTTP/1.0”, Internet-Draft (work in progress), March 1995, <URL:ftp://nic.nordu.net/internet-drafts/draft-ietf-http-v10-spec-00.ps>.

[BERNERS-LEE 94b]

T. Berners-Lee, L. Masinter & M. McCahill, “Uniform Resource Locators”, Internet-Draft (work in progress), <URL:ftp://nic.nordu.net/internet-drafts/draft-ietf-uri-url-08.txt>.

[CONNOLLY]

Daniel Connolly, “HyperText Markup Language (HTML)”, <URL:http://www.w3.org/hypertext/WWW/MarkUp/MarkUp.html>.

[DRAHOTA 94]

A. Drahota, D. Hutcheson, “DAIS and its use at Hydro-Electric”, ANSAworks 94, Cambridge, UK, April 1994.

[EDWARDS 95]

N.J. Edwards, “A Stub Compiler for CGI: The Programmer’s Guide”, Technical Report APM.1465, APM Ltd., Cambridge UK, April, 1995.

[JAVA 95]

“Hot Java Home Page”, Sun Microsystems, 1995, <URL:http://java.sun.com/>.

[McCOOL]

Rob McCool, “The Common Gateway Interface”, <URL:http://hoohoo.ncsa.uiuc.edu/cgi/overview.html>.

[NCSA 94]

“NCSA Mosaic for X Documentation”, National Center for Supercomputing Applications, USA, December 1994 <URL:http://www.ncsa.uiuc.edu/SDG/Software/XMosaic/mosaic-docs.html>.

[OMG 95]

“CORBA 2 Interoperability Draft”, OMG Document Number 95-3-10, March 1995. (Available by anonymous ftp from ftp.omg.org, file: pub/docs/95-3-10.ps).

[OMG 93]

“The Common Object Request Broker: Architecture and Specification”, Revision 1.2, Draft, December 1993, OMG Document Number 93-12-43. (Available by anonymous ftp from ftp.omg.org, file: pub/docs/93-12-14.ps.z).

[OUSTERHOUT 94]

John S. Ousterhout, “Tcl and the TK Toolkit”, Addison-Wesley, 1994.

[RAYL 93]

Susan Rayl, “Telescript Programming”, General Magic Inc., 2465 Latham Street, Suite 100, Mountain View, CA 94040, USA, April 1993.

[SOLLINS 94]

K. Sollins, L. Masinter, “Functional Requirements for Uniform Resource Names”, Internet-Draft (work in progress), <URL:ftp://nic.nordu.net/internet-drafts/draft-ietf-uri-urn-req-01.txt>.

[THOMAS 94]

Gomer Thomas, Rob van der Linden, “Remote Database Queries in Open Distributed Systems”, APM.1138, APM Ltd., Cambridge, U.K., June 1994.