A Web of Distributed Objects

Owen Rees
Nigel Edwards
Mark Madsen
Mike Beasley
Ashley McClenaghan

Abstract:

This paper describes work on interoperability between the World Wide Web and Distributed Object systems being carried out under the ANSA workprogramme, using the Object Management Group's CORBA architecture. The approach described offers the opportunity to bring the benefits of distributed object technology to the Web by using the CORBA infrastructure. This also offers a way to solve the current engineering problems by using available industry standard technology.

Keywords:

Objects, CORBA, evolution, protocols, IDL, HTTP, interoperability

Introduction

This paper describes work on interoperability between the World Wide Web and Distributed Object systems being carried out under the ANSA workprogramme. The primary objective of this work is to bring the added benefits of distributed object technology to the Web, without losing any of the features of the Web that have made it so successful.

We have been motivated in our work by the following observations:

- Current Web infrastructure is suffering from a number of early engineering decisions that have not scaled to cope with the demand.
- Web clients and servers are tending to become larger monolithic applications; a conventional library does not provide the necessary flexibility for change, optimization for performance, and scaling down for small devices (e.g. consumer appliances).
- The Web is document-centric, and provides little support for connecting interactive services, or for interacting with "intelligent content".

These have to be balanced against the success of the Web to date. Our ambition is to develop an evolutionary strategy which overcomes these limitations without imposing new restrictions.

The strategy for achieving this objective is to enable seamless interoperability between the World Wide Web and Distributed Object systems based on the Object Management Group's Common Object Request Broker Architecture (CORBA) [1]. Interoperability will make CORBA-based systems

accessible to Web browsers, and make the information resources on the Web accessible to CORBA-based systems. Since CORBA is the leading industry standard for systems integration, it provides a pathway for linking the Web to a wide range of electronic information and business services.

Some of the benefits of using distributed objects are:

- Ease of programming: the underlying object model, the Interface Definition Language (IDL), and the supporting tools combine to simplify the task of writing distributed applications.
- Extensibility and manageability: systems built from interacting objects are inherently extensible since objects can be easily added or replaced. Extensibility allows customised management interfaces to be added to the system.
- Encapsulation and systems integration: distributed object technology has proved very effective in encapsulating "legacy systems" as objects. These objects offer interfaces that can then be used to implement an integrated system.

The work described here not only makes systems built with this technology accessible, but also uses this technology to build the gateways that provide the required interoperability. This approach demonstrates both the tools and the migration path that make it possible to move the Web onto a CORBA infrastructure. Following this path will allow the Web to exploit the work on protocol engineering that has been done for CORBA platforms, and also the results of the work now in progress on real-time and multi-media protocols for future CORBA systems [2].

The principles that are guiding the work are:

- Evolution not revolution: the results of this work must fit with what already exists, and offer an opportunity to exploit new facilities without demanding an irrevocable commitment in advance.
- A usable version of the technology must be made freely available.
- Use existing work, and especially freely available technology in order to promote compatibility with other initiatives, and widespread uptake.

Comparing CORBA and WWW

Object Request Brokers (ORBs), as defined by the OMG CORBA [1] specification, provide the mechanisms by which objects transparently make requests and receive responses. HTTP is, according to the Internet-Draft of March 1995 [3], a generic stateless object oriented protocol, and based on a request/response paradigm. The similarities offer an opportunity to interoperate between CORBA based systems and the World Wide Web, and the challenge is to combine their strengths, rather than to compound their weaknesses.

Methods

In CORBA the methods a service supports are defined in an Interface Definition Language (IDL). Objects communicate with each other using ORBs which provide a transport protocol for passing invocation requests and replies between objects.

Many WWW services use HTTP. 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), and these, together with POST are supported, and actively used, on most of the currently deployed servers.

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 IDL stubs and server IDL skeletons 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.

The use of IDL and tools has proved very effective in the development of distributed systems, and in particular, in interfacing to so-called "legacy systems". The ability to define interface types, with tools to help build clients and servers for those types, is one of the main strengths of the CORBA approach.

Transport

ORBs can support multiple transport protocols and hide from the programmer, many aspects of distribution (e.g. object location). In the future all CORBA compliant ORBs will be required to include support for the General Inter-ORB Protocol (GIOP) [4] which specifies the message formats; and the Internet Inter-ORB Protocol (IIOP) which specifies how GIOP messages are exchanged using TCP/IP.

HTTP, as generally implemented, defines both message formats and mapping to TCP/IP. The draft specification for HTTP [3] attempts to separate these issues, but is constrained by the need to capture the existing practice.

One well known problem with HTTP is that a new TCP connection is opened for each request, and closed after the response has been delivered. This is an inefficient use of system and network resources and a source of delay. By contrast, GIOP is designed to allow a connection to be used for multiple requests, and also to allow overlapping requests; amortizing setup costs over many requests/replies. Several requests may be sent over the same connection without waiting for a reply, and the replies may be delivered in any order. The replies are matched to requests by the use of identifiers.

In a CORBA system, the ORB provides connection and session management to make best use of the available resources. The application need not be concerned with these matters, and does not need to be changed or reconfigured to adapt to different communication resources.

HTTP via CORBA

There are a number of possible approaches to carrying HTTP requests and responses over a CORBA-based system. The simplest is to carry the whole request or response as a sequence of octets without attempting to resolve its structure in any way. This would allow the connection to be re-used, but would do little to provide interoperability.

A more useful approach is to define an IDL mapping of HTTP that promotes interoperability. The

objective is to allow CORBA technology to be used to develop clients that can use resources from the Web, and servers that can provide resources to the Web. This means looking at HTTP as a set of methods with various arguments and results, and constructing gateways that convert between the HTTP representations and CORBA representations.

The IDL and stub compiler technology associated with CORBA means that it is possible to experiment with various mappings of HTTP, with a reasonable level of effort.

Strategy for Interoperability

A strategy for full interoperability must enable the benefits of the new technology, without losing the use of the old resources. Interoperability is achieved through gateways and an important test of the completeness of the interoperability is to be able to put together a pair of gateways that together behave like an HTTP proxy, but interact via a different protocol - IIOP. Figure 1 shows this situation.

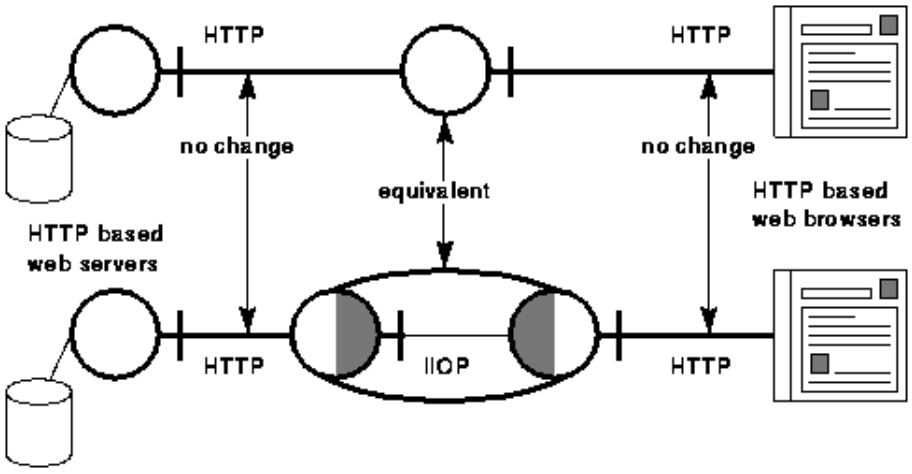

Figure 1. A pair of complementary HTTP-IIOP and IIOP-HTTP gateways appear to act like a proxy from the point of view of the client and server.

Existing clients can connect to the gateway by using their proxy mechanism. Most browsers have such a mechanism, and any other clients, such as proxies, that can use a proxy, can also use the gateways.

Once the gateways are available, it is then possible to provide clients and servers that use IIOP as their native protocol, and which can interwork with the existing Web through the gateways, as well as interacting amongst themselves, and with other CORBA objects.

First Stage: Gateways Exploit the Proxy Mechanism

The first stage of the evolution path is to implement basic versions of the gateways that convert HTTP interactions into the corresponding interactions described by the IDL. The two gateways are I2H which forwards IIOP requests to HTTP, and H2I which forwards HTTP requests to IIOP.

Where should the gateways be located?

By locating the I2H gateway near to the server, and the H2I gateway near to the client, the IIOP protocol can be used over much of the route. Where this part of the route accounts for a significant part of the round trip time, being able to re-use an existing connection should show an improvement. Figure 2 shows this arrangement.

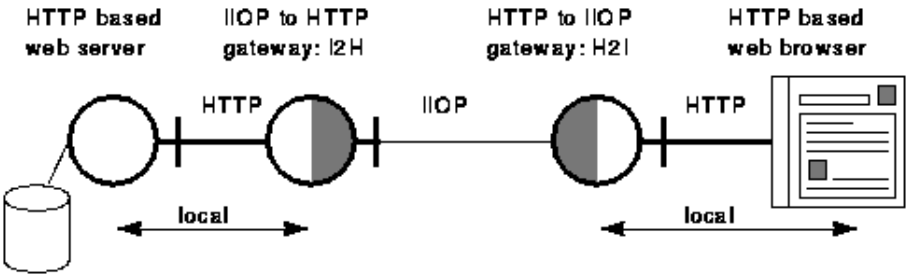

Figure 2. The collocation of the basic IIOP-HTTP and HTTP-IIOP gateways with the server and client respectively.

An important part of the work will be detailed performance testing, both to measure where the time is being spent, and how much difference it makes to go via the gateways over various kinds of connection.

Choosing the route

Most current Web clients have a mechanism that allows them to connect to a proxy, and this can be used to route requests to the H2I gateway. To retrieve resources from servers that do not have an I2H gateway, H2I has to be a full HTTP proxy as well as a gateway to IIOP. This presents H2I with the problem of knowing when to use IIOP, and where to route the requests.

This problem is handled in our scheme by a locator. The locator is defined as a separate service so that various strategies for deciding when to use IIOP can be explored. Figure 3 shows the locator service interface being used by the client-side gateway.

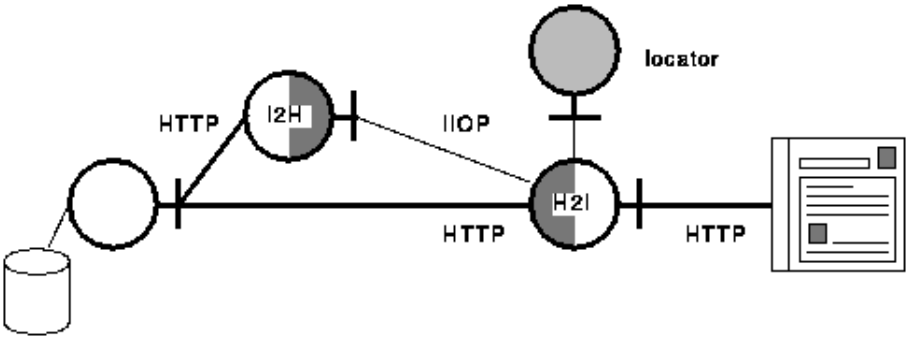

Figure 3. The locator service interface is used by the client-side gateway to obtain an interface reference for the appropriate server-side gateway, and the client-side gateway then makes an IIOP connection to the server-side gateway. If the locator cannot provide an interface reference to a server-side gateway, the client-side gateway passes the request straight through to the server via HTTP.

Defining the interface between H2I and the locator in IDL means that distributed object technology can be used to defer the decision about how the locator and H2I interact. H2I is written as if it is invoking the locator operations directly, and the locator is written as if it is being invoked directly. They can then either be linked into a single program, or linked to the generated stubs and skeletons so that they interact via the ORB. The ORB allows them to be located as close together or as far apart as you like.

Defining the locator as a distributed system object also means that H2I does not need to know whether it is a single object, or a collection of objects. The locator may consist of a number of objects located in various places, and interacting in order to ensure that the required data is available quickly whenever and wherever it is needed.

How does the locator get its data?

There are various options for gathering the data that the locator needs.

- HTTP servers with an IIOP gateway send an extra header in HTTP responses. This is ignored by existing clients, but passed to the locator by the H2I proxy/gateway. The locator is responsible for understanding this header and updating its database. The problem with this approach is that the HTTP server must be modified to send the extra header, which conflicts with the original objective of requiring no change to existing clients and servers.
- Attempt IIOP, fall back to HTTP if it fails. In this case, H2I must report the result to the locator, and in the event of failure, be told what to do next. The advantage is that there is no need to change the existing clients or servers, but at the expense of a failed attempt that will introduce delay. In this case, the locator must have a way to guess a likely IIOP route for the resource which will have been identified by a URL.
- Use a *trader*. The locator calls another service to find out if there is a suitable IIOP service. The ISO ODP concept of a *trading service* [5] can be applied here, once again exploiting the results of existing work. A trading service allows a client to find service offers that match a set of constraints on the values of properties. A service provider exports its offer to a trader by listing the properties of the service and their values. A client can then import a service from a trader by specifying constraints on the properties. In general, the trading service is provided by a number of traders that interact in order to make the offers available throughout the trading space.

These options are not exclusive. There must be some strategy for dealing with failed IIOP attempts however the IIOP route may have been chosen. The locator can take account of information passed back in responses, whatever strategy it chooses in the absence of data. The locator may keep the data itself, or it may pass it on to a trading service where other locators can benefit from its experience.

By defining the locator interface, the various strategies can be explored without having to change H2I. It will also be possible to have a variety of locators offering the same interface but implementing different strategies to suit different circumstances.

Second Stage: Native IIOP Web Server and Client

Once the gateways are available, clients and servers that use IIOP as their native protocol can be implemented. An IIOP client would not need to use the H2I gateway; an IIOP server would not need to

use the I2H gateway.

The IDL mapping of HTTP provides an opportunity to replace the textual headers with a more compact, and easier to process representation. The best way to represent the header information in IDL is being studied. This is another case where it is very useful to be able to experiment with various IDL mappings. Although a more compact representation will be beneficial where there is a bandwidth limit, the advantage of reduced processing time will not be realised until the HTTP stages of the transmission are eliminated.

IIOP native browser

The simplest way to implement a native IIOP browser will be to add an IIOP protocol module to the W3C Reference Library (libwww - previously known as the Common Code Library) in a browser that is built over that library. This will reuse much of the code already written for H2I, including the locator since the browser will have the same protocol selection requirement as H2I.

A disadvantage of this approach is that it contributes to the trend of browsers becoming larger and more complicated as more and more features are added to them. However, one of the advantages of distributed object technology is that it becomes easy to write "thin clients" that can run on small machines and interact with a server that implements what would normally be part of the application. An application composed of interacting objects can have those objects located on various machines so as to match the available processing, storage and communication resources. The idea of "thin clients" will become increasingly important due to the use of the Web by non-technical users, and the convergence of the Internet and consumer electronics.

One of the objectives of this work is to simplify the process of designing a modular browser that can take advantage of new resource discovery schemes, data formats, and anything else that may become available, simply by interacting with a new object. An example of this has been cited already: the locator is designed as a separate object. Whether to include the locator (locally) in the same address space as the browser, or to use the ORB to interact with it (remotely), can be decided at link time.

This technique of defining new services in IDL can be used to add additional features to the browser, deferring the decision of whether those features are implemented locally or remotely until link time. Using IDL also makes it easier to upgrade these "helper" services: a smarter locator using metadata could be used with no change to the client as the interface would remain unchanged (or perhaps inherit from the earlier version). This would make the deployment of URN/URC schemes [6] easier.

IIOP native server

One of the fundamental objectives of this work is backwards compatibility. A native IIOP server should be able to serve exactly the same file system as a current generation HTTP server using the "http://" scheme name. Thus exactly the same URLs can be used to access resources using HTTP or IIOP. Our architecture supports this by introducing a level of indirection in a browser or gateway attempting to use IIOP: given a URL of the form "http://...." the locator is consulted to see if the resource can be retrieved using IIOP. The native IIOP server will also provide support for CGI [7].

The simplest way of building a native IIOP server would to adapt an existing HTTP server by reusing the code written for I2H. This would make it easy to provide the same resources through both protocols.

Indeed, the server could deal with HTTP and IIOP requests on the same TCP port: the first four bytes in an IIOP request are the four characters "GIOP" [4], enabling the server to distinguish requests using IIOP. This would make the task of the locator easier.

The immediate consequence of building such a server is that the need to use a gateway is removed. Thus we benefit from IIOP's connection management (discussed earlier) which should improve performance. However, the main advantages of distributed object technology are extensibility and modularity.

This extensibility and modularity can be exploited to add a management interface to the server. Putting management operations into an interface that is separate from the normal service interface has been a valuable technique in distributed systems. Having a separate management interface avoids the problem of having to find a safe way to perform management through the interface seen by normal users, and also avoids the need to perform management operations through the file system of the host that is running the server, and other local and system specific facilities. The ability to define extra methods for a server can also be used as an alternative to CGI to build gateways to other applications (see later).

As an alternative to running both IIOP and HTTP servers, or constructing a server that supports both, the H2I gateway can be used to provide HTTP access to an IIOP-only server.

Implementation Experience

Prototypes of the gateways have been written, and work is continuing to implement improved gateways as well as the other components.

A simple IDL mapping of HTTP

The simple IDL mapping of HTTP which is used by the initial prototypes is shown below.

```
interface http {
/*
  HTTP headers are represented as name value pairs and
  encoded as strings, e.g. "accept text/html"
*/
  struct header {
    string name;
    string value;
  };
/*
  A request consists of a URL (encoded as a string) and a
  sequence of headers. This is sufficient for all methods
  except PUT and POST.
*/
  struct request {
    string URL;
    sequence< header > headers;
  };
/*
  PUT and POST require a body. So the request has an extra
  field: a sequence of octet. An octet is an 8 bit quantity
  which will not undergo any conversion when transmitted by
  the communication system.
```

```
*/
  struct request_body {
    string URL;
    sequence< header > headers;
    sequence< octet > body;
  };
/*
  Responses to all requests consist of the status code and
  its correpsponding reason string, a sequence of headers,
  and the body as a sequence of octets. For responses with no
  body, a sequence of length zero is sent.
*/
  struct response {
    short respcode; /*The HTTP status code e.g. "200"; OK*/
    string resptext; /*The HTTP Reason phrase*/
    sequence< header > headers;
    sequence< octet > body;
  };
  /* The standard HTTP methods*/
  response GET   (in request      request);
  response HEAD  (in request      request);
  response POST  (in request_body request);
  response PUT   (in request_body request);
  response DELETE(in request      request);
  response LINK  (in request      request);
  response UNLINK(in request      request);
};
```

This IDL is passed through a stub compiler to generate the header files, client stubs and server skeletons for the gateways. The stub compiler uses the IDL compiler front end made available by SunSoft Inc., through OMG, with a back end written for this project.

The simple mapping above has a number of deficiencies, including:

- It does not attempt to match applicable headers to methods
- It bundles everything up into one argument and one result

Once we have completed the implementation of the infrastructure described in this paper we plan to experiment with different, possibly more efficient mappings of HTTP. Once obvious possibility is a binary encoding of the common standard HTTP headers, using strings only for unusual and extension headers. Another is to deliver bodies through stream objects that support incremental and concurrent processing of the incoming data.

The I2H gateway provides the interface specified in the IDL. The first working prototype of I2H used the W3C Reference Library (libwww) version 3.1pre1, the Inter-ORB Engine written by SunSoft and published via OMG, and the headers and server skeletons generated from the IDL by the stub compiler. Figure 4 shows the architectural arrangement of these components in I2H.

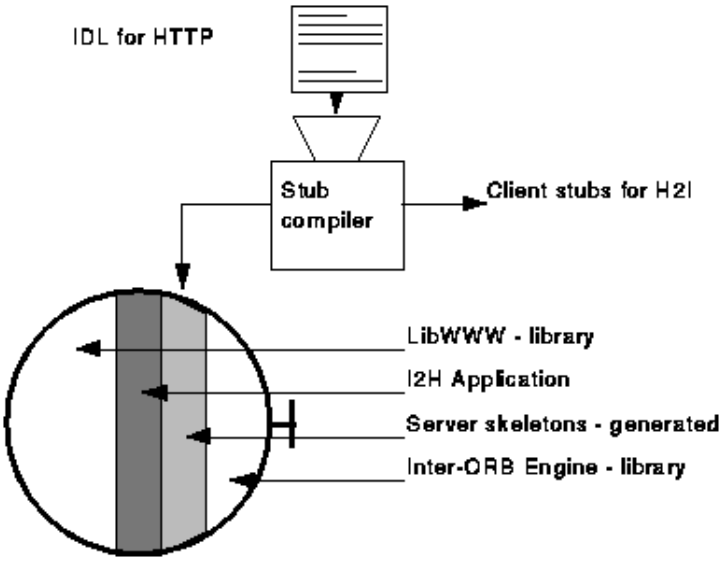

Figure 4. The architecture of the I2H gateway, showing how I2H binds the WWW Reference Library (LibWWW) components to the Inter-ORB Engine via the automatically generated server skeletons and a bespoke application layer.

Much of the work of processing the IIOP requests and responses is done by the generated skeletons, and the library functions called from the skeletons. Since the W3C Reference Library performs most of the work of accessing resources via HTTP (and other protocols), only a small amount of code is needed to connect these together.

The H2I gateway

The H2I gateway acts as a proxy to which existing Web clients, usually browsers, can connect. The first working prototype was a modified version of CERN HTTPD version 3.0, since most of the required mechanisms are already available. The first prototype had a fixed policy for using IIOP - all requests for HTTP URLs are sent via IIOP to an I2H gateway specified through an environment variable. This was sufficient to exercise the data paths, but was quickly replaced by a second prototype with a dynamic selection mechanism. Figure 5 shows the architecture of this second prototype for H2I.

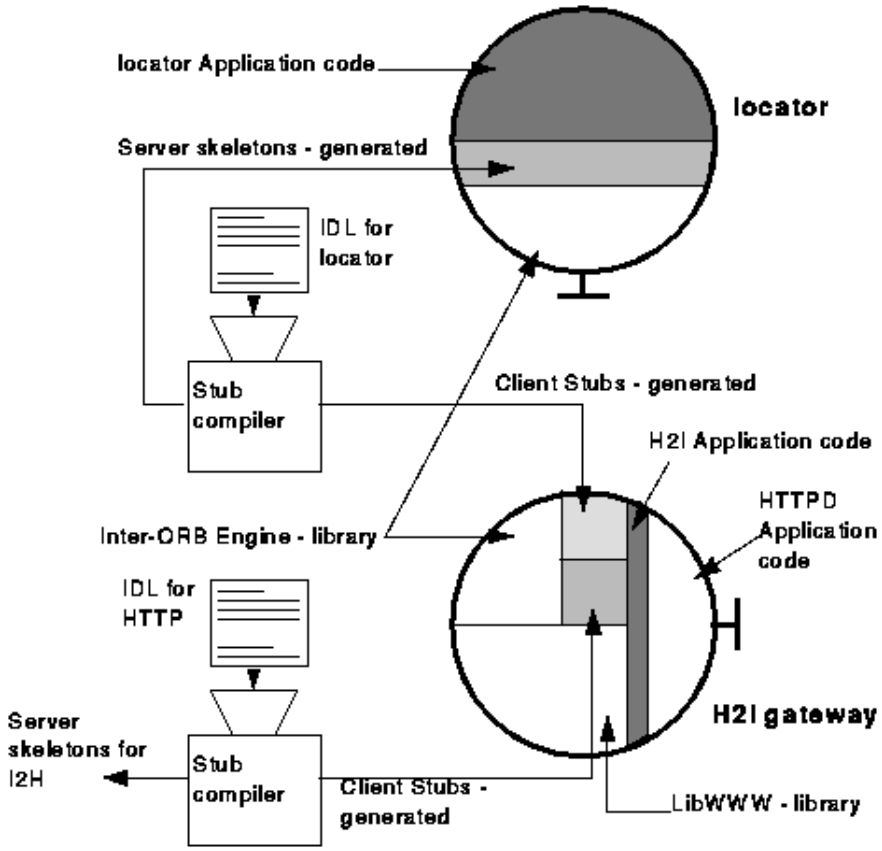

Figure 5. The architecture of the H2I gateway and the locator with which it interoperates. In addition to the component set used by I2H, H2I contains the client stubs derived from the IDL for the locator, and HTTPD-based application code. The locator has a simple layered server structure.

The H2I prototype is being used to explore the options for providing data to the locator. The first experiments use data delivered in extension HTTP headers that indicate the availability of an IIOP gateway. A modified CERN HTTPD is being used to send the extra HTTP header.

Stub Compiler for CGI

The initial approach of the project was to begin by making it easy to integrate distributed technology into the Web at the server side. The aim was to make distributed object systems rapidly available via the Web through HTML Forms.

One of the limitations of integrating legacy systems into the Web is the amount of time taken in creating the CGI [7] programs to access them and generating the forms to control those programs. This problem is dealt with by using an IDL to describe the service being integrated.

A stub compiler for CORBA IDL was developed which generates the necessary template forms for the front end, as well as the server skeletons for the CGI program which unmarshall the parameters. The advantages of this approach include:

- The entire process of stub generation is automatic, and in particular the unmarshalling of CGI parameters is automatically correct.
- The template HTML forms are syntactically correct HTML.
- The template HTML forms that are created require only a few minutes of editing to be made fully suitable for the required task (generally this requires only sizing of input fields and addition of explanatory text).

The functioning of the IDL stub compiler for CGI is shown in Figure 6.

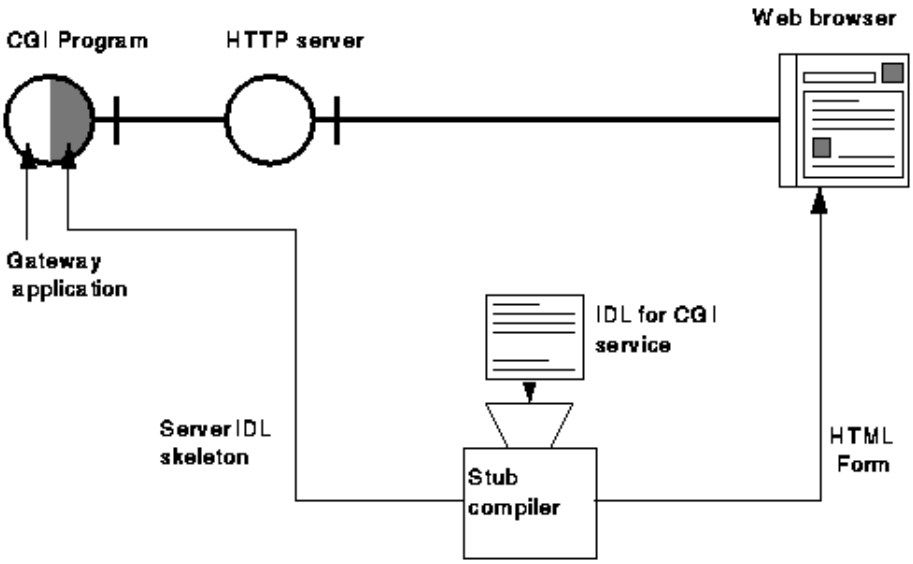

Figure 6. The generation of both the server-side IDL skeleton and the corresponding client-side HTML Form, from the IDL for the CGI program, by the stub compiler.

The principal limitation of the current implementation is that template forms do not take full advantage of HTML form functionality. Further work is needed to expand the range of types that can be mapped to form facilities, especially where there are choices of how to map a type to form facilities (e.g. an enumeration to radio buttons or a menu).

Another way of integrating existing systems into the Web is to define extension methods for a Web server which access other services. For an ORB or IIOP based server much of the work of adding the new methods to the server would be done by the stub compiler which would generate server skeletons for the new methods. HTML forms could be used to invoke the new methods. This approach may make the Web server stateful, since new methods cannot be guaranteed to support stateless interaction. Some may regard this as a disadvantage. In contrast the CGI approach keeps the Web server stateless. However, since a new CGI process is forked to deal with each request, any state required by the gateway must be stored externally (e.g. on the local file system).

The ANSAweb stub compiler for CGI is available by anonymous FTP from the ANSA FTP server <URL:ftp://ftp.ansa.co.uk/>. It is known to run on HP-UX and Solaris platforms. The package includes complete documentation and examples. A more detailed overview is given in reference [8].

Related Work

There are presently a number of projects aimed at putting objects into the WWW. Of these, the most closely related are the DCE-Web project, built around DCE, and the Web* project. These are described in further detail below.

DCE-Web and Wand

The OSF Research Institute has a number of projects which are applying the Distributed Computing Environment (DCE) to the Web. The most closely related to our work are the DCE-Web project [9] and its native DCE-Web server Wand [10].

In DCE-Web, HTTP protocol messages are packaged as DCE RPC data by the DCE-Web layer, a technique known as protocol tunnelling. Thus the HTTP protocol messages are communicated unchanged, minimising the changes required to the core protocol processing functions of existing browsers and servers.

Our approach is to provide a mapping of HTTP to CORBA IDL. Complete backwards compatibility is maintained by use of gateways and dynamic protocol switching. (We intend to explore a number of mappings with a view to improving efficiency e.g. removing the need to parse ASCII headers.)

The main benefits of the DCE approach are security and naming. The DCE security services are used to provide fine grained security and the DCE naming service can be used to name Web objects. This is complementary to our work, which emphasizes the benefits of extensibility and modularity inherent in distributed objects. Thus we are able to create browsers which consist of collaborating objects (e.g. the use of the locator) and are able to define new user methods for our Web servers, as well as defining separate management interfaces.

Web*

The Web* project [11] has developed a Tcl library which uses CORBA's Dynamic Invocation Interface to access Orbix services [12]. (Orbix is a commercial implementation of CORBA.) The Tcl Orbix clients are executed as CGI programs by either the NCSA or CERN Web servers. Using this technology Web* has successfully used Orbix to provide a WWW interface to Oracle databases - another demonstration that CORBA technology is good at integrating existing information systems.

Our work focuses on applying CORBA technology to Web servers and browsers - the front end of the Web. A native IIOP Web server could communicate with an Orbix service directly using IIOP, or it could use CGI to launch a Web* Tcl client.

The CGI stub compiler we have built is complementary to the Web* project. It uses CORBA IDL to describe the interface offered by the CGI program and from this generates template HTML forms and IDL server skeletons for the CGI program. This technique could be used by Web* to generate HTML forms to access its Tcl Orbix clients. Similarly its Tcl Orbix clients could benefit from skeletons which automatically unmarshalled parameters sent by the Web server.

Summary

This paper has described work being carried out as part of the ANSA workprogramme to provide interoperability between the World Wide Web and CORBA-based distributed object systems. The key elements of the design are the representation of HTTP in IDL, and gateways that translate between HTTP and the IIOP encoding of the interfaces.

The gateways allow HTTP to be carried over IIOP as well as providing full interoperability between Web and CORBA clients and servers, as is shown in Figure 7 below.

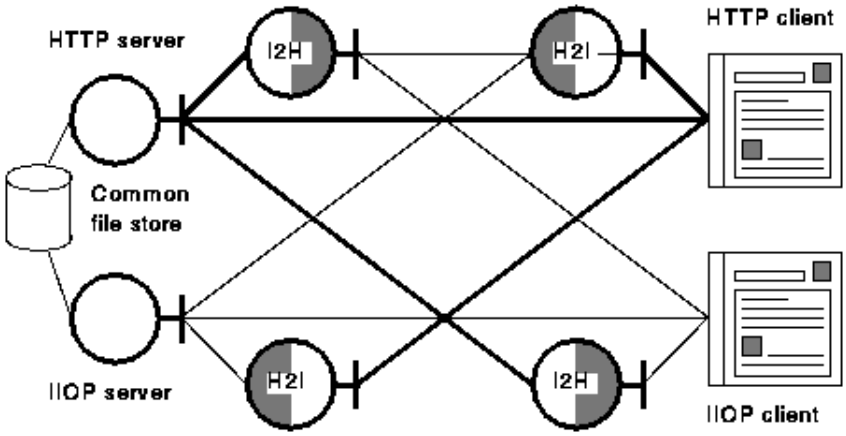

Figure 7. The architecture for interoperability between clients and servers with native IIOP and HTTP modes respectively. For clarity, the locator is not shown. This architecture is an automatic consequence of the manner in which the I2H and H2I gateways have been designed.

Prototypes of these gateways have been implemented, and work is continuing to explore the opportunities offered by distributed object technology in the Web, and Web technology in distributed object systems.

This work has already shown that object technology can be used to provide interoperability between the Web and distributed objects. The necessary protocol support for interoperability is provided by IIOP. We have also shown that IIOP can be seamlessly integrated into the Web via servers and proxies. In this way, it is already possible for the Web to reap the benefits of distributed object technology:

- Ease of programming;
- Extensibility and manageability;
- Encapsulation and systems integration.

Distributed object technology will help the Web to evolve into an information environment of the kind demanded by its users and by commercial interest. At the same time, distributed object technology now has the potential to be used and tested in the biggest distributed system ever built.

Ultimately, this will mean that the Web can be used as an integrator for distributed object systems, while those object technologies will benefit from having the Web client architectures providing them with a globally uniform front end.

Acknowledgements

The authors would like to acknowledge the contributions of all members of the ANSA team, past and present, and in particular Andrew Herbert, towards the ideas discussed in this paper. We would also like to thank the sponsors of the ANSA workprogramme for providing the resources used in this work.

References

1. Object Management Group Inc, *The Common Object Request Broker: Architecture and Specification, Revision 1.2*, December 1993. OMG Document Number 93.12.43; <URL:ftp://ftp.omg.org/pub/docs/93-12-43.ps>

2. Guangxing Li, *An Overview of Real-Time ANSAware 1.0*, *Distributed Systems Engineering* 2(1), 1995, pp28-38.

3. Tim Berners-Lee, Roy Fielding and Henryk Frystyk Nielsen, *Hypertext Transfer Protocol--HTTP/1.0*, work in progress within the HTTP Working Group of the IETF.

4. Object Management Group Inc, *Universal Networked Objects*. March 1995. OMG Document Number 95.3.10; <URL:ftp://ftp.omg.org/pub/docs/95-3-10.ps>

5. Douglas Kosovic, *Trader Information Service*. <URL:http://www.dstc.edu.au/AU/research_news/odp/trader/trader.html>

6. Karen Sollins and Larry Masinter, *Functional Requirements for Uniform Resource Names*, *Internet RFC 1737*, 20 December 1994. <URL:ftp://ds.internic.net/rfc/rfc1737.txt>

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

8. Nigel Edwards, *Object Wrapping (for WWW)--The Key to Integrated Services?*, Document Identifier APM.1464, 1995. Architecture Projects Management Ltd, Cambridge. <URL:http://www.ansa.co.uk/ANSA/ISF/1464/1464prt1.html>

9. *DCE-Web Home Page* <URL:http://www.osf.org/www/dceweb/DCE-Web-Home-Page.html>

10. *Wand--Web and DCE Server*. <URL:http://www.osf.org/www/wand/index.html>

11. *WEB* Home Page*. <URL:http://webstar.cerc.wvu.edu/>

12. *The Orbix Home Page at IONA Technologies*. <URL:http://www.iona.ie:8000/www/Orbix/index.html>

About the Authors

Owen Rees <URL:http://www.ansa.co.uk/Staff/rtor.html>
The ANSA Project, Architecture Projects Management Ltd, Poseidon House, Castle Park, Cambridge CB3 0RD, UK.
Tel +44-1223-568926; Fax +44-1223-359779.
<rtor@ansa.co.uk>

Nigel Edwards
Hewlett-Packard Laboratories, Filton Road, Stoke Gifford, Bristol, BS12 6QZ.
Tel +44-117-9228490; Fax +44-117-9229285.
<nje@hplb.hpl.hp.com>

Mark Madsen <URL:http://www.ansa.co.uk/Staff/msm.html>
The ANSA Project, Architecture Projects Management Ltd, Poseidon House, Castle Park, Cambridge CB3 0RD, UK.
Tel +44-1223-568934; Fax +44-1223-359779.
<msm@ansa.co.uk>

Mike Beasley <URL:http://www.ansa.co.uk/Staff/mdrb.html>
Architecture Projects Management Ltd, Poseidon House, Castle Park, Cambridge CB3 0RD, UK.
<mdrb@ansa.co.uk>

Ashley McClenaghan <URL:http://www.ansa.co.uk/Staff/am.html>
The ANSA Project, Architecture Projects Management Ltd, Poseidon House, Castle Park, Cambridge CB3 0RD, UK.
Tel +44-1223-568924; Fax +44-1223-359779.
<am@ansa.co.uk>