ACCESSING CORBA OBEJCTS ON THE WEB•

Atul Kumar, Deepak Gupta, Pankaj Jalote
Department of Computer Science and Engineering,
Indian Institute of Technology Kanpur - 208016, India
E-mail: atul@iitk.ac.in, deepak@iitk.ac.in, jalote@iitk.ac.in

ABSTRACT

With the emergence of distributed object technology, there is an increasing agreement that the benefits of distributed object technology should be brought to the World Wide Web. The work presented in this paper is an attempt to integrate CORBA objects into the Web. We propose a URI scheme for addressing CORBA objects. A URI for an object not only identifies the object but may also optionally include the name of the method to be invoked on the object and the parameters required. We have implemented the URI scheme by extending Sun Microsystems' HotJava browser and also by implementing a proxy server that acts as a protocol level gateway between CORBA and HTTP.
With this kind of integration, different web services can be described using CORBA, thereby making the Web extensible for different needs and bringing the richness of distributed objects to the web.

KEYWORDS

URI Scheme for Objects, Object Browser

1. INTRODUCTION

The popularity of the World Wide Web has gone beyond any expectations. Web browsers have become the universal user interface for many computer users. Some file managers are now integrated with the web browser to provide a single interface for all the work that a normal user does using a desktop. A paradigm shift has already taken place toward web-based computing.
The web was designed originally as a repository of static HTML documents. Need for dynamic content was felt soon and the Common Gateway Interface (CGI) standard was added to the web. Using CGI programs, web pages can be generated dynamically depending on the user inputs and other state information. Interactivity can be provided by embedding code written in scripting languages like JavaScript and VBScript. Web page presentation can be further enhanced using style sheets, content positioning and downloadable fonts which are supported by Dynamic HTML. Java applets and ActiveX Control (for Microsoft Internet Explorer browser) can be added to a web page to provide powerful interactivity for web based applications.
Beside the web, another development that has been taking place in distributed systems is in the area of distributed objects. This trend has been strengthened by the popularity of object oriented programming and the preference of client-server model over the traditional centralized model. There is an increasing agreement that distributed systems can be best modeled as a distributed collection of interacting objects, and distributed objects provide an appropriate framework for integrating heterogeneous and autonomous computing resources. Object Management Group's Common Object Request Broker Architecture (CORBA) is a leading standard intended to support object oriented distributed application [1]. Another widely used standard is Microsoft's Distributed Component Object Model (DCOM).
By integrating web and CORBA, distributed object technology can be deployed to the wide application base of web. A number of ways for achieving this integration were proposed in last few years [2, 3, 4, 5]. Most of these proposals addressed the problem using one of the four major approaches. The first approach integrates CORBA client server structure into the web making web server act as CORBA client. CORBA applications are wrapped using CGI programs. A user accesses these applications through normal web browser which communicates with the web server using HTTP. Several mechanisms are suggested to make the integration an easy process by providing tools that automatically generate CGI programs and HTML forms from the object interface definition. This approach requires least modification to the existing web. The second

• This work has been partially supported by a grant from Avaya Communications, New Jersey, USA.

approach adds interoperability of HTTP with other communication protocols such as Internet Inter-ORB Protocol (IIOP). This approach relies on some gateway programs to bridge the client and server. Normal web clients access CORBA based servers by contacting the gateway that can translate the HTTP requests into CORBA requests by using a predefined mapping. The third approach combines clients and servers from either CORBA or web using Java-enabled design. A java applet that comes with a web page can invoke operations on the CORBA objects using IIOP. The fourth approach uses specially designed message formats to pass requests and responses between client and server. Special tags are added to some existing markup language for embedding objects in the pages written in that language.

There are several commercial products that provide some level of integration between web and CORBA. CORBA applications can be integrated into the Netscape Application Server (NAS) [6]. Java Servlet technology provides Web developers with a simple mechanism for extending the functionality of a Web server. Distributed object applications developed using Java RMI and CORBA can be deployed on the web using Java servlets. OrbixWeb from IONA is a software environment that allows to build and to integrate ORB based enterprise systems on the Internet [7]. ColdFusion is a complete Web application server for developing and delivering scalable e-business applications [8]. ColdFusion Markup Language (CFML) cleanly integrates with HTML for user interface and XML for data exchange. ColdFusion supports Java and C++, and fully integrates with CORBA. These products help in developing and deploying CORBA applications on the web. Programmers that use distributed object technology for developing business applications use these products to make integration of web applications with the web server an easy process. No extra capability, other than what can be provided by applets and embedded scripting languages, is required at the client side.

Some recent proposals such as SCOAP (OMG TC document orbos/00-09-03), CORBA to WSDL/SOAP Interworking (OMG document number mars/2002-06-03) and WSDL/SOAP to CORBA Interworking (OMG document number mars/2002-06-04) use Simple Object Access Protocol (SOAP) for packing the parameters and return values in an XML document.

In this paper we present an approach for making distributed objects available on the web as a first step towards integrating distributed objects and the web. Our approach is to access the CORBA objects from a browser directly without sending the request to a web server. We believe that a client should directly access the business applications developed using the distributed object technology. At present all such communications takes place via web servers using HTTP as transport protocol. Everything else is hidden behind the web server interface. No CGI wrapping for the CORBA applications is required if a browser can use IIOP to communicate with CORBA ORBs. We have designed a Uniform Resource Identifier (URI) scheme that can address CORBA objects. A browser has been implemented that, given an URI for a CORBA object, locates the object and accesses the Interface Repository associated with the object to get its interface information. CORBA Dynamic Invocation Interface (DII) is then used to invoke a method on the object if the method name and parameters are specified in the URI.

The rest of the paper is organized as follows. Section 2 describes the design of the proposed URI scheme for CORBA objects. Section 3 describes the implementation of Object Browser. Section 4 discusses the security extension that is added to the browser in order to access objects deployed on the Secure ORBs. Section 5 concludes the paper.

2. URI SCHEME FOR CORBA OBJECTS

Now we discuss our proposed approach. In the web, a user accesses a resource by specifying the URI for the resource. In our scheme a user accesses a CORBA object by giving the URI for the object. However, a CORBA object is semantically much richer than a web document and more needs to be done to make objects accessible through URIs.

The URI scheme for accessing objects through the web browsers, besides being able to specify the object, must be able to handle addressing of methods on that object and supplying parameters for invocation. Following is a list of important requirements for the URI scheme

- **Scheme name:** New scheme names are required to distinguish the object URIs from other URIs (for example http:// and ftp://).
- **Object location:** A way to unambiguously address an object is the first requirement of any URI scheme for addressing objects. CORBA Naming Service does not encode hostname or IP address of a machine in the name assigned to an object. The URI scheme can use CORBA Naming Service

names for locating the objects once the Initial Reference of CORBA Naming Service is known. Some ORB implementations allow an ORB to be initialized with an arbitrary Naming Service running on any ORB as default Naming Service if the host name (or IP address) and the TCP port number of Naming Service program are known. This provides an opportunity to unambiguously locate an object with the host name, TCP port of the Naming Service object and the name string of object bound with that Naming Service.

- **Specifying method name:** optionally a method name can be specified in the URI along with the object location. Since method names are strings these can be easily added to an object URI by defining some character(s) as delimiter. If the method name is not present in the URI, the browser should fetch the complete interface information of the object and it should generate HTML forms for each method with input fields for each IN and INOUT parameter.
- **Parameter Names, Types and Values:** parameters can be supplied in the URI to invoke a method. There should be some standard way to convert the IDL data typed values into strings. Parameter names and types can also be added to make the scheme more flexible (otherwise the order of the values will become significant). If the parameters are not given in the URI, browser should generate a HTML form for the specified method. If insufficient number of parameters are given then a partially filled HTML form should be generated.

The URI scheme that we propose has four parts: scheme name, object location (name/address/reference), method name, and parameter list. Each is discussed below.

Scheme Names

Two new naming schemes starting with `ior://` and `iiopname://` are added to the existing set of naming schemes for URIs. `ior://` is used when an object is specified by its stringified Interoperable Object Reference (IOR). `iiopname://` is taken from the Interoperable Naming Service proposal submitted to the OMG by four organizations [9]. This proposal discusses two new schemes – `'iioploc'` and `'iiopname'`. These schemes are easy to use in TCP/IP and DNS-centric environments such as the Internet. `'iioploc'` URLs are used to specify the location of a CORBA service. If a CORBA application uses some basic CORBA services such as Naming Service then initial references for these services are required to be specified at the time of initializing an ORB. The proposal discusses command line arguments such as `-ORBInitRef` and `-ORBDefaultInitRef` to specify the initial references for CORBA Services using `'iioploc'` URLs. `'iiopname'` scheme extends the capabilities of the `'iioploc'` scheme to allow URLs to denote entries in the CORBA Naming Service.

Object Location

- **ior:** Following is the syntax for specifying interoperable object reference in the 'ior' scheme.
  `ior://Stringified_IOR_as_hex-octets`
  The hexadecimal strings are generated by first turning an object reference into an IOR, and then encapsulating the IOR using the encoding rules of Common Data Representation (CDR), as specified in General Inter-ORB Protocol (GIOP) version 1.0. The content of the encapsulated IOR is then turned into hexadecimal digit pairs, starting with the first octet in the encapsulation and going until the end. The high order four bits of each octet are encoded as a hexadecimal digit, then the low four bits. An example of the 'ior' scheme is:
  `ior://0032A3CF....`
- **iiopname:**An iiopname URI contains an address and an object name. Address consists of some DNS host name (or IP address) and a TCP port number. Object name consists of a hierarchical CORBA name. The full syntax is:

```
iiopname = "iiopname://"addr_list["/"name_string]
addr_list = [address ","]* address
address = [version host [":" port]]
host = DNS-style_Host_Name | ip_address
```

  version = major "." minor "@" | empty_string

```
port = number
major = number
minor = number
name_string = string | empty_string
```

  Where:
  host: a DNS-style host name or IP address. If not present, the localhost is assumed

version: a major and minor GIOP version number, separated by ',' and followed by '@'. If the version is absent, 1.0 is assumed.
ip_address: numeric IP address (dotted decimal notation)
port: port number the agent is listening on. Default is 9999.
name_string: a CORBA Naming Service hierarchical string whose components are separated by '/'.
An example of 'iiopname' scheme is:

```
iiopname://somehost.com/a/string/path/to/obj
```

This URI specifies that at host somehost.com, an object of type NamingContext (with an object key of NameService) can be found, or alternatively, that an agent is running at the location which will return a reference to a NamingContext. The string `a/string/path/to/obj` can then be used to obtain the object reference using resolve operation on that NamingContext.

Method Names

Method name follows the object location part of the URI. Two consecutive colons (::) are used as separator between the object location part and the method name. Since a method name is just a CORBA-IDL identifier name, no conversion is required for including it in the URI. Following are the examples of URIs with method name:

```
iiopname://objectserver.com/object/name::do_job
ior://00040A32E3.....1F97::do_job
```

Parameter Names, Types and Values

We use a scheme similar to the CGI for passing the parameter values to the method. The URI with parameter values looks like:

```
iiopname://objectserver.com/one/two/object::do_job?par1=val1&par2=val2
```

Above scheme does not require the parameters to be passed in the same order as defined in the IDL interface. This is because the parameter name is included in the URI with its value. However, some ambiguity may exist if the method is overloaded. Adding parameter type will remove such ambiguity. The URI now looks like:

```
iiopname://a.com/some/object::do_job?int-par1=val1&string-par2=val2
```

Specifying type of the parameters is mandatory and no default type is assumed by the scheme.

Values of different IDL data types are mapped to the URI string as follows. Value of all basic data types is expressed as a string (as used by standard io-libraries of various programming languages for displaying the values on terminal in human readable form). Octet type values are converted into string by two hexadecimal digits (high order first). 'wstring' values can be represented as strings. 'fixed' type is the fixed point decimal number of upto 31 significant digits. 'fixed' type values are easily included in the URI as its string equivalent contains only digit characters and a decimal point. Escape conventions described in RFC 2396 are used wherever required. Following is an example URI with parameter values:

```
iiopname://a.com/object/name::do_job?int-x=34&char-y=c&float-z=3.56
```

Parameter name for basic components of compound data types can be specified in the URI in an unambiguous way. However, we have not yet implemented the dynamic invocation of methods which have compound type parameters.

3. IMPLEMENTATION

A prototype browser for accessing CORBA objects is developed by extending HotJava web browser. New protocols (naming schemes) can be added to HotJava by writing Protocol Handlers [10]. A proxy server is also developed for the new naming schemes. This is useful when it is difficult to add the complete functionality for supporting new naming schemes in a browser. A browser in that case be extended to accept user defined proxy server settings for the new naming schemes and to pass respective URIs to the proxy server using HTTP transport. We have used Java programming language as it is the only programming language that can be used to extend the HotJava browser. JavaORB, a freely available CORBA 2.3 implementation from Distributed Object Group is used to provide CORBA ORB classes [11].

Our implementation assumes that an interface repository is available to the object and its IDL interface is registered with the interface repository. Although, neither an Interface Repository nor the registration of objects' interfaces with the Interface Repository are necessary for objects to provide some service.

The Browser (and the proxy server) works as follows. A given URI is parsed to extract its different components. All parsing errors are handled during the parsing process and an error page is returned to the browser if an error occurs.

If the scheme name given in the URI is 'iiopname', the browser initializes an instance of the ORB by creating an 'iioploc' URI using the host name (or IP address) given in the URI and NameService as an object key. Any error that occurs such as "invalid host name", "naming service not running at the specified location" or "connection time out with host" is reported immediately and further action is aborted. In case the ORB is initialized with the specified Naming Service, the hierarchical name components are resolved and the reference to the target object is obtained. In case of an 'ior' URI, IOR string is "destringified" using an appropriate ORB operation.
Once the object interface is obtained, the Interface Repository associated with the object is queried to obtain its interface information. Any error such as "Interface Repository not available" or "IDL interface not registered for this object" are reported immediately and further action is aborted. If a method name is not specified in the URI, HTML Forms for all methods defined in the object's interface are generated using the interface information obtained from the Interface Repository. Input fields are provided for all IN and INOUT parameters by using the IDL to HTML mapping described in the previous section. The Form ACTION URL for a method, when combined with the input data submitted, forms a complete URI to invoke that method. If a method name is specified in the URI but no parameter is given, a form is generated only for the specified method. If parameters given in the URI are not sufficient for method invocation, a form with some input filled with given values is generated.
If values for all IN and INOUT parameters are supplied in the URI and these values are consistent in data type with the interface definition, CORBA DII mechanism is used to invoke the specified method on the object. A "Request" object is created which can be used for dynamic invocation. Arguments are added to the "Request" object using the appropriate method depending on the parameter's passing type (IN, OUT or INOUT) and parameter's data type. After the operation invocation is successful, return value and the values of OUT and INOUT parameters, if any, are extracted (depending on the data type). In case the return value is an Object Reference, a hyperlink with 'ior' URI corresponding to the Object Reference is provided in the result page. In case the operation is "one way", a message informing successful invocation is returned. In case an Exception is raised during the invocation, the stack trace of the exception is displayed.
Results of the method invocation are presented as a HTML page. In order to achieve this, a CORBA-IDL to HTML mapping is required. Objects can be inserted into a HTML page using special tags as defined in [12]. But this requires modification in the HTML rendering code of the browser as a standard browser does not understand these tags. We use the string representation of typed values to present the results in a HTML page. However other possibilities, such as using the 'data' URL scheme proposed in RFC 2397, which provides a mechanism to supply immediate data in the URL, can be explored to present the results.

4. ACCESSING SECURE OBJECTS

Restricting access for certain services to only authorized users is a basic requirement for many services that are accessed through the Internet. Normally, a method on a CORBA server object can be invoked by any CORBA client that can access ORB of the server object over the network. CORBA application developers can make use of the CORBA Security Service for restricting access to some or all the objects available on an ORB. Authentication and access control can be managed by the CORBA Security Manager and the objects need not to have code for enabling and managing security for their methods. Objects with similar security requirements can be grouped in the same domain and they can share the same access control policy [13].
A number of authentication mechanisms are supported in CORBA Security Service [14]. The simplest of these is username-password based authentication. In order to use a protected service, a CORBA client program needs to supply the correct username and password to the principal authenticator object on the server object's ORB. If the authentication is successful, a credential object is created and the reference to that object is returned to the client program. Now the client program can present this credential to access an object on the secure ORB. If this authenticated user is allowed the access to the object and method in question then the secure ORB invokes the request otherwise an exception is raised. An exception is also raised if a secure object is accessed without the authentication step.
If the browser intercepts the NO_PERMISSION exception while invoking the requested method on the target object, it prompts the user for the username and password. The browser then uses these values to authenticate the user with the CORBA Security Service on the remote ORB. The browser receives 'Credentials' for the user after the authentication is successful. These 'Credentials' are presented to the remote ORB while invoking the method again.

5. CONCLUSION

Our goal for this work was to demonstrate the feasibility of providing web services using CORBA objects. So far, our work has emphasized accessibility of operations served by CORBA objects using web like URIs through a web browser. With this prototype in place, we are now working on modeling of existing web objects such as MIME typed objects, cookies, and encrypted sessions using CORBA interfaces. Once this can be done, we will have an integration of web and general purpose distributed objects.
This will make the web much more powerful as a wide range of applications developed using the CORBA can be deployed on the web without any special arrangement for interfacing them with the web server. Range of objects that can be accessed will eventually become unlimited. This will further help in solidifying the trend of making computing collaborative and web-centric.

REFERENCES

1. Object Management Group's CORBA Website. URL: http://www.corba.org/
2. Joint W3C/OMG Workshop on Distributed Objects and Mobile Code, June 24-25, 1996, Boston. URL: http://www.w3.org/OOP/9606_Workshop/
3. OOPSLA96 Workshop: Toward the Integration of WWW and Distributed Object Technology, October 6, 1996, Almaden. URL: http://www.eng.uci.edu/~peilei/index.html
4. Jeff Sutherland, Integrating Java, Objects, Databases, and the Web, Proceedings of International Conference on Object Oriented Information Systems, December 16-18, 1996, London. URL: http://jeffsutherland.com/oopsla96/suth.html
5. Senta Fowler, Issues on Integrating Objects on the Web. URL: http://www.cs.wvu.edu/~fowler/exam1.html
6. Integrating NAS, Netscape Extensions and CORBA. URL: http://developer.netscape.com/docs/manuals/appserv/cookbook/NASandCORBA.html
7. OrbixWeb 3.2 White Paper. URL: http://www.iona.com/products/docs/orbixweb32-wp.pdf
8. The ColdFusion Web Application Server. URL: http://www.allaire.com/products/ColdFusion/index.cfm
9. Interoperable Naming Service, Joint Revised Submission by BEA Systems, DSTC, IONA, and Inprise, OMG Document orbos/98-10-11. URL: http://cgi.omg.org/cgi-bin/doc?orbos/98-10-11
10. Mark Wutka, et. al., Adding Additional Protocols to HotJava, JAVA Expert Solutions, Chapter 35. URL: http://docs.rinet.ru/JSol/ch35.htm
11. Distributed Object Group's JavaORB. URL: http://dog.intalio.com/details_javaorb.html
12. Dave Raggett, editor, Inserting Objects into HTML, World Wide Web Journal, Volume I, issue 3, Summer 1996. URL: http://www.w3journal.com/3/s2.raggett.html
13. Bob Blakley, CORBA Security. The Addision-Wesley Object Technology Series, Chapter 5 (p45-60), 2000.
14. CORBA Security Service Specification, URL: http://www.omg.org/technology/documents/formal/security_service.htm