# Tunneling Across Firewalls by Using XML and Servlet: An Experiment on CORBA

Wing Hang Cheung, Michael R. Lyu, Kam Wing Ng
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong SAR, PRC
{ whcheung | lyu | kwng } @cse.cuhk.edu.hk

## ABSTRACT

*In this paper, we describe how we use XML and Servlet to tackle some communication problems with firewalls. Common firewalls block many different communication protocols; CORBA IIOP is one good example. We introduce our mechanism by using XML and Servlet to support CORBA general calls tunneling through firewalls with HTTP. We further extend our mechanism to support CORBA callbacks. Then, we give an example of using the proposed tunneling method to implement a scalable mediator-based query system, and perform an evaluation on our method. Our approach is generic, and can be applied to other communication protocols as well.*

## Keywords

Firewall Tunneling, Firewalls, XML, Java Servlet, CORBA, IIOP, CORBA callbacks

## 1. INTRODUCTION

With the rapid expansion of the Internet, the use of firewalls is also becoming more and more common nowadays. Firewalls are used in the gateways between the local networks and the public Internet, in order to protect the computers in the internal networks by enforcing some security policies. Their role is to control external access to internal information and services. Using packet filtering by a router in the network layer to enforce certain rules is one of the most common mechanism used by the firewalls. But firewall systems can include elements that operate at layers above the network layer in the application level. Application level gateways for Telnet, File Transfer Protocol (FTP), and Hypertext Transfer Protocol (HTTP) are in common use.

Common firewalls block many less common applications, such as the communication protocols for agents, and also the Internet InterORB Protocol (IIOP) used in Common Object Request Broker Architecture (CORBA) [1]. This is because those common firewalls may not be able to decode the message bodies of those protocols. Using CORBA IIOP as an example: IIOP is the Object Management Group (OMG) specified network protocol for communication between object request brokers, which employs TCP/IP and can be handled by common firewalls at network and transport level with packet filters. But at the application level, the message body of IIOP is encoded in Common Data Representation (CDR) and firewalls are unable to decode it. Therefore, firewalls cannot base filtering decisions on IIOP messages.

With the blockings of some protocols by firewalls, the scalability of system development and system integration would be limited. There exist specific firewalls with certain some protocols, but they are usually not generic and may have some limitations. Take CORBA IIOP as an example again: There are a number of firewalls for CORBA IIOP, such as IONA Orbix Wonderwall [2] and Visibroker Gatekeeper [3], but they cannot solve all firewall problems. First, they are not commonly used. Secondly, they may be vendor-dependent and proprietary. Also, some CORBA features, such as callbacks, may not be handled.

Elenko and Reinertsen [4] have suggested a communication perspective of the cooperation between XML and CORBA by employing XML, Servlet and HTTP calls to substitute for CORBA IIOP communications. Applying HTTP calls to transport XML parameter contents can eliminate the

complicated firewall issue of IIOP, as application level gateways for HTTP are in common use.

Consequently, we took CORBA IIOP as our target and developed a simple solution by using HTTP, XML and Java Servlet for tunneling through the firewalls for the support the CORBA IIOP calls in a generic way. In the next section, we present how we use XML and Servlet to support CORBA IIOP calls and tackle the firewall problem. In Section 3, we address how we further enhance our mechanism to allow CORBA callbacks which are not feasible behind many CORBA firewalls. In Section 4, we briefly describe how we can automatically generate the necessary components to support our mechanism, by referring to the design of IDL (Interface Definition Language) for a CORBA system. In Section 5, we provide an example of using our approach to implement a scalable query system. We address the evaluation of our approach in Section 6 and conclude our work in the last section. In general, our approach can be applied to other communication protocols as well.

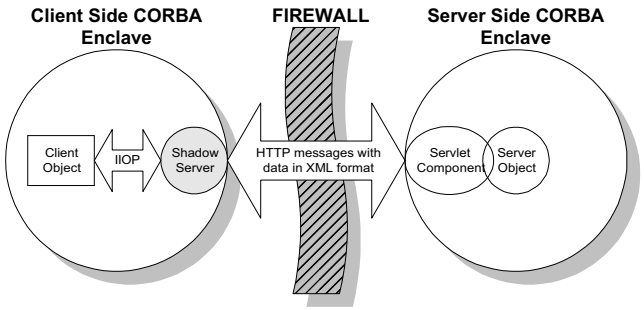


Fig 1. Our mechanism to support general CORBA IIOP across the firewalls

## 2. SUPPORTING GENERAL CORBA CALL

In order to support IIOP calls between two CORBA enclaves that are separated by firewalls, the main approach we use is to convert the contents of IIOP calls into HTTP calls, as HTTP calls can go through the firewall blockings. Figure 1 shows the mechanism of our tunneling solution. In this case, we need two components to do the conversions automatically: one is at the client side to convert request messages from IIOP messages to HTTP messages (i.e., the one named as *Shadow Server* in Figure 1), while another one is at the server side to convert the HTTP request messages back to normal IIOP messages (i.e., the one named as *Servlet Component* in Figure 1). Their duties will be inter-changed when the server returns the computation results back to the client side. We explain now the details of these two components.

### 2.1 At Client Side

We call the client-side conversion component as *Shadow Server*, as in the client objects' viewpoint, this conversion component will allow client objects to make requests. Then this component will return the results to the clients, so client objects can just regard it as a proxy server object. That is why we call it as *Shadow Server*.

The *Shadow Server* object must be very similar to the real target server object in terms of their interface. They must share the same interface such that the client objects will not notice the differences between them when making requests. Other than the common interface, all the internal implementation of the methods would be different. The Shadow Server will not do any real computation or manipulation to the data passed by the clients, but it will convert the parameters and other related information to HTTP messages and send them to the real server object. It also converts all received resulting HTTP messages to ordinary IIOP messages and returns them to the client objects.

### 2.2 At Server Side

We use Java Servlet [5] component in the server side to communicate with the *Shadow Server* in the client side. Java Servlet is a web component, managed by a container, that generates dynamic content. Servlets interact with web clients via a request-response paradigm implemented by the Servlet container. This request-response model is based on the behavior of HTTP.

In the server side, the server objects will combine with a Servlet component when they are to be called by some client objects outside their CORBA enclave. When the Servlet component receives a request from the client side, it will read and extract the parameters and the related information of HTTP message, and then invoke the related server object. The Servlet

component will also convert the returned data into an HTTP message and send them back to the client.

### 2.3 Data in Transmission

Extensible Markup Language (XML) [6] plays a very important role in the transmission of HTTP messages. XML has the flexibility defining new tags on top of its semi-structure feature, so that it can well represent most of the complicated data structures [7]. Even in the case of unlimited-multilevel recursive data structures, such as tree structures, XML can still handle them nicely. Figure 2 shows a tree structure and its corresponding XML representation. We can see that the XML data can represent data with great flexibility. Hence, we use HTTP to send stream of XML data between the client and server sides. By using the Data Type Definition (DTD) of XML data, we can provide a grammar for the XML data transmission format. Hence we can make a compromise on the interpretation data transmission formats for both sides.

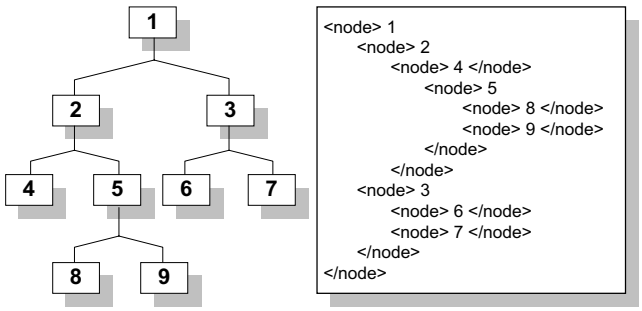


Fig 2. A tree structure and its corresponding XML data describing its structure

Besides the flexibility of data representation, the readability and the ease of manipulation of XML information also provide great flexibility of server as well as client implementation. As long as we follow the DTD of the data transmission format, it is not limited for the client side or the server side to be implemented by CORBA objects. Hence, programmers can have great freedom to choose different implementation methods.

## 3. SUPPORTING CORBA CALLBACK

The mechanism introduced in Section 2 can only be used in general CORBA calls. CORBA provides an interesting callback feature, which cannot be handled by this mechanism. When client programs need to react to changes or updates that occur in the server side, it would be inefficient for the client programs to check the server periodically. Instead, it would be more efficient if the server can notify the clients whenever there is an update in the server side, hence the client programs can react to changes with a faster response. This approach is the callback feature. As the callback feature requires both sides to be capable of starting a communication, we require both sides to have the shadow objects and Servlet components.

We describe our system design for callbacks in Figure 3. The client object should be combined with a Servlet component when it is expected to have callbacks at the very beginning, which can be known from the system IDL (Interface Design Language) design. The client will first get a reference to the shadow server in the client-side CORBA enclave. The shadow server sends information, such as IP address, port number and call method of the client object and its Servlet component to the server side.

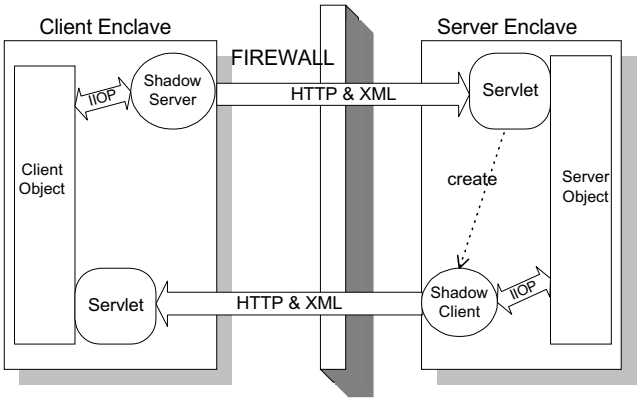


Fig 3. Diagram showing the mechanism that supports CORBA callbacks

At the server enclave, if the server object's Servlet component does not have the record of the client object, it will automatically create a new *Shadow Client* object. That *Shadow Client* object will be initialized by the information of the real client

and its Servlet component, so that it will know how to set up the connection with the real client later.

The real server object then gets the reference of the *Shadow Clients* (the newly created ones in the server side) that require callbacks. Whenever the server is updated, it can call the *Shadow Clients* to invoke and notify the client object. During the data transmission, we still employ similar XML data format as described in the previous section. By this mechanism, we can support IIOP calls for CORBA callbacks by integrating XML, Servlet and HTTP calls.

## 4. AUTOMACTIC CODE GENERATION

When building a CORBA system, it is a must to have an IDL file to generate necessary source codes for server skeletons, client stubs and other system architectures. The IDL design of a CORBA system provides the interface definitions of all the objects in the system. As the add-on Servlet components and the shadow objects concern only the interfaces of the server and client objects, we can use the IDL files to generate these add-on components automatically. The IDL files can provide the following interface information:

- Interface names;
- Method names provided by each interface;
- The return type of each method;
- All parameters types and their orders in prototypes of each method;
- All exceptions in each method; and
- The possibility of having callbacks features (i.e., when there is a CORBA object interface which has another CORBA object interface as one of its parameters).

We have written a compiler for IDL files which will analyze the interface design and generate the following artifacts:

- Source code for shadow server/client objects;
- Source code for Servlet components; and
- DTD of the transmitted messages.

The generation of these source codes reduces the extra programming work for those add-on components as they usually contain many similarities, especially in the part of converting the internal data structures to XML formats. The generation of DTD files, on the other hand, provides a standard for information exchange in XML formats, such that based on the DTD, system developers can have implementations other than using CORBA for their client or server components.

## 5. EXPERIMENT ON A QUERY SYSTEM

In order to provide a clearer picture of our approach, we demonstrate an example of an Internet application using our tunneling method. We have implemented a mediator-based query system [8] to illustrate the steps we introduced.

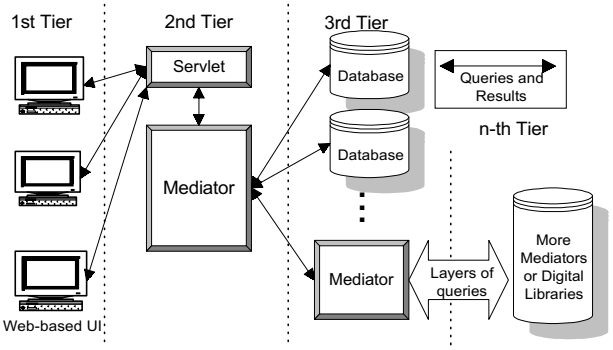


Fig 4. Diagram showing the architecture of our query system example CORBA

Mediators are the middleware between the client objects and the database objects, where clients can be end users or another mediator objects. Mediators forward the client queries to the appropriate database objects, and then integrate the returned answers and forward them back to the clients, forming an n-tier distributed system. Figure 4 shows the architecture of our system, which consists of *Mediator* objects, *Database* objects and user interfaces. The system provides a good method for system integration, as it can abstract multiple databases into a single conceptual interface to the users.

Now, suppose we want to make a query from a mediator object in the client CORBA enclave, to another mediator object in the server CORBA enclave, in which they are separated by firewalls. In this query system, we have implemented a *Shadow Server* object, named *MediatorGateway* to serve the

purpose of connecting to another mediator object in a server enclave behind a firewall. From the viewpoint of the objects in the client enclave, the *MediatorGateway* object, which is also inside the same client enclave, is the same as the target *Mediator* object in the server enclave, as both of them implement the same interface. The objects in the client enclave can call *MediatorGateway* the same way as they call *Mediator* objects. This provides transparency to other CORBA objects.

The *MediatorGateway* object converts all the necessary parameters into an XML message, and then send it to the target mediator by HTTP calls. Figure 5 shows the mechanism of such calling. In Figure 5, Mediator *M* wants to call another Mediator *SM*, which is integrated with Servlet. Mediator *M* would call the *MediatorGateway* shadow object *G*, and *G* converts the queries and parameters into an XML message and sends it to *SM* by HTTP calls going through the firewall. *SM* further makes queries to other databases in the server enclave and returns the answers, which are also converted in XML format and sent to *G* by HTTP calls. *G* would return the answers to Mediator *M* using ordinary IIOP calls. In this case, mediator *M* just performs a normal call and the external object reference is transparent to *M*.

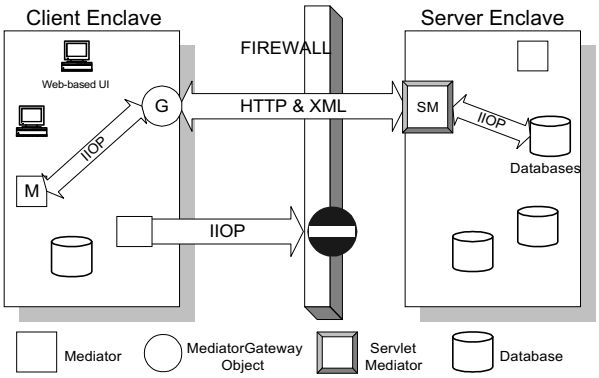


Fig 5. Diagram showing the mechanism that supports general CORBA calls in our sample system

To better help the users in obtaining the information they need, one important feature of modern information system is allowing users to specify some topics of information they want to subscribe. Whenever there is an update of the specified information, the database objects can inform the subscribed users immediately. This feature requires callbacks.

We introduce a new object, *agent* object, which asks for subscribed information. It will register the subscribed topic on the Mediator object, and then wait for the Mediator object to callback. You may refer to Figure 6 for the callback mechanism. *Agent* object, *SA*, which has also associated with it a Servlet component for later callbacks, sends a message to *MediatorGateway*, *MG*, for registering the subscription to the mediator. The message, in XML format and sent by HTTP call, will carry information about the IP address, port number and other relative information of the Servlet Agent object.

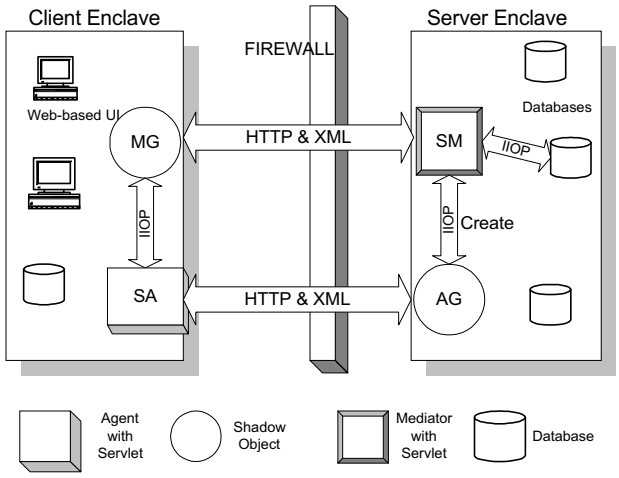


Fig 6. Diagram showing the mechanism that supports CORBA callbacks in our sample system

In the server side, when the Servlet component of the mediator notices that there is a need to callback, but it does not have to make a record of that client object, it will create an *AgentGateway* object, *AG*, the *Shadow Client*, for the mediator to callback later. Whenever the mediator notices that there is an update in the databases, it will immediately give a callback to the *AgentGateway* object. The *AgentGateway* object will invoke the corresponding *Agent* object via HTTP calls and this will return the result back to the real *Agent* object in the client side. That is how we implement the callback feature of CORBA with our mechanism.

## 6. EVALUATION

### 6.1 Performance Statistics

We have evaluated the performance of the CORBA-based query system described in Section 5 in order to find out the overhead of our approach. The query system was installed in a number of personal computers with Pentium III 500MHz CPU and 10Mbps network connection. We tested our system in a low workload environment such that it will not be influenced by other factors. The system is Java-implemented and the queries in our system would return a few hundred bytes of information. We kept track of the time each process used and the results were shown in Table 1.

**Table 1. Performance Statistics of the Query System Described in Section 5.**

| Processes | Milliseconds |
|---|---|
| 1. Mediator objects | 20 - 80 |
| 2. Databases | 180 - 800 |
| 3. IIOP communications within the same CORBA enclave (Local Area Network) | 10 - 100 |
| 4. Shadow Client or Server objects | 20 - 100 |
| 5. Servlet Components with Tomcat Servlet Engine [9] | 120 - 250 |
| 6. HTTP communications in the Internet | 240 - 2200 |

From Table 1, we can find out that when comparing to other objects in the system, the Servlet components and the shadow objects carry similar time to process. And the most time-consuming part of the whole process is the Internet connection, which is unavoidable in the communications in the world-wide area. When comparing to the communications in the Internet, the time spent in those add-on components are not significant.

### 6.2 Pros and Cons of Our Approach

Our implementation of using XML, Servlet and HTTP calls to substitute IIOP connections enjoys the following advantages:

- It can solve the incompatible IIOP firewall problems and provide vendor-independence and callbacks support. Common normal firewalls cannot block the communication between CORBA objects and hence the scalability of system design and construction can be greatly increased.
- The newly-added components to the system are transparent to the original objects. Internal CORBA objects would not notice the difference between the real target object and the shadow object, thus no special modification or implementation is needed for ordinary internal objects, which increases the system transparency properly.
- Systems can maintain good security, as external CORBA objects outside the enclave can only call the objects integrated with the Servlet components, and we can protect other internal CORBA objects from being called externally. Moreover, we are exploiting some very common products like Java Servlet or HTTP calls, whose security properties are well developed.
- No information loss or distortion, as using XML can represent the information in the transmitted messages well, even when the parameter structures of the invoking calls are complicated. This properly enhances the system interoperability.
- Our mechanism can also be used as a gateway to inter-cooperate with other non-CORBA modules. As long as the DTD of transmitted messages is defined and agreed between both clients and servers, we include any kind of implementation in the server and the client sides.

Our design has some drawbacks, however. First, using HTTP calls is slower than using the IIOP communication. Moreover, we have one more Servlet layer in the implementation and thus the system requires extra workload to initialize and engage the Servlet components. Nevertheless, the time for this overhead is negligible when comparing with the average Internet access delay.

## 7. CONCLUSION

In this paper, we have addressed the problem of the common use of firewalls in the Internet, which can block many communication protocols and affect the scalability of system development and system

integration. We have proposed a mechanism to tackle this Internet communication problem with CORBA IIOP as an example. As CORBA IIOP communications may be blocked by common firewalls, the method we have proposed integrates XML, Servlet, and HTTP calls to perform tunneling and for the IIOP connections. We have further addressed how we could enhance our design to support the CORBA callback feature, which may not be supported by other CORBA firewall gateways.

Then we have briefly addressed how we can generate the related source code and components automatically and in a generic way by engaging the interface design (IDL) of a system. We have also presented a real example by applying this mechanism to implement a scalable mediator-based query system across firewalls. The advantages and disadvantages of our system have been evaluated and presented.

## 8. REFERENCES


[1] Object Management Group, Inc. *The Common Object Request Broker: Architecture and Specification. Revision: CORBA 2.4.1*, November 2000.

[2] IONA Technologies. *Orbix Wonderwall Administrator's Guide*, June 1999.

[3] Visigenic Software, Inc. *Visigenic Gatekeeper Guide*, version 3.2 edition, February 1998.

[4] Mark Elenko and Mike Reinertsen. XML & CORBA. Application development trends, September 1999.

[5] Sun Microsystems, *Java Servlet Specification Version 2.3*, October 2000.

[6] World Wide Web Consortium, http://www.w3.org/TR/2000/REC-xml-20001006. Extensible Markup Language (XML) 1.0 (2nd Ed), 2000

[7] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: from relations to semistructured data and XML.* Morgan Kaufmann Publishers, San Franciso, USA, 1999.

[8] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer Volume 25 Number 3*, March 1992.

[9] Apache Software Foundation. *Jakarta Project Subprojects:Tomcat.* http://jakarta.apache.org/tomcat