FRAMEWORK FOR ACCESSING CORBA OBJECTS WITH INTERNET AS THE BACKBONE

By

MEENAKSHI SUNDAR SETHURAMAN

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2001

I dedicate this thesis to my parents

ACKNOWLEDGMENTS

I offer my highest gratitude to Dr. Sanguthevar Rajasekaran for providing me with guidance and motivation to complete this thesis. I also thank Dr. Sartaj Sahni and Dr. Panos Pardalos for serving on my thesis committee. I feel very proud to have people of such stature related to my work.

I would like to take this opportunity to thank my family and friends for providing me with continued support and love.

TABLE OF CONTENTS

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

FRAMEWORK FOR ACCESSING CORBA OBJECTS WITH INTERNET AS THE BACKBONE

By

Meenakshi Sundar Sethuraman

August 2001

Chairman: Dr. Sanguthevar Rajasekaran
Major Department: Computer and Information Science and Engineering

To communicate to an object implemented with in an ORB via the Internet, IIOP protocol is used. Because of security issues, firewalls block all protocols except for HTTP. As a result the IIOP protocol is also blocked. The Framework has been designed to overcome difficulties to communicate to an ORB-object via the Internet. The framework uses XML for communication, which is text-based and hence can be transferred over HTTP. As the firewall is configured to allow HTTP protocol, the XML message is not blocked. The XML document uses the SOAP message format for its data representation. On the server side is a translator, which translates the XML message to equivalent CORBA call, thus invoking the remote implemented object. The result is sent back to the client as an XML document in SOAP message format.

.

CHAPTER 1
INTRODUCTION

E-commerce has changed the face of business. The Internet can be used to drastically improve efficiency, reduce costs, and increase sales for an organization by automating business-to-business (B2B) relationships with their partners.

Growth Of E-commerce

In a short period of time, the Internet has changed the face of communication. A vast amount of information is available to millions of users using the Internet. Apart from ordinary users corporate business also effectively uses the Internet. Almost all the transactions among corporate partners are electronic transactions through the Internet. Consumers can use the Internet to buy items and suppliers can use it to sell their items. We can view the Internet as an intermediary where buyers, suppliers, distributors and partners can satisfy their requirements.

Web Technology

The advent of web browsers and Hypertext markup language (HTML) has resulted in the tremendous growth of web technology. The web browser became an important client interface. They were capable of displaying static HTML documents. The HTML allows organizations to publish information in a format that any user on any system can display. They have a predefined set of tags, which are mainly used for data formatting and representation. These predefined tags were part of the evolving HTML standard. In early days, web applications were simple and contained static information.

Enhanced Web Technology

Web technology enhancement has added support for dynamic content. Web servers can connect to corporate backend systems and dynamically generate HTML to display information. Web applications are available to access information from data warehouses and live corporate data.

E-Commerce

Consumer-based solutions or e-commerce applications have become an essential part of day-to-day business. Customers, suppliers, distributors and other partners can use the web to browse catalogs, place orders, execute transactions, monitor inventory levels and conduct other operations.

Business-to-Business E-commerce

Present applications are browser-based. These applications need human interventions and are oriented for human users. The Internet can be used for exchanging automated information between two corporate systems, which is effectively between two application systems [1]. The solutions that focus on integrating inter-corporate application can significantly reduce costs through well-integrated B2B deployments. The automation of B2B relationships with suppliers, distributors and other partners will greatly improve efficiency, will reduce costs and will increase sales [1]. The Internet can be used for automated procurement, replenishment and distribution operations.

CORBA Middleware

One of the most popular approaches to application integration or well-integrated B2B deployment is to implement interfaces using standard distributed middleware such as CORBA, DCOM or RPC [1]. Middleware is very effective in building internal

distributed applications and is also used to implement business-to-business integration across the Internet. CORBA is one form of the popular middleware available in the industry.

CORBA

The Common Object Request Broker Architecture (CORBA) is one of the most important middleware specifications available to the industry. CORBA provides a framework for development and execution of distributed applications. CORBA allows a distributed, heterogeneous collection of objects to interoperate [2]. The Object Request Broker (ORB) is the object bus or the distributed service that implements location transparency. It locates the remote object on the network.

Inter ORB Architecture

The General Inter-ORB protocol (GIOP) specifies a set of message formats and common data representations for communicating to an ORB with the Internet as the backbone. The Internet Inter-ORB protocol (IIOP) specifies how GIOP messages are exchanged over a TCP/IP network [3]. The IIOP makes it possible to use the Internet itself as a backbone to communicate to an ORB.

Problem Definition

Issues Relating to ORB Communication via Internet

Firewall Issues

Security imposes impediments on cross-corporate application integration. The organizations have firewalls set up to protect their systems from malicious misuse. In order to support request from browsers, the firewalls are configured to allow HTTP requests to get in but to disallow or stop requests that use other protocols [1]. The firewall

does not have the capability to effectively distinguish between authorized middleware users and malicious hackers. To communicate to an ORB via the Internet, it requires IIOP protocol to be used, which unfortunately will be blocked by the firewall. In a present day scenario it is not possible to communicate to an ORB with the existing framework and Internet as the backbone.

Performance Issues

CORBA users have also tried to implement cross-corporate applications using firewalls that can be configured to accept CORBA IIOP requests [1]. This solution works only if the particular firewall has been implemented at all sites involved in the application. It has been also found that it suffered serious set backs performance vise.

Other Security Issues

Even if the request is allowed to penetrate the firewall, the e-commerce application also needs to implement its own security protection to authenticate users, authorize access and protect data. Most middleware solutions provide security services, but they rely on internal security systems that don't easily extend outside of the organization.

Framework for ORB Communication via the Internet

XML

Extensible Markup Language (XML) is a subset of Standard Generalized Markup Language (SGML) [4, 5]. An XML document can be used as a self-describing message that can convey information between applications across the Internet. XML does not have a predefined set of tags. The tags defined in the XML are user defined and convey specific information to the parser.

SOAP

Simple Object Access protocol (SOAP) is a lightweight protocol for exchange of information in a decentralized, distributed environment [6]. It is an XML based protocol. The framework uses only the format for the exchange of information defined in SOAP.

XML-CORBA Framework

Although middleware provides the kind of services required to implement cross-corporate ORB communications it has limited applicability on the Internet. The limited capability is due to the fact that middleware uses its own protocol, which is blocked by the firewall. The solution to this problem is a design that uses HTTP as the medium of carrier across the Internet. Designing a framework using HTTP, XML and a translator, which can translate between XML request and equivalent ORB request, can solve the problem. The client sends the message in XML using SOAP format via the HTTP. On the server side the translator converts the service request in XML to equivalent CORBA request. The request is dynamically created, populated with arguments, sent to the service object and results obtained. The service object is identified using the target reference. The results are converted back to XML in SOAP format and sent back to the client.

Scope of the Thesis

The thesis assumes that naming standards evolve and that the corporate use these standards to query services. The client sends in their requests using XML in the SOAP format. Any other format is out of scope of the framework. Non-XML communications and transfer of the data is also not within the scope of the framework.

CHAPTER 2
CORBA AS MIDDLEWARE

This chapter deals with the basic concepts of CORBA and its architecture. CORBA was designed to allow components to discover each other and interoperate on an object bus. It specifies a system, which provides interoperability between the objects in a heterogeneous, distributed environment. It is based on the Object Management Group's component object model.

CORBA defines interface specifications that are written in Interface Definition Language (IDL). The CORBA IDL is declarative in nature and does not provide any implementation details. The Interface Definition Language (IDL) is a descriptive language and not a programming language [2]. It describes the interfaces being implemented by the remote objects. The IDL is used to define the name of the interface and the names of each of the attributes and methods. Once the IDL file has been created, it can be used to generate client stubs and server skeletons. The client IDL stubs provide the static interfaces to the remote service-objects. The precompiled stubs define the invocation of the remote services by clients. The stub acts like a proxy for the remote server object [2]. The middleware that establishes connection between the client stubs and the target objects or the service objects is the ORB. By using an ORB, the client stubs can transparently invoke a method on the server object. The ORB intercepts the client calls and is responsible for finding the remote-object. The ORB also passes the parameters to invoke the remote-object and finally returns the results. The client at first obtains an object reference of the remote service-object. Using the object reference and

the precompiled stubs the client issues a request to the target object via the ORB. The following diagram depicts the overall architecture of the CORBA specification.

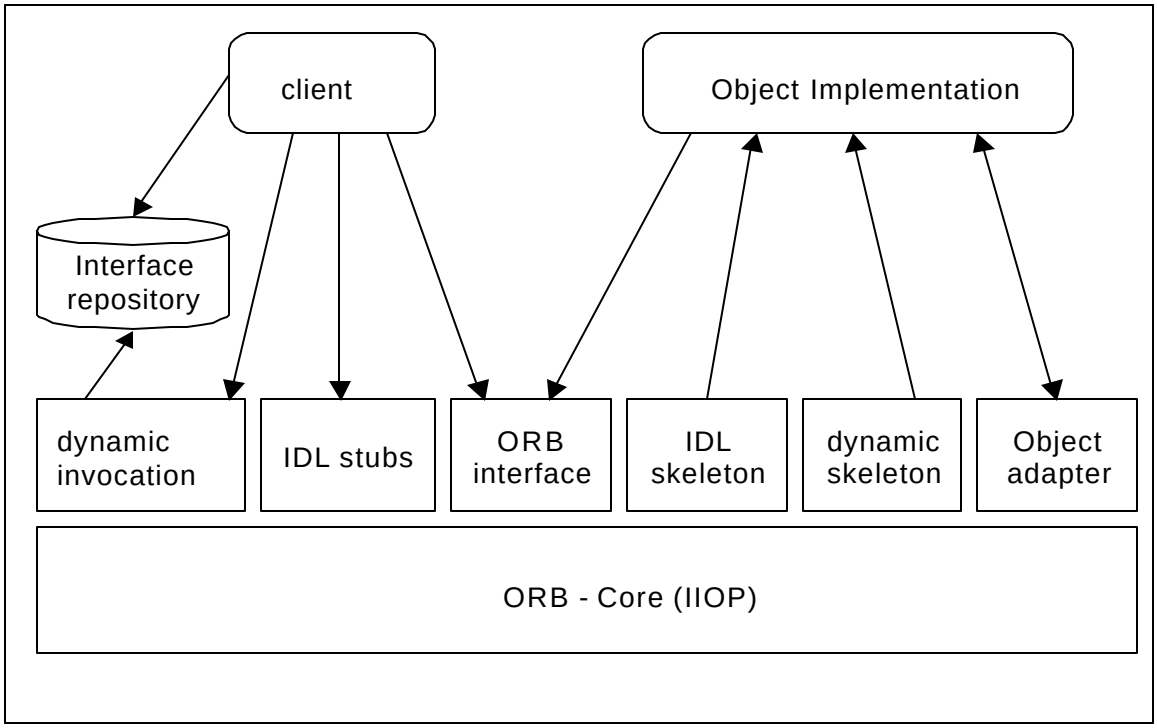

Figure 1. CORBA Architecture

Overview of Architectural Components

Object Request Broker

The ORB is the heart of any CORBA implementation. It enables objects to transparently make requests and receive responses in a distributed environment. Due to the presence of the ORB, the client applications invoke the remote-objects, as if they are present locally.

IDL Stubs

The client IDL stubs provide the static interface for the object services. To the client it is a local call to a proxy object, which in turns communicates with the remote

service object. The stubs also perform marshaling. It has the capability of encoding and decoding the operation and the parameters into the flattened message formats.

IDL Skeletons

The IDL skeletons are equivalent to the IDL stubs. The skeleton is the bridge between the ORB and the actual code that implements the method associated with the object's interface.

Dynamic Invocation

The functionality, which allows the dynamic construction and dispatch of the object invocation without the stubs, is called dynamic invocation. A stub routine is specific to a particular operation. But the dynamic invocation allows any object to be invoked. To dynamically invoke the client supplies information about the parameters and the object to be invoked. The framework uses the dynamic invocation for invoking the remote object. The server cannot tell the difference between static and a dynamic invocation as they both have the same message semantics.

Dynamic Skeleton

The dynamic skeleton provides a run time binding mechanism for servers to invoke the server-objects without the server skeletons. The dynamic skeleton looks at the values of an incoming message to figure out the target object and the method. The precompiled skeleton interface, in contrast to a dynamic skeleton, is for a particular object implementation.

Object Adapter

The object adapter accepts requests on behalf of the server's objects. It provides the run-time environment for instantiating server objects, passing the requests to them and assigning them with their respective object ids or references. The object adapter also

registers the classes it supports and their run time instances with the implementation repository.

Interface Repository

The interface repository is a run time distributed database that contains machine-readable IDL-defined interfaces. The application allows components to dynamically access, store and update metadata information. Thus by using the information of the interface repository it is possible for a program to determine the operations that are valid on the interface.

Internet Inter ORB Protocol

The ORB interoperability allows communication between independent implementations of the CORBA standards. The interoperability also allows a client of one vendor ORB to invoke operations on an object implemented within different ORB. To achieve above transaction the ORB should communicate using a standard protocol. The protocol for ORB interoperability is the General Inter-ORB protocol (GIOP) [3]. It is defined for any connection oriented transport layer protocol. The specification of the GIOP designed for the TCP/IP in the transport layer is the IIOP protocol. IIOP defines how GIOP message formats are exchanged over a TCP/IP network [3]. So for an ordinary client to communicate to an ORB through the Internet, IIOP messages have to be constructed. Another drawback of the IIOP is that firewalls block the entry of all the protocols except for the HTTP. This makes it impossible to communicate to an object implemented within an ORB.

CHAPTER 3
XML AND SOAP

This chapter deals with the basic concepts of XML and SOAP. The framework uses XML to convey the service and the parameters required. The XML data uses SOAP format for the communication across the Internet using HTTP. The XML is an acronym for Extensible Markup Language. XML is a meta-language used to define other languages. XML is also a markup language that specifies neither the tag set nor the grammar for the language. The XML document is enclosed within a SOAP message format. The SOAP is an acronym for Simple Object Access Protocol.

Basic Concepts of XML

A XML document can be parsed if it is well formed. A well-formed document is one that has every user-defined tag closed. Though the XML document does not follow grammatical rules it still should conform to a general set of principles. These principles are then used by XML – aware applications and parsers to make sense of the document and perform some action with the data.

A XML document is not required to be valid. A valid document is one that conforms to its DTD. The DTD is an acronym for document type definition. A DTD defines the grammar and tag set for a specific XML formatting [4]. If a document specifies a DTD and follows that DTD'S rules, it is said to be a valid XML. DTD is being replaced by more powerful XML-schema. The following section discusses various acronyms and specifications in a XML document.

```
<?xml version="1.0"?>

<dining-room>
        <table type="rectangle" wood="rosewood">
                <manufacturer>Timberland</manufacturer>
                <price>$200.00</price>
        </table>

        <chair wood="Teakwood">
                <quantity>6</quantity>
                <quality>good</quality>
        <chair>
</dining-room>
```

Figure 2. XML Document

Acronyms and Specifications in XML Document

PI

A PI in an XML document is a processing instruction [4]. A processing instruction tells an application to perform some specific task. A processing instruction is distinguished from other XML data because it represents a command to either the XML parser or a program that would use the XML document. The first line in the sample XML document is a PI.

Data Element

The elements are represented by arbitrary names and must be enclosed in angle brackets. The names of the elements must start with a letter or underscore and then may contain any number of letters, numbers, underscores, hyphens or periods [5]. They may not contain embedded spaces. Every opened element must in turn be closed. These data elements are purely user-defined. They give meaning to the textual data and helps parser parse the required information.

DTD

A DTD is a document type definition [5]. A DTD establishes a set of constraints for an XML document or a set of documents. DTD is not a specification on its own, but is defined as part of the XML specifications. A DTD defines the way an XML document should be constructed.

Namespaces

Namespaces is one of the XML-related concepts that define a mapping between an element prefix and a URI. This mapping is used for handling namespace collisions and defining data structures that allows parsers to handle collisions.

The Root Element

The root element is the highest-level element in the XML document and must be the first opening tag and the last closing tag with in the document. XML specifies that there may be only one root element within the document.

XML Schema

XML schema is designed to replace and amplify DTDS. XML schema offers XML-centric means to constraint XML documents. DTD has no knowledge of hierarchy; they have difficulty handling namespace conflicts. But XML Schema on the other hand represents the data in the form of hierarchy and using XML itself.

XML Applications

The most popular use for XML is to create a separation of content and presentation. XML can be used to represent the data content in client-server communications and is not tied to any single type of client or server. The XML solution is system and vendor independent. Above all, XML is purely text based and transmission

is easy and efficient. The transmission occurs over HTTP protocol and is not blocked by the firewalls. The above-mentioned advantage makes XML more suitable for business-to-business e-commerce.

Basic Concepts of SOAP

SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment [6]. It is an XML based protocol that consists of three parts. An envelop that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types and a convention for representing remote procedure calls and responses. SOAP codifies the use of XML as an encoding scheme for request and response parameters using HTTP as a transport. In particular, a SOAP method is simply an HTTP request and response that compiles with the SOAP encoding rules. A SOAP endpoint is simply an HTTP-based URL that identifies a target for method invocation. The project uses the SOAP format for its communications between the applications. The following diagram describes a sample SOAP document.

```
<SOAP-ENV:Envelope
        xmlns:SOAP-ENV="http://specification.soap.org/"
        <SOAP-ENV:Body>
                <m:Getprice xmlns:m="www.xyzstock.com"
                        <symbol>XA</symbol>
                        <company>COMPANYA</company>
                </m:Getprice>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 3. SOAP Document

SOAP Messages

A SOAP message is an XML document that consists of a mandatory SOAP envelope, an optional header and a mandatory SOAP body [6]. SOAP defines a simple semantics for applications to express data in a specified format and thus achieving a standard format.

SOAP Envelop

The envelope is the top-bounding element of the XML document representing the message. The grammatical rules regarding the envelope are as follows. All SOAP messages should contain the envelope data element and it should be the top element. The element may contain namespace declaration as well as additional attributes.

SOAP Header

The SOAP header data element conveys to the recipient as to how the SOAP message should be processed [6]. The Header element is if present should be present as the first immediate child element of the SOAP envelope. All immediate child elements of the SOAP Header element are called header entries. The SOAP parser looks for specific information from the SOAP header before going ahead to the next element.

SOAP Body

The SOAP Body element provides a simple mechanism for transferring required information to the recipient of the message [6]. The Body element is encoded as an immediate child element of the SOAP envelope XML element. If a header element is present then the Body element must immediately follow the header element, otherwise it must be the first immediate child element of the envelope element. The framework uses the SOAP body to enclose the XML data. The server application parses the SOAP document and looks for the enclosed XML file.

SOAP and XML Over IIOP

SOAP and XML messages can be exchanged over the HTTP protocol, as they are text based. They are mainly used to convey data or information in a meaningful and a more defined way. The above-mentioned qualities make XML and SOAP more suitable for the XML-CORBA Framework. They can easily pass through the firewall and can be used to exchange meaningful information. The SOAP and XML can use the HTTP secure transaction to ensure the safe transfer of the data.

CHAPTER 4
THE XML-CORBA FRAMEWORK DESIGN

This chapter deals with the design and the functionalities of different components in the XML-CORBA framework. The chapter provides sufficient explanation to what each component does. Extending the server-side application to the Internet involves connecting with the client systems using XML (Extensible Markup language) and HTML. The Framework connects server side CORBA with XML-based applications to. At run time, information from the XML-based application is channeled to a CORBA Wrapper or a Translator, which converts the request to equivalent CORBA calls. Then the call is directed to the service object implemented with in the ORB. The following diagram depicts the different component of the framework.

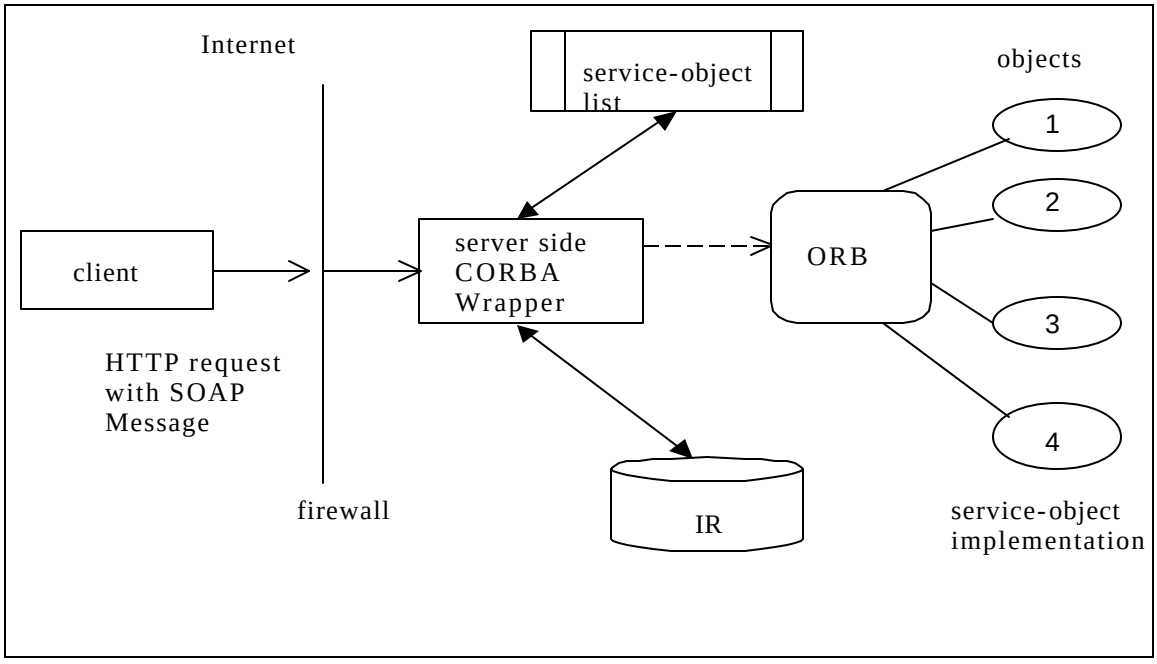

Figure 4. Different Components of the XML-CORBA Framework

Overall Design

From the figure we can note that the client is a web-enabled browser or a URL object, capable of sending HTTP request to a specified URL. The server interface is a servlet, which accepts incoming HTTP request. After capturing the HTTP request, the XML data is extracted and passed on to the translator. The translator translates the incoming XML data to equivalent CORBA call and is aptly named as CORBA wrapper. The CORBA Wrapper communicates to the Service-Object list and the Interface repository to gain more information on the object implementation. The Interface Repository is a part of the CORBA specification. Once the CORBA call is created, it is sent to the corresponding object, which provides the service. Once a result is processed, it is converted back to XML response in the SOAP format and then sent back to the client. This is a brief description of the overall design. The functionalities of the components are dealt in detail in the coming sections.

Functionality of the Components

Client

The client application can be more described as, a cross-corporate application seeking a service for some of its business requirements. The client is a web enabled browser or a simple URL object. In the present day scenario one of the popular way of communication is by using XML. XML is text based and hence it is efficient and easy to use. The XML document contains the name of the service and all the parameter required for the invoking the object. The XML document is enclosed within a SOAP messages that consist of a mandatory SOAP envelop, an optional header and a mandatory SOAP body.

CORBA Wrapper

The CORBA wrapper is the heart of the framework. The wrapper receives the XML message in the SOAP format and converts it to the equivalent CORBA equivalent call. The result obtained from the invoked object is converted to a XML message and sent back to the client. The wrapper forwards the request to the server object and can be called as CORBA client. The various components of the CORBA Wrapper are depicted in the following figure.

Servlet

The servlet component acts as the interface between Internet and the CORBA wrapper. The servlet interacts with the client and receives the HTTP request from them. The received request contains the XML document in the SOAP format. The XML document is passed on to the XML parser.

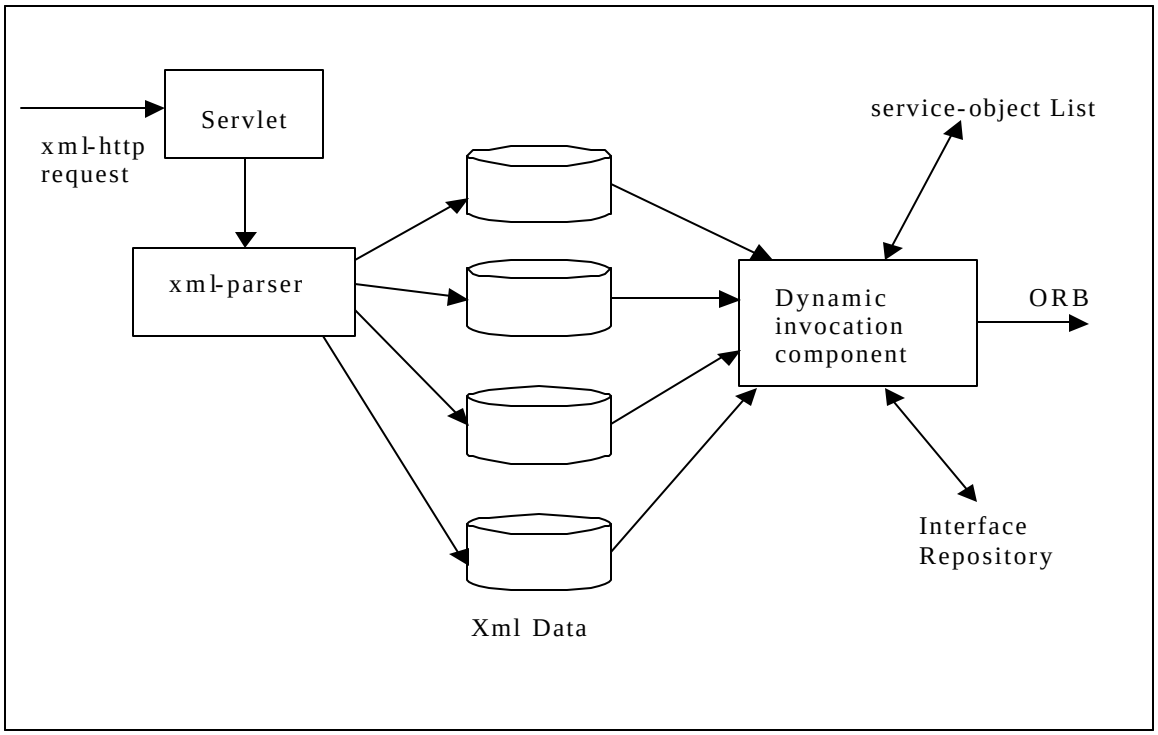

Figure 5. CORBA wrapper

XML parser

The XML parser parses the received XML document and extracts the data, to obtain the requested service and the required parameters. This component handles the important task of taking a raw XML document and making sense of the document. The parser ensures that the XML document is well formed and it would also ensure the document is valid, if a DTD or schema is referenced. The result of parsing a XML document is a tree structure, which can be easily manipulated, traversed and handled by other applications.

Dynamic invocation component

After the data is extracted from the XML document it is passed on to the dynamic invocation component. This component is responsible for creating a request object dynamically and for invoking it on the requested service. To invoke the requested service the object reference of the target object is obtained. Then a request object is dynamically created, populated with the required arguments, and then invoked on the requested service using the target reference. The dynamic invocation contrasts with the default static invocation, which requires stubs for each type of CORBA object that the client intends to invoke. In other words, a client that uses static invocation declares in advance the types of objects it will invoke. A client that uses the dynamic invocation makes no such declaration. The advantage of the dynamic invocation is flexibility it can be used to support generic clients that can invoke any object, precisely what the functionality of the Wrapper is.

Interface Repository

The Interface Repository (IR) contains the descriptions about the CORBA object interfaces. As mentioned earlier the IDL files are used to describe the interfaces that are

implemented by the remote service-objects. The data in an interface repository is the same as that in IDL files, containing descriptions of modules, interfaces, operations, and parameters [2]. These descriptions are for runtime access. The information is stored as hierarchies of objects whose methods divulge information about the interfaces. Interface repository is a part of the CORBA specification. The CORBA vendors have interface repository implemented as the part of the package. The interface files can be registered with the interface repository with special command line utilities.

Service-Object List

The client is supplied with the name of the service (method) that has to be invoked and the parameters required for the invocation. All the other details like name of the object implementing the method, location of the implementation and similarly other details are hidden from the client. The service-object list is an implementation that maps the name of the service to their Identifiers (ID), thereby helping to locate the object containing the implementation. This gives the flexibility to the client to know very little information for invoking a service.

ORB

The server objects are implemented within the ORB, hereby giving location transparency. The ORB helps in the invocation of the server objects by the client requests. The inner details are hidden from the user. The ORB is a part of the CORBA specification and is implemented by all the vendors.

Service-Objects

The server objects are implemented within the ORB. These objects are the business logics behind a corporate. They implement different services depending on the requirements of the company and its partners. There are different components that act as

intermediate between these service-objects and the ORB. A Portable Object Adapter (POA) is intermediary and routes request between the two [7]. A servant is a programming object that provides the implementation of the abstract service object. The POA determines which servant object to invoke in correspondence to the client request thus invoking the abstract implementation.

CHAPTER 5
IMPLEMENTATION OF THE FRAMEWORK

The following section deals with the implementation of the different components within the framework. The most important component of the framework is the CORBA wrapper. The other major components are ORB, XML parser, interface repository and so forth. The ORB component comes along with CORBA package. Many CORBA vendors support ORB implementation. The CORBA package also includes the implementation of interface repository, naming service and other components. Many XML parsers are available in the industry. The XML parsers are capable of validating the XML document with DTD or XML-schema.

Implementation Details of the Components

Client Side Implementation

The client is a web browser or a URL object capable of sending HTTP requests. For the framework, the client is implemented using a URL object. The java URL class has methods, which helps to connect to a server. The server connection is established using a uniform resource locator (URL). The client after connecting to the server opens an output stream and sends the XML file in SOAP format. After the transfer of the XML file, the client awaits the result from the server.

CORBA Wrapper Implementation

Servlet interface

The Wrapper has a servlet interface so that it has the capability to receive HTTP requests. The servlet implementation allows application logic to be embedded in HTTP

request-response process. The servlet API is specified in two java extension packages. They are javax.servlet and javax.servlet.http. The javax.servlet.http contains classes and interfaces specific to the HTTP. HTTP contains two methods GET and POST through which a request or response is communicated. The servlet contains implementations to process both type of request using doget and doPost methods. Since the application sends the request in POST format the doPost is invoked which handles the request.

Parsing the XML document

The XML document has to be parsed so that data can be manipulated with. It has applications, which can be used to parse the document. SAX is simple API for XML [4]. It provides an event-based framework for parsing XML data. SAX is the process of reading through the document and breaking down the data in to usable parts. The other standard application is the DOM. The DOM is an acronym for document object model [4]. While SAX only provides access to the data with in a XML document, DOM provides a representation of an XML document as a tree. Since it is represented as a tree it has the advantage of being traversed, and manipulated easily. The CORBA wrapper parses the XML document using DOM.

Implementing dynamic invocation

The dynamic invocation is achieved using a stub-less run time binding approach using, the dynamic invocation interface. To dynamically invoke, the reference of the requested target-object is required. This is obtained using the service-object list. Once the reference of the target-object is acquired then the following steps are used to invoke the method on the remote target-object

CORBA objects are introspective. They can provide the applications information about themselves. The get_interface method when invoked on the object it returns a

InterfaceDef object that fully describes this interface [7]. The InterfaceDef object is defined inside the interface repository.

Once the InterfaceDef object is obtained it can be used as an entry point for navigating all the information about the interface, the methods it supports, the parameters required and return types. The client issues a lookup_name to find the method it wants to invoke and then issues a describe call to obtain the method's full definition [7]. A describe_interface is an alternative, which gives full description of the interface and about the method that needs to be invoked.

The next issue is to create the parameter list for invoking the service on the target object. CORBA specifies a self-defining data structure for passing parameters, which it calls Named Value List. The list is implemented using an NVList pseudo object [7]. The list is created by invoking a create_list and remaining items are added using add_item calls.

The important part of the dynamic invocation is to create a request. A request is a pseudo object that contains the name of the method, the argument list, and the return value. A request is invoked using create_request and passing along with it the name of the method, arguments and return value.

The last step is the invocation of the request. This can be done using the invoke call. The invoke call sends the request object and obtains the results.

Service-Objects Implementation

The remote target-objects are implemented with in the ORB. Before the registering of the target objects the server needs to be set up as given below. The ORB has to be initialized. After the initialization of the ORB the Portable Object Adaptor (POA) has to be set up which acts as an intermediary between the ORB and the service

objects. In other words it determines which servant should be invoked when a client request is received. A servant is a programming object that provides the implementation of an abstract object or the remote service object.

Portable object adapter

Portable Object Adapters provide portability on the server side. A POA is the intermediary between the implementation of an object and the ORB. In its role as an intermediary, a POA routes requests to servants. In the ORB at least one POA must be present, which is called the rootPOA [7]. The rootPOA is created automatically as soon as the ORB is initialized. The set of POAs is hierarchical and all POAs have the rootPOA as their root parent. When an abstract object is assigned to a servant, it is called an active object and the servant is said to implement the active object. Every POA has one Active Object Map which keeps track of the object IDs of active objects and their associated active servants. An Active Object Map is a table that maps active CORBA service objects through their object IDS to servants. The object ID is a way to identify a CORBA object with in the object adapter. An object ID can be assigned by a application and should be unique with in the object adapter in which it was created. Below is a descriptive diagram of the POA, active object map, servant manager and the servant object.

Servants and servant managers

The servant managers are responsible for finding and returning the corresponding servant object which in turns invokes the remote service object. They allow the POA to activate objects when a request for an inactive object is received. Servant managers are optional. A servant is an active instance of an implementation. The Portable Object Adapters (POA) maintains a map of active servants and the object id’s of the servant.

When a client request is received, the POA first looks the map for the object id. If it is found then the servant manager is asked to locate and activate the appropriate servant.

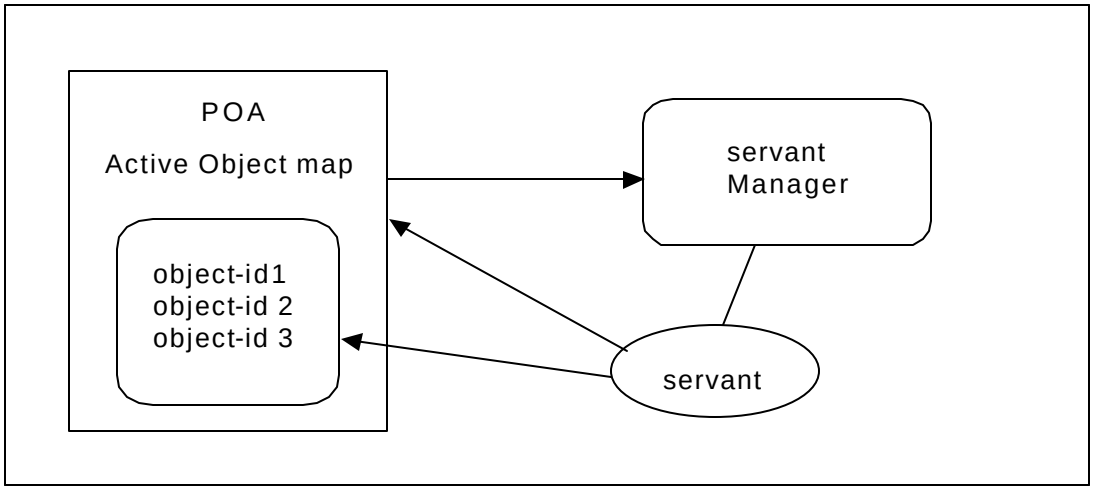

Figure 6. POA Overview

CHAPTER 6
CONCLUSION

This chapter deals with the goals accomplished in the thesis and concludes with a few ideas on future work in the same.

Goals Accomplished

The CORBA-XML framework makes it possible for a client to communicate to a target object implemented with in an ORB. It removes the hurdles of firewall; security concerns and provides better performance. The client can be a simple implementation using the HTTP protocol. The usage of the XML for transfer of data provides extensibility to the data. The framework accomplishes a way to talk to an ORB-object with the Internet as the backbone. The IIOP, which was designed for such a communication, cannot be practically used due to firewall blockages. An alternative to talk to the ORB-object in the presence of a firewall is what the thesis has accomplished.

Future Vision

The Framework provides a way for objects to interoperate, across the Internet and with in a corporate. It uses ORB for location transparency. The future e-business can be imagined as vast collection of objects, which interoperate among them-selves. They require less human intervention. The services in turn can manage a company's resource, manage their supply-chain, place orders to their suppliers, provide their clients with required information, take orders from their client, shipping them directly to the end

customers and so forth. This increases the pace of the business, the ease with which a transaction can occur and the efficient handling of the resources.

Future Work

CORBA is one of the middleware available to the industry. There are many other middleware implementations available in the market. The framework can be extended to provide compatibility with other middleware components.

LIST OF REFERENCES

[1] Anne Thomas. Web Methods B2B Integration Server white paper. http://tni.webmethods.com/, 1998.

[2] Robert Orfali and Dan Harkey. Client/Server Programming with Java and CORBA. Wiley Publications, 1998.

[3] I-Planet E-Commerce solutions. CORBA/IIOP white paper, Catching the Next Wave. http://developer.iplanet.com/docs/wpapers/corba/index.html/, 1997.

[4] Brett McLaughlin. Java and XML. O'Reilly Publications, 2000.

[5] W3C Recommendation. The XML specification. http://www.w3.org/TR/REC-xml, 2000.

[6] W3C Recommendation. The SOAP specification. http://www.w3.org/TR/SOAP, 2000.

[7] VijayKumar Natarajan, Stefan Reich and Bhaskar Vasudevan. Programming with Visibroker, A developer's guide to Visibroker for Java. Wiley Publications, 2000.

BIOGRAPHICAL SKETCH

Meenakshi Sundar is a native of Chennai, India. He earned his high school diploma from Vana Vani Matriculation, IIT-Campus, Chennai. He earned his Bachelor of Engineering degree in the field of Computer Science at Sathyabama College of Engineering, University of Madras, India. In May 1999 Meenakshi Sundar earned his bachelor's degree and graduated with distinction. He immediately went on to pursue his master's at the University of Florida, Gainesville in the field of Computer Engineering. He likes the game of cricket, a popular game played in India.