

CORBA Compare to SOAP/Web Services

Table of Content

Abstract	3
Introduction	4
1. SOAP	5
1.1. SOAP and Firewalls	6
1.2. The Payload Format	6
1.3. Envelope and Types	6
1.4. Array Types	7
1.5. Structure Types	7
1.6. Enumeration	7
1.7. Response Example	8
1.8. SOAP and Legacy Systems	8
1.9. SOAP vs. XML- RPC	8
1.10. Web service technology success stories	9
2. CORBA	10
2.1. CORBA ORB Architecture	11
2.2. The Notion of Client- Server	12
2.3. Object Request Broker (ORB)	12
2.4. ORB Interface	12
2.5. Interface Definition Language (IDL)	12
2.6. CORBA IDL stubs & skeletons	12
2.7. Built- in Types and other Features	12
2.8. Dynamic Invocation Interface (DII)	13
2.9. Object Adapter	13
2.10. Dynamic Skeleton Interface (DSI)	13
2.11. Communication Scenario	13
2.12. CORBA success stories	13
3. CORBA and Web services: Comparative Analysis	14
Conclusion	16
Recommendation	16
Glossary	17
References	18
Appendix	20

Abstract

The convergence of Internet with EDI (Electronic Data Exchange) and standard middleware like CORBA and technological changes has given rise to a new computing paradigm called SOAP/Web Services. Direct interaction between computers is now possible due to the rapid development of XML and SOAP technologies. There is, however, a lack of precise definition about Web Services and its capabilities compare to a well-defined middleware technologies such as CORBA.

In this report, we first evaluate the two technologies comparing and contrasting the features offered by each. Second, we provide the analysis on the circumstances in which these technologies should be used. Finally, we recommend the best one for a distributed computing system.

Introduction

"The Internet was initially designed for unstructured information exchange but rapidly grew in use as a mechanism for e-Business."(www2002.org) However, there was no uniform mechanism for accessing electronic services. This implied that interoperability between services was little if not non-existent. This was due largely to the fact that the Web content/services had been designed primarily for human use rather than for machine consumption.

To facilitate automated access to complex services, a group of companies led by Microsoft and IBM (and now being handled by the XML Protocol Activity group under W3C) standardized on **SOAP** (W3C, SOAP) as a lightweight protocol based on XML for exchanging messages over the Web. Similar efforts are underway to define higher level service layers such as **WSDL** (W3C, WSDL) and **UDDI** (UDDI).

The most popular distributed application frameworks are **COM** and **DCOM**. These are Microsoft specific, **EJB** which is Java specific, and **CORBA** which is both platform and language independent. CORBA in particular has found success as a distributed component framework in a number of areas ranging from telecommunications, finance, e-commerce, healthcare, to the graphical user interface of Linux desktop (Miguel de Icaza and Jonathan Blandford, 1999).

A key difference between CORBA and the Web service technologies (UDDI/WSDL/SOAP) is that CORBA provides true object-oriented component architecture unlike the Web services, which are primarily message based (SOAP, despite its name, does not really deal with objects). Moreover, CORBA also comes with a standard set of services (Events, Naming, Trading) that allow application writers to focus on the business logic rather than on the details of the communication infrastructure.

In this report, we will present both technologies and see how they can be compared. We will argue that these two technologies should actually be used in concert. Furthermore, we will help define the environment where each plays best.

This report is organized as follows. Section 1 and 2 gives a brief overview of Web services and CORBA along with application domains where these technologies have been successful. From these application scenarios we attempt to extract some key features that a complex distributed service must possess, and compare the two technologies along these dimensions in Section 3. Conclusions are presented in Section 4 and finally our recommendation.

1. SOAP

The success of **XML-RPC** has led to the design of a new flexible lightweight protocol for exchange of information in a distributed environment. **SOAP** is XML-based and uses the HTTP framework. Services are described in terms of the messages accepted and generated. Users of such services do not need to know anything about the details of the implementation (object model, programming language, etc.); they only need to be able to send and receive messages.

The SOAP specification includes: (1) a syntax to define messages (envelope, with optional header and body), (2) encoding/serialization rules for data exchange, and (3) conventions for representing RPCs.

The SOAP message exchange model can be defined as follows. Upon receiving a message, the local application must:

1. Identify the parts of the message intended for it,
2. Verify that the parts from step 1 are supported by the local application and process them accordingly (if not, the message is discarded),
3. If the local application is not the final destination, the parts from step 1 have to be removed before the message is forwarded to its final destination.

The specifications (i.e., interface) of services can be described using **WSDL** (Web Services Description Language). WSDL is a general framework (based on XML) for describing network services as collections of communication endpoints capable of exchanging messages. It describes where a service is located, what operations are supported, and the format of the messages to be exchanged based on how the service is invoked. WSDL does not mandate a specific communication protocol used (it supports various bindings such as SOAP and HTTP).

The Web service vision foresees a proliferation of services which in turn requires the availability of public directories that can be used for the registration and lookup of services. **UDDI** provides a mechanism for service providers to advertise their services in a standard form and for service consumers to query services of interest, thereby paving the way for interoperability between services. A **UDDI** entry consists of white pages (e.g., address, contact information), yellow pages (e.g., industrial characterization based on standard ontologies), and green pages (e.g., references to specifications of services). **UDDI** is itself implemented as a Web service using **SOAP** as the message protocol.

High-level component architecture for Web services is presented in Figure 1.

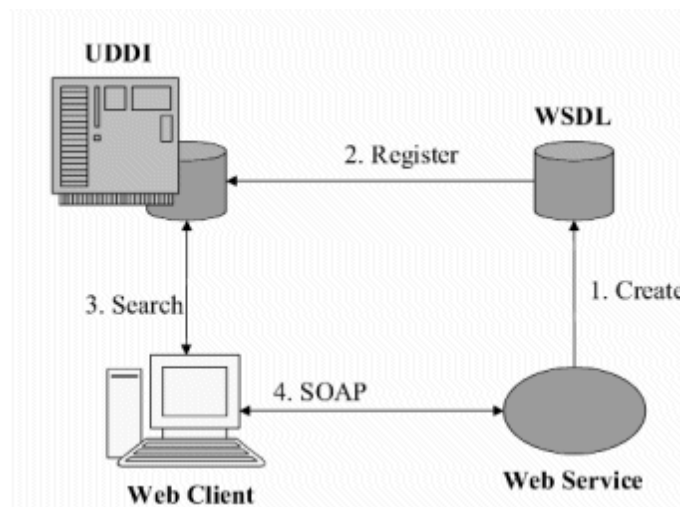


Figure 1. Components in a Web Service

1.1. SOAP and Firewalls

Since SOAP is piggybacking HTTP, it is particularly effective at passing through firewalls. Firewalls prevent many RPC and interprocess communications protocols from reaching their destinations. Most corporate firewalls allow web browsing which allows SOAP to slip through transparently.

1.2. The Payload Format

A SOAP request is an XML document. It contains an envelope **<SOAP: Envelope>** which contains an optional header element **<SOAP: Header>** and a mandatory body element **<SOAP: Body>**. The order is important. The header is a mechanism for adding features to SOAP messages without prior agreement between communication parties. Often used for intermediaries on the message path.

1.3. Envelope and Types

<SOAP: Envelope> must include the following three standard namespace declarations:

```
xmlns: xsi="http://www.w3.org/1999/XMLSchema/instance"
xmlns: xsd="http://www.w3.org/1999/XMLSchema/instance"
xmlns: SOAP="urn: schemas- xmlsoap- org: soap. v1"
```

As we shall see, other declarations are also possible.

- SOAP supports all the simple types and types derived from simple types defined in W3C XML Schema Primer. (<http://www.w3.org/TR/xmlschema-0/#CreatDt>)
- The data type is passed in the "xsd: type" attribute.
- NULL values are indicated with an attribute "xsi: null= 'true'".

- If a simple type does not include an "xsd: type" attribute, then it's up to the application to determine the data type based on the needs of the application.

1.4. Array Types

- Array types use the attribute “**SOAP: arrayType=‘ item[]‘**” to indicate that the contents of the array element is an array of <item> elements.

```
<stuff SOAP: arrayType= 'item[] '>
  <item xsd: type=" integer"> 42</ item>
  <item>
    <x xsd: type=" float"> 100</ x>
    <y xsd: type=" float"> 200</ x>
  </ item>
  <item xsd: type=" string"> All is well.</ item>
  <item xsi: null=" true"/>
</ stuff>
```

- Arrays can contain simple types, structures, or arrays, and they can be mixed as shown above.

1.5. Structure Types

- Structure Types contains elements for each member of the structure. The names of the elements are the names of the members.

```
<corner>
  <x xsd: type=" float"> 100</ x>
  <y xsd: type=" float"> 200</ y>
</ corner>
```

- Structures can contain simple types, structures, or arrays.

1.6. Enumeration

- Enumerations: set of distinct names

```
<element name=" EyeColor" type=" tns: EyeColor"/>
  <simpleType name=" EyeColor" base=" xsd: string">
    <enumeration value=" Green"/>
    <enumeration value=" Blue"/>
    <enumeration value=" Brown"/>
  </ simpleType>
<Person>
  <Name> Henry Ford</ Name>
  <Age> 32</ Age>
  <EyeColor> Brown</ EyeColor>
</ Person>
```

1.7. Response Example

```
POST http://soap1.develop.com/cgi-bin/ServerDemo.pl?class=Geometry HTTP/ 1.0
SOAPMethodName: http://soap1.develop.com/cgi-bin/ServerDemo.pl? class=Geometry#
calculateArea
Host: soap1.develop.com
Content-Type: text/xml
Content-Length: 494
<? xml version=" 1.0"?>
```

```
<SOAP: Envelope
xmlns: xsi="http://www.w3.org/1999/XMLSchema/instance"
xmlns: xsd="http://www.w3.org/1999/XMLSchema/instance"
xmlns: SOAP=" urn: schemas- xmlsoap- org: soap. v1">
<SOAP: Body>
  <calculateArea>
    <origin>
      <x xsd: type=" float"> 10</ x>
      <y xsd: type=" float"> 20</ y>
    </ origin>
    <corner>
      <x xsd: type=" float"> 100</ x>
      <y xsd: type=" float"> 200</ y>
    </ corner>
  </ calculateArea>
</ SOAP: Body>
</ SOAP: Envelope>
```

Example from lwprotocols.Org

1.8. SOAP and Legacy Systems

XML and SOAP are in many ways to integrate new systems with legacy systems. Creating a wrapper around legacy databases or legacy systems in general establishes a layer of indirection for system integration. The wrapper communicates with SOAP between the XML world and the legacy database.

1.9. SOAP vs. XML-RPC

XML-RPC is simpler but less flexible. SOAP is more efficient in particular for requests. Many applications already use XML-RPC since it came before SOAP. However many are migrating to SOAP. There are object libraries already written for SOAP (in Java). These make the implementation even simpler by hiding the protocol:

- [http:// www.alphaworks.ibm.com](http://www.alphaworks.ibm.com)
- <http://develop.com/SOAP/>

1.10. Web service technology success stories

A listing of web services available on <http://www.xmethods.com> shows an abundance of available services, but with most of them being very simple, such as a stock quote service, or a Celsius to Fahrenheit conversion service.

However, in anticipation of rapid growth in Web services, a number of companies are developing the basic infrastructure pieces required for rapid deployment of such services, ranging from base technologies such as XML parsers, XML Schema validators, to complex services such as UDDI directories, etc. Companies are also building web application servers that provide a unified platform integrating UDDI, WSDL, and SOAP support (e.g., IBM's WebSphere).

Microsoft in particular is moving heavily into .NET, which is a framework for building distributed Web services. An instantiation of that framework is named .NET My Services (formerly called HailStorm) which includes a list of services such as Profile (name, address information), Contacts (address book), Inbox (email, voice mail access), Calendar, Wallet (receipts, coupons and other transaction records), etc. One goal is that the availability of such standardized services will make it easy for them to be used as components of a larger application.

2. CORBA

The Common Object Request Broker Architecture (**CORBA**) is an open standard for distributed object computing defined by the Object Management Group (OMG, July 95). CORBA is an object bus enabling the client to invoke methods on remote objects at the server independent of the language the objects have been written in, and their location. The interaction between client and server is mediated by object request brokers (**ORBs**) on both the client and server sides, communicating typically via IIOP (Internet Inter-ORB Protocol) (David Alpher, Jan. 2001). CORBA objects can be either collocated with the client or distributed on a remote server, without affecting their implementation or use.

The details are taken care of by the ORBs. The capabilities of CORBA objects (operations or methods) are defined using the Interface Definition Language (IDL).

Operations defined on the interface accept input parameters and return values (both corresponding to some CORBA data-types) and can raise exceptions. The implementation languages supported by CORBA include C, C++, Java, Ada95, COBOL as well as some scripting languages such as Perl, Python, Javascript. Furthermore, CORBA is designed to be independent of the OS and runs on many OS platforms, including Win32, UNIX and real-time embedded systems. Moreover, the communication protocols used by CORBA for ORB communication include TCP/IP, IPX/SPX, ATM, etc.

Figure 2 illustrates the components in the CORBA 2.x reference model, all of which collaborate to provide the portability, interoperability and transparency outlined above.

(Steve Vinoski, July/August 1993 & October 1998).

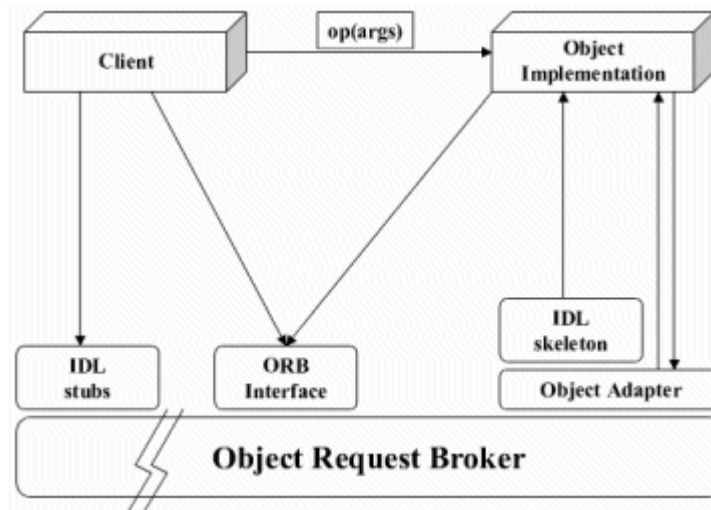


Figure 2. Components in the CORBA 2.x Reference Model

The typical life cycle of a CORBA application is as follows:

1. Define the service as interfaces in IDL,
2. Compile the IDL to generate client stub and server skeletons,

3. Implement the service and associate it with the skeletons via the portable object adapter (POA) shown the Figure 2,
4. Publish the service with a Naming or Trading Service for use by clients.

The CORBA client processing involves the following:

1. Contact the Naming Service for the desired service and retrieve the appropriate object reference,
2. Invoke operations on the object reference using the IDL-compiler generated stubs. Alternatively, clients can infer the operations supported by the service by consulting an interface repository (IR) and dynamically create requests populating them with the appropriate parameters using the dynamic invocation interface (DII),
3. Process incoming reply or exceptions.

CORBA also defines a set of application domain-independent services specification called the Common Object Services Specification (COSS) (Object Management Group, December 1998). These services are useful when building applications based on distributed objects and include the Naming, Trading Object, Security, Property, Persistence, Transaction, Event, and Life Cycle services, among others.

2.1. CORBA ORB Architecture

A CORBA programming entity that consists of an identity, an interface, and an implementation the program entity that invokes an operation on an object implementation. Object Management Group. (The Common ORB: Architecture and Specification, September 2001.)

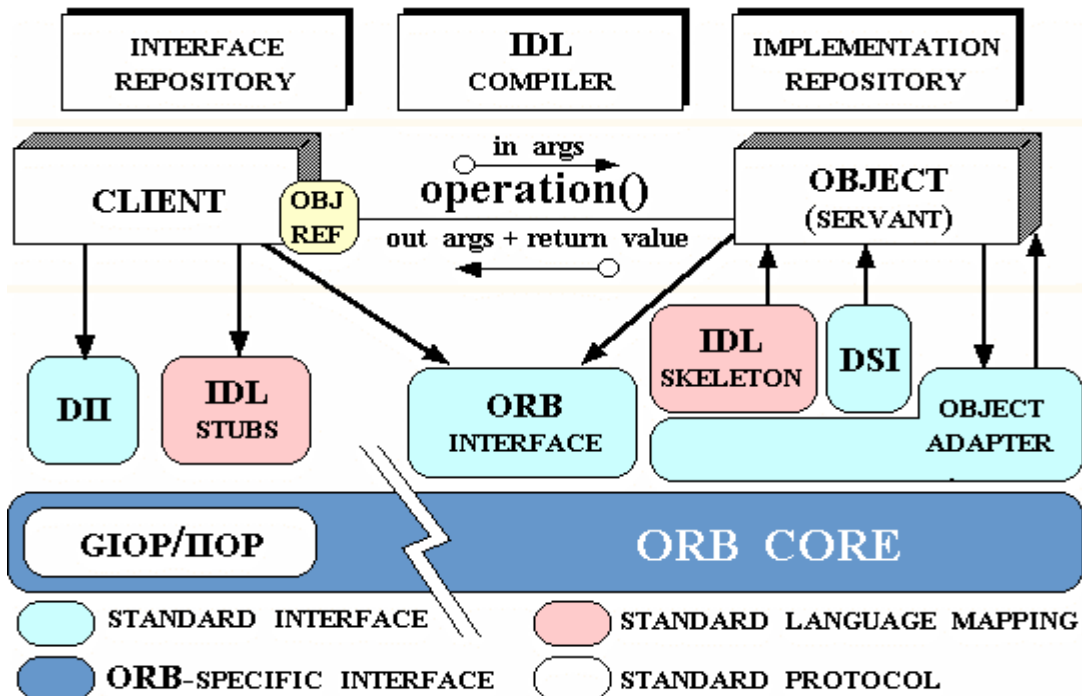


Figure 3. CORBA ORB Architecture

2.2. The Notion of Client- Server

With CORBA there is not rigid notion of a client and a server; components communicate with others on a peer- to- peer basis. Client and server are roles filled on a per-request basis. A component can be a client and a server at the same time: client for other services and server for the services it provides.

2.3. Object Request Broker (ORB)

The ORB provides a mechanism for transparently communicating client requests to target object implementations. The ORB simplifies distributed programming by decoupling the client from the details of the method invocations. This makes client requests appear to be local procedure calls. When a client invokes an operation, the ORB is responsible for finding the object implementation, transparently activating it if necessary, delivering the request to the object, and returning any response to the caller.

2.4. ORB Interface

An ORB is a logical entity that may be implemented in various ways (such as one or more processes or a set of libraries). To decouple applications from implementation details, the CORBA specification defines an abstract interface for an ORB. This interface provides various helper functions such as converting object references to strings and vice versa, and creating argument lists for requests made through the dynamic invocation interface described later.

2.5. Interface Definition Language (IDL)

IDL is a declarative language that defines types of objects by separating interfaces, named operations and parameters. This is a means by which the object implementation tells clients what operations are available and how to invoke them. IDL is mapped to a particular language. When compiled, it produces stubs and skeletons

2.6. CORBA IDL stubs & skeletons

CORBA IDL (Interface Definition Language) stubs and skeletons serve as the “glue” between the client and server applications, respectively, and the ORB. CORBA IDL compiler (Language Mappings) automates the transformation between CORBA IDL definitions and the target programming language. The use of a compiler reduces the potential for inconsistencies between client stubs and server skeletons and increases opportunities for automated compiler optimizations.

2.7. Built- in Types and other Features

The built in types are: Long (32 bit), long (64 bits), short (16 bits) float, double, long double, char, octet, Boolean, enum. The constructed types can be done with *struct*. Template types: string (string< n>), sequence (sequence< string, n>), fixed (fixed< 5, 2>). Object references. Interface Inheritance.

2.8. Dynamic Invocation Interface (DII)

This interface allows a client to directly access the underlying request mechanisms provided by an ORB. Applications use the DII to dynamically issue requests to objects without requiring IDL interface- specific stubs to be linked in. Unlike IDL stubs (which only allow RPC- style requests), the DII also allows clients to make non- blocking deferred synchronous (separate send and receive operations) and one-way (send- only) calls.

2.9. Object Adapter

This assists the ORB with delivering requests to the object and with activating the object. More importantly, an object adapter associates object implementations with the ORB. Object adapters can be specialized to provide support for certain object implementation styles (such as OODB object adapters for persistence and library object adapters for non-remote objects). An implementation must be registered with the OA. The OA maps requests to the appropriate implementation.

2.10. Dynamic Skeleton Interface (DSI)

This is the server side's analogue to the client side's DII. The DSI allows an ORB to deliver requests to an object implementation that does not have compile- time knowledge of the type of the object it is implementing. The client making the request has no idea whether the implementation is using the type- specific IDL skeletons or is using the dynamic skeletons.

2.11. Communication Scenario

IDL compilation results in stubs (client- side) and skeletons (server-side). Client gets an object reference, and makes requests using it. ORB translates the request into a form suitable for transmission. OA assists ORB in determining the exact implementation. IFR helps ORB do appropriate type checking. The IFR (Interface Repository) is a service that provides persistent objects that represent IDL information in a form available at run- time (it provides type info to issue requests). ORB decodes request, passes it on to the skeleton. Skeleton services request, stub and skeleton pass it back to the client (along with exception info, if any).

2.12. CORBA success stories

CORBA has been deployed in a wide range of industries including aerospace and defense, banking and finance, chemical/petrochemical, consulting, education, electronic commerce, Government, healthcare and insurance, human resources, manufacturing, publishing and multimedia, real estate, research, retail, software/hardware, telecommunications, transportation, and utilities (see Table 1). We now give brief descriptions of two of those selected applications from the Telecommunications domain (For details of CORBA success stories, refer to <http://www.corba.org>).

Domain	Sample Application
<i>Banking & Finance</i>	Online Account Access
	Online Bill Payment
	Stock Trading
<i>E-commerce</i>	Online Shopping
<i>B2B</i>	Supply Chain Networks
<i>Healthcare</i>	Insurance Claim Handling system
	Hospital Patient Record Management Systems
<i>Telecommunications</i>	Service Provisioning
	Network Management
<i>Enterprise</i>	Virtual Customer Care Center
<i>Entertainment</i>	Pay-per-view Subscription Service

Table 1. Examples of CORBA-based Applications

Telecommunications Service Provisioning:

Telecommunication equipment vendors such as Lucent and Nokia are using CORBA to develop and produce telecommunication products enabling service providers to rapidly create, deploy, and manage value added services based on a common Intelligent Network (IN) architecture. Such products need to communicate with a large number of disparate telephony network elements. Thus, there is a need to use an extensible and flexible integration technology, which is provided by CORBA.

Network Management:

Long distance carriers such as Sprint have adopted high-efficiency object technology to manage its worldwide network. The network comprises large amount of equipment such as routers, hubs, switches, etc. running on several different hardware and software platforms. In order to have a single, distributed integrated system that can manage all these equipment, object technology such as CORBA is required. In addition, CORBA allows reuse, modular construction, and reduced development cycle times compared to other technologies.

3. CORBA and Web services: Comparative Analysis

This section compares the CORBA and Web service technologies based on two different aspects. First, we provide comparisons based on the computing model. Next, we compare them based on the features supported by each technology. (Object Management Group. OMG XMLDOM: DOM/Value Mapping Specification, April 2001.)

Please refer to Table 2 Comparison between CORBA and Web services in appendix.

One important observation concerning CORBA and Web Services is that whatever can be accomplished by CORBA can be accomplished using Web service technologies and vice versa, although the amount of effort required would be noticeably different. In particular, one can implement CORBA on top of SOAP, or SOAP on top of CORBA.

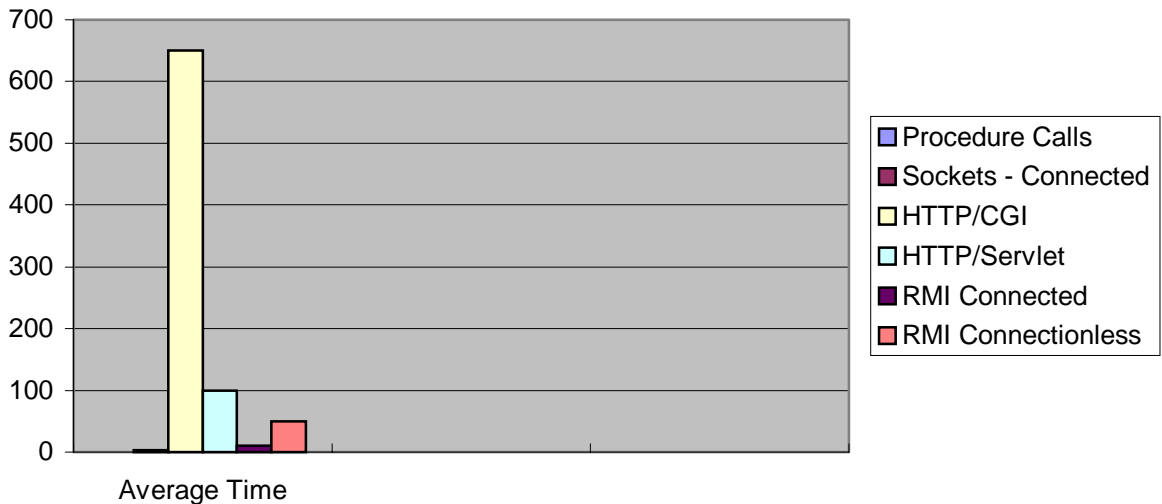
Table 2 provides an overview of comparisons between the two technologies along several architectural dimensions. Table 3 on the other hand provides a high-level comparison of the technology stack comprising the layers that come into play when building a distributed service.

CORBA stack	Web Services stack
IDL	WSDL
CORBA Services	UDDI
CORBA Stubs/Skeletons	SOAP Message
CDR binary encoding	XML Unicode encoding
GIOP/IOP	HTTP
TCP/IP	TCP/IP

Table 3. CORBA and Web services technology stacks

Table 4 is providing performance test data among different protocol. (Dave Orchard, September 1997).

Access Type	Average Time (ms)
Procedure Calls	0.01
Sockets - Connected	3
HTTP/CGI	650
HTTP/Servlet	100
RMI Connected	10
RMI Connectionless	50
CORBA IIOp - Connected	6
CORBA IIOp - Connectionles	50
COM	1
DCOM	10



Graph of table 4

Conclusion

It's amazing that many in the SOAP community have basically dismissed CORBA/IIOP as being too "difficult" to use, and not suitable for the Internet. They state that XML is so much easier to use, but yet they don't actually expose the XML to the programmer. Instead, they develop incompatible, proxy-based frameworks based on the same designs as the CORBA vendors. CORBA is heavy-duty because distributed applications appear to require that. Once the Web Services community discovers all the missing pieces, Web Services will be quite heavy duty also. Depending on the framework selected, SOAP programming is no easier, and possibly more work than the equivalent CORBA approach. And regardless of the framework selected, the resultant SOAP code *is not portable* between frameworks.

In the long run, the hype about Web Services will diminish, and CORBA and Web Services will both have their place. Just as CORBA and COM, and for that matter RMI, coexist. When necessary, bridges will be used to tie everything together. It looks like we're going to use SOAP to talk to the outside world, and we'll be using CORBA in the inside. In the short term, it will take a number of years for SOAP to mature, and the *missing* pieces to be defined.

Some argue that a thorough understanding of Software Architecture would provide you with the true benefits and disadvantages of the platforms and technologies of the past, the present and the future. Those people will be few because this requires great skill and expertise. Also, we developers are so busy continuously reinventing the wheel that there rarely is time to actually evaluate the architectural aspects of a system and learn from them.

Some may find that there isn't a single thing that could be called *innovation* in the entire SOAP and Web Services area. It's just repackaging of already solved problems and, unfortunately, with many mistakes of the past being repeated. Although being a strong statement, it certainly has some truth in it. Many "technologies" are driven more by marketing, hype and revenue rather than sound technical decisions that would encourage incremental progress.

Recommendation

SOAP and CORBA are not exclusive but rather should be seen as complementary technologies that need to coexist.

Glossary

Component Object Model (COM)
Common Object Services Specification (COSS)
Common Object Request Broker Architecture (CORBA)
DCOM (Distributed COM)
Dynamic Invocation Interface (DII)
Dynamic Skeleton Interface (DSI)
Electronic Data Exchange (EDI)
Enterprise Java Beans (EJB)
Hypertext Transport Protocol (HTTP)
Interface Definition Language (IDL)
Interface Repository (IFR)
Internet Inter-ORB Protocol (IIOP)
Intelligent Network (IN)
Object Management Group (OMG)
Object Request Broker (ORB)
Remote Procedure Call (RPC)
Simple Object Access Protocol (SOAP)
Universal Description, Discovery and Integration (UDDI)
Web Service Description Language (WSDL)
eXtensible Markup Language (XML)

References

<http://www2002.prg/CDROM/alternate/395/>

David Alpher. The application server state of the union, January 2001, from http://e-serv.ebizq.net/aps/alpher_2.html

Miguel de Icaza and Jonathan Blandford. Components in the GNOME project, 1999, from <http://developer.gnome.org/doc/whitepapers/Components/>.

Common Object Request Broker Architecture, OMG, July 1995.

Steve Vinoski. Distributed Object Computing with CORBA. C++ Report, 5(6), July/August 1993.

Steve Vinoski. New Features for CORBA 3.0. Communications of the ACM, 41(10):44--52, October 1998.

W3C. SOAP, Simple Object Access Protocol, from <http://www.w3.org/2000/xp/>.

W3C. WSDL, Web services description language from <http://www.w3.org/TR/wsdl/>.

UDDI. From <http://www.uddi.org/>.

Object Management Group. CORBA Services: Common Object Services Specification, Updated Edition, 95-3-31 edition, December 1998.

Elliot Lee. CORBA applications in GNOME.

Object Management Group. CORBA Messaging Specification, OMG Document orbos/98-05-05 edition, May 1998.

Object Management Group. OMG XMLDOM: DOM/Value Mapping Specification, ptc/01-04-04, ptc/01-04-04 edition, April 2001.

Object Management Group. The Common Object Request Broker: Architecture and Specification, 2.5 edition, September 2001.

OMG. CORBA web services. TC Document orbos/2001-06-07.

OMG. Simple CORBA object access protocol (SCOAP). From <ftp://ftp.omg.org/pub/docs/orbos/00-09-03.pdf>.

Douglas Schmidt and Steve Vinoski. Object interconnections: CORBA and XML -- three part series. In *C/C++ Users Journal*, 2001.

Rogue Wave Software. XML CORBA. From
<http://www.roguewave.com/products/xml/xorba/>.

Dave Orchard, Sep 9 1997 from
<http://pacificspirit.com/Courses/SIGSSep97pacificspirit.com/daveshome.html>

Appendix

Table 2. Comparison between CORBA and Web services

Aspect	CORBA	Web services
Data model	Object model	SOAP message exchange model
Client-Server coupling	Tight	Loose
Location transparency	Object references	URL
Type system	IDL	XML schemas
	Static + runtime checks	Runtime checks only
Error handling	IDL exception	SOAP fault messages
Serialization	Built into the ORB	Can be chosen by the user
Parameter passing	By reference	By value (no notion of objects)
	By value (<i>value type</i>)	
Transfer syntax	CDR used on the wire	XML used on the wire
	Binary format	Unicode
State	Stateful	Stateless
Request semantics	At-most-once	Defined by SOAP
Runtime composition	DII	UDDI/WSDL
Registry	Interface Repository	UDDI/WSDL
	Implementation repository	
Service discovery	CORBA naming/trading service	UDDI
	RMI registry	
Language support	Any language with an IDL binding	Any language
Security	CORBA security service	HTTP/SSL, XML signature
Firewall Traversal	Work in progress	Uses HTTP port 80
Events	CORBA event service	N/A