Reinventing the Wheel? CORBA vs. Web Services

Aniruddha Gokhale Bharat Kumar Arnaud Sahuguet

Bell Labs, Lucent Technologies
101 Crawfords Corner Road
Holmdel, NJ 07733

Bell Labs, Lucent Technologies
600 Mountain Ave
Murray Hill, NJ 07974

{agokhale,bharat,sahuguet}@lucent.com
Off. (732) 949-5979, (908) 582-5487, (908) 582-6491

Abstract

Technological advances in and convergence of the World Wide Web with Electronic Data Interchange and standard middleware such as CORBA has given rise to a new computing paradigm based on a loosely coupled service-oriented architecture called Web Services. Direct machine to machine interactions that were hitherto deemed infeasible are now possible due to the rapid technological advances in XML and SOAP technologies. There is, however, an unprecedented hype surrounding this new paradigm. In reality, the Web services paradigm lacks a precise definition. Furthermore, the use-case scenarios for Web services and its pros and cons compared to existing, well-defined middleware technologies such as CORBA are not well understood.

This paper provides three contributions to the study of Web services and the role played by the emerging XML/SOAP standards as well as existing, integration technologies such as CORBA. First, we evaluate the two technologies comparing and contrasting the features offered by each. Second, we provide our analysis on the circumstances in which these technologies should be used. We argue that most often these technologies need to coexist. Finally, we derive examples from the telecommunications' Intelligent Networking domain to support our conclusions.

Keywords: Web services, XML, SOAP, WSDL, UDDI, CORBA, IDL, distributed, components, objects

Approximate word count: 8000 words

1 Introduction

The World Wide Web was initially designed for unstructured information exchange but rapidly grew in use as a mechanism for e-commerce. However, there was no uniform mechanism for accessing e-services, and each service exposed its interfaces in an ad-hoc manner. 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 (Simple Object Access Protocol [20]) as a light-weight protocol based on XML for exchanging messages over the Web. Similar efforts are underway to define higher level service layers such as WSDL (Web Service Description Language [21]) and UDDI (Universal Description, Discovery and Integration [16]). Now, *Web Services* are being touted as a mantra to solve all e-commerce ills.

Distributed application frameworks required to build complex services have been around for a while. Popular ones are (or have been) COM (Component Object Model), DCOM (Distributed COM), and COM+ which are Microsoft specific, EJB (Enterprise Java Beans) which is Java specific, and CORBA (Common Object Request Broker Architecture) 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, health-care, to the graphical user interface of your Linux desktop (GNOME [2]).

A key difference between CORBA and the Web service technologies (UDDI/WSDL/SOAP) is that CORBA provides a 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. Comparing this with the ongoing work on Web service technologies, we are forced to ask the question: *are we reinventing the wheel?*

Another question of great importance to the authors' employer arises from the fact that the telecommunications industry is looking at ways to rapidly create new services to satisfy customer demand and to generate new revenues. As a result, one important area of interest is the convergence between traditional telephony services and Web services. The telecommunication industry would like to leverage existing/new Web services and take advantage of the new technologies being developed as part of the Web services revolution in order to provide new and innovative applications to the user. For instance, a cell-phone user would like to know the names and locations of the best restaurants in his or her immediate neighborhood. By combining telephony location services with yellow pages (e.g. `yahoo.com`) and restaurant ratings (e.g. `zagat.com`), such a service could be offered. Therefore, another question that arises is: *how should this convergence be achieved.*

In this paper, we provide some arguments to answer the questions raised above. More precisely, we will present both technologies and see how they can be compared. In the process, we will also try to debunk a few myths surrounding them. 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 paper is organized as follows. Section 2 gives a brief overview of CORBA and Web services 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. Section 4 examines some application scenarios and discusses which of these two technologies would be more appropriate in each of those. Section 5 examines how the two technologies may interplay to provide a solution and discusses this in the context of a sample application from the telecommunications area. Related work and discussions are presented in Section 6. Conclusions are presented in Section 7.

2 Overview

This section provides a brief overview of the CORBA and Web services technologies. We refer the reader to the corresponding specifications for further details.

2.1 CORBA

The Common Object Request Broker Architecture (CORBA) is an open standard for distributed object computing defined by the Object Management Group (OMG). 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).* 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

*The Extensible Transport specification allows ORBs to use protocols in addition to IIOP, however, IIOP support is mandatory.

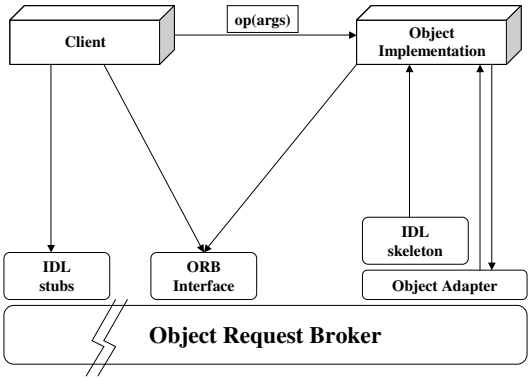

Figure 1: Components in the CORBA 2.x Reference Model

communication protocols used by CORBA for ORB communication include TCP/IP, IPX/SPX, ATM, etc.

Figure 1 illustrates the components in the CORBA 2.x reference model, all of which collaborate to provide the portability, interoperability and transparency outlined above. A good description of the various components of the CORBA reference model and new features included in the CORBA component model (CCM) can be found in [18, 19].

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 1,
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) [7]. These services are useful when building applications

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

Table 1: Examples of CORBA-based Applications

based on distributed objects and include the Naming, Trading Object, Security, Property, Persistence, Transaction, Event, and LifeCycle services, among others.

2.1.1 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[†].

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

[†]For details of CORBA success stories, we urge the reader to refer to http://www.corba.org

such as CORBA is required. In addition, CORBA allows reuse, modular construction, and reduced development cycle times compared to other technologies.

2.2 Web Services

Web services are an emerging distributed middleware technology that uses a simple XML-based protocol to allow applications to exchange data across the Web. 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.

At the core of Web services is the Simple Object Access Protocol (SOAP), an XML based communication protocol for interacting with Web services. 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 used for the registration and lookup of services. UDDI (Universal Description, Discovery and Integration) 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.

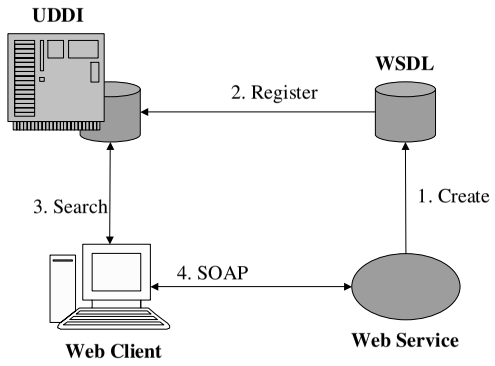

Figure 2: Components in a Web Service

A high level component architecture for Web services is presented in Figure 2.

2.2.1 Web service technology success stories

The Web service revolution is still in its early stages. A listing of web services available on `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 Hail-Storm) which includes a list of services such as Profile (name, address information), Contacts (address book), Inbox (email, voice mail access), Calender, 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.

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.

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 static + runtime checks	XML schemas 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 (*valuetype*)	by value (no notion of objects)
Transfer syntax	CDR used on the wire binary format	XML used on the wire Unicode
State	stateful	stateless
Request semantics	at-most-once	defined by SOAP
Runtime composition	DII	UDDI/WSDL
Registry	Interface Repository Implementation repository	UDDI/WSDL
Service discovery	CORBA naming/trading service RMI registry	UDDI
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

Table 2: Comparison between CORBA and Web services

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.

In the following sections, we compare both technologies along the dimensions specified in Table 2. Furthermore, for each of these characteristics, we identify the pros and cons of both approaches.

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

Table 3: CORBA and Web services technology stacks

3.1 Distributed computing models in CORBA and Web services

Below we outline the differences between CORBA and Web services based on their computing model and how they deal with complexities arising out of distribution.

Data Model: One important distinction that comes into play when examining CORBA and Web services is how an application is modeled in either case. CORBA is a true object-oriented component framework, whereas Web services are centered around a message passing paradigm with no notion of objects. As mentioned earlier, SOAP, despite its name, does not deal with objects. There was a belated effort to reassign the name SOAP to mean Service Oriented Access Protocol, but the XML Protocol Activity Group did not like this acronym first, description later approach. So in the latest W3C working drafts, it is simply called SOAP 1.2, with no explanation of what the name means.

In CORBA, there is a tight coupling between the client and the server. First, both must share the same interface – with a stub on the client-side and the corresponding skeleton on the server-side – and must run an ORB at both ends. Second, the interaction between client and server can be done directly, with no need for further intermediation (except from the ORB of course). The client obtains a handle to a CORBA object and applies a method on the it. The result of the call is possibly another CORBA object on which it can apply further methods.

In Web services, everything is decoupled. The client sends a message and receives a message. The response does not give an immediate access to the next step.

Request semantics: In order to maintain data consistency, the infrastructure should provide *at most once* semantics. These semantics guarantee against multiple executions of the same client request on the server.

CORBA ORBs are required to provide *at most once* semantics, thus ensuring data integrity. CORBA also specifies a Transaction Service that provides support for multiple transaction models such as *flat* and *nested*.

In Web services, the message semantics are defined by the protocol underlying SOAP (e.g., by HTTP, which does not provide any at-most-one semantics). There are some additional issues. SOAP has a notion of exceptions/errors (SOAP-Fault specification) that can be returned by a Web service. However, that portion of the SOAP specification has been criticized since it requires a service returning a SOAP-Fault to also return an `HTTP 500` error status (when using SOAP over HTTP), thus breaking the true layering nature of the protocols involved.

Scalability and reliability: Several applications such as B2B transactions and banking/stock trade transactions, among others should scale reliably to several million transactions per day.

In CORBA, the Portable Object Adapter (POA) policies combined with the Fault-tolerant CORBA features and the Load-balancing CORBA service provide the desired scalability to CORBA applications. The Fault tolerant CORBA uses the entity redundancy paradigm to provide fault tolerance to CORBA objects.

These issues are not part of the Web services standards, but again, are left to the components implementing these standards. Application servers (e.g. IBM's WebSphere) implement their own mechanisms to handle scalability and reliability.

Serializability: Serializability impacts various issues such as persistence, performance, extensibility, and ease of interoperation with other frameworks.

The CORBA Objects-by-Value (*valuetypes*) specification provides a language-independent equivalent of Java language's *serializable* functionality. The *valuetypes* specification enables features such as reverse Java-to-IDL mapping that allows Java RMI objects to interoperate as CORBA objects; and the XML/Value mapping [8] that allows XML documents to be represented as native CORBA types.

SOAP allows for user-defined serialization mechanisms. The built-in SOAP encoding is XML-based and provides some advanced features like sparse arrays to reduce transport costs. It has been argued that SOAP, being XML based, is too verbose as compared to CORBA IIOP's binary format, and thus introduces a big performance hit. That may be true for certain kinds of applications, but in most complex business applications, the cost of message passing is dwarfed by the cost of processing the business logic. Hence, the verbose nature of SOAP might not be an issue.

Static and runtime checks: The programming platform should offer some static and run-time guarantees about the execution behavior of the application.

CORBA uses IDL as a contract language. IDL is strongly typed and provides some static guarantees. CORBA Dynamic Invocation Interface (DII) on the other hand does not provide such static checks. The implementation can also take advantage of some properties of the target language (e.g. Java) with some runtime checks (e.g., array bound checks, etc.).

When building Web services, there is no standardized infrastructure support to offer static checking. At runtime, only the structure of the SOAP message is checked, and the payload is only required to be a piece of well-formed XML. It is the responsibility of the application to verify the payload further (i.e., validate against a schema). In future, WSDL could be used to generate a mapping to a programming language to offer some static guarantees.

Note however that the overall checks involved during SOAP message validation are much more fine grained than the checks based on IDL. XML Schemas are more expressive than IDL and define some syntactic checks (e.g., regular expression describing a date format, etc.), as well as some semantic checks (e.g., range constraints), which cannot be captured in IDL.

3.2 Feature support in CORBA and Web services

Below we outline the common features and requirements of representative applications such as those described in Section 2.1.1, and Section 2.2.1. We also describe how CORBA and Web services provide support for each such feature.

Location transparency: Client applications should be able to seamlessly interoperate with the services, without concern for the location of the service.

CORBA client applications obtain references to objects and invoke operations on them to perform application tasks without concern for whether the objects are remote or collocated relative to the client.

Web services client applications (using SOAP) refer to services using URLs which implicitly encode the location (IP address) of the service. However, location information can be changed via DNS. A side-effect of the encoding of network information in URLs, is that it permits some semantic manipulation of network addresses directly at the level of the application, making it sometimes easy to write proxy services.

Registry: Efficient repositories for services that maintain all the service-specific information should be available.

The CORBA standard defines an Interface Repository (IR) that provides run-time information about IDL interfaces. At run-time clients can use the IR to discover the operations that can be performed on an object, and make invocations on it using the Dynamic Invocation Interface (DII). The CORBA standard also defines an Implementation Repository that contains information that allows an ORB to activate servers to process a request, along with other server-specific information such as administrative control, resource allocation, security, and activation modes.

In Web services, UDDI registries optionally can store some structural information about the service in the form of a WSDL specification which defines the set of interfaces and messages used by the service.

Service Discovery: Service lookup should be easy (e.g., accessing a service via its name, or functionality offered).

The CORBA Interoperable Naming Service defines a URL-formatted object reference called *corbaloc* that applications can use to reach remote services. A second URL format called *corbaname* allows applications to directly invoke the remote service. The Trading Object Service supports an advanced lookup, registration and discovery of services based on service type.

In Web services, UDDI registries permit discovery of services matching a given interface. UDDI is itself a Web service accessible via a SOAP interface.

Firewall traversal: Application requests should be able to seamlessly traverse firewalls after proper security audits. This is crucial for applications that span enterprises.

The upcoming CORBA Firewall Traversal specification allows CORBA requests to be routed through firewalls. Explicit routing information is added to the interoperable object references (IOR). The clients can control security parameters such as end-to-end SSL or permit unencrypted requests via gateways/proxies. Note that this is still a draft specification, and currently not supported by ORB vendors, or supported via proprietary solutions only.

For Web services, the preferred transport protocol is SOAP over HTTP. Because HTTP is the ubiquitous Web protocol with a well-defined port, firewalls are usually configured to allow inbound and outbound HTTP traffic, thus enabling SOAP messages to cross firewall boundaries easily. The presence of Web service does not require any change in the firewall configuration. SOAP over secure HTTP (using SSL) is handled in the same way.

Security: Security in distributed applications involves a variety of features such as authentication, authorization, encryption, data integrity, delegation, non-repudiation and auditing.

All these features are supported by the CORBA Security service. However, there are no standardized security services in the Web services technology stack. Some aspects of security can be dealt with at the level of the transport protocol though. SOAP does not specify any security features but rather makes it easy to build security using Internet technologies such as SSL or XML-Signature for maximum interoperability. Moreover, Web service vendors have started offering some security services (e.g. Microsoft's Passport and Sun's Liberty Alliance for network identity). However, the security solutions being provided for Web services are akin to *reinventing the wheel* since significant R&D efforts have gone into developing a well defined CORBA Security service.

Persistence: Certain applications such as online shopping maintain customer state in the form of *shopping carts*. Such information should be made persistent to handle failures or as mandated by the business logic.

The CORBA Persistent State Service (PSS) provides the mechanism to maintain a persistent state of the object. The PSS is an abstraction layer that provides an unique API to use any kind of datastores like files, databases, directory, and so on.

Web services do not define a standard persistence mechanism, instead leaving it to the applications to do so. For example, applications written in Java can make use of Java serialization to create persistent objects.

Ease of deployment and assembly: The framework must be amenable to creating, assembling, and deploying services by integrating new as well as custom-built legacy components, while still offering standardized, open interfaces to clients.

The CORBA Component Model (CCM) provides a container environment similar to EJB, and can support EJBs as CORBA components. Furthermore, CCM provides the ability to develop components in multiple languages. The specification provides a multi-platform software distribution format that includes an installer and XML-based configuration tools. Due to its platform independent feature, CORBA is very well suited to wrap legacy applications as CORBA objects and bind them together.

The main motivation for SOAP (and Web services more generally) is to address the complexity of deploying and assembling software components. Even though it is too early to draw any conclusion about this new technology, SOAP tries to address these two issues in the following ways.

SOAP relies on Web data-formats and protocols that have been successful, such as XML and HTTP. SOAP champions interoperability by imposing a simple message-based protocol, which is model, language and platform independent. The underlying data-model is XML-based (XML Schemas) which makes it expressive and extensible. SOAP is being touted as a middleware technology for middleware.

SOAP is being enriched by new technology (e.g., WSDL, UDDI) to make the authoring, assembly and deployment of Web services easy. Development platforms such as `.NET` and WebSphere make this task easier.

Platform independence: New service offerings might potentially have to leverage from components built on heterogeneous platforms involving different hardware and software including different operating systems and implementation languages.

CORBA has been designed to be platform independent. This includes hardware, operating systems, and programming and scripting languages.

In Web services, all message communication is via SOAP, and no other restrictions (e.g., platform similarities) are imposed on the client or the server. The only requirement is to be able to read and write SOAP messages (i.e. XML documents).

Footprint: The use of mobile embedded devices such as PDAs mandate the need for a small footprint client-side (or even server-side) application.

The Minimum CORBA specification provides a subset of the CORBA standard and is meant primarily for embedded systems. The OMG is also working on a Wireless CORBA specification [10] that allows wireless access, terminal mobility, and service provisioning in CORBA. A related work in the OMG community deals with CORBA for smart transducers. Note however that all these specifications still require some minimum infrastructure ("Orb-lite") on the client side.

In Web services, depending on the complexities of the client and server (e.g., is WSDL/UDDI required?), the footprint requirements will vary. In extreme cases (e.g., thin clients accessing simple services), it is not hard to imagine that the client might not even require a full XML parser.

4 Determining Applicability of Web services and CORBA

In this section, we identify various characteristics of a distributed application and determine which of the two technologies i.e., Web services and CORBA, best provides the features required to implement that application.

Web interfaces: For business applications that require a web interface of some sort (e.g., to enable a ubiquitous remote access capability), SOAP is an obvious choice because it relies on XML and typically uses HTTP as the transport protocol. The body of a SOAP message can be easily transformed (e.g. via XSLT) into HTML for display on a browser. Examples of such an applications include business workflows that span both human and machine agents; one task in the workflow can be displayed as a form in a web interface, that a human needs to fill out and submit.

Providing a web interface over a pure CORBA framework would require more work. There are some web clients that have built in ORBs (e.g., Netscape Communicator 4.x has a Visibroker for Java ORB), however, they are used more for lower-level communication purposes, rather than for generating HTML interfaces.

Secure architectures with firewalls: For deployment architectures with firewalls, SOAP is the best choice so far, mainly due to the fact that HTTP is the preferred transport for SOAP. Since HTTP is associated with a well defined port, most firewalls have been configured to accept HTTP traffic. SOAP messages can piggy-back on HTTP messages, and can hence can tunnel through firewalls.

CORBA IIOP does not use a specific port for communication. This is not a technical shortcoming of CORBA but more a sociological fact. To address this issue, ORB vendors have in the past provided their own proprietary solutions for firewall traversal. The OMG is working on a firewall traversal specification for CORBA IIOP. However, at this time, due to lack of implementations, its impact cannot be estimated.

It is important to keep in mind however, is that providing security in a distributed application goes beyond the ability of a protocol to be "firewall friendly".

Presence of "legacy" components: The need to interface with legacy components forces a strong reason to "talk" the same protocol for ease of integration. For example, a lot of telecommunication infrastructure is already built using CORBA, so it does not make sense to rewrite the infrastructure using SOAP, even if a convergence of telephony and web services is desired. Another huge advantage that CORBA has is its ability to easily interface with other popular object-oriented component frameworks (e.g., the CORBA component model is a superset of the EJB object model which makes the integration of both seamless). Analogously, COM/DCOM components are part of the `.NET` architecture, and thus can easily interface with Web services (though bridging with CORBA is also easy).

Finally, CORBA-SOAP gateways already exist and can be used to move from one methodology to another (see Section 5.2). The impact on performance due to such gateways is less understood, though some vendors claim a very low performance degradation (e.g., less than 2% on the overall throughput).

Stateful applications: In CORBA the state of an application can be captured inside the instance of an object. The state is modified by calling methods on the object.

SOAP on the other hand is a stateless protocol. Thus, maintaining state across several exchanges is somewhat tedious. Either the entire state would need to be passed around in the messages belonging to one session, or the state would be maintained by the server, and a session-id passed around. This is complicated if more than one remote server is involved.

The same comments are applicable to transaction-based applications. For example, applications have demonstrated 2PC (two-phase commit) across multiple ORBs and J2EE implementations, but implementing the same with Web services (via SOAP) is much harder.

Mobile environment: In a mobile environment the client (and possibly the server) keeps moving which requires dealing with changing network addresses (and hence the need for forwarding services), and unreliable connections.

For this kind of scenario, CORBA is not well-suited. The mobility factor creates some additional constraints which have to be taken care off ahead of time (because of the tight coupling between the client and server). The ORBs have to be able to handle mobility or a special forwarding service has to be set-up. In any case, the transition is not transparent to the application. Although the OMG is working on a Wireless CORBA specification that addresses several of these issues, we are not aware of any implementations and hence cannot judge its impact.

With SOAP, the mobility factor can be taken care off by proxies that route message accordingly. The big advantage is that the sender of the message and the final recipient do not have to be aware

of the proxies. The message-based approach is also practical to handle disconnection, In this case, the message simply needs to be sent again.

Thin clients: With the advent of the wireless Web, millions of users will be able to communicate with each other and with services. These users will carry thin clients (such as cell-phones, PDAs) with limited memory and computing power.

CORBA imposes an all-or-nothing approach. An thin client willing to speak CORBA will have to support the ORB libraries. The footprint issue is being addressed though by the Minimum CORBA specification, which provides a subset of the CORBA standard. The on going work on the Smart Transducer specification also addresses the footprint issues related to devices such as chips, sensors, etc. However, in all such cases, an ORB will still be required at the client end, even if the footprint is kept small. This might potentially increase the cost of such devices (larger memory, faster processor requirements).

SOAP on the other hand does not require much. The contract is to be able to send and receive SOAP messages. In some cases, one can imagine that the parser is fine-tuned to support only the kind of messages specific to the application (e.g., no need for a full fledged XML parser).

Transparent Proxies: There are certain applications that would benefit from the addition of filtering proxies between the client and the server, that provide some value-added services while remaining transparent to both the client and the server.

CORBA has support for a few specific services in this area (e.g., load-balancing, replication, caching etc.). However, for services that are not part of the suite, it is not clear how easy it is to add new ones because it would require expensive changes at the ORB level.

SOAP makes it easier to interface new components in the architecture. A new component simply receives a message, processes it as needed, creates a modified message, and forwards it. This can be done in a fashion transparent to both the client and server. In a way, we can say that SOAP is proxy-friendly. As an example, SOAP enables the *easy* creation of filtering/routing proxies that can range from being *service agnostic* to various degrees of *service aware*; e.g., building a proxy that handles all requests to one particular service in a special manner, and simply forwarding the rest. Once the proxy has been configured in such a manner, any changes to the interface for that service (operations on the service, parameters passed etc.) need not be percolated to the proxy, and hence do not affect the behavior of the proxy. However, doing the same in the CORBA framework would be hard, if not impossible.

Need for transparent addressing: In CORBA, references to network objects is done via IOR. An IOR is a CORBA encoding of a network address as defined by the CORBA specification. IORs are used and resolved by ORBs. The meaning of the IOR is completely opaque to the application: there is no way to understand nor alter the meaning of the IOR (which is good because it makes it impossible to forge object references).

In SOAP, addressing is done via URLs, which have their location and other semantic information embedded in them. However, this is sometimes also an advantage; they can be manipulated, stored and replayed, based on the embedded information. For instance, in a load-balancing scenario, a proxy can rewrite a request addressed to a specific service into a new service (e.g., a request to `http://www.lucent.com/service1` to `http://www.lucent.fr/service1`).

Real-time performance: CORBA has already been deployed for real-time applications (finance, aeronautics, etc.). Over ten years of research and development have contributed to highly efficient and fine-tuned implementations.

It is too early to expect from SOAP similar performance. Moreover, the major concern of SOAP is interoperability with components that possibly are not aware of each other at deployment time. It is not clear that we should expect real-time performance between such components.

We conclude this section by mentioning two other important aspects to be cognizant about:

- *Human factors:* This relates to the degree of familiarity with the technology, learning curve, and number of developers required/available (where Web services have an upper hand, since they are based on well known and heavily used protocols/formats, namely HTTP and XML).
- *Maturity of the technology:* The maturity of the technology also plays a crucial role in deciding which technology to use. Maturity of a technology can be gauged from the amount of standardization and the number of tools available. Here CORBA wins since a lot of effort has gone into improving the CORBA specification, and the ORB implementations.

5 Interoperation between CORBA and Web Services

This section describes how the features of CORBA and Web services can be combined to implement new value-added services that otherwise would be tedious, if not impossible, to implement using only one of the two technologies. To elicit our point, we focus on an example from the telecommunications domain described below.

The cost of making phone calls has been dropping steadily over the years with most of the cost savings being passed to the end-customer. Hence, carriers are actively looking to rapidly create new, revenue-generating services. Examples of such applications include call forwarding, caller ID, anonymous number blocking, etc. However, these are still traditional telephony applications. Carriers are now looking to leverage the rapid growth in Web services to provide such enhanced services.

We now examine a real-world application that illustrates the convergence of traditional telephony services and Web services. We use this example to motivate why a combination of existing middleware

technologies such as CORBA and the emerging Web services is necessary to implement new, valued-added services.

Example 5.1 (Mobile Restaurant Locator) Consider a scenario where a user is on a WAP-enabled phone and wants to find out the good restaurants in his/her current neighborhood. The user sends a corresponding WAP request to a restaurant web site which needs to determine the user's current location so that it can send back a list of the five nearest restaurants. Hence, the web site sends a request to the telephony network, in particular, to the Gateway Mobile Location Center (GMLC), to determine the user's location. Based on the information the web site receives from the GMLC, it then generates the corresponding response to be sent back to the user.

In this case, the restaurant web site can send a SOAP request over HTTP to the telephony carrier's network. The carrier on receipt of the SOAP request can use the standard telephony APIs/protocols to communicate with the GMLC before formulating a correct response to be sent to the Web service. The Web service, in turn, generates a WML response to be sent back to the cell phone. □

5.1 Implementing the Mobile Restaurant Locator Service

As mentioned above, carriers are working on providing new value-added services that leverage the web infrastructure. In some cases (like the example provided above), the carriers themselves might not provide the value-added service to the end-customer, but might allow third-parties (e.g., the restaurant web site), *access* to the telephony infrastructure to do so. The question then becomes, *how should the access be given without compromising on the integrity of the carrier's data?*

In order to answer this question, let us briefly examine the current way of allowing applications to control the telephony network. The number of network elements that control different aspects of the telephony network are staggering, with each talking possibly a different, specialized protocol. Thus, in order to rapidly create new, value-added services in an environment involving so much heterogeneity required a technology that could bind these pieces together.

An obvious solution was to create a middleware infrastructure that provides a standardized interface for the applications while insulating them from the complexities of interaction with the network elements. The middleware is known by different names by different equipment vendors (including Lucent), but to our knowledge, in almost all cases, the middleware is built using the CORBA framework[‡]. It is tedious and infeasible to replace CORBA with Web services (using XML/SOAP) to address the issues related to performance, scalability, and reliability that is critical to network element operations while simultaneously addressing interoperability issues arising out of heterogeneous, low-level protocols.

Up until now, however, most applications that controlled the telephony services resided in the carrier's network only. However, as the above example illustrates, convergence between Web services

[‡]The interface provided to applications is via a standardized protocols such as Parlay over CORBA

and telephony services required the carriers to *open* up their networks for access by third-parties. Carriers obviously do not want third-parties to have complete and arbitrary access to their network. Instead, they want to provide a limited access layer, in a controlled fashion. Secondly, any protocol established between the carrier and the web site is application specific, and thus must be flexible and easy to modify if the application demands change in the future.

SOAP seems to be a natural choice to resolve these problems. In the above example, SOAP messages could be used to tunnel requests between the carrier's and web site's firewalls. Moreover, the flexibility offered by SOAP headers would be beneficial in a couple of different ways. It would be easy for the web site to create generic SOAP messages that work with a variety of carriers wherein each carrier might simply extract information relevant to it from the message and discard the rest. Moreover, even if the response from each carrier is different (e.g., latitude/longitude of the user, or only the wireless cell where the user is calling from etc.), having a flexible message format supported by SOAP would allow this to be handled easily. Using CORBA to achieve this has some disadvantages, namely (a) hard to create a flexible and easy to modify (without requiring changes by all parties involved) message format, and (b) lack of currently available solutions to firewall traversal, although there is ongoing work in OMG to resolve this issue.

In the following sections, we examine the implementation details from the perspective of the carrier. In particular, we focus on the interoperability between Web services and CORBA. As explained earlier, CORBA is required to provide a scalable and reliable solution to bind together the carrier's network elements that communicate via different protocols. On the other hand, SOAP enables a third party with a web-based, limited access to the carrier's telephony network. When combined in this manner, the two technologies can be used to provide new, value-added, revenue-generating services.

5.2 SOAP to CORBA gateway

To enable web access to services built using CORBA (or vice versa), the ideal scenario would be to install gateways that perform automatic conversion between SOAP and CORBA IIOP messages.

To illustrate the interoperability solutions between SOAP and CORBA, we present some details of SOAP-CORBA gateways based on the description of XORBA [14], SCOAP [12], and soap2corba bridge [15]. In the following, we will assume that the role of the gateway is to (a) accept SOAP requests, (b) forward them to established or new CORBA based servers, and (c) return the result as a SOAP response. The processing of an incoming SOAP request involves the following steps:

1. the SOAP request is parsed,
2. the gateway looks up the IDL description of the CORBA service,
3. the gateway looks up the WSDL description of the SOAP request,
4. a dynamic CORBA request is built and sent to the server (using DII),

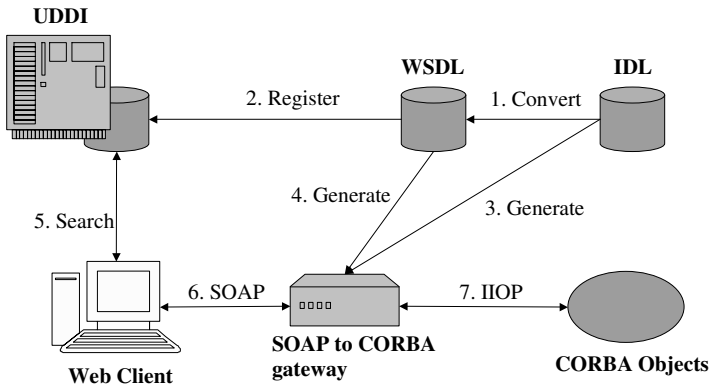

Figure 3: Accessing CORBA objects via SOAP

5. a SOAP response is built out of the CORBA response (using some type information from the repository).

Figure 3 shows the architecture involved in creating such a gateway [11], where CORBA services need to be exposed to web clients.

An intrinsic part of the protocol conversion is mapping the structure and content of the data between the two message protocols. Since SOAP messages are (at some level) simply XML documents, this implies a need to map from XML Schemas and/or DTDs to corresponding IDL constructs. In the remainder of this section, we examine the issues involved with the data mapping.

5.3 Mapping between XMLSchema/DTD and IDL

Applications that process XML data typically convert XML documents into an internal tree representation and manipulate it via the DOM API (a W3C standard). One way to send XML data between two applications is to first convert the DOM tree into another form (say a string) [13], send the string on the wire (which might require further marshaling/unmarshaling by IIOP), and then recreate the DOM tree on the receiving side.

Early CORBA specifications dealt with the issue of object references only. There was no way to pass copies of objects from a server to a client such that operations defined by the IDL on those objects could be performed locally on the client. However, this implied that in some cases, an additional level of data marshaling had to be done by the client and server (e.g., to pass XML data). In order to alleviate these problems, the notion of *value types* was introduced.

CORBA value types are essentially the language-independent implementation of Java's serialization functionality. They enable new features such as Java-to-IDL mapping, as well as mapping XML documents to/from native CORBA types. This feature is part of the CORBA 2.3 specification (June 1999), however it has not been (fully) implemented by all ORB vendors.

XML documents may or may not have a DTD for describing document structure. If a DTD exists, it can be used to create a static mapping to CORBA value types [8]. In this way, DOM trees representing XML documents can be passed between client and server, without requiring an additional step of serializing from the DOM representation to a different wire representation (e.g., a string). A dynamic mapping for XML documents has also been defined in the case where a DTD is not available. However it is less easier to use, though being more flexible. This ensures that applications can create XML documents in-memory from scratch (or read in from an external source), and the validity of the document can be guaranteed before it is written out to the wire.

The mapping from DTD to value types is straightforward. However, since DTDs are being deprecated in favor of XML Schemas, the question is if a similar mapping can be defined between XML Schemas and CORBA value types.

The CORBA Web services document [11] describes how to expose CORBA services as web services, and provides mapping between IDL and XMLSchema, as well as IDL and WSDL (and by definition, to SOAP). This permits web clients to access legacy CORBA applications. However, they do not address the question of how to map XML schemas to valuetype IDL so that XML documents can easily be passed on the wire (without doing additional marshaling), while at the same time preserving document validity. For example, range constraints, patterns, key/uniqueness constraints, foreign key constraints etc, cannot be mapped to IDL. So the server would need to perform an extra validation step to validate the document being received.

CORBA, however, has a notion of *interceptors* that allow application-level hooks into the ORB and the stubs/skeletons generated by the IDL compiler. For example, the stub/skeleton interceptor can be used to access the parameters being exchanged and check/modify them before they are passed to the application. As an example, constraints in XML Schema can be mapped into code that acts as an interceptor at the unmarshaling layer and performs constraint checking on the fly.

6 Discussions and Related Work

This section describes examples of CORBA and Web services platform. We also mention other work comparing CORBA and Web services.

There exist several CORBA vendors specializing in ORB infrastructure for enterprise and/or embedded/real-time applications. There are both commercial and public domain implementations of such ORBs for different languages. The Netscape browser and server uses an ORB internally. Today, CORBA platforms are often bundled as part of so-called application servers that include other protocols such as SMTP, SNMP, etc. [1] provides a good survey of such application servers.

Web service platforms are still in their infancy and most of them are at the stage of prototypes. The only support service agreed upon and offered as a standard is UDDI. Some platforms offer their

own proprietary services.

Microsoft's `.NET` is a Web services platform comprising development tools, specialized servers and some Web services. Applications can be written in any language supported by the Common Language Infrastructure (e.g., VB, C++, C#). These services (called `.NET My Services`) seem to be targeted to the individual user. Moreover, they are meant to be used by developers for applications that require access to users' information or for the integration with Microsoft desktop applications. `.NET My Services` offer among other things: .NET Presence, `.NET Location`, `.NET Calendar`, `.NET Contacts`, `.NET Inbox`, and `.NET Lists` and network user authentication (`Passport`).

Sun's `ONE` (Open Network Environment) platform consists of the iPlanet application server and J2EE. The platform is presented as a service jukebox. The announced services are Instant messaging network user authentication (Liberty Alliance). IBM's WebSphere core consists of a J2EE application server and offers a large suite of components that will ultimately be accessed as Web services.

An interesting aspect that is not captured by either CORBA or Web services is the possibility of code-shipping: the remote execution of some local code or the local execution of some remote code. This feature is not supported mainly for two reasons: security and language independence. However, Java with RMI can be used to provide such a feature.

There has been some other work on comparing CORBA and Web services such as in [13], which is a three-part article series on comparing CORBA with XML and SOAP. The first article addresses versioning in CORBA IDL and XML. The second article focuses on solutions to represent XML as a CORBA IDL data types. The solutions range from simple CORBA *string* data type to the upcoming XMLDOM [8] specification that makes use of CORBA *valuetype* to represent XML data. The final article describes SOAP and web services and how they compare to CORBA. They focus on comparing SOAP with IIOP in terms of the transfer syntax, interoperability with non CORBA systems, and firewall traversal issues. In comparing CORBA with Web services, they elucidate the differences in CORBA and web services' application choreographies. They also demonstrate several ways of combining CORBA and web services by mapping the application choreographies. Finally, the authors conclude that web services is a middleware integration technology rather than a middleware replacement technology.

Our approach differs from the above in that we delve into the choreographies of several applications derived from different domains. We then factor out the requirements patterns and determine if CORBA and web services support each of these requirements. Based on this analysis, we provide a recipe for determining web services and CORBA applicability.

7 Conclusions and Future Work

With all the hype surrounding XML and Web services, it is difficult for people to understand the new differentiating features this technology has to offer over existing technologies such as CORBA. In this

paper, we have presented an argumentative comparison of both technologies showing where they relate and where they diverge. By looking at the requirements of different distributed applications representing several different domains, we have identified some key aspects that should motivate the decision for one over the other. We have also presented solutions and challenges for interoperation between both.

An over simplified view is to consider Web services as middleware for middleware that would sit on top of CORBA and relegate CORBA as a lower-level implementation platform. Decisions should be more subtle. As an illustration from the telephony networks, CORBA sometimes sits on top of SOAP-like applications. The SIP protocol used for signaling (i.e. getting an agreement between both ends of a call, regarding voice encoding, etc.) is HTTP-like and is used by a higher-level call control API (Parlay) usually implemented in CORBA. The choice of a HTTP-like architecture for signaling is motivated by the nature of the task and the simplicity of implementation. The choice of CORBA for call control is motivated by the object-orientedness of modeling of calls and the need for performance.

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

References

[1] David Alpher. The application server state of the union. `http://e-serv.ebizq.net/aps/alpher_2.html`.

[2] Miguel de Icaza and Jonathan Blandford. Components in the GNOME project.

[3] ebXML. http://www.ebxml.org/.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Massachusetts, 1995.

[5] Elliot Lee. CORBA applications in GNOME.

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

[7] Object Management Group. *CORBAServices: Common Object Services Specification, Updated Edition*, 95-3-31 edition, December 1998.

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

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

[10] Object Management Group. *Wireless Access and Terminal Mobility in CORBA*, telecom/01-02-01 edition, February 2001.

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

[12] OMG. Simple CORBA object access protocol (SCOAP). ftp://ftp.omg.org/pub/docs/orbos/00-09-03.pdf.

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

[14] Rogue Wave Software. XML CORBA. http://www.roguewave.com/products/xml/xorba/.

[15] SourceForge. SOAP to CORBA bridge. http://soap2corba.sourceforge.net.

[16] UDDI. http://www.uddi.org/.

[17] Bill Venners. Jini vs. CORBA. Posted on the Jini Forum, Aug. 11 1999.

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

[19] Steve Vinoski. New Features for CORBA 3.0. *Communications of the ACM*, 41(10):44–52, October 1998.

[20] W3C. Simple object access protocol. `http://www.w3.org/2000/xp/`.

[21] W3C. Web services description language. `http://www.w3.org/TR/wsdl/`.