Abstract—The use of computer systems have been widely used in such applications, scientific, manufacturing and business. The main idea of distributed computing systems is to use a client-server architecture in combination with remote method invocation. The invocation of these methods is to send data from a client to a server for processing and then the server returns data to the client for presentation. In general, distributed platforms Java-RMI, CORBA and Web Services provide a range of methods to communicate with distributed objects on remote computers. This paper presents a comparative study (measurement of processor usage, memory, packets on the network, etc.) Between CORBA and Web Services, implementing two distributed services, a basic arithmetic operations and sending strings.

Index Terms—CORBA, Web Services, distributed computing.

I. INTRODUCTION

WEB services present another alternative distributed computing infrastructure; an alternative that is being strongly promoted as preferable to the use of distributed object middleware such as Java RMI[8][9][10][12] or CORBA [1][2][3].

Web Services differ from the distributed object technologies in that they have reverted to an earlier “remote service” model similar to that in DCE. There is no concept of an object reference; instead a service is defined simply by an end-point that supports various operations[2][4][7]. In terms of Java-RMI or CORBA, a Web Service is like a singleton server object. The singleton server character of a Web Service means that a stateless-server architecture is preferred; though there are mechanisms permitting the implementation of stateful-servers.

Web Service implementations support different client-side application programmer interfaces; client code may work by constructing “call” objects that are dispatched to the server, or may use a higher level interface that hides the communications level entirely through the use of client-side stub objects with an operational interface that mimics that of the server.

The client-stub approach results in code that is very similar to Java-RMI or CORBA clients[12][14]. The analogous mechanisms for generating client and server components for Web-Services, Java-RMI, and CORBA are illustrated in Figure 1. When auto-generated client-side stubs are used for Web Services, the development processes and the code complexity for both client- and server-side are virtually identical for Web Service, Java-RMI, and CORBA solutions. Typically one starts with an interface definition for the service. A client-side stub is auto-generated from this interface. On the server-side, the interface is processed to yield a base class for the implementation class that must be written by the developer. With the mechanisms and costs of development being very similar, other factors will determine the choice of a technical solution. Systems developers will have to choose between interoperability where Web Services have advantages, and performance that will favor Java RMI or CORBA [14].

Java RMI and CORBA use optimized connection-oriented communications protocols that are either language specific, or have detailed rules defining how data-structures and interfaces should be realized. In contrast, Web Services (application-to-application) are based on the ubiquitous technologies that have grown up to support WWW-services (human-via browser-to-application)[6][7]. Communications use HTTP. HTTP is universally supported, and HTTP traffic can normally pass through firewalls. Tiresome practical details like common data representations are avoided through the use of textual representations. All numeric and other data are converted to text. Meta-data, defining structure, are provided in situ as XML mark-up tags[2][9][13]. XML parsers allow client and server implementations to construct their distinct but equivalent representations of any data structures.

II. TECHNOLOGY REVIEW

A. CORBA

The Common Object Request Broker Architecture essentially is an object bus, enabling a client to invoke methods on remote objects contained in a server, independently of the programming language the objects have been written in and of their location.

The client-server interaction is mediated by Object Request Brokers (ORBs), present both on client and on server sides and communicating usually via the Internet Inter-ORB...
Protocol (IIOP)[1][5].

CORBA objects can be either collocated with the client or distributed on a remote server, without affecting their implementation or use: the ORBs will take care of the details. The methods of CORBA objects are defined using the Interface Definition Language (IDL): they accept as input parameters and return values some CORBA data-types and can raise exceptions.

CORBA has been designed to be independent from the Operative System: it runs on many OS platforms, ranging from the Win32/UNIX up to the real-time embedded systems and supports a vast majority of programming languages as well as some scripting languages. Communication protocols used by CORBA for ORB communications include TCP/IP, IPX/SPX, ATM, etc.

In Figure 1 are depicted the elements of the CORBA reference model, which collaborate to provide the properties briefly outlined in this section.

![Figure 1: Component model in CORBA Architecture.](image)

B. Web Services

Web Services are a distributed middleware technology that uses simple XML-based protocol to allow applications to exchange data across the Web. Services are described in terms of the messages accepted and generated. It is not important the component itself (object model, programming language, etc.), the only requirement is that the service is able to process certain well-defined XML documents[7][12].

The XML Document contains all the application-specific information that a service consumer sends to the service for processing. The documents a service can process are described using an XML schema. Two processes involved in a Web Services conversation must agree on the same description, in order to correctly validate and interpreter the documents they exchange[13].

Actually, at the core of Web Services there is SOAP, an XML based communication protocol for interacting with Web Services; it erroneously stood for “Simple Object Access Protocol” although it has nothing to do with accessing object. The SOAP specification includes:

- Syntax to define messages with optional header and body. The header holds all system information, while body contains the XML document that the Web service has to process.
- The encoding/serialization rules for data exchange.
- The conventions for representing RPC.

The increasing popularity of Web Services have produced a proliferation of services, which in turn have required the availability of public directories that could be used for the registration and lookup of services themselves. Universal Description, Discovery and Integration (UDDI) provides a mechanism for service providers to advertise their services in standard form and for service consumers to query services of interest, opening the way for interoperability between services. UDDI is itself implemented as a Web service using SOAP as the message protocol. In Figure 2 is presented high-level component architecture for Web Services.

![Figure 2: Component model of Web Services.](image)

III. COMPUTING MODEL AND FEATURE EVALUATION

A. Data model

At first, one of the most important differences between CORBA and Web Services is how an application is modeled in either cases: while CORBA is a true object-oriented component framework, in Web Services there is no notion of objects: they are centered around a message passing paradigm. Moreover in CORBA 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 it: the result of the call is possibly another CORBA object on which it can apply other 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 [1][4].

B. Data consistency

To assure data consistency, the infrastructure should provide semantics that guarantee against multiple executions of the same client request on the server3. While CORBA ORBs are required to provide at most once semantics, in Web Services the message semantics are defined by the protocol underlying SOAP. For instance, HTTP, SMTP or TCP/IP, that do not provide any at most once semantics[9][10].

C. Type Checking

CORBA uses IDL as a contract language, which is strongly typed and provides some static warranties about the execution behavior of the application. On the other hand, CORBA
Dynamic Invocation Interface (DII) does not provide such static checks. The implementation can also take advantage of some properties of the target language with some runtime checks (e.g. array bound check in Java)[7].

In Web Services there is no standardized infrastructure support to offer static checking; only the structure of the SOAP message is checked at runtime, and the payload is only required to be a piece of well-formed XML. Further checks to the payload (i.e. scheme validation) fall under the responsibility of the application itself.

It must be said, however, that SOAP message validation is more fine grained than the check based on IDL: XML is more expressive than IDL and defines some syntactic (e.g., date format regex) and semantic (e.g., range constraints) checks which cannot be captured in IDL.

D. Scalability and Reliability

CORBA applications can obtain the desired scalability and reliability by combining the Portable Object Adapter (POA) policies with the Fault-tolerant CORBA features and the Load-balancing CORBA service[3].

In Web Services these issues are left to the components. For instance, application servers such as IBM’s WebSphere, RedHat JBoss or Apache Tomcat implements its own mechanism to handle scalability and reliability [3][4].

E. Location independence

In CORBA, client applications invoke operations on an opaque object’s reference, disregarding whether the object itself is remote or local.

Web Services client applications refer to services by URLs, which implicitly encode the location of the service. However location information can be changed via DNS, exposing the whole infrastructure to the risk of security threats. On the other hand, encoding the network information in URLs, gives the possibility to write proxy services simply manipulating the network address at application’s level [6].

F. Security

CORBA security service supports a wide variety of features such as encryption, authentication, delegation, auditing, etc. Web Services do not provide any standardized security services, although some aspects of security can be dealt with at transport protocol level. In particular, SOAP does not specify any security feature but rather makes it possible to exploit Internet technologies such as XML-Signatures or SSL to achieve maximum interoperability[13][14].

Recently, several important Web services vendors have started offering some proprietary security solutions. In most of the cases, these solutions have been already implemented in - or are derived directly from - the well-established CORBA Security service. So, almost in these cases, they represent a superfluous development effort [13].

G. Service Subscribing

The CORBA standard defines an Interface Repository (IR) on which (at run-time) clients can discover the operations that can be performed on an object, and makes invocations on it via the Dynamic Invocation Interface (DII). More over it is also defined an Implementation Repository, containing server-specific information such as administrative control or security. UDDI provides a standard managing framework for the Web Services throughout searchable central repositories that use a subscribing mechanism to store service definitions[2][5][6].

H. Various Considerations

For what has been reported above, Web Services mark an important paradigm shift in data integration. But the so-called birth of Web Services does not mean that application integration simply becomes yesterday’s problem. Consider Web Services as “middleware for middleware”, that sit on top of CORBA and relegate CORBA as a lower-level implementation platform would be an over simplified view[1]. Unfortunately, there couldn’t be a worse misconception in today’s integration world. Web services will take application integration from its actual status (requiring expensive boutique solutions) to a much more egalitarian status of global concern, that will be massively distributed to nearly every company engaged in data exchange initiatives.

IV. EXPERIMENTAL SETUP

The tests were conducted on a LAN (Local Area Network) with a speed of 100Mb / s. Stress testing and performance are conducted with a desktop PC DELL Optiplex 740 (2.3GHz), 2Gb RAM and Opet systems GNU / Linux Ubuntu 10.04. Customer acting as a client and server use a macbook pro 5.5 , at 2.3GHz, 8GB RAM and OS-X systems opetarivo Lion.

These measurements were performed in a laboratory conducting the tests only for each of the technologies in isolation, so that the processor, memory and latency in the messages were not interrupted by processes other than this experiment.

The measurements to be performed:

• Memory used on the client.
• Using the client processor.
• Use a server processor.
• Latency in a simple request
• Latency in multiple requests.
• Total bytes transferred.
• Packet Counting

Applications used in this work are:

• A simple arithmetic operations.
• The sending of a text string.

These two applications were chosen for their simple implementation and are the basis of transactions and complex calculations. Finally, the implementation follows the steps in Figure 3, where in the rectangle of "Implementation" added particular implementations of each technology.
V. RESULTS

A summary of result is show in tables 1-3, Latency, traffic and memory and cpu usage were the main measurements in this work.

<table>
<thead>
<tr>
<th>Technology</th>
<th>Total Latency in seconds</th>
<th>Total Packets</th>
<th>Total Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Services</td>
<td>0.11s</td>
<td>16</td>
<td>3338</td>
</tr>
<tr>
<td>CORBA</td>
<td>0.5s</td>
<td>8</td>
<td>1111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
<th>Technology</th>
<th>Total packages</th>
<th>Total in Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elemental Arithmetic Operations</td>
<td>WS</td>
<td>48,931</td>
<td>10,360,814</td>
</tr>
<tr>
<td></td>
<td>CORBA</td>
<td>10,007</td>
<td>1,400,851</td>
</tr>
<tr>
<td>String passing</td>
<td>WS</td>
<td>55,617</td>
<td>16,053,312</td>
</tr>
<tr>
<td></td>
<td>CORBA</td>
<td>10,007</td>
<td>2,236,661</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
<th>Elemental Arithmetic Operations</th>
<th>String passing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>CPU Client</td>
<td>CPU Server</td>
</tr>
<tr>
<td>Web Server</td>
<td>15.0s</td>
<td>6.0s</td>
</tr>
<tr>
<td>CORBA</td>
<td>2.3s</td>
<td>0.8s</td>
</tr>
</tbody>
</table>

VI. CONCLUSIONS AND FUTURE WORK

In this paper we present a comparative study of two alternative technologies free for the development of distributed systems. The results show that local environments (read intranet), the most viable alternative is the use of CORBA, since it has the best performance with the least in terms of memory usage, processor and bandwidth. Now if the needs of distributed systems are oriented to the use of the Internet, web services solution that provides the best results, and their integration scalability and flexibility are superior to CORBA. While CORBA can integrate different languages.

REFERENCES