# *Web Services: An Architectural Overview*

Roseli Persson Hansen
Cassia T. Santos
Sérgio Crespo C. S. Pinto
Guilherme L. Lanius
Fernando Massen

Email:{roseli,cassiats,crespo,speka,ferhr}@exatas.unisinos.br
Post-Graduation InterSubjective Program in Applied Computing - PIPCA
UNISINOS

## Abstract

*Web Services* represents an architectural structure that allows communication between applications. The use of eXtensible Markup Language (XML)-based Technology allows exploitation of services without needing to know what platform or language was used to create those services.
This document aims to offer an *overview* of the main approaches about Web Services architecture, stablishing what are the common points between them.

## 1 Introduction

*Web Services* represent an architectural structure that allows communication between applications. An service can be invoked remotely or be used to employ a new service together with another services. Beyond the basic *Web Services* structure, this paper will be presenting some standard technologies to build *Web Services*, such as: *Web Services Description Language* (WSDL) [10], *Simple Object Access Protocol* (SOAP) [6] and *Universal Description, Discovery and Integration* (UDDI) [11]. These technologies are based in *eXtensible Markup Language* (XML) [3], and they allow service reutilization and invocation, without needing to know the underlying language or platform. This article approachs yet the main Web Services architectures, proposed by Microsoft, HP, Sun and IBM.

Section 2 presents an brief concept of *Web Services*, such as the components that make part of an basic structure: Service Register, Service Requestor and Service Provider. Setion 3 describes the main standard technologies employed in *Web Services*. In the Section 4 the main approaches of the architecture are described, such as some development platforms that are being deployed today. Then the Section 4 ends with an comparison between architecture and platform proposals.

## 2 What is *Web Services*?

*Web Services* are modularized applications that can be described, published and invoked under a network, commonly being WEB-based. I.e., *Web Services* are interfaces that describes an collection of operations that are accessible through the network by throwing standard XML messages. *Web Services* allow serivce integration in an fast, effective way ([21, 8]).

A *Web Service* is a software component that independs on the implementation and platform. It can be described using an service description language, published in a registry and discovered by an standard mechanism. Thus, it can be invoked from a *Application Program Interface* (API) through the network, built in round of another services.

*Web Services* contain the best accomplishments of the Web-based component development. Like its software components counterparts, *Web Services* contains *black-box* functionalities that can be reused without caring in what language it will be used ([12]).

The *Web Services* terminology describes specific functionalities of the exposed business, usually through Internet connections with the purpose to provide some way to exploit this service.

The exposure of services enrolls:

- Identify or define value functions in the process or business.

- Define a service-based interface to the processes.
- Describe these interfaces in a Web-based form.

To make a service available in the Web is commonly necessary:

- Publish the service interface, allowing users find and use it.
- Accept requests and send responses using the XML message protocol.
- Stablish a binding between external requests and business functions implementation..

By using a message-form document, a program sends a request to a *Web Service* through the network and (optionally) receives some response, also in a XML-based message-form document. The usage of XML raises the importance of provide such extensible mechanism to describe the messages and its contents. Furthermore, Web-based services standards define the format of the message and mechanisms to publish and discover *Web Services* interfaces.

*Web Services* are not accessed via specific object-model protocols such as *Distributed Component Object Model* (DCOM), *Remote Method Invocation* (RMI) or *Internet Inter-ORB Protocol* (IIOP); actually they are accessed through transport protocols like HTTP, FTP, etc. ...

## 2.1 *Web Services* Architecture

In a Web Service architecture a service description covers all the necessary details to grant the service interaction, including messages's format, the transport protocols and physical location.

This interface hides service implementation details, allowing that the same service can be used independently of the underlying hardware or language, thus making Web Services-based applications component-oriented, making those components available for reuse.

The *Web Services* architecture presented in Figure 1 is built upon a three-role interaction:

- **Service Provider:** this is the service's owner from the business perspective. From the architectural approach, this is the platform that is accessed in the service request. It is also the entity that creates the *Web Service*, being responsible to make its description in some standard format and publish its details in a central registry.

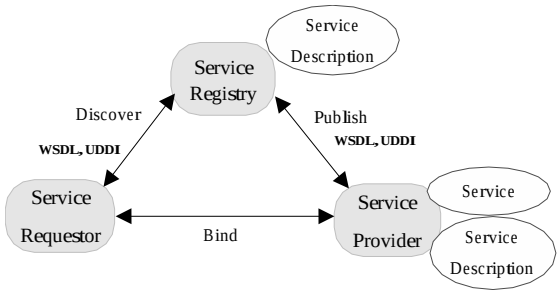


Figure 1: *Web Services* roles, operations and crafts ([21])

- **Service Requestor:** it is an application that invokes or initializes some interaction with the service. It could be a *web browser* or even a non-user interface program such as another *Web Service*. By using the service description it's possible to discover and invoke *Web Services*.
- **Services Registry:** it's the place where service providers publishes their service descriptions. Service Requestors search the Registries, fetching binding and description information - both during the development time (static bindings) or run time (dynamic bindings).

As mentioned early, there is the **Service Description** whose contents describes interface and implementation details, including data structures, operations and network binding informations. Also it contais data to easy **Service Requestors**'s searching process. The **Service** is the software deployed through the network by the **Service Provider**.

Afterwards, there are common operations used in that architecture, such as:

- **Bind:** when a service must be accessed, this operation invokes and initializes interaction within its caller in runtime, using binding information provided by the service description to both locate and contact it.
- **Publish:** a service must be published in an **Service Registry** to be accessed. The **Service Provider** thus contacts the **Service Registry** to publish the service.
- **Discover:** A **Service Requestor** finds a description of the service or queries a **Service Registry** for the required service type. A **Service Requestor** can find a service interface description in both run time or development

time. Then, the necessary information regarding bindings and locales to invoke a service are found and contacted.

## 2.2 *Web Services* Conceptual Layers

The Picture 2 presents the *Web Services* conceptual layers, described below ([21], [17]):

- **Network Tier:** it's the base layer that represents protocols such as HTTP, FTP, SMTP, POP, etc. ... This tier is used accordingly to the needs of the application: security, availability, performance and reliability.
- **XML-Based Message:** this layer uses the SOAP protocol as the message exchange technology standard, which stands for exchange of information in a distributed, decentralized environment.
- **Service Description:** the description of the service is done using the *Web Services Description Language (WSDL)*, which defines the interface and interaction mechanisms of the service, further describing additional informations such as context, Quality of Service and service-to-service relationship.
- **Service Publishing and Discovery:** these two layers use the *UDDI* standard to discovery and publish information regarding *Web Services.*

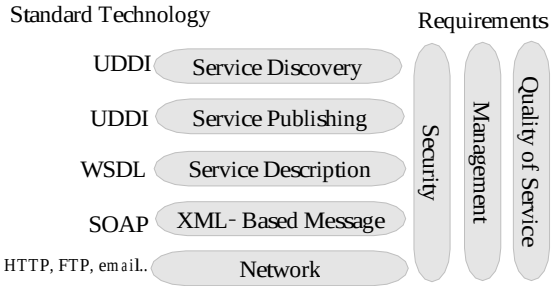


Figure 2: *Web Services* Conceptual Layers ([21])

# 3 Standard Technologies Used in *Web Services* Architectures

Three technologies were previously mentioned regarding *Web Services* structure. Lying between these technologies WSDL, SOAP and UDDI are addressed in a most proper form in the subsections that follows.

## 3.1 The *Web Services Description Language* (WSDL)

The WSDL is a XML-based language to describe services as a collection of accessible operations through messages ([10]). Defines interfaces and service interaction mechanisms, binding protocols and network services details. It presents additional descriptions such as context, Quality of Service and service-to-service relationship. By using its methods, it's possible describe a object in a transparent, implementation-independent way.

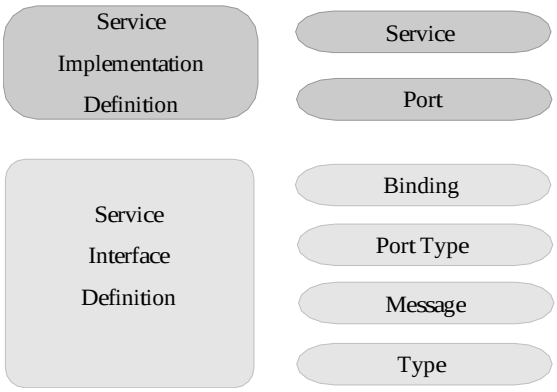


Figure 3: Service Description Layer ([21])

The usage of WSDL also allows a basic service description in two parts as shows the Picture 3.

This basic definition is separated in both service interface and implementation, allowing these parts to be reused separately. A brief description of the layers from the Picture 3 ([21, 10]) is explained below.

### Service Interface Description Layer

A service interface contains a WSDL service definition. This definition allows that a interface can be used, instantiated and referenced by multiple service implementation definitions ([4]).

The WSDL:*binding* directive describes the protocol, data format, security and other attributes to a particular service interface (WSDL: *portType*). The WSDL: *port Type* tag informs the *Web Service* operation elements.

The WSDL:*message* tag is used to define data *input* and *output* regarding a operation.

Also the WSDL:*type* tag defines the complex data types inside a message.

A interface file describes the *Web Service*, including the methods that are invoked, parameters that are

sent and codification that's being utilized.

### Service Implementation Description Layer

It is a WSDL document that describes how a service interface is implemented by a service provider. A WEB-based service is modeled as a WSDL element which contains a WSDL: *port elements* collection (associated port and service searching) with a WSDL: *binding* interface-definition element.

A implementation file describes where the *Web Service* is installed and how it could be accessed.

Likewise the interface and implementation definitions, the WSDL specifies extensions for protocol bindings and message formats, such as SOAP, HTTP GET/POST and MIME ([10]).

## 3.2 *Simple Object Access Protocol* (SOAP)

It is a protocol for exchanging information in a decentralized, distributed environment, thus allowing communication between various applications ([6]). This communication is made via XML-format message exchanging, including calling parameters and result requirements. This implies that the messages could be understood by almost all hardware platforms, operating systems, programming languages and network hardware ([25]). This protocol is mostly used to invoke a *Web Service* and also can be used to publish and locate *Web Services* through a UDDI registry.

The SOAP protocol could be potentially used in combination with a myriad of another protocols, such as HTTP, SMTP e JMS ([17, 13, 14]). It supports *remote procedure calls* (RPC), and asynchronous messaging style functionality also.

The SOAP data model offer data type definition to the most used structures like *string*, *integer*, *float*, *double* e *date*. The process of translating from data (parameters and results) to XML is often called *codification*.

A SOAP package consists of four parts:

- SOAP Envelope: defines a *framework* that contains the message content, who can process that message and the status of obligation to process it. It's a SOAP message structure whose syntatic elements of the message are encapsulated.
- SOAP Codification: defines serialization mechanisms that could be used to exchange instances or data types defined by an application;
- RPC SOAP: specifies how encapsulate *remote procedure calls* and responses inside the message, thus invoking remote procedures;
- Binding *Framework* and SOAP Transport: Defines a abstract *framework* to exchange SOAP envelopes between applications using a simple transport protocol.

In the SOAP specification, another concepts are also important ([15, 6]):

- SOAP Client: it's a program that creates a XML document containing the necessary information to invoke remotely a method within a distributed system (it could be a WEB or application server);
- SOAP Server: it's responsible for executing a SOAP message and acts like a interpreter and distributor of documents;
- Mensagem SOAP: it's the basic communication form between SOAP nodes;

```
<env:Envelope xmlns:env="">
  <env:Body>
    <m:getLastTradePrice
      env:encodingStyle="
http://www.w3.org/2001/06/soap-encoding"
      xmlns:m= "
http//www.w3.org/2001/06/quotes"
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </env:Body>
</env:Envelope>
```

Figure 4: SOAP Envelope Sample Code ([6])

A SOAP message is a envelope containing optional headers and a body actually containing a message with its parameters or results. The SOAP messages are written in XML. Seeing Picture 4, we have a sample code from a SOAP message.

The SOAP envelope presents the structure shown in Picture 5, where we can see the parts that composes the envelope. The SOAP block is a syntatic construction used to enclosure data that constitutes logically one only computational unity. The block is identified by an external element called URI *namespace*. The SOAP header is a collection of zero or more blocks, whose could be redirected to some known SOAP receptor inside the message's path. Finally, the SOAP body is a collection of zero or more blocks redirected to the last SOAP receptor.

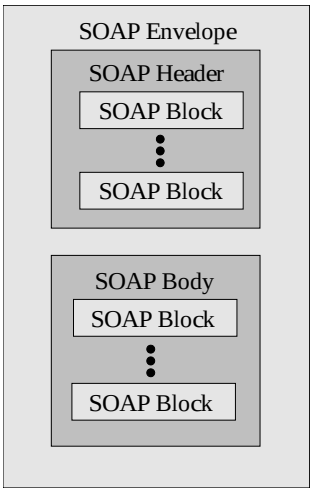


Figure 5: SOAP Envelope Structure ([6])

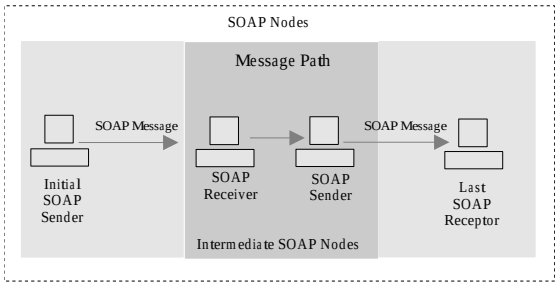


Figure 6: Message's Path Througth SOAP Nodes

The picture 6 shows how the message navigate through the SOAP nodes. Once SOAP doesn't guarantee routing, it knows which SOAP sender created the message and knows who'll be the last receptor of the message through zero or more intermediary nodes.

When a SOAP node receives a message, it must execute a process, generate success or fail messages or, if necessary, return additional messages.

The invocation of the service using SOAP occurs likewise Picture 7. The Application (1) requests a SOAP message and invokes the service operation through a *Web Service* provider. The **Service Requestor** presents the message including the network address of the *Web Service* provider. The network infrastructure (2) ships the message to a SOAP server. This SOAP server redirects the requested message to the **Web Service Provider**. The WEB server (3) is responsible to process a request message and then build a response. The message (4) is redirected through the SOAP infrastructure. When a

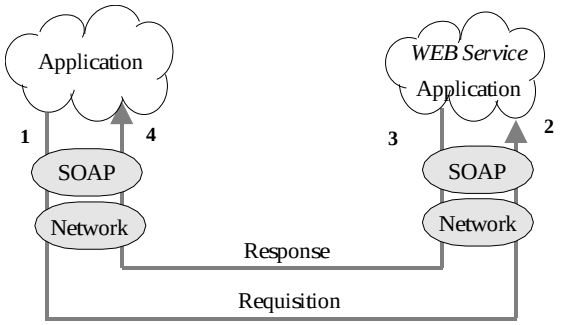


Figure 7: XML Message using SOAP ([21])

XML message arrives at the requisitor, is converted to a programming language, being shipped then to the application.

To use a service, it's mandatory to the client to find it. This search can be done through UDDI, which will be presented in the next section.

## 3.3 *Universal Description, Discovery and Integration* (UDDI)

The UDDI specification is a common effort to define and create a standard service registry. This registry can be accessed by clients and these can locate all the services that they need ([2, 11]).

The central component, called UDDI *Project* manipulates a public, global registry called UDDI *business registry*. All the information maintained in the public registry are available to general queries. A private registry can add security controls to provide data integrity and prevent non-authorized access. By that way, it only can store private information, contain a subcollection from the public registry, even a combination of both. The information that's offered by *business registry* consists of three components:

- "white pages" - address, contact and identifiers that are known;
- "yellow pages" - industrial categorization;
- "green pages" - information.

The UDDI implementation is a *Web Service* registry server that provides a mechanism to publish and locate *Services*. A UDDI registry server contains categorized information about businesses, services that the lasts offers, and associations with *Web Services* specifications. These specifications normally are also done in WSDL through a UDDI registry ([24]).

The main information model used by UDDI registry is defined through a XML schema, which defines four types of information:

- business information;
- service information;
- bindings information;
- service-specific information.

The information that concludes the registry of a service consists of four types of data structures. ([11]). This division in a information-type fashion offers simple partitions to help in the fast search and comprehension of the different data that builds the registry. The five structures are shown in Picture 8.

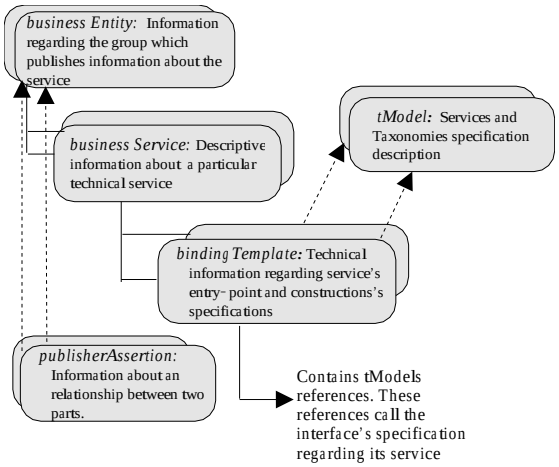


Figure 8: UDDI Structure ([11])

The ***businessEntity*** supplies information about a business, containing one or more ***businessServices***. The technical and business descriptions for a specific *Web service* are defined in its ***businessService*** and ***bindingTemplate*** elements. Each ***bindingTemplate*** contains a reference to one or more ***tModels***. One ***tModel*** is used to define the technical specification of a service.

### The *businessEntity*

This structure represents all the whole information known from an specific business or entity which publishes descriptive information about the entity, likewise its supplying services. From the XML point of view, the ***businessEntity*** is the *top-level* data structure whose information regards description about an business or entity.

### The *businessService*

Represents a logical classification of the service. The element's name contains the term "*business*" aiming to describe the purpose of this hierarchy level in service description. Thus, each ***businessService*** structure belongs to an only ***businessEntity*** structure.

### The *bindingTemplate*

The ***bindingTemplate*** structures are technical descriptions of *Web Services* that are stored via individual ***bindingTemplate*** instances. These structures provide support for one access the services remotely. The support for technology-specific parameters of the application and configuration files are encountered also. Then, it describes how the ***businessService*** uses various technical information.

### The *tModel*

The **tModel** structure is represented through metadata (data about data). The purpose of a ***tModel*** inside a UDDI registry is to provide a reference system.

### The *publisherAssertion*

Many business aren't effectively represented by only one ***businessEntity***. As a consequence, many ***businessEntity*** structures can be published. Even then, they continue to represent an simple grouping and could demand that one of its relationships should be visible in their UDDI registers. In this matter, two related businesses use ***publisherAssertion*** messages, as a way to publish declarations on how they relate with each other. One or more *publishers* should agree that the relationship between them is valid. Both should publish exactly the same information, rendering their relationship visible.

An UDDI registry, for this matter, runs inside a UDDI server. That registry is an WEB application that can be accessed via *browser* or by an programmable API, through SOAP protocol.

## 3.4 Publishing and Finding WSDL Service Descriptions in the UDDI *registry*

The UDDI data entities provides support to describe information about business and services. The information about a service description inside WSDL complements the information found in the UDDI

registry. UDDI provides support also to various kinds of service description, althought it doesn't supports WSDL or another mechanism of service description directly ([4, 5]).

A complete WSDL service description is a combination of a service interface and its implementation document. This implementation describes services's instances; each instance is defined using a WSDL service element.

A service implementation is published in a UDDI registry as a ***businessService*** or within one or more ***bindingTemplates***. The ***businessService*** is published by the service provider.

Once the service interface represents a reusable definition, this interface can be published inside a UDDI registry using a ***tModel***. Such operation must be done before the service implementation is published as a ***businessService***.

The standard technologies are putting the efforts of some companies to offer development architectures targeting *Web Services*.

# 4 *Web Services* Development *Architectures and Platforms*

From the basic *Web Services* architecture, some companies are offering their proposals such as their own development platforms. The following subsections presents the IBM, HP, SUN and Microsoft proposed platforms.

## 4.1 IBM's *Web Services* Model

The IBM proposal is based upon the general *Web Services* architecture presented in the Picture 1. IBM developed the *WebSphere* family of server-based tools, mainly aimed to raise the functionalities of the standard WEB server. That platform also extends a development environment adding Internet-based development standards such as *Java2 Enterprise Edition* (J2EE), CORBA, XML and *Web Services*.

The *WebSphere Studio Application Developer* presents the Java language as the technology of choice to *Web Services* development. The *Application Developer* uses "*Web Services Wizards*", which makes the development of *Web Services* easier ([27]):

### *Web Service Wizard*

The Web Service Wizard starts at the step 2 of the Picture 9:

- From a existing *JavaBean* is possible specify which methods should be exposed as a *Web Service*. The wizard then generates the WSDL files accordingly.
- The wizard also creates a ISD file, that's the SOAP development descriptor for an specific *Web Service* (dds.xml).
- It is also mandatory to specify the SOAP codification, which tells to the runtime how to translate data structures written in Java to SOAP-XML and vice-versa.
- It is possible to create a *Web Service proxy* class likewise the 5th step. This class contains code to cofigure the parameters regarding input and output's codification styles. Also this is the class that does the SOAP requisition accordingly the 6c step.
- It allows generate a JSP application which invokes *Web Service* methods and return its responses (step 6b).
- The wizard adds two servlets to the application: the Apache RPC SOAP and the Apache Message SOAP. These servlets receives SOAP requests coming from a SOAP client (step 6c), invoke a *Web Service* (step 6d) and returns its SOAP response.

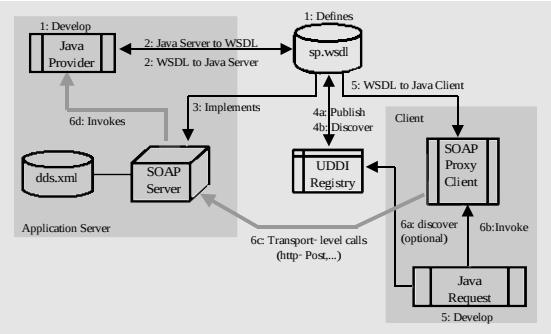


Figure 9: *Web Services* Development Process. ([27])

### *Web Service Client Wizard*

- The *Web Service Client Wizard* allows generation of a Java *proxy* class from a existing WSDL file (step 5 - Picture 9).
- The Java *proxy* class encapsulates the code in order to make a SOAP requisition and call the *Web Service*.

- This assistant is part of the *Web Service Wizard*. It can generate, deploy and run a WEB sample application to test the *Web Service*.

### *Web Service Skeleton JavaBean Wizard*

This wizard creates a *JavaBean* from a WSDL document. The *JavaBean* contains method definitions accordingly the WSDL described operations. With this resource, it's possible to implement various methods from a basic structure, thus finishing the *Web Service* implementation process.

### *Web Service DADX Group Configuration Wizard*

The wizard allows create a *Document Access Definition Extension* (DADX) group. A DADX file specifies how create a *Web Service* using a collection of operation that are defined through SQL declarations or DAD files. The DAD files describes a XML document element mappings and, also specifically, the columns of a DB2 database query. It's used by DB2 XML *extender* to store and retrieve XML documents inside a DB2 database ([9]).

### *UDDI browser (import, export)*

The *Application Developer* contains a WEB application that compounds the IBM UDDI *explorer*. It allows the following functionality:

- Find a *businessEntity* (step 4b);
- Publish a *businessEntity* (step 4a);
- Find a service interface and implementation (step 4b);
- Publish a service interface and implementation.

## 4.2 Hewlett Packard's *Web Services* Model

From the known standards applied to the *Web Services* architecture, the Picture 10 presents already known components such as:

- Registry Server that provides mechanisms to publish and locate *Web Services*;
- Service User that uses the available services;
- The Service Provider providing an application as a service and possibly publishing the service.

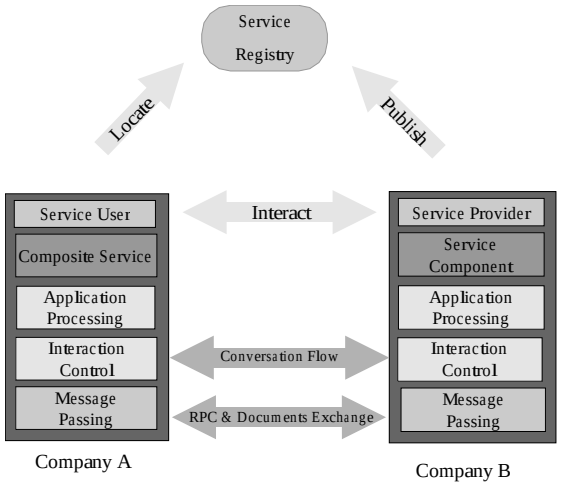


Figure 10: HP *Web Services* structure ([18])

Through these elements, the HP platform supports *Web Services* interactions, partitioning the architecture in three key areas such as: message transference, interaction control and application processing. Based in this structure, the HP model provides standards to create, deploy and access WEB services. The main HP *Web Service* core technologies are XML and Java. It also uses emerging standards like UDDI, SOAP and WSDL, also supplying support and development tools ([18]).

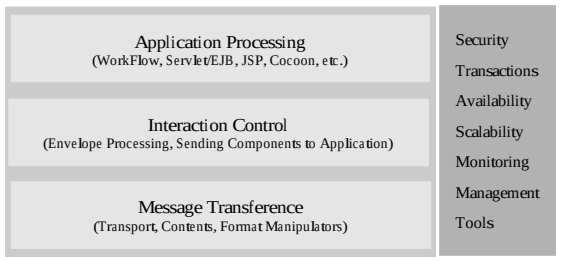


Figure 11: HP *Web Services* 1.0 Platform ([18])

The HP model supports *Web Services* interactions via a modular architecture that can be seen in Picture 11 and will be briefly described below:

- **Message Transference:** support low-level communication protocols between participating *Web Services*. Supplies morphing and sending of component-processing messages to alternative formats (non-XML formats, for instance). It enables SOAP-RPC support based on asynchronous messages over HTTP/HTTPS, such as XML messages pro-

cessing and content translation, even including SAX, DOM and XSTL support.

- **Interaction Control:** manages the message exchange in the business context, generating the whole high-level processing in the protocol. It also manages messages validation and exchanging. Its characteristics includes SOAP, version 1.1 envelope processing, including sending support, J2EE-based logical component adaptors and workflow communications systems.

- **Application Processing:** enables the business logic that's the *Web Service*'s core. This logic can be contained inside databases and corporative systems. The application processes, thus, extends the J2EE's basic functionalities, integrating the service with legacy systems and databases. It also enables the using of Cocoon2[1]-based services to other *Web Services*.

Afterwards, regarding this architecture, development, shipping and configuration tools are presented below:

- UDDI Registry navigation and *Web Service* description publishing in UDDI registries tools.

- It supplies a private UDDI Registry for development use.

- It provides SOAP messaging support and Cocoon2-based UDDI registry that enables component development regarding another *Web Services*.

- Provides a *proxy* generator which offers remote-client *Web Services* invocation.

- Supports *Integrated Development Environment* (IDEs) use, commonly used to develop *Web Services* components.

- Supplies scripts and configuration tools that enables the adoption of the platform regarding customer's needs.

## 4.3 Sun Microsystems's *Web Services* Model

Sun Microsystem's *Web services* architecture is divided in three logical components, presented in Picture 12. The **Service Requestor** is capable to ship

[1]Cocoon2 it's a *framework* to process XML documents. The HP platform uses this framework to deliver the results to the browser in HTML, WML or another specific format.

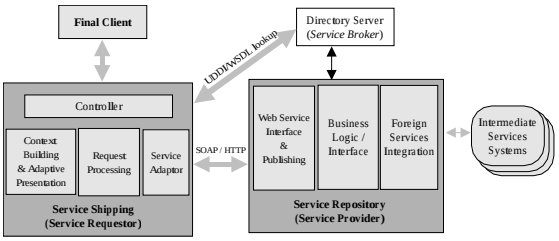


Figure 12: Interaction between components ([29])

services, thus serving both agents and clients requests. The **Service Container** can verify the **Directory Service** *(Service Broker*), which contains metadata about registered *Web Services* regarding service interface bindings and service invocation bindings.

All these activities can be modeled through a interface called **Service Processor** or in an abstract module, within various useful processors such as: e-mail processor, search processor, print processor, etc., aiming access to search/bind/invoke basic services ([29]).

In addition to the components shown above, there is also the **Controller** component. It's used to coordenate the various interactions and service invocations, and it's responsible to general process management regarding these operations.

From this architecture perspective, Sun presents, through the *Java 2 Plataform, Enterprise Edition* (J2EE) platform, *Web Services* building components. The Picture 13, shows the *Web Services* development cycle using J2EE, where one can see some forms used by clients to access services. These accesses are made through an *Web Browser* (HTTP), from some application or within *applets* (IIOP). Clients can connect to databases using *Java Database Connectivity* (JDBC) thorught SQL, for example. Services can also be accessed through SOAP, UDDI, WSDL and ebXML ([1]) technology-based clients.

To make these relationships possible, the J2EE development model concentrates its efforts in XML and Java technologies. Accordingly to this, Sun offers a development platform - *Java Web Services Developer Pack* (JWSDP), which contains a set of Java and XML APIs, also including its own version of Tomcat JSP and *servlet container*, and a WSDP Registry Server as well. through its APIs it's possible access XML documents and use their operations. Their APIs are briefly described below ([1, 19]):

- Java API for XML Processing (JAXP) - this

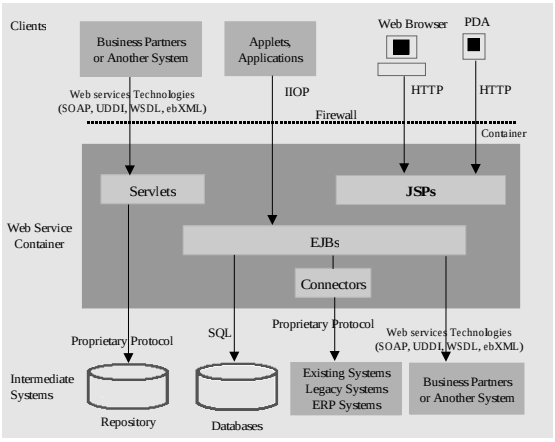


Figure 13: J2EE *Web Services* Development ([19])

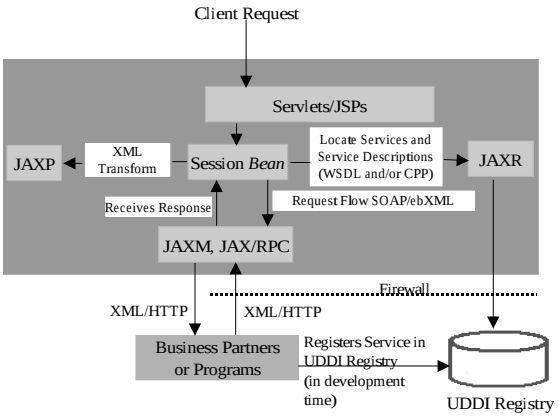


Figure 14: Using JAX* APIs for *Web Services* invocation ([19])

API's objective is create, access and modify XML documents via Java programming. It Proccess XML documents using various *parsers* such as the *Simple API for XML parsing* (SAX) and *Document Object Model* (DOM). It also supports the XML *Stylesheet Language Transformations* (XSLT), thus giving the developer control to enable and convert data to and from XML or another formats.

- Java API for XML Messaging (JAXM) - sends SOAP messages over the internet in some standard way. It can be extended to work within message protocols such as ebXML.

- Java API for XML Registries (JAXR) - presents a standard way to access business registries and related informations. It allows write applications in Java programming language supplying a uniform access path to registries based on standards such as ebXML or UDDI.

- Java API for XM, RPC-Based (JAX-RPC) - enables one to write a Java programming language application that uses SOAP to invoke a *remote procedure call* (RPC). It can be used also in the message request and response. Thus, this API enables the developer to define its own XML schema and use it to send XML fragments and documents.

These APIs enable one easily access XML-based services using Java programming language accordingly to Picture 14.

## 4.4 Microsoft *Web Services* Model

Microsoft's .NET platform supports the standard XML*Web Services* architecture. The .NET platform itself and its technologies presents a new collection of XML-based products, emphasizing the SOAP protocol ([16]).

In this manner, the data can be easily programmed, edited, exchanged and organized between *Web Sites*, applications and devices. In the Picture 15 we take note that the communication between different *Web Services* is done using this communications standards ([28, **?**]).

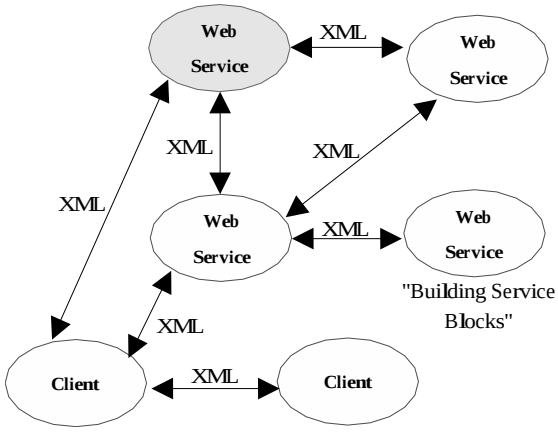


Figure 15: Microsoft *Web Services* Model ([**?**])

The XML *Web Services* model proposed by Microsoft consists in building blocks to move data via distributed computing.

The Picture 16 represents the method used by an application to supply data and services to provide

data and services to another applications or users, over the Internet, using a common supply chain. Applications access Web Services via WEB protocols (world-widely available) and data formats such as HTTP, XML, UDDI e SOAP without needing to accomplish specific operating system or implementation details.

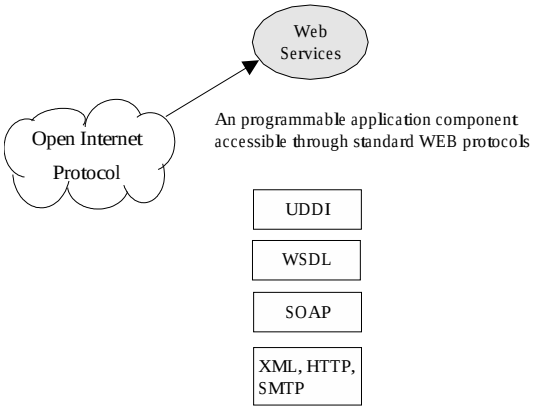


Figure 16: *Web Services* ([**?**])

The .NET platform enables the creation and use of applications, processes and XML-based WEB sites, such as services which *design* share and combine informations and functionalities from another platforms and devices ([**?**]).

The .NET *Web Services* development components follows ([20]):

- ASP+ *runtime*: represents an programming abstraction that allow developers to build *Web Services* from an existing structure.
- .NET Class Libraries: Offer a collection of various classes to built and manipulate *Web Services*.
- *Web Services Description Language Tool*: gera um código de serviços para XML *Web Services* e serviços WEB clientes a partir de arquivos WSDL e arquivos *XML Schema Definition* (XSD).
- *Web Services Discovery Tool*: Discovers XML-*Web Services*'s URLs located inside a WEB server and records locally documents and metadata related to each XML-*Web Service.*
- WEB *References*: It's an graphical interface that incorporates the functionality of the *Web Service Description Language Tool*.
- *Server and Solution Explorer Tool*: It's an graphical interface to navigate among *Web Services* and manage such projects.

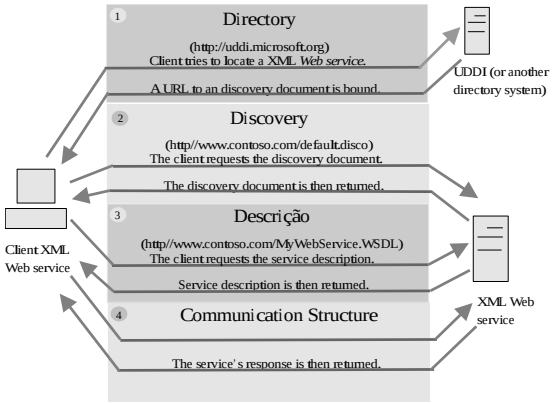


Figure 17: XML *Web Services* .NET Infrastructure ([20])

Microsoft's XML *Web Services* structure is referenced in Picture 17, which presents three entities or roles: The XML *Web service client*, the XML *Web service* and the *Directory Service*. As it follows, the operations made by them will be described.

1. The client is interested in a credit card validation service. UDDI then provides the service directory and search functionalities througth a standard API.
2. A client requests a discovery document regarding the service, which resides, by convention, at the WEB server's root directory. A discovery document contains pointers or endpoints to service descriptions and its locations. In the case a client already knows the *Web Service*'s URI, this step can be omitted.
3. The *Web Service*'s client requests then a service description in the form of a WSDL document, througth an standard URL, using standard HTTP. The then-returned WSDL document contains all the technical informations needed to invoke the service (parameter list, data types, etc. ...).
4. The client then requests through the service a SOAP message containing the names of the methods, its parameters, etc. ... that it needs. Then the XML *Web Service* processes the request and return a SOAP envelope containing the computed results.

There are various definitions of XML *Web Services*, althought all definitions have some common points:

- XML *Web Services* exposes functionalities useful to the users through a standard WEB protocol. In most cases, SOAP is the used protocol.
- XML *Web Services* provides such way to describe its interfaces with sufficient details, that it even allows the user to build an client application to talk to them. This description is generally offered in a XML document called WSDL.
- XML *Web Services* are registered in such way that users can find them easily. That search is done through UDDI.

A comparison between the main approaches is shown below.

### 4.5 An Architecture Comparison

Considering the mentioned information regarding the main proposed architectures, in this section we'll analise briefly its differences and similarities to understand better its technologies and standards.

| | .NET (Microsoft) | J2EE (SUN) | IBM *Web Services* | HP *Web Services* |
|---|---|---|---|---|
| Technology | .NET | J2EE | J2EE | J2EE |
| Development Environment | Visual Studio.NET | J2EE+JWSDP | J2EE+WebSphere | J2EE+Cocoon |
| Server Platform | Windows NT/.NET | JVM *Compliant* | JVM *Compliant* | JVM *Compliant* |
| Services Registry | UDDI or another Directory System | UDDI | UDDI | UDDI |
| Service Discovery | WSDL (DISCO document) | WSDL | WSDL | WSDL |
| Communication Protocol | SOAP | SOAP | SOAP | SOAP |

Table 1: Comparison between Architectures

The Table 1 represents a comparison between employed technologies in Sun, Microsoft, IBM e HP proposals. The adoption of standard technologies such as SOAP, WSDL e UDDI by the different companies is driven by interplatform operability objective. We can note, however, the use of two concurrent technologies such as J2EE e .NET.

J2EE technology is adopted by most of corporations whose efforts are being redirected to *Web Services* application development.

Otherwise, Microsoft's proposed technology seems try to integrate the *Web Services* architecture to a proprietary layer, driven by the .NET classes, which aren't open neither shared by other development tools aside Visual Studio.NET. There is a comparison between these two technologies in the Table 2. It is possible envise their characteristics in relation to various points such as used interpreters, dynamic WEB page or content servers, database access among others.

| Characteristics | .NET | J2EE |
|---|---|---|
| Technology Profile | Product | Standard |
| Intermediate Solutions | Microsoft | 30+ |
| Interpreter | CLR | JRE |
| Dynamic WebPage Generation | ASP.NET | JSP |
| Middle- tier Components | .NET Management Components | EJB |
| Database Access | ADO.NET | JDBC, SQL/J |
| SOAP, WSDL, UDDI | Yes | Yes |
| Implicit Architecture Support (load- balancing, etc) | Yes | Yes |

Table 2: Comparison between J2EE e .NET technologies ([26])

## 5 Conclusion

This paper tried to present the main *Web Services* approaches, likewise its technologies and standards as well. The different architecture implementations shows the fact that standards such as SOAP, UDDI and WSDL represents a form to allow *Web Services* communication built among different platforms.

We could note that companies like Sun, IBM and HP are redirecting their efforts to build their development platform in a non-proprietary, standard fashion, remarkably contrary to the Embrace and Extend characteristics regarding Microsoft's .NET strategy.

The common effort to use standards built upon the XML language allows that applications can communicate between each other using a common, standard and human-readable messaging technology. By this meanings, the programming language in which its implementations are done becomes transparent.

Another *Web Services* Architecture's visible characteristic is the possibility to create new applications only composing and combining services that already exists through the Internet.

# References


[1] Amrstrong, E.; Bodoff, S.; Carson, D.; Fisher, M.; Green, D and Haase, K. *The Java Web Services Tutorial*. http://java.sun.com/webservices/dowloads/webservicesturorial.html. Last access - 16/04/2002.

[2] ARIBA, INC., International Business Machines Corporation and Microsoft Corporation. *UDDI Data Structures Reference*. September, 2000.

[3] Bray, T.; Paoli, J.; Sperberg-McQueen, C. M. and Maler, E. Extensível Markup Language (XML) 1.0. Second Edition. http://www.w3.org/TR/2000/REC-xml-20001006. Last Access - October 2000.

[4] Brittenham, P.; Cubera, F.; Ehnebuske, D.; Graham, S. *Understanding WSDL in a UDDI registry, Part 1*, September 2001.

[5] Brittenham, P. *Understanding WSDL in a UDDI registry, Part 2*, September 2001.

[6] Box, D; Ehneubuske, D.; Kakivaya, G.; Layman, A.; Mendelsohn, N; Nielsen, h. F.; Thatte, S.; Winer, D. *SOAP: Simple Object Access Protocol*, April 2000.

[7] Clarke, N. W. *Silverstream and Web Services - A Technical Overview & Developer's Perspective of SOAP*, June 2000.

[8] Clements, T. *Overview of SOAP Web Services - Tecnical Overviews*. Sun Microsystems, August 2001.

[9] Cheng, J. and Xu, J. *IBM DB2 XML extender*. February 2000 www-4.ibm.com/software/data/db2/extenders/xmlext/xmlextbroch.pdf. Last Access - 26/04/2002

[10] Christensen, E.; Curbera, F.; Meredith, G. and Weerawarana, S. *Web Services Description Language (WSDL)*, March 2001.

[11] Ehnebuske, D.; Rogers, D.; Riegen, C. V. *UDDI Version 2.0 Data Structure reference*. June 2001.

[12] Graham, S.; Simeonov, S.; Boubez, T.; Davis, D.; Daniels, G.; Yuichi, N. and Neyama, R. Building Web Services With Java. SAMS, 2002. 581 pages.

[13] Gudgin, M. ; Handley, M.; Moreau, J. and Nielsen, H. F. *SOAP Version 1.2 Part 1: Messaging*. W3C, October 2001.

[14] Gudgin, M. ; Handley, M.; Moreau, J. and Nielsen, H. F. *SOAP Version 1.2 Part 2: Adjuncts*. W3C, October 2001.

[15] Gudgin, M.; Hadley, M. Moreau, J.; Nielsen, H. F. *SOAP Version 1.2*. W3C, July 2001.

[16] Hanson, J. *Use Web services to integrate web aplications with EISs*. http://www.javaworld.com/javaworld/jw-02-2002/jw-0208-eis_p.html. Last Access - 22/02/2002.

[17] Hendricks, M.; Galbraith, B.; Irani, R.; Milbery, J.; Modi, T.; Tost, A.; Toussaint, A.; Basha, S. J. and Cable, S. *Professional Java Web Services*. Wrox, January 2002. 588 pages.

[18] *HP Web Services Plataform Architecture - an oveview*. October, 2001. http://www.bluestone.com/downloads/pdf/web_services_architecture.pdf. Last Access - 25/03/2002.

[19] Kao, J. *Developerś Guide to Building XML-based Web Services with the java 2 Plataform, Enterprise Edition (J2EE*). Sun Microsystem, Inc. June 2001.

[20] Kraft, Reiner. *How IBM WebSphere Studio Aplication Developer Compares with Microsoft Visual Studio .NET - Part1: Conceptual Differences*. http://www7b.boulder.com/wsdl/techjournal/0202_kraft/kraft.html. Last Access - 27/02/2002.

[21] Kreger, H. *Web Services Conceptual Architecture*. IBM Software Group, May 2001.

[22] McLaughlin, B. *Java and XM*L. O'Reilly, 2001, 528 pages.

[23] Microsoft Corporation. *XML WEb Services - A new Opportunity for ISV*s. http://www.microsoft.com/serviceproviders/strategicwp/xml_web_services.asp, 2001. Last Access - 25/03/2002.

[24] Rajaram A. *Overview of UDDI. Web Services - Tecnical Overviews*. Sun Microsystems, August 2001.

[25] Seely, S. *SOAP Cross Platform Web Service Development Using XML*. Prentice Hall PTR, 2002. 391 pages.

[26] Vawter, C. and Roman, E. *J2EE vs. Microsoft.NET - A Comparision of Building XML-based Web Services*, June 2001.http://www.midleware-company.com/. Último acesso - 25/03/2002.

[27] Wahli, U.; Tomlinson, M.; Zimmermann, O.; Deruyck, W. and Hendriks, D. *Web Services Wizardry with WebSphere Studio Application Developer. Redbooks* - IBM. April 2002, 598 pages.

[28] Wolter, R. *XML Web Services Basic*s. Microsoft Corporation, December 2001.http://msdn.microsoft.com/library/default.asp?url=/library/en-us/Dnwebsrv/html/webservbasics.asp?frame=true. Last access - 25/03/2002.

[29] Yin, K. *A Reference Architecture for Smart Web Services*. Sun Microsystems, August 2001.