

A survey and case-study evaluation of Web Services security technology

Miguel Pardal

Instituto Superior Técnico,

Department of Information Systems and Computer Engineering

miguel.pardal@dei.ist.utl.pt

<http://mega.ist.utl.pt/~mflpar/>

Abstract

Organizations want to make their information systems more agile so they can better answer the challenge of adapting to changes in business requirements. Web Services and Service-Oriented Architectures propose systems with greater flexibility, reuse and interoperability. However, the essential security standards and implementations have yet to be sufficiently evaluated in practical uses.

This paper presents a survey of Web Services technology with additional detail for security standards and implementations. It also evaluates the technology using a complex and valuable business case-study: real estate transactions. A prototype evidenced insufficiencies in the available implementations.

Keywords

Service-Oriented Architectures

Web Services

Security

Enterprise Information Systems

Enterprise Applications Integration

Distributed Systems

1 Introduction

The *Internet* is the main infrastructure of the information society. As a large scale public network, it allows for an *open and dynamic business environment*, where information and communication technologies enable new and innovative ways to work and create value [Laudon02].

Enterprise applications are the software part of information systems for electronic business on the Internet [Fowler02]. They have many users, must deal with large volumes of complex data, and use complex and changeable business rules to process it. They also need to be integrated with other systems typically built for distinct purposes, by other organizations and using different technologies. All of this requires advanced tools and qualified people. Therefore, building enterprise applications is a significant technical challenge.

One of the main challenges of enterprise applications is *change*: customers change, businesses change and their information systems must also change [Laudon02]. This means that applications need to be more *agile*, i.e., they need to more easily adapt to changes in the business requirements. Programming methodologies and techniques are important, but the underlying technology must also be an enabler and not an obstacle.

Web Services (WS) [Curbera05] and *Service-Oriented Architectures* (SOA) [Krafzig04] are currently being proposed to address this need for agility, at the technology and architecture levels, respectively, with the goals of maximizing *flexibility, reuse* and *interoperability*. The *services* are the units of enterprise applications, granting access to data and functional resources.

For most enterprise applications, *security* is an essential requirement: *valuable resources require adequate protection* [Anderson01]. With the increase in openness brought by services come new security challenges. One can no longer rely on network configuration and firewalls to draw the border between the organization and the outside world. The services must “know” the principals, so they can be used by those who are trusted, leaving the rest out. This is easier said than done. The services must have effective mechanisms for *access control* (authentication, authorization), *message protection* (confidentiality, integrity) and *configuration flexibility* (for choosing the settings most suited for each invocation).

The focus of Web Services Security is on the *reuse* and *integration* of existing cryptography-based security technologies like X.509 [Housley99], Kerberos [Kohl93] and HTTPS [Rescorla00]; rather than reinvention.

1.1 Objectives and outline

This paper answers the following two questions:

- *What are Web Services?*
- *What are the protection capabilities of current Web Services Security technology?*

To do so, first a *survey* of Web Services technology is presented, focusing on security standards and implementations. Next, the *evaluation* results from a case-study prototype and other implementation tests are presented and discussed. The paper ends with a *conclusion* that recaps the main issues and points directions to future work.

2 Survey

This section presents the survey on Web Services technology, covering the *platform* as a whole, including *standards* and *implementations*, and an *example* service in detail.

The *scope* of the survey was limited to technological aspects, leaving out SOA aspects, like: methodologies, service modeling, service orchestration and choreography, and management of meta-information.

2.1 Web Services platform

A Web Services *platform* is an implementation of a set of Web Services standards that enable the creation and execution of Web Services applications. The core standards are XML [Bray04], XML Schema [Fallside04], SOAP [Gudgin03] and WSDL [Booth05]. These standards are broadly available in current implementations.

There are several standardization processes in progress, like WS-Policy [Schlimmer06], for instance. Each proposal is usually sponsored by companies, like Microsoft, IBM, Sun Microsystems and Oracle, then is submitted to a organization, like W3C, OASIS, IETF or WS-I, where there it is further discussed and finalized.

To keep the platform design coherent, there are several *core technical guidelines* that standards and implementations should abide to:

- *Message orientation* - services communicate exclusively by messages, that have a life time that can span more than one transmission on a given transport;
- *Encapsulation* - services are described in standard and public contracts, but their implementation is kept private;
- *Autonomy* - each service can be managed individually and should have the least dependencies possible from other services;
- *Standards composition* - the standards used by the services should be structured in block-like fashion so they can be custom composed according to the needs of a particular application;

- *Standards-based interoperability* - no precondition is assumed beyond those explicit in the standards. The standards should be publicly accessible and open.

2.2 Standards

Standards enable interoperability. If two implementations are to cooperate, they must follow a compatible set of standards. WS standards can be used as-needed by applications, but first they must be understood individually and how they fit together.

WS-Map [Pardal06a] is a contribution to Web Services developers and researchers, to help them find their way around so many standards. It is a visual index of standards categories, shown in figure 1. It aims to give a broad and vendor-independent view of the technology. The following sections describe the Web Services standards by categories, as specified in WS-Map.

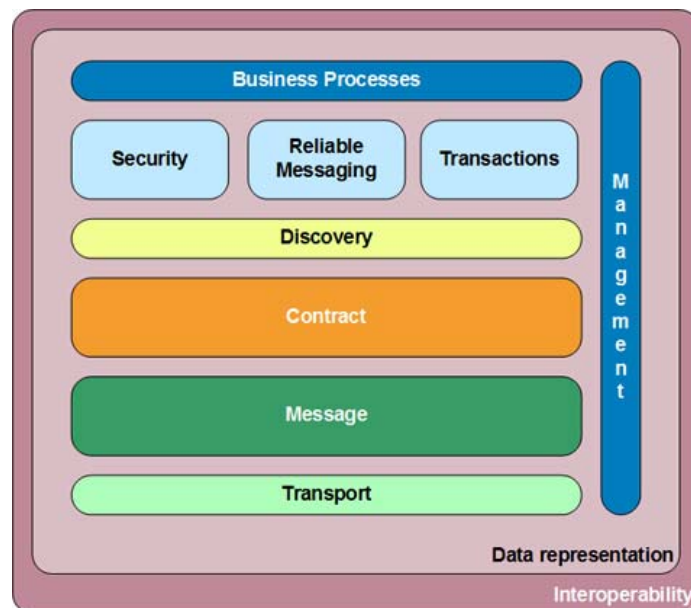


Figure 1 – Web Services standards map, divided in categories [Pardal06a].

2.2.1 Data representation

The data representation standards address the problem of heterogeneous data representation i.e. how to represent data in a format that is equally understood by everyone.

XML [Bray04] is a text-based tag language that allows the representation of data in a structured and self-describing way. A document that conforms to XML

syntax is called *well-formed*. XML is used as canonical format for data exchanged in Web Services.

XSD [Fallside04] is a grammar to define XML document schemas, including: elements, attributes, order, cardinality, data types and default values. A well-formed document that conforms to a XSD is called *valid*.

2.2.2 Transport

The transport standards define ways to establish a communication channel between a client and a service. The communication can be synchronous or asynchronous, meaning that the client blocks waiting for an answer from the service or not, respectively.

The most used transport for Web Services is HTTP [Fielding99]. A common asynchronous transport is SMTP [Klensin01]. There are also non-standard transport implementations using message queue systems.

The transport can provide additional features to the communication. However relying on them makes the Web Service transport-dependent. For example, HTTPS [Rescorla00] can be used to provide security for a Web Service using HTTP as transport.

2.2.3 Message

The message standards define the structure of the communication units and the ways they can be exchanged between services.

SOAP [Gudgin03] defines the Web Services messages as XML documents. The SOAP envelope separates the *header* (with platform data) from the *body* (with application data or fault). The header can contain data from different Web Service extension protocols.

MTOM/XOP [Gudgin05a] is used to transport binary data in SOAP messages, replacing former approaches.

WS-Addressing [Box04] allows for the addressing and forwarding of SOAP messages in a transport-independent way.

WS-Enumeration [Geller04a] enables the creation of data enumeration sessions encompassing several requests and responses.

WS-Eventing [Geller04c] and WS-Notification [Graham04] are competing standards for asynchronous event notification using Web Services, making polling unnecessary. Both have subscribers, subscription managers and event consumers. WS-Polling [Davis05] specifies mechanisms for successive requests when asynchronous notifications are made impossible by a firewall. In these cases, one of the endpoints must periodically connect for information updates.

2.2.4 Contract

The contract standards accurately describe the data, functions, policy and resources of a Web Service. They are used for client-server binding.

The Web Service interface is described with WSDL [Booth05] and its data types are described with XML Schema [Fallside04]. The WSDL contract is necessary but not sufficient to describe a service. Beyond the interface, there are operational aspects (ex. which SOAP version to use), commercial aspects (ex. usage costs), and non-functional aspects (ex. security).

The Web Service policy is defined with WS-Policy [Schlimmer06]. The policy states additional requirements that must be fulfilled by the client and by the service so that the interaction between them can occur. The normal form of a WS-Policy has the generic structure shown in example 1.

Example 1 – WS-Policy normal form.

```
<wsp:Policy ... >
  <wsp:ExactlyOne>
    ( <wsp:All>
      ( <Assertion ...> ... </Assertion> )*
    </wsp:All> )*
  </wsp:ExactlyOne>
</wsp:Policy>
```

A policy states a set of configuration alternatives supported by the service. The client only has to support one of the alternatives. For each non-functional requirement domain, like security, there is a support library that should be used to implement the required standards.

WS-Transfer [Geller04b] and WS-ResourceFramework [Czajkowski04] are two competing standard proposals to describe and to explicitly manage Web Service resources as informational entities with XML representation that can be created, read, updated and deleted.

2.2.5 Discovery

The discovery standards define ways to publish and discover Web Services.

UDDI [Clement04] defines a directory that allows dynamic registration and querying of Web Services. The UDDI data model allows three types of queries: by service business type and required capabilities (*yellow pages*), by company contacts (*white pages*), and by endpoint address (*green pages*).

WS-MEX [Curbera04] enables Web Service self-description with a protocol to access XSD, WSDL, WS-Policy, and other meta-information.

2.2.6 Security

The main concerns for Web Services security are:

- *Message protection* – how to make sure that the message contents are not read and or written while going through the network or intermediate nodes;
- *Access control* – which principals can access the service and when;
- *Configuration flexibility* – which security configuration should be applied for a given service invocation.

XML-Signature [Eastlake02a] and XML-Encryption [Eastlake02b] are the two core cryptography standards for XML documents dealing respectively with digital signatures and content ciphering.

WS-Security [Nadalin04] states how to protect SOAP messages with XML-Signature and XML-Encryption and how they can carry tokens in headers. Security tokens are security-related data items, like cryptographic keys, digital certificates, assertions, etc. Tokens enable WS-Security to bind to existing security technologies, like X.509 [Housley99] and Kerberos [Kohl93].

WS-SecurityPolicy [Kaler05] is a WS-Policy vocabulary for specifying security policies.

SAML [Cantor04] is an assertion format for the exchange of authentication, authorization and attributes information between different security domains. It also defines a communications protocol.

WS-Trust [Gudgin05b] defines a trust model for Web Services based on STS (Security Token Services) that perform trust brokering, i.e. emit, renew and validate security tokens enabling new trust relationships.

WS-SecureConversation [Gudgin05c] specifies how to establish a secure session encompassing several messages, using session keys for more efficient and robust cryptography.

2.2.7 Reliable messaging

Reliable messaging standards address the reliability of Web Service message exchanges in a transport independent way.

WS-Reliability [Iwasa04] and WS-ReliableMessaging [Ferris05] are two competing proposals for assured delivery, duplicate elimination and correct ordering in Web Services messaging.

2.2.8 Transactions

The transactions standards address the problem of providing well-defined semantics for the combined result of a group of Web Services operations on distributed resources.

The goal semantics can be ACID (where atomicity, consistency, isolation and durability properties hold) or a more relaxed version of it. To achieve this kind of solution, the models assume temporary and recoverable fault models for the machines and networks.

There are two frameworks for transactions in Web Services: WS-Coordination [Feingold05] and WS-CompositeApplicationFramework [Little03].

2.2.9 Business processes

The business process standards leverage all the other Web Services technologies and define development tools at an abstraction level closer to the views and needs of the business people.

WS-BPEL [Thatte03] is an approach based on orchestration where the business process is represented by a graph, with the nodes being the business activities and the arcs being the information and control flows that enable composition of existing services.

WS-CDL [Kavantzias04] is an approach based on choreography that describes business processes declaratively, stating pre-conditions and post-conditions for the execution of activities. The way the process is actually executed can change, as long as the stated conditions still hold true.

2.2.10 Management

The management standards address the problem of keeping the Web Services up-and-running. This problem has two complementary aspects: the management of the Web Services themselves and the management of the machines and networks where they execute. Currently, the standardization efforts are focused on the latter kind of management. There are two competing standards for the management of machines and networks using Web Services: WS-Management [Geller04d] and WS-DistributedManagement [Sedukhin05].

2.2.11 Interoperability

The interoperability standards are called *profiles* and are necessary because of the ambiguities in standards that result in differences in implementations. Each profile covers a set of Web Services standards and provides: implementation guidelines, example applications, and compatibility test toolkits.

WS-I (Web Services Interoperability Organization) is an organization that gathers the main vendors of Web Services tools and defines general purpose profiles, like basic interaction [Ferris04] and security [Barbir05].

The WS-DeviceProfile [Schlimmer05] is not defined by WS-I and has a different scope. In this case, the goal is to select a subset of standards for devices with limited resources.

2.3 Implementations

Standards define the platform, but implementations are needed to make it a reality where applications are actually developed and deployed.

The *base implementations* cover the core standards, including data representation, transport, message, contract and discovery. *Extension implementations* are necessary to implement non-functional requirements like *security*, reliable messaging, transactions and the remaining categories.

2.3.1 Base implementations

Before Web Services

Distributed systems middleware didn't start with Web Services. Similar efforts trace back to DCE [Lockhart94], CORBA [OMG91], DCOM [Eddon98], and

Java RMI [Sun97]. The main distinction of Web Services is the emphasis on flexibility, reuse and interoperability.

The starting point for Web Services was XML in 1998 as a data representation standard. The first uses of XML were “data islands” in HTML pages. Next, ways were developed to exchange XML documents over the network, like XML-RPC [Dumbill01].

In 2002 Microsoft released Dot Net including a XML remote procedure calls, coining the term Web Services. This “grand entrance” caught the attention of Microsoft’s competition that began working on similar offerings.

First generation: Dot Net, Axis and JAX-RPC

As mentioned, the first implementation of Web Services was Microsoft Dot Net [MacDonald03]. A Java open-source implementation followed in Apache Axis [Graham01]. Then, Sun Microsystems led the reference implementation for Java, called JAX-RPC [McGovern03], later implemented on Sun, IBM, Oracle and BEA products.

The “genetic traits” of this generation are: use of SOAP 1.0/1.1, WSDL 1.0/1.1, UDDI 1.0/2.0 and HTTP as exclusive transport. Its main drawbacks are: excessive emphasis in remote procedure call model with no support for asynchronous invocations, use of custom SOAP encoding with no schema reuse and transport-dependent addressing.

Second generation: WSE, Axis2 and JAX-WS

The second generation, currently available in release products, was once again started by Microsoft, with the launch of WSE (Web Services Enhancements) that pioneered security and addressing and a user-friendly configuration process.

The Java platform responded with the improvement of JAX-RPC, renamed to JAX-WS 2 and integrated with JAX-B 2 for data binding. Asynchronous invocations and transports other than HTTP were also added.

Apache Axis2 is an alternative open-source implementation that allows several transports and external modules for extensions.

The “genetic traits” of this generation are: use of SOAP 1.1/1.2, WSDL 1.1/2.0 and XML schema data encoding. UDDI is mostly being replaced by service self-description. Despite support for other transports, HTTP is still dominant.

Third generation: WCF, WSIT

The third generation, still in development and in beta products approaching release, promises greater interoperability between Dot Net and Java. Microsoft will release Windows Communications Foundation (WCF) and Sun Microsystems will release WSIT (Web Services Interoperability Technology / Project Tango). WCF will consolidate Microsoft's adoption of service-oriented computing, replacing previous distribution middleware, like Dot Net Remoting. The new slogan is ABC: addresses, bindings and contracts.

WSIT uses the JAX-WS 2 architecture and extends it to fit together with WCF. IBM and Oracle follow at a distance, still not decided to follow Sun's path. Oracle has the leading implementation of BPEL. Apache Axis2 is improving significantly.

The "genetic traits" of this generation are: use of SOAP 1.1/1.2, WSDL 2.0 with support for several message exchange patterns (synchronous, asynchronous, request-answer, notification, etc.), service self-description, transport independence with support for message queues, implementation of non-functional requirements like security using policy-driven configuration.

2.3.2 Security implementations

The main security implementations currently available for Web Services are:

- WSE 3 (Web Services Enhancements 3) for Microsoft Dot Net 2 [Microsoft05];
- WSS4J (Web Services Security for Java), for Apache Axis2, for Java [Apache06];
- XWSS (XML and Web Services Security), for JAX-WS 2, for Java [Sun06].

Table 1 details the standards supported by each implementation.

Table 1 – Standards supported by current Web Services security implementations.

Provider	Implementation	Supported standards
Microsoft	WSE 3: Dot Net Framework 2.0, Visual Studio 2005, Web Services Enhancements 3.0	WS-Security: Username, X.509, Kerberos WS-Secure Conversation, WS-Trust SAML
Apache	WSS4J: Apache Axis2, Rampart module of Web Services Security for Java (WSS4J)	WS-Security: Username, X.509 WS-Policy SAML
Sun Microsystems	XWSS: Java Web Services Developer Pack 2.0, XML and Web Services Security 2.0	WS-Security: Username, X.509 SAML

2.4 Web Service example

As mentioned before, a Web Service gives access to data and functional resources that can be used in enterprise applications. The following example is a *Notary service* that allows the submission of documents to be validated. Figure 2 illustrates the interaction process between a client and the service.

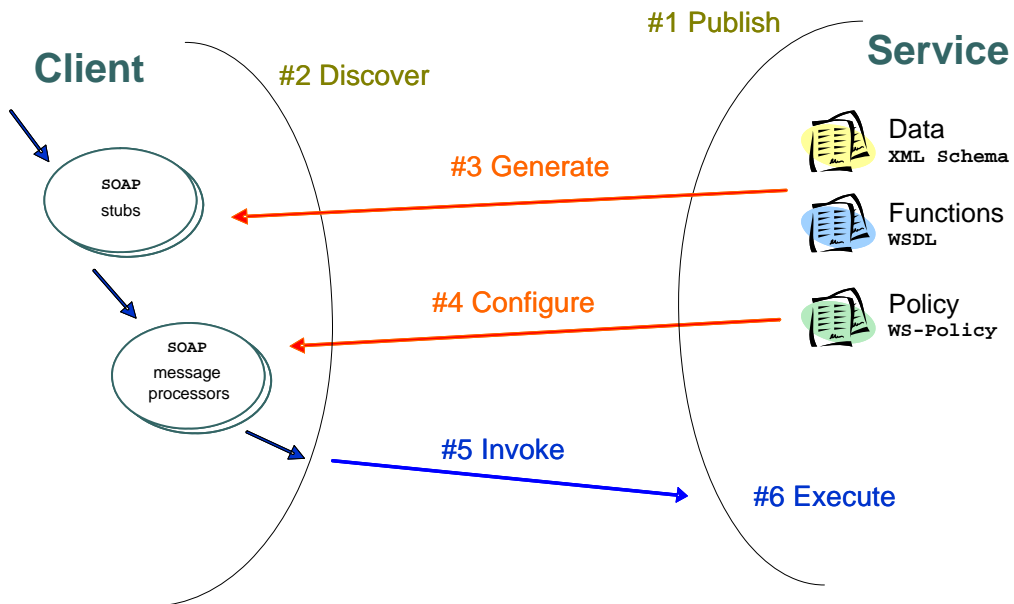


Figure 2 – Web Service client-server interaction.

The service is created and its endpoint is published. Its network location and meta-data contracts are stored in a descriptor or service registry. The *data contract* describes the data types used by the service interface, using XML Schema, like shown in example 2.

Example 2 – XML Schema for Notary service specifying the data structures and using an external schema for the sales contract document.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="urn:org.notary"
  xmlns:c="urn:org.seller:sale-contract"
  targetNamespace="urn:org.notary" version="1.0">
  <xs:import namespace="urn:org.seller:sale-contract"
    schemaLocation="http://seller.org:8080/sale-contract.xsd"/>
  <xs:element name="submitContract" type="ns1:submitContractType"/>
  <xs:complexType name="submitContractType">
    <xs:sequence>
      <xs:element name="contractId" type="xs:string" minOccurs="0"/>
      <xs:element name="contract" type="c:contractType" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="submitContractResponse"
type="ns1:submitContractResponseType"/>
  <xs:complexType name="submitContractResponseType"/>
  <xs:element name="InternalError" type="ns1:InternalErrorType"/>
  <xs:complexType name="InternalErrorType">
    <xs:sequence>
      <xs:element name="message" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="ContractNotAccepted" type="ns1:ContractNotAcceptedType"/>
  <xs:complexType name="ContractNotAcceptedType">
    <xs:sequence>
      <xs:element name="message" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

The *interface contract* describes the service functions through the input, output and fault messages, using WSDL, like shown in example 3.

Example 3 – WSDL for Notary service specifying the submitContract operation.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:tns="urn:org.notary"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="urn:org.notary"
  name="notaryService">
  <types>
    <xsd:schema>
      <xsd:import namespace="urn:org.notary" schemaLocation="notary.xsd" />
    </xsd:schema>
  </types>
  <message name="submitContract">
    <part name="parameters" element="tns:submitContract"/>
  </message>
  <message name="submitContractResponse">
    <part name="parameters" element="tns:submitContractResponse"/>
  </message>
  <message name="InternalError">
    <part name="fault" element="tns:InternalError"/>
  </message>
  <message name="ContractNotAccepted">
    <part name="fault" element="tns:ContractNotAccepted"/>
  </message>
  <portType name="notaryPortType">
    <operation name="submitContract">
      <input message="tns:submitContract"/>
      <output message="tns:submitContractResponse"/>
      <fault name="ContractNotAccepted" message="tns:ContractNotAccepted"/>
      <fault name="InternalError" message="tns:InternalError"/>
    </operation>
  </portType>
  <binding name="notaryPortBinding" type="tns:notaryPortType">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
```

```

    <operation name="submitContract">
      <soap:operation/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
      <fault name="ContractNotAccepted">
        <soap:fault name="ContractNotAccepted" use="literal"/>
      </fault>
      <fault name="InternalError">
        <soap:fault name="InternalError" use="literal"/>
      </fault>
    </operation>
  </binding>
  <service name="notaryService">
    <port name="notaryPort" binding="tns:notaryPortBinding">
      <soap:address location="http://server.org/notaryApp"/>
    </port>
  </service>
</definitions>

```

The *policy contract* describes additional requirements for service interaction, using WS-Policy, like shown in example 4.

Example 4 – WS-Policy for Notary service, using WS-SecurityPolicy assertions stating the use of Kerberos v5 protection.

```

<wsp:Policy>
  <sp:SymmetricBinding>
    <wsp:Policy>
      <sp:ProtectionToken>
        <wsp:Policy>
          <sp:KerberosV5APREQToken sp:IncludeToken=".../Once" />
        </wsp:Policy>
      </sp:ProtectionToken>
      <sp:SignBeforeEncrypting />
      <sp:EncryptSignature />
    </wsp:Policy>
  </sp:SymmetricBinding>
  <sp:SignedParts>
    <sp:Body/>
    <sp:Header Namespace="http://xmlsoap.org/ws/2004/08/addressing" />
  </sp:SignedParts>
  <sp:EncryptedParts>
    <sp:Body/>
  </sp:EncryptedParts>
</wsp:Policy>

```

The client is an application that wants to use the service resources. The client queries the service registry to find the service's network location and its contracts. The data and functions contracts are used to automatically generate invocation stubs that generate SOAP messages, like the one shown in example 5.

Example 5 – SOAP message of a request for the Notary service.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="urn:org.seller.sale-contract"
  xmlns:ns2="urn:org.notary">
  <soapenv:Body>
    <ns2:submitContract>
      <contractId>758032672389</contractId>
      <contract>
        <ns1:seller>
          <ns1:name>John Williams</ns1:name>
          <ns1:citizenId nr="32434333"
            issueDate="2003-03-07T12:56:11.000Z" />
          <ns1:fiscalId nr="1231233277" />
          <ns1:address>1319 South Avenue, City 99999, NY</ns1:address>
        </ns1:seller>
        <ns1:buyer>
          <ns1:name>Peter Patterson</ns1:name>
          <ns1:citizenId nr="123456789"
            issueDate="2004-12-02T12:22:11.000Z" />
          <ns1:fiscalId nr="23123122" />
          <ns1:address>2020 Hill Street, Town 88888, NY</ns1:address>
        </ns1:buyer>
        <ns1:terms>
          <ns1:clause nr="1">...</ns1:clause>
          <ns1:clause nr="2">...</ns1:clause>
          <ns1:clause nr="3">...</ns1:clause>
        </ns1:terms>
        <ns1:date>2006-07-22T11:00:00.000Z</ns1:date>
      </contract>
    </ns2:submitContract>
  </soapenv:Body>
</soapenv:Envelope>
```

The policy is used to configure message processors and support libraries, to add non-functional requirements like security. After the *client-service binding* is complete, the client invokes the service, using the stubs and configured processors to produce and send a message. The service receives the message, verifies it and executes. In some cases, it returns a response message.

Every exchanged message and contract is XML-based. This means that client and server can be developed in different implementations (ex. Java and Microsoft Dot Net) and interoperability is assured as long as standards are followed closely.

3 Evaluation

The survey described the main Web Service standards and implementations. They were evaluated using a prototype for a business case-study and custom tests. The evaluation *results* are presented in this section after the *case-study overview*.

3.1 Case-study overview

The case-study chosen for the evaluation was “real-estate contracts” because of its realism and complexity (several actors are involved) and also because of the value of the items, given the desired security emphasis.

The full business process and informational entities were modeled using a service-oriented extension of a methodology proposed by Guerra and Pardal [Guerra04] for enterprise architecture.

The prototype focused on the “agreement of sale between seller and buyer”. The prototype use-cases and interaction diagrams were modeled using UML [Fowler99]. The prototype specification and development explicitly accounted for *binding*, *invocation* and *key distribution*, shown in figures 3, 4, 5, respectively.

All the modeling can be consulted in detail in the publication by Pardal [Pardal06b].

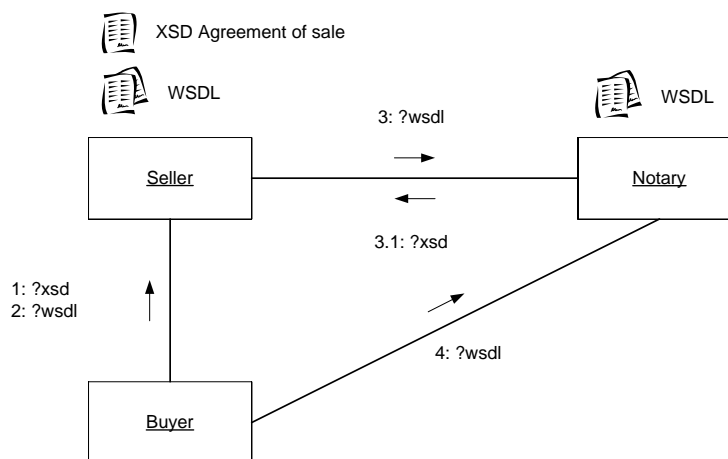


Figure 3 – Binding interaction diagram between seller, buyer and notary in the case-study. The agreement contract schema is proposed by the seller.

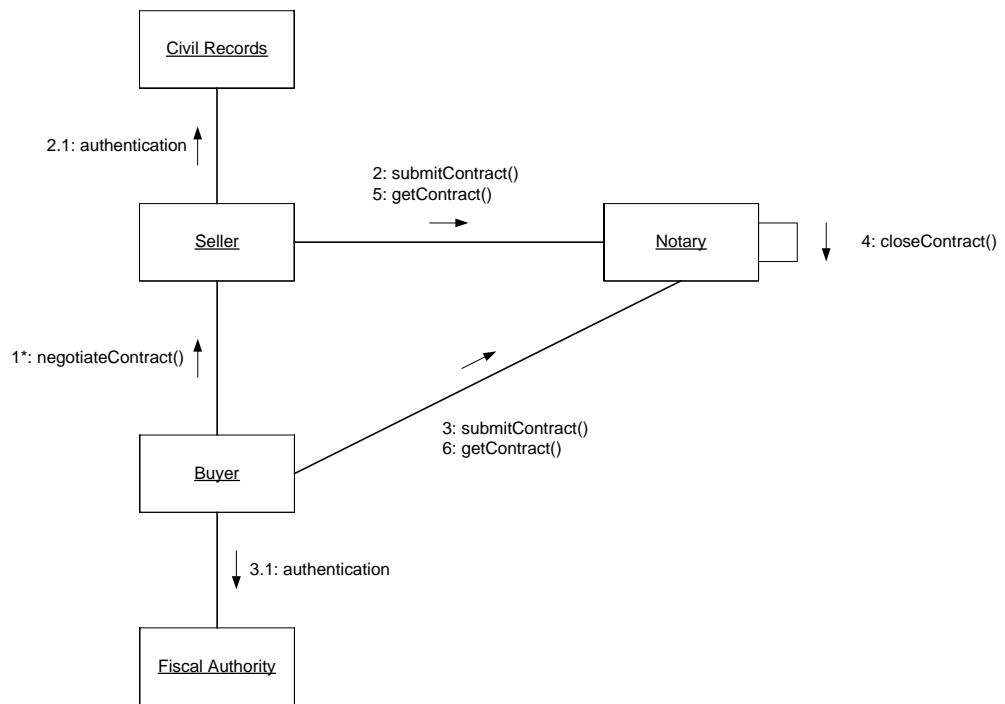


Figure 4 – Invocation interaction diagram between seller, buyer and notary in the case-study. The contract is submitted by both seller and buyer to the notary that verifies it.

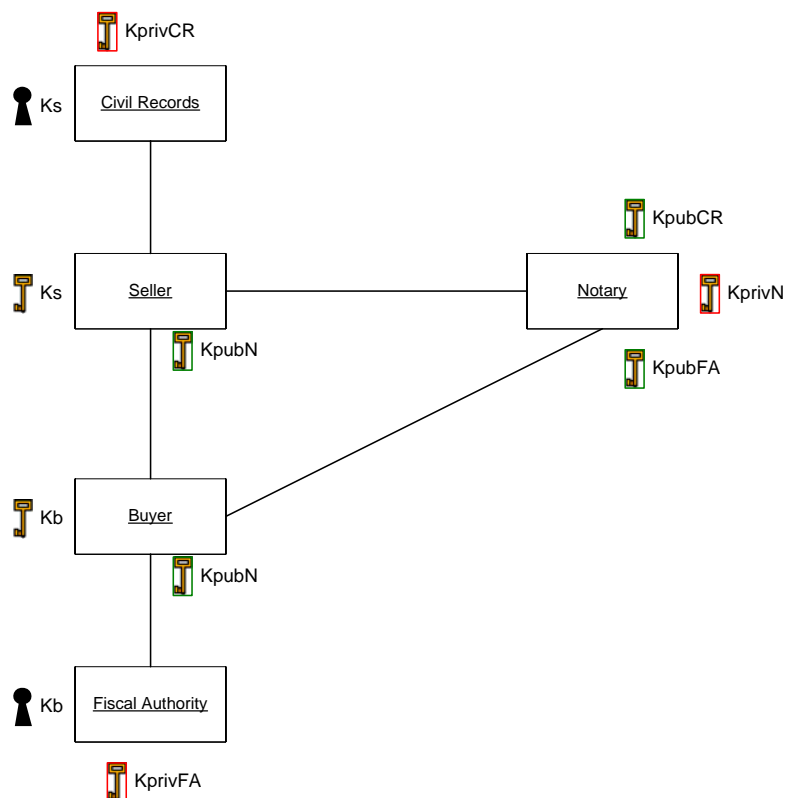


Figure 5 – Key distribution diagram. Legend: Kb (buyer’s secret key), Ks (seller’s secret key), Kpriv Kpub (asymmetric cryptographic key-pair) for CR (civil records that authenticates the seller), FA (fiscal authority that authenticates buyer) and N (notary).

3.2 Results

3.2.1 Web Services development

Tools used in prototype

The XML Schema, WSDL and WS-Policy documents were edited with a plain text editor instead of being automatically generated. The idea was to avoid platform biasing. However this was not an easy process, as the document's structure has several indirection levels, making them hard and tiresome for direct human usage.

The prototype development tools were JAX-B 2 and JAX-WS 2 for XML-Java data-binding and clients and services development, respectively. The configuration is based on code annotations and configuration files that require some effort to keep them coherent during development.

Dynamic binding

The binding was explicitly described in the prototype to allow the evaluation of data binding, function binding and policy binding features. The service binding is *static* if it's done in development-time or deployment-time and it's *dynamic* if it is done in invocation time.

JAX-B 2 and JAX-WS 2 and the other implementations enable static and dynamic binding of data and functions. However, policy binding is static and uses custom configuration formats.

The data and function binding is not very interesting in the general case of business Web Services because of the added complexity or human intervention necessary to achieve a valid interpretation context of operations. However, dynamic policy binding is much more interesting because it enables a service invocation to adapt to the circumstances in which it is happening, keeping the functionality semantics the same. The difference from functional binding is that whereas a functional domain is open to different business areas and semantics, the non-functional domain is actually much closed, with a restricted set of configuration alternatives that can be chosen automatically. For instance, there are only a handful of different security technologies for choosing.

Separation of data and function binding

JAX-WS 2 enables the separation of data binding (XSD) and function binding (WSDL), with each contract in an autonomous document. This custom binding mechanism is very useful, because *data schema sharing* is very common in business applications. “WSE 3 / Dot Net 2” and “WSS4J / Axis2 / Java” did not support this flexible mapping.

Data schemas reuse is very important to enterprise applications. A shared information catalog of schemas could even be the basis of enterprise information architecture for integration, as proposed by Guerra and Pardal [Guerra04], following work by Spewak [Spewak93].

Extensible data schemas

Data schemas can have *extensibility elements* (xsd:any) that allow document instances to have additional data. This enables service upgrades with backward compatibility and the transport of opaque information items.

JAX-B 2 handles extensibility elements as XML tree documents or converting them to Java classes when a data-binding context is available.

Data-binding limitations

JAX-WS 2 and JAX-B 2 handle data-binding for simple, complex, arrays and exceptions data types. However there are limitations inherent to the data-binding approach.

Sometimes it would be useful to switch from XML view to Java classes and vice-versa, without the cost of conversions. This could be achieved by *wrapping* the XML document in generated Java interfaces instead of classes, keeping the data in XML. This approach is currently not available in implementations. The most similar approach is navigating the XML document using XPath expressions. Another problem is the *memory usage* of full XML documents. AXIOM from Axis2 enables a gradual processing of the XML document, making it more efficient to parse large messages, with only a small increase in coding complexity that now has to deal with the reading state of the document.

3.2.2 Web Services protection

Transport security versus message security

In several cases, transport security can be enough protection for a Web Service. This approach is also called *point-to-point* because the scope of protection is the connection between two network nodes. The most common transport protection is HTTPS.

However, if the message goes through intermediate nodes before arriving at its destination or if it requires persistent confidentiality then message security is required. This approach is also called *end-to-end* because the scope of protection goes from the sender to the final receiver. The message protection technology is WS-Security.

The main advantage of transport security is simplicity, leveraging capabilities of application servers. A disadvantage is the authentication granularity because there is a single digital certificate for a server and a single set of trusted clients, so each service cannot have individual settings. Another disadvantage is that equal protection is provided for all data. Also, it's not possible to partially expose the parts of the message to intermediate nodes, if necessary.

The main advantage of message security is individual protection of parts of the message. Besides, security tokens can carry keys, certificates and assertions. It also allows transport independence. The main drawback is performance, with typically slower processing times and higher memory requirements. The messages themselves became larger and their structure more complex.

Transport security is simpler to use, but message security is more expressive and flexible. Both approaches can be combined. Indeed, a common practice is to sign the message to assure integrity and authenticity, and use HTTPS to assure data confidentiality. This solution is much more flexible than simple transport security and also has a much better performance than message cipher [Adams04].

Security mechanisms summary

Table 2 summarizes the message security mechanisms supported in current implementations. The following symbols are used, meaning:





































-  “Fully supports”;
-  “Partially supports”;
-  “Doesn’t support”.

Table 2 – Security mechanism support in current implementations.

Security mechanism		XWSS	Axis2	WSE 3
Authentication	User-password			
	X.509 certificates			
	SAML assertion			
	Kerberos			
Authorization	authentication based			
	using operating system			
	SAML assertion			
Message protection	Digital signature			
	Encryption			
	Repeated messages detection			
	Security sessions			

For configuration, XWSS uses a *callback* model for requesting security parameters in run-time. WSE 3 has generic *turn-key scenarios* for the most common configurations and *custom assertions* enabling a consistent configuration extension mechanism. However, there currently is no service implementation capable of dynamic policy binding.

Policy support

Contract-oriented security configuration using WS-SecurityPolicy is one of the most innovative features promised for Web Services security.

WS-SecurityPolicy specifies a XML format for describing available configuration alternatives. This enables *policy negotiation with client* (intersection of alternatives supported by client and service) and *policy merge with server during deployment* (union of alternatives supported by the service and by the server where it is being deployed). Example 4 shows a WS-SecurityPolicy example.

WS-SecurityPolicy will enable automatic security binding between client and service. However in current implementations it still isn't supported and the service's security configuration can only be applied on deployment-time, not on execution-time.

Cross-domain security information exchange

Other of the innovative features in Web Services security is cross-domain security information exchange using SAML.

SAML specifies an XML format for authentication, authorization and user attributes assertions. Any principal can create an assertion and whoever receives the assertion makes the trust decision. Example 6 shows a SAML authentication assertion example.

Example 6 – SAML authentication assertion example. It says that the user with e-mail address user@example.com has been authenticated by the asserter. The digital signature assures the information's integrity and authenticity.

```
<Assertion>
  <Conditions NotBefore="2006-07-22T12:02:00Z" NotOnOrAfter="2006-07-
22T13:02:00Z">
    <AudienceRestrictionCondition>
      <Audience>http://www.example.com/Members</Audience>
    </AudienceRestrictionCondition>
  </Conditions>
  <Advice>
    <AssertionIDReference>id</AssertionIDReference>
    <Assertion>...</Assertion>
  </Advice>
  <AuthenticationStatement AuthenticationMethod="urn:ietf:rfc:2246"
    AuthenticationInstant="2006-07-22T12:02:00Z">
    <Subject>
      <NameIdentifier
Format="urn:oasis:names:tc:SAML:1.0:assertion#emailAddress">
        user@example.com
      </NameIdentifier>
    </Subject>
  </AuthenticationStatement>
  <ds:Signature>...</ds:Signature>
</Assertion>
```

SAML will ease the exchange of security information across organizational borders, but in current implementation it still isn't secure enough to be trusted, because adequate digital signature support is missing.

4 Conclusion

This final section states the main *contributions* of this paper and identifies interesting *future work*. It concludes with the *final remarks*.

4.1 Contributions

Web Services technology was thoroughly surveyed, with an up-to-date account of standards and implementations and a simple example.

The protection capabilities of Web Services Security technology were evaluated using a prototype for a business case-study and custom tests in all implementations.

The implementation of non-functional requirements should be performed as an independent aspect of functional requirements. Web Services security is all about access control, message protection and configuration flexibility. The Web Services platform's features required for supporting security are the following:

- Requirement declaration;
- Configuration specification;
- Management of execution contexts;
- Message handling interception;
- Operation processing interception.

The *requirement declaration* is performed with a service policy that declares the security requirements. This capability is needed, for instance, to declare that a service can be invoked with transport security or with message security. This mechanism is still not supported in current implementations, but is expected in the next generation of implementations.

The *configuration specification* selects the security mechanisms to engage and the parameters to request from the application in run-time (ex. which digital certificate is used to sign messages). All current implementations support configuration files, although each uses its own custom format.

The *management of execution contexts* deals with state variables related with security. Contexts enable data sharing between the service platform and the application. Some relevant context scopes are: application, session, operation and thread. For instance, the session context allows the storing of a cryptographic key used to store a set of messages. Contexts are mostly supported in the

implementations, but their implementation is not consistent and flexible enough and should be improved.

The *message handling interception* provides access to the messages contents (headers and body) and routing. These capabilities allow, for instance, the forwarding of a rejected incoming message, sending it to a security node for reporting. In current implementations only linear interception flows is supported. There are prototypes for more elaborate flows, like SPEF (SOAP Profile Enabling Framework) [Malek05] from Fujitsu.

The *operation processing interception* allows decision points before the service code is actually executed. This implies that objects (ex. business logic, data access, remote stubs) in the application must be created in factories that can be customized to modify behavior. Using this feature it's possible, for instance, to implement generic authentication mechanisms. This feature is not supported in current implementations.

Of all the evaluated implementations, WSE 3 is the most advanced and robust, but it still doesn't support WS-Policy or SAML. WSS4J for Axis2 is also promising, but it's still unstable and its scarce documentation needs to be improved. XWSS for JAX-WS 2 is reasonably robust, supports the most important mechanisms, is extensible and has acceptable documentation. WSS4J and XWSS also have the advantage, from an evaluator's perspective, of being open-sourced.

4.2 Future work

Several interesting future work topics were identified.

Firstly, *the survey needs to keep pace with new developments* of standards and implementations. WS-Map [Pardal06a] is one effort in this direction.

Reliable messaging and transactions extensions also need to be evaluated, perhaps following a similar approach as the one followed for security i.e. using business case-studies.

There clearly is work to be done in *service-oriented development methodologies* that bridge SOA to Web Services, ranging all the way from requirements, to specification, to implementation.

Development tools should focus more on service contracts. An approach centered on them, with direct specification of data schemas, functions and policy, is more suitable for service development than Java or Dot Net centric approaches, because

the latter map their own concepts making the contracts less explicit and therefore more difficult to manage and maintain.

The main goal of tools should be *simplifying XML programming*. Data-wrapping instead of data-binding could enable more programming control with less data conversions. Also new abstractions could be used to broker context between services platform and applications. For instance, a WS-Security could perform automatic processing and produce a “security report” stating all security checks performed and the used parameters (ex. keys, certificates), leaving the final trust decisions to the application.

4.3 Final remarks

Currently, Web Services are a good way to integrate applications, especially if they are developed in different environments, like Java or Dot Net. Web Services can be protected with user-password and X.509 WS-Security at the message level or using HTTPS at the transport level. However, more advanced service-oriented features like policy binding and cross-domain information exchange are still not available. In the near future, both are expected to improve significantly.

Whenever possible, the security mechanisms should be implemented automatically by the platform, but they must share context with applications using meaningful abstractions to delegate trust decisions and others, in a simple, effective way. The “security report” proposal is an example of such an approach.

Web Services technology is indeed promising, but it still has a way to go before it can be regarded as truly revolutionary.

References

- Adams, H., (2004). Best Practices for Web services, Part 11: Web services security, IBM Developer Works, <http://www.ibm.com/developerworks/webservices/library/ws-best11/>
- Anderson, R., (2001). Security Engineering, Wiley
- Apache, (2006). Securing SOAP Messages with WSS4J, http://ws.apache.org/axis2/modules/rampart/1_0/security-module.html
- Barbir, A.; Gudgin, M.; McIntosh, M. & Morrison, K.S., (2005). WS-I Basic Security Profile Version 1.0, WS-I, Nortel Networks, Microsoft, IBM, Layer 7, <http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html>
- Booth, D. & Liu, C.K., (2005). Web Services Description Language (WSDL) Version 2.0, W3C, Hewlett-Packard, SAP Labs, <http://www.w3.org/TR/2005/WD-wsdl20-primer-20050803>
- Box, D. & Curbera, F., (2004). Web Services Addressing (WS-Addressing), W3C, Microsoft, IBM, BEA, SAP, <http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/>
- Bray, T.; Paoli, J.; McQueen, C.M.S.; Maler, E. & Yergeau, F., (2004). Extensible Markup Language (XML) 1.0 (Third Edition), W3C, Textuality and Netscape, Microsoft, Sun Microsystems, <http://www.w3.org/TR/2004/REC-xml-20040204>
- Cantor, S.; Kemp, J.; Philpott, R. & Maler, E., (2004). Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, OASIS, Internet2, Nokia, RSA Security, Sun Microsystems, <http://xml.coverpages.org/SAML-core-20-CD-01.pdf>
- Clement, L.; Hatley, A.; von Riegen, C. & Rogers, T., (2004). UDDI Version 3.0.2, OASIS, Systinet, IBM, SAP AG, Computer Associates, <http://uddi.org/pubs/uddi-v3.0.2-20041019.htm>
- Curbera, F. & Schlimmer, J., (2004). Web Services Metadata Exchange (WS-MetadataExchange), MSDN, Microsoft, IBM, Computer Associates, SAP, BEA Systems, Sun Microsystems, webMethods, <http://msdn.microsoft.com/ws/2004/09/ws-metadataexchange/>
- Curbera, F.; Leymann, F.; Storey, T.; Ferguson, D. & Weerawarana, S., (2005). Web Services Platform Architecture: Soap, WSDL, WS-Policy, WS-Addressing, WS-Bpel, WS-Reliable Messaging and More, Prentice Hall
- Czajkowski, K.; Ferguson, D.F.; Foster, I.; Frey, J.; Graham, S.; Sedukhin, I.; Snelling, D.; Tuecke, S. & Vambenepe, W., (2004). The WS-Resource Framework Version 1.0, Computer Associates International, Inc., Fujitsu Limited, Hewlett-Packard Development Company, International Business Machines Corporation and The University of Chicago, <http://www.globus.org/wsrp/specs/ws-wsrf.pdf>
- Davis, D., (2005). Web Services Polling (WS-Polling), W3C, IBM, <http://www.w3.org/Submission/2005/SUBM-ws-polling-20051026/>
- Dumbill, E.; Johnston, J. & Laurent, S.S., (2001). Programming Web Services with XML-RPC, O'Reilly

- Eastlake, D.; Reagle, J. & Solo, D., (2002). XML-Signature Syntax and Processing, W3C, <http://www.w3.org/TR/2002/REC-xmlsig-core-20020212/>
- Eastlake, D. & Reagle, J., (2002). XML Encryption Syntax and Processing, W3C, <http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/>
- Eddon, G. & Eddon, H., (1998). Inside Distributed COM, Microsoft Press
- Fallside, D.C. & Walmsley, P., (2004). XML Schema Part 0: Primer Second Edition, W3C, <http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/>
- Feingold, M., (2005). Web Services Coordination (WS-Coordination) Version 1.0, IBM, Microsoft, Hitachi, Arjuna Technologies, IONA, <http://www.ibm.com/developerworks/library/specification/ws-tx/>
- Ferris, C.; Liu, C.K.; Nottingham, M.; Yendluri, P.; Gudgin, M.; Ballinger, K. & Ehnebuske, D., (2004). WS-I Basic Profile Version 1.1, WS-I, Microsoft, IBM, SAP, BEA Systems, webMethods, <http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html>
- Ferris, C. & Langworthy, D., (2005). Web Services Reliable Messaging Protocol (WS-ReliableMessaging), Microsoft, IBM, BEA, TIBCO Software, <http://msdn.microsoft.com/library/en-us/dnglobspec/html/WS-ReliableMessaging.pdf>
- Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P. & Lee, T.B., (1999). Hypertext Transfer Protocol -- HTTP/1.1, IETF, <http://www.w3.org/Protocols/rfc2616/rfc2616.txt>
- Fowler, M.; Rice, D.; Foemmel, M.; Hieatt, E.; Mee, R. & Stafford, R., (2002). Patterns of Enterprise Application Architecture, Addison Wesley
- Fowler, M. & Scott, K., (1999). UML Distilled, Addison-Wesley
- Geller, A., (2004). Web Service Enumeration (WS-Enumeration), Microsoft, Systinet, Sonic Software, BEA, Computer Associates, <http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-enumeration.pdf>
- Geller, A., (2004). Web Service Transfer (WS-Transfer), Microsoft, Systinet, Sonic Software, BEA, Computer Associates, <http://msdn.microsoft.com/ws/2004/09/ws-transfer/>
- Geller, A., (2004). Web Services Eventing (WS-Eventing), Microsoft, IBM, TIBCO Software, BEA Systems, Computer Associates, Sun Microsystems, <http://www-128.ibm.com/developerworks/webservices/library/specification/ws-eventing/>
- Geller, A., (2004). Web Services for Management (WS-Management), Microsoft, Sun, Intel, AMD, Dell, <http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-management1004.pdf>
- Graham, S.; Simeonov, S.; Boubez, T.; Davis, D.; Daniels, G.; Nakamura, Y. & Neyama, R., (2001). Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI, Sams Publishing
- Graham, S. & Niblett, P., (2004). Web Services Notification (WS-Notification) Version 1.0, IBM, Sonic Software, TIBCO Software, Akamai Technologies, SAP AG, Globus, Argonne National Laboratory, Hewlett-Packard, <http://ifr.sap.com/ws-notification/ws-notification.pdf>

- Gudgin, M.; Hadley, M.; Mendelsohn, N.; Moreau, J. & Nielsen, H.F., (2003). SOAP Version 1.2 Part 1: Messaging Framework, W3C, Microsoft, Sun Microsystems, IBM, Canon, <http://www.w3.org/TR/2003/REC-soap12-part1-20030624/>
- Gudgin, M.; Mendelsohn, N.; Nottingham, M. & Ruellan, H., (2005). SOAP Message Transmission Optimization Mechanism, W3C, Microsoft, IBM, BEA, Canon, <http://www.w3.org/TR/2005/REC-soap12-mtom-20050125/>
- Gudgin, M. & Nadalin, A., (2005). Web Services Trust Language (WS-Trust), Microsoft, IBM, OpenNetwork, Layer 7, Computer Associates, VeriSign, BEA, Oblix, Reactivity, RSA Security, Ping Identity, VeriSign, Actional, <http://www.ibm.com/developerworks/library/specification/ws-trust/>
- Gudgin, M. & Nadalin, A., (2005). Web Services Secure Conversation Language (WS-SecureConversation), Microsoft, IBM, OpenNetwork, Layer 7, Computer Associates, VeriSign, BEA, RSA Security, Ping Identity, Actional, Computer Associates, <http://www.ibm.com/developerworks/library/specification/ws-seccon/>
- Guerra, M.; Pardal, M. & da Silva, M.M., (2004). An Integration Methodology based on the Enterprise Architecture, Proc. of the 2004 Conference of the UK Academy for Information Systems (UKAIS 2004), <http://mflpar.googlepages.com/GuerraPardalUkais2004.pdf>
- Housley, R.; Ford, W.; Polk, W. & Solo, D., (1999). Internet X.509 Public Key Infrastructure, IETF, <http://www.ietf.org/rfc/rfc2459.txt>
- Iwasa, K., (2004). Web Services Reliable Messaging TC WS-Reliability 1.1, OASIS, Fujitsu Limited, Novell, Inc., Oracle Corporation, Sun Microsystems, <http://docs.oasis-open.org/wsrn/ws-reliability/v1.1>
- Kaler, C. & Nadalin, A., (2005). Web Services Security Policy Language (WS-SecurityPolicy) Version 1.1, Microsoft, IBM, VeriSign, RSA Security, <http://www.ibm.com/developerworks/library/specification/ws-secpol/>
- Kavantzas, N.; Burdett, D. & Ritzinger, G., (2004). Web Services Choreography Description Language Version 1.0, W3C, Oracle, Commerce One, Novell, <http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/>
- Klensin, J., (2001). Simple Mail Transfer Protocol, IETF, <http://www.ietf.org/rfc/rfc2821.txt>
- J. Kohl, C.N., (1993). The Kerberos Network Authentication Service (V5), IETF, <http://www.ietf.org/rfc/rfc1510.txt>
- Krafzig, D.; Banke, K. & Slama, D., (2004). Enterprise SOA: Service-Oriented Architecture Best Practices, Prentice Hall PTR
- Laudon, K. & Laudon, J., (2002). Management Information Systems, Pearson Prentice-Hall
- Little, M., (2003). Web Services Composite Application Framework (WS-CAF) version 1.0, Sun, Oracle, IONA, Arjuna, Fujitsu, http://www.arjuna.com/library/specs/ws_caf_1-0/WS-CAF-Primer.pdf
- Lockhart, H., (1994). OSF DCE Guide to Developing Distributed Applications, McGraw-Hill

MacDonald, M., (2003). Microsoft .NET Distributed Applications: Integrating XML Web Services and .NET Remoting, Microsoft Press

Malek, H.B. & Durand, J., Kitsuregawa, M. (eds.), (2005). A SOAP Container Model for e-Business Messaging Requirements, Proceedings of WISE 2005, Springer-Verlag, , LNCS 3806, 643–652

McGovern, J.; Tyagi, S.; Stevens, M. & Matthew, S., (2003). Java Web Services Architecture , Morgan Kaufmann

Microsoft, (2005). Microsoft Web Services Enhancements (WSE) 3.0 documentation,
<http://msdn.microsoft.com/webservices/webservices/building/wse/default.aspx>

Anthony Nadalin, C.K., (2004). Web Services Security: SOAP Message Security 1.0 (WS-Security 2004), OASIS, IBM, Microsoft, Verisign, Sun, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

OMG, (1991). The Common Object Request Broker: Architecture and Specification (CORBA)

Pardal, M., (2006). WS-Map: Web Services Standards Map, WWW,
<http://mega.ist.utl.pt/~mflpar/ws-map>

Pardal, M.F.L., (2006). Security of Enterprise Applications in Service Architectures, Instituto Superior Técnico, <http://mflpar.googlepages.com/MScMflp20060908.pdf>

Rescorla, E., (2000). HTTP Over TLS, IETF, <http://www.ietf.org/rfc/rfc2818.txt>

Schlimmer, J., (2005). Devices Profile for Web Services, Microsoft, Ricoh, Intel, Lexmark, <http://specs.xmlsoap.org/ws/2005/05/devprof/devicesprofile.pdf>

Schlimmer, J., (2006). Web Services Policy Framework (WSPolicy) Version 1.2, Microsoft, IBM, VeriSign, Sonic Software, SAP, BEA Systems,
<http://specs.xmlsoap.org/ws/2004/09/policy/ws-policy.pdf>

Sedukhin, I. & Vambenepe, W., (2005). Web Services Distributed Management: Management of Web Services (WSDM-MOWS) 1.0 and Management Using Web Services (MUWS 1.0), OASIS, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm

Spewak, S. & Hill, S., (1993). Enterprise Architecture Planning, John Wiley & Sons

Sun, (2006). Java Web Services Developer Pack, Sun Microsystems Web Site,
<http://java.sun.com/webservices/>

Sun, (1997). Java Remote Method Invocation (RMI), Sun Microsystems Web Site,
<http://java.sun.com/products/jdk/rmi/index.jsp>

Thatte, S., (2003). Business Process Execution Language for Web Services Version 1.1, Microsoft, IBM, Siebel Systems, BEA, SAP,
<http://www.ibm.com/developerworks/library/specification/ws-bpel/>

Acronyms

Acronym	Meaning
AXIOM	Axis Object Model
CORBA	Common Object Request Broker Architecture
DCE	Distributed Computing Environment
DCOM	Distributed Component Object Model
HTML	HyperText Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	HTTP over SSL
JAX-B	Java Architecture for XML Data Binding
JAX-RPC	Java APIs for XML-based RPC
JAX-WS	Java API for XML Web Services
MTOM	Message Transmission Optimization Mechanism
RMI (Java)	Java Remote Method Invocation
RPC	Remote Procedure Call
SAML	Security Assertion Markup Language
SMTP	Simple Mail Transfer Protocol
SOA	Service-Oriented Architecture
SOAP	Service Oriented Architecture Protocol , Simple Object Access Protocol, (no meaning just a name)
SPEF	SOAP Profile Enabling Framework
SSL	Secure Sockets Layer
STS	Security Token Service
UDDI	Universal Description, Discovery, and Integration
UML	Unified Modelling Language
WCF	Windows Communication Foundation
WS	Web Service(s)
WS-BPEL	Web Services Business Process Execution Language
WS-CDL	Web Services Choreography Description Language
WSDL	Web Service Description Language
WSE	Web Services Enhancements
WS-I	Web Services Interoperability Organization
WSIT	Web Services Interoperability Technology (Project Tango)
WS-MEX	WS-MetadataExchange
WSS4J	Web Services Security For Java
XML	eXtensible Markup Language
XOP	XML-binary Optimized Packaging
XPath	XML Path
XSD	XML Schema Definition
XWSS	XML and Web Services Security