WEB SERVICES

Mohand-Said Hacid
University Claude Bernard Lyon 1 - France

Abstract: In the emerging world of Web services, services will be combined in innovative ways to form elaborate services out of building blocks of other services. This is predicated on having a common ground of vocabulary and communication protocols operating in a secured environment. Currently, massive standardization efforts are aiming at achieving this common ground. We discuss aspects related to services, such as possible architectures, modeling, discovery, composition and security.

Key words: Web services architecture, Web services modeling, Web services discovery.

1. INTRODUCTION

A Web service is programmable application logic accessible using standard Internet protocols. Web services combine the best aspects of component-based development and the Web. Like components, Web services represent functionality that can be easily reused without knowing how the service is implemented. Unlike current component technologies which are accessed via proprietary protocols, Web services are accessed via ubiquitous Web protocols (ex: HTTP) using universally accepted data formats (ex: XML).

In practical business terms, Web services have emerged as a powerful mechanism for integrating disparate IT systems and assets. They work using widely accepted, ubiquitous technologies and are governed by commonly adopted standards. Web Services can be adopted incrementally at low cost. Today, enterprises use Web services for point-to-point application integration, to reuse existing IT assets, and to securely connecting to

business partners or customers. Independent Software Vendors embed Web services functionality in their software products so they are easier to deploy.

From a historical perspective, Web services represent the convergence between the service-oriented architecture (SOA) and the Web. SOAs has evolved over the last years to support high performance, scalability, reliability, and availability. To achieve the best performance, applications are designed as services that run on a cluster of centralized application servers. A service is an application that can be accessed through a programmable interface. In the past, clients accessed these services using a tightly coupled, distributed computing protocol, such as DCOM, CORBA, or RMI. While these protocols are very effective for building a specific application, they limit the flexibility of the system. The tight coupling used in this architecture limits the reusability of individual services. Each of the protocols is constrained by dependencies on vendor implementations, platforms, languages, or data encoding schemes that severely limit interoperability. Additionally, none of these protocols operates effectively over the Web.

The Web services architecture takes all the best features of the service-oriented architecture and combines it with the Web. The Web supports universal communication using loosely coupled connections. Web protocols are completely vendor-,platform-, and language-independent. The resulting effect is an architecture that eliminates the usual constraints of distributed computing protocols. Web services support Web-based access, easy integration, and service reusability.

A Web service is an application or information resource that can be accessed using standard Web protocols. Any type of application can be offered as a Web service. Web services are applicable to any type of Web environment: Internet, intranet, or extranet. Web services can support business-to-consumer, business-to-business, department-to-department, or peer-to-peer interactions. A Web service consumer can be a human user accessing the service through a desktop or wireless browser, it can be an application program, or it can be another Web service. Web Services support existing security frameworks.

1.1 Characteristics of Web Services

A Web service exhibits the following characteristics:

- A Web service is accessible over the Web. Web services communicate using platform-independent and language-neutral Web

protocols. These Web protocols ensure easy integration of heterogeneous environments.

- A Web service provides an interface that can be called from another program. This application-to-application programming interface can be invoked from any type of application client or service. The Web service interface acts as a liaison between the Web and the actual application logic that implements the service.
- A Web service is registered and can be located through a Web service Registry. The registry enables service consumers to find services that match their needs.
- Web services support loosely coupled connections between systems. They communicate by passing messages to each other. The Web service interface adds a layer of abstraction to the environment that makes the connections flexible and adaptable.

1.2 Web Services Technologies

Web services can be developed using any programming language and can be deployed on any platform. Web services can communicate because they all speak the same language: the Extensible Markup Language (XML). Web services use XML to describe their interfaces and to encode their messages. XML-based Web services communicate over standard Web protocols using XML interfaces and XML messages, which any application can interpret.

However, XML by itself does not ensure effortless communication. The applications need standard formats and protocols that allow them to properly interpret the XML. Hence, three XML-based technologies are emerging as the standards for Web services:

- Simple Object Access Protocol **(SOAP)** [1] defines a standard communications protocol for Web services.
- Web Services Description Language **(WSDL)** [3] defines a standard mechanism to describe a Web service.
- Universal Description, Discovery and Integration **(UDDI)** [2] provides a standard mechanism to register and discover Web services.

The rest of the chapter is organized as follows: Section 2 gives an overview of the classical approach to Web services (architecture and components). Section 3 introduces semantic Web services. We conclude in section 4.

2. WEB SERVICES ARCHITECTURE

Distributed computing has always been difficult. Now the business world has lined up behind the term "Web services" to try and build services that are highly reliable and scalable.

Many Web services architectures today are based on three components (figure 1): the service requestor, the service provider, and the service registry, thereby closely following a client/server model with an explicit name and directory service (the service registry). Although simple, such an architecture illustrates quite well the basic infrastructure necessary to implement Web services: a way to communicate (SOAP), a way to describe services (WSDL), and a name and directory server (UDDI). SOAP, WSDL and UDDI are nowadays the core of Web services. Specifications covering other aspects are typically designed based on SOAP, WSDL and UDDI. This is similar to the way conventional middleware platforms are built, where the basic components are interaction protocols, IDLs, and name and directory services.

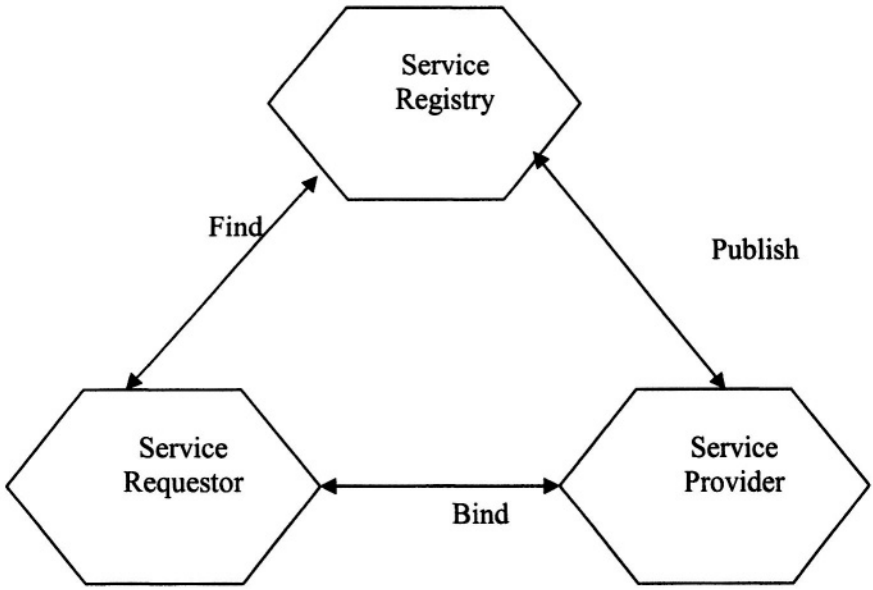

Figure 1. Web Services Architecture

Figure 2 shows how the main components of a Web service architecture relate to one another. When a service provider wants to make the service

available to service consumers, he describes the service using WSDL and registers the service in a UDDI registry. The UDDI registry will then maintain pointers to the WSDL description and to the service. When a service consumer wants to use a service, he queries the UDDI registry to find a service that matches his needs and obtains the WSDL description of the service, as well as the access point of the service. The service consumer uses the WSDL description to construct a SOAP message with which to communicate with the service.

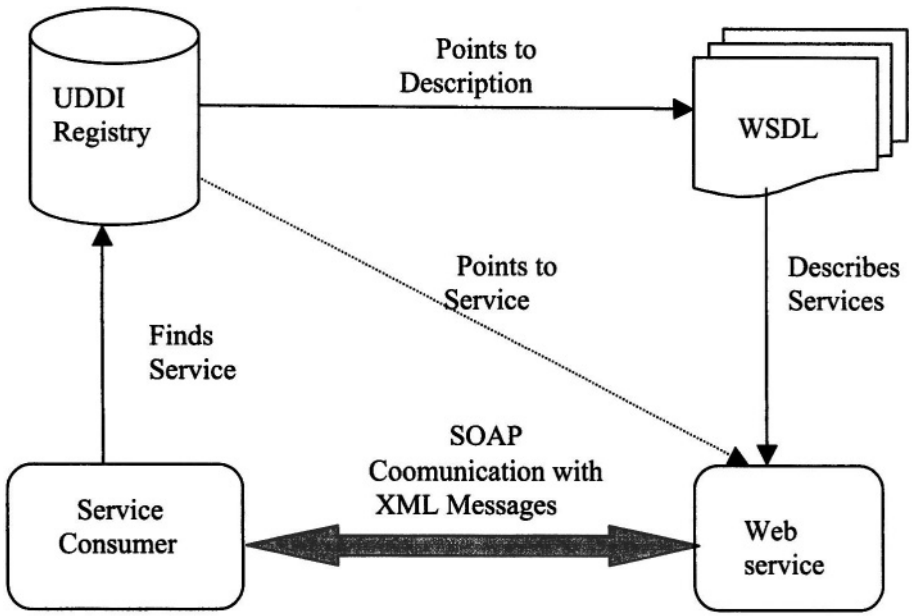

Figure 2. Web services Components – current technologies

2.1 SOAP

SOAP is an extensible XML messaging protocol that forms the foundation for Web Services. SOAP provides a simple and consistent mechanism that allows one application to send an XML message to another application. Fundamentally, SOAP supports peer-to-peer communications (figure 3). A SOAP message is a one-way transmission from a SOAP sender to a SOAP receiver, and any application can participate in an exchange as either a SOAP sender or a SOAP receiver. SOAP messages may be combined to support many communication behaviors, including request/ response, solicit response, and notification.

SOAP was first developed in late 1999 by DevelopMentor, Microsoft, and UserLand as a Windows-specific XML-based remote procedure call (RPC) protocol. In early 2000 Lotus and IBM joined the effort and helped produce an open, extensible version of the specification that is both platform-and language-neutral. This version of the specification, called SOAP 1.1 (see http://www.w3.org/TR/SOAP/), was submitted to the World Wide Web Consortium (W3C). W3C subsequently initiated a standardization effort.

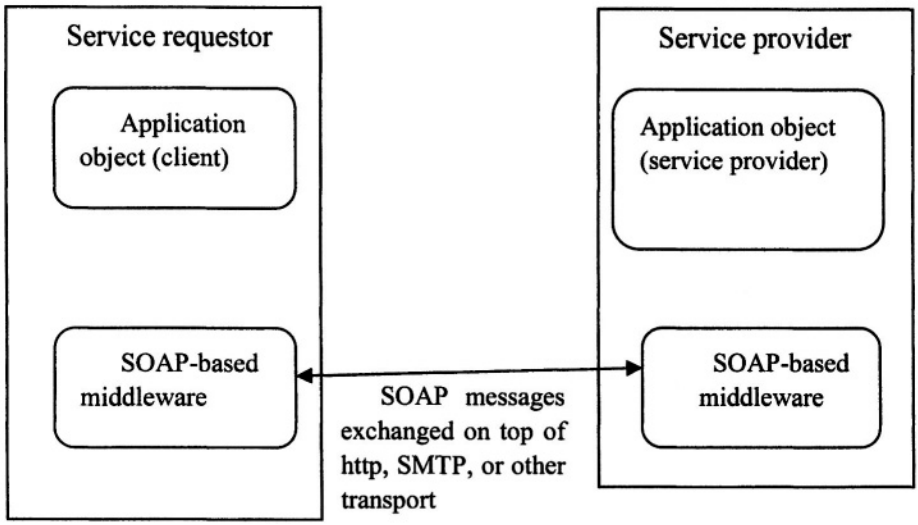

Figure 3. Clients can invoke Web services by exchanging SOAP messages

A pictorial representation of the SOAP message is given in figure 4.
Clients can invoke Web services by exchanging SOAP messages.

- **SOAP Envelope.** The SOAP envelope provides a mechanism to identify the contents of a message and to explain how the message should be processed. A SOAP envelope includes a SOAP header and a SOAP body. The SOAP header provides an extensible mechanism to supply directive or control information about the message. For example, a SOAP header could be used to implement transactions, security, reliability, or payment mechanisms. The SOAP body contains the payload that is being sent in the SOAP message.

- **SOAP Transport Binding Framework.** It defines bindings for HTTP and the HTTP Extension Framework.
- **SOAP Serialization Framework.** All data passed through SOAP messages are encoded using XML, but there is no default serialization mechanism to map application-defined datatypes to XML elements. Data can be passed as literals or as encoded values. Users can define their own serialization mechanism, or they can use the serialization mechanism defined by the SOAP encoding rules. The SOAP encoding style is based on a simple type system derived from the W3C XML Schema Part 2:Datatypes Recommendation (see http://www.w3.org/TR/xmlschema-2/). It supports common features found in the type systems of most programming languages and databases. It supports simple scalar types, such as "string ", "integer ", and "enumeration ", and it supports complex types, such as "struct" and "array ".
- **SOAP RPC Representation.** SOAP messaging supports very loosely coupled communications between two applications. The SOAP sender sends a message and the SOAP receiver determines what to do with it. The SOAP sender does not really need to know anything about the implementation of the service other than the format of the message and the access point URI. It is entirely up to the SOAP receiver to determine, based on the contents of the message, what the sender is requesting and how to process it. SOAP also supports a more tightly coupled communication scheme based on the SOAP RPC representation. The SOAP RPC representation defines a programming convention that represents RPC requests and responses. Using SOAP RPC, the developer formulates the SOAP request as a method call with zero or more parameters. The SOAP response returns a return value and zero or more parameters. SOAP RPC requests and responses are marshaled into a "struct" datatype and passed in the SOAP body.

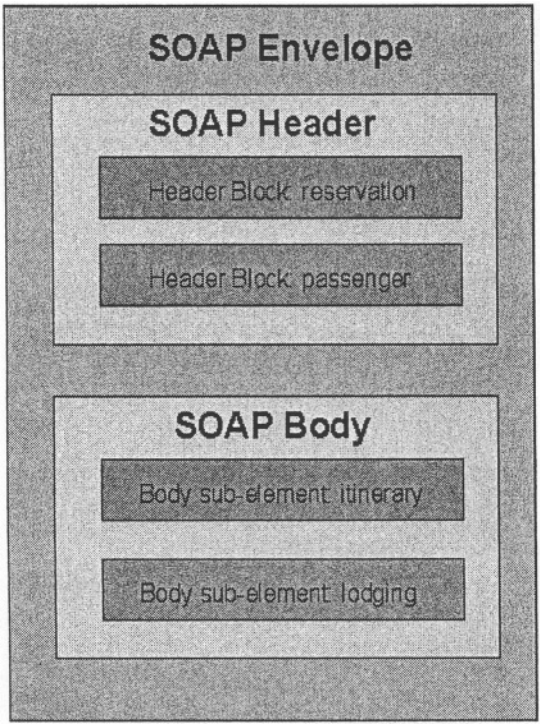

Figure 4. Structure of SOAP messages

2.1.1 SOAP Message Exchange

SOAP is a simple messaging framework for transferring information specified in the form of an XML infoset between an initial SOAP sender and an ultimate SOAP receiver. The more interesting scenarios typically involve multiple message exchanges between these two nodes. The simplest such exchange is a request-response pattern. Some early uses of SOAP emphasized the use of this pattern as means for conveying remote procedure calls (RPC), but it is important to note that not all SOAP request-response exchanges can or need to be modeled as RPCs. The latter is used when there is a need to model a certain programmatic behavior, with the exchanged messages conforming to a pre-defined description of the remote call and its return. A much larger set of usage scenarios than that covered by the request-response pattern can be modeled simply as XML-based content exchanged in SOAP messages to form a back-and-forth "conversation", where the semantics are at the level of the sending and receiving applications.

2.2 WSDL

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. WebMethods' Web Interface Definition Language (WIDL), one of the pioneering specifications for description of remote Web services, was an XML format that took familiar approach (which was accessing functionality on a remote machine as if it were on a local machine) to users of remote procedural technologies, such as RPC and CORBA. There was some fit between WIDL and the XML-RPC system by UserLand. The former has since faded away, as message-based XML technologies have proven more popular than their procedural equivalents. The latter seems to be giving way to SOAP, which has support for message-oriented as well as procedural approaches.

The Web services Description Language (WSDL) is an XML-based language used to describe the services a business offers and to provide a way for individuals and other businesses to access those services electronically. WSDL is the cornerstone of the Universal Description, Discovery, and Integration (UDDI) initiative spearheaded by Microsoft, IBM, and Ariba. UDDI is an XML-based registry for businesses worldwide, which enables businesses to list themselves and their services on the Internet. WSDL is the language used to do this.

WSDL is derived from Microsoft's Simple Object Access Protocol (SOAP) and IBM's Network Accessible Service Specification Language (NASSL). WSDL replaces both NASSL and SOAP as the means of expressing business services in the UDDI registry.

2.2.1 WSDL Document Types

To assist with publishing and finding WSDL service descriptions in a UDDI Registry, WSDL documents are divided into two types: *service interfaces* and *service implementations* (see figure 5).

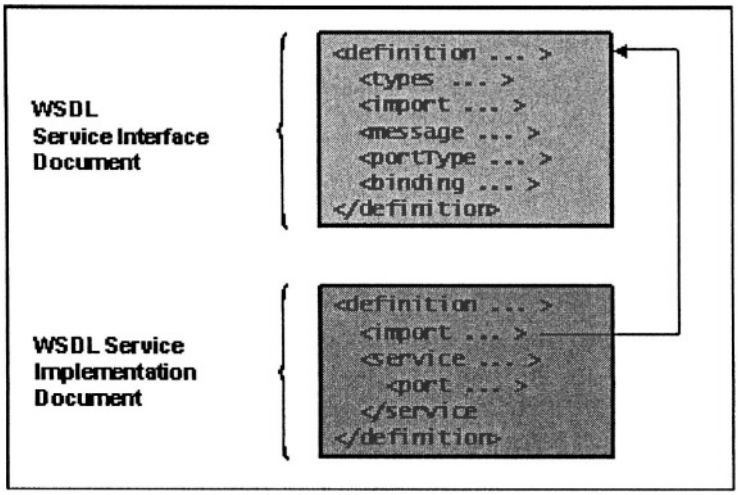

Figure 5. WSDL document types

A service interface is described by a WSDL document that contains the *types, import, message, portType,* and *binding* elements. A service interface contains the WSDL service definition that will be used to implement one or more services. It is an abstract definition of a Web service, and is used to describe a specific type of service.

A service interface document can reference another service interface document using an import element. For example, a service interface that contains only the message and portType elements can be referenced by another service interface that contains only bindings for the portType.

The WSDL service implementation document will contain the *import* and *service* elements. A service implementation document contains a description of a service that implements a service interface. At least one of the import elements will contain a reference to the WSDL service interface document. A service implementation document can contain references to more than one service interface document.

The import element in a WSDL service implementation document contains two attributes. The *namespace* attribute value is a URL that matches the *targetNamespace* in the service interface document. The *location* attribute is a URL that is used to reference the WSDL document that contains the complete service interface definition. The *binding* attribute on the port element contains a reference to a specific binding in the service interface document.

The service interface document is developed and published by the *service interface provider*. The service implementation document is created and published by the *service provider*. The roles of the service interface provider and service provider are logically separate, but they can be the same business entity.

A complete WSDL service description is a combination of a service interface and a service implementation document. Since the service interface represents a reusable definition of a service, it is published in a UDDI registry as a tModel. The service implementation describes instances of a service. Each instance is defined using a WSDL service element. Each service element in a service implementation document is used to publish a UDDI businessService.

When publishing a WSDL service description, a service interface must be published as a tModel before a service implementation is published as a businessService.

2.3 UDDI

Just before WSDL emerged, a consortium of 36 companies, including IBM, Ariba, and Microsoft, launched the Universal Description, Discovery and Integration (UDDI) system, an initiative to provide a standard directory of on-line business services with an elaborate API for querying the directories and service providers.

The key item of consideration in the UDDI specifications is the "Web service." A Web service describes specific business functionality exposed by a company, usually through an Internet connection, for the purpose of providing a way for another company or software program to use the service. The UDDI specifications define a way to publish and discover information about Web services. UDDI aims to automate the process of publishing your preferred way of doing business, finding trading partners and have them find you, and interoperate with these trading partners over the Internet.

Prior to the UDDI project, no industry-wide approach was available for businesses to reach their customers and partners with information about their products and Web services. Nor was there a uniform method that detailed how to integrate the systems and processes that are already in place at and between business partners. Nothing attempted to cover both the business and development aspects of publishing and locating information associated with a piece of software on a global scale.

Conceptually, a business can register three types of information into a UDDI registry. The specification does not call out these types specifically, but they provide a good summary of what UDDI can store for a business:

- **White pages.** Basic contact information and identifiers about a company, including business name, address, contact information, and unique identifiers such as tax IDs. This information allows others to discover your Web service based upon your business identification.
- **Yellow pages.** Information that describes a Web service using different categorizations (taxonomies). This information allows others to discover your Web service based upon its categorization (such as being in the manufacturing or car sales business).
- **Green pages.** Technical information that describes the behaviors and supported functions of a Web service hosted by your business. This information includes pointers to the grouping information of Web services and where the Web services are located.

2.3.1 Why UDDI?

Most eCommerce-enabling applications and Web services currently in place take divergent paths to connecting buyers, suppliers, marketplaces and service providers. Without large investments in technology infrastructure, businesses of all sizes and types can only transact Internet-based business with global trading partners they have discovered and who have the same applications and Web services.

UDDI aims to address this impediment by specifying a framework which will enable businesses to:

- Discover each other.
- Define how they interact over the Internet.
- Share information in a global registry that will more rapidly accelerate the global adoption of B2B eCommerce.

2.3.2 UDDI Business Registry

UDDI relies upon a distributed registry of businesses and their service descriptions implemented in a common XML format.

The UDDI Business Registry provides an implementation of the UDDI specification. Any company can access the registry on the Internet, enter the description of its business, reach a UDDI site and search through all the business services listed in the UDDI registry. There is no cost to access

information in the registry. Though based on XML, the registry can also describe services implemented in HTML, CORBA, or any other type of programming model or language.

3. TOWARDS SEMANTIC WEB SERVICES

3.1 Introduction

Semantic Web services are emerging as a promising technology for the effective automation of services discovery, combination, and management [25, 21, 20]. They aim at leveraging two major trends in Web technologies, namely Web services and Semantic Web :

- Web services built upon XML as vehicle for exchanging messages across applications. The basic technological infrastructure for Web services is structured around three major standards: SOAP, WSDL, and UDDI [33, 16]. These standards provide the building blocks for service description, discovery, and communication. While Web services technologies have clearly influenced positively the potential of the Web infrastructure by providing programmatic access to information and services, they are hindered by lack of rich and machine-processable abstractions to describe service properties, capabilities, and behavior. As a result of these limitations, very little automation support can be provided to facilitate effective discovery, combination, and management of services. Automation support is considered as the cornerstone to provide effective and efficient access to services in large, heterogeneous, and dynamic environments [10, 33, 20]. Indeed, until recently the basic Web services infrastructure was mainly used to build simple Web services such as those providing information search capabilities to an open audience (e.g. stock quotes, search engine queries, auction monitoring).
- Semantic Web aims at improving the technology to organize, search, integrate, and evolve Web-accessible resources (e.g., Web documents, data) by using rich and machine-understandable abstractions for the representation of resources semantics. Ontologies are proposed as means to address semantic heterogeneity among Web-accessible information sources and services. They are used to provide meta-data for the effective manipulation of available information including discovering information sources and reasoning about their capabilities. Efforts in this area include the

development of ontology languages such as RDF, DAML, and DAML+OIL [18]. In the context of Web services, ontologies promise to take interoperability a step further by providing rich description and modeling of services properties, capabilities, and behavior.

By leveraging efforts in both Web services and semantic Web, semantic Web services paradigm promises to take Web technologies a step further by providing foundations to enable automated discovery, access, combination, and management of Web services. Efforts in this area focus on providing rich and machine understandable representation of services properties, capabilities, and behavior as well as reasoning mechanisms to support automation activities [25, 11, 21, 20, 13, 8]. Examples of such efforts include DAML-S, WSMF (Web services Modeling Framework) [21], and METEOR-S (http://lsdis.cs.uga.edu/proj/meteor/SWP.tm).Work in this area is still in its infancy. Many of the objectives of the semantic Web services paradigm, such as capability description of service, dynamic service discovery, and goal-driven composition of Web services remain to be reached.

3.2 Web Services and their Complexity

Many Web service description languages distinguish between elementary and complex Web services. Elementary Web services are simple input/output boxes, whereas complex Web services break down the overall process into sub-tasks that may call other web services. Strictly speaking, such a distinction is wrong and may lead to mis-conceptualizations in a Web service modeling framework. It is not the complexity of the Web service that makes an important distinction. It is rather the complexity of its *description* or its interface (in terms of static and dynamic) that makes a difference. A complex Web service such as a logical inference engine with a web interface can be *described* as rather elementary. It receives some input formulas and derives--after a while--a set of conclusions. A much simpler software product such as a simple traveling information system may be broken down into several Web services around hotel information, flight information, and general information about a certain location. Therefore, it is not the inherent complexity of a Web service, it is the complexity of its external visible description that makes the relevant difference in our context. This insight may look rather trivial, however, it has some important consequences:

- Many Web service description approaches do not make an explicit distinction between an internal description of a Web service and its external visible description. They provide description means such as

data flow diagrams and control flow descriptions without making clear whether they should be understood as interface descriptions for accessing a Web service, or whether they should be understood as internal descriptions of the realization of a Web service. Often, the internal complexity of a Web service reflects the business intelligence of a Web service provider. Therefore, it is essential for him not to make it publicly accessible. This is the major conceptual distinction between an internal description of the workflow of a Web service and its interface description.

- The dichotomy of elementary and complex Web services is too simplistic. As we talk about the complexity of the *description* of a Web service it is necessary to provide a scale of complexity. That is, one starts with some description elements and gradually upscale the complexity of available description elements by adding additional means to describe various aspects of a Web service.

3.3 Functionalities Required for Successful Web Services

UDDI, WSDL, and SOAP are important steps in the direction of a web populated by services. However, they only address part of the overall stack that needs to be available in order to eventually achieve the semantic Web services vision. [9] identifies the following elements as being necessary to achieve scalable Web service discovery, selection, mediation and composition:

- **Document types.** Document types describe the content of business documents like purchase orders or invoices. The content is defined in terms of elements like an order number or a line item price. Document types are instantiated with actual business data when a service requester and a service provider exchange data. The payload of the messages sent back and forth is structured according to the document types defined.
- **Semantics.** The elements of document types must be populated with correct values so that they are semantically correct and are interpreted correctly by the service requesters and providers. This requires that vocabulary is defined that enumerates or describes valid element values. For example, a list of product names or products that can be ordered from a manufacturer. Further examples are units of measure as well as country codes. Ontologies provide a means for defining the concepts of the data exchanged. If ontologies are available document types refer to the ontology concepts. This

ensures consistency of the textual representation of the concepts exchanged and allows the same interpretation of the concepts by all trading partners involved. Finally, the intent of an exchanged document must be defined. For example, if a purchase order is sent, it is not clear if this means that a purchase order needs to be created, deleted or updated. The intent needs to make semantically clear how to interpret the sent document.

- **Transport binding.** Several transport mechanisms are available like HTTP/S, S/MIME, FTP or EDIINT. A service requester as well as a service provider has to agree on the transport mechanism to be used when service requests are executed. For each available transport mechanism the layout of the message must be agreed upon and how the document sent shall be represented in the message sent. SOAP for example defines the message layout and the position within the message layout where the document is to be found. In addition, header data are defined, a requirement for SOAP message processing.
- **Exchange sequence definition.** Communication over networks is currently inherently unreliable. It is therefore required that service requester and service provider make sure themselves through protocols that messages are transmitted exactly once. The exchange sequence definition achieves this by defining a sequence of acknowledgment messages in addition to time-outs, retry logic and upper retry limits.
- **Process definition.** Based on the assumption that messages can be exchanged exactly once between service requester and service provider, the business logic has to be defined in terms of the business message exchange sequence. For example, a purchase order might have to be confirmed with a purchase order acknowledgment. Or, a request for quotation can be responded to by one or more quotes. These processes define the required business message logic in order to derive to a consistent business state. For example, when goods are ordered by a purchase order and confirmed by a purchase order acknowledgment they have to be shipped and paid for, too.
- **Security.** Fundamentally, each message exchange should be private and unmodified between the service requester and service provider as well as non-reputable. Encryption, as well as signing, ensures the unmodified privacy whereby non-repudiation services ensure that neither service requester nor service provider can claim not to have sent a message or to have sent a different one.

- **Syntax.** Documents can be represented in different syntaxes available. XML is a popular syntax, although non-XML syntax is used, too (e.g. EDI).
- **Trading partner specific configuration.** Service requesters or service providers implement their business logic differently from each other. The reason is that they establish their business logic before any cooperation takes place. This might require adjustments once trading partners are found and the interaction should be formalized using Web services. In case modifications are necessary, trading partner specific changes have to be represented. Current Web service technology scares rather low compared to these requirements. Actually, SOAP provides support on information binding. Neither UDDI nor WSDL add any support in the terms enumerated above. Many organizations had the insight that message definition and exchange are not sufficient to build an expressive Web services infrastructure. In addition to UDDI, WSDL and SOAP standards for process definitions as well as exchange sequence definitions are proposed such as WSFL [23], XLANG [32], ebXML BPSS [35], BPML [5] and WSDL [12]. Still, there are important features missing in all of the mentioned frameworks. Very important is to reflect the *loose coupling* and *scalable mediation* of Web services in an appropriate modeling framework. This requires mediators that map between different document structures and different business logics as well as the ability to express the difference between publicly visible workflows (public processes) and internal business logics of a complex Web service (private processes). Therefore, a fully-fledged Web service Modeling Framework (WSMF) [4] was proposed.

3.4 Semantic Markup for Web Services

To make use of Web service, a software agent needs a computer-interpretable description of the service, and the means by which it is accessed. An important goal for Semantic Web markup languages, then, is to establish a framework within which these descriptions are made and shared. Web sites should be able to employ a set of basic classes and properties for declaring and describing services, and the ontology structuring mechanisms of DAML+OIL provide the appropriate framework within which to do this.

Services can be simple or primitive in the sense that they invoke only a single Web-accessible computer program, sensor, or device that does not rely upon another Web service, and there is no ongoing interaction between

the user and the service, beyond a simple response. Alternatively, services can be complex, composed of multiple primitive services, often requiring an interaction or conversation between the user and the services, so that the user can make choices and provide information conditionally. DAML-S is meant to support both categories of services, but complex services have provided the primary motivations for the features of the language. The following tasks are expected from DAML-S [14, 27, 14, 15]:

- **Automatic Web service Discovery.** Automatic Web service discovery involves the automatic location of Web services that provide a particular service and that adhere to requested constraints. For example, the user may want to find a service that makes hotel reservations in a given city and accepts a particular credit card. Currently, this task must be performed by a human who might use a search engine to find a service, read the Web page, and execute the service manually, to determine if it satisfies the constraints. With DAML-S markup of services, the information necessary for Web service discovery could be specified as computer-interpretable semantic markup at the service Web sites, and a service registry or ontology-enhanced search engine could be used to locate the service automatically. Alternatively, a server could proactively advertise itself in DAML-S with a service registry, also called middle agent [17, 37, 24], so that the requesters can find it when they query the registry.

- **Automatic Web service Invocation.** Automatic Web service invocation involves the automatic execution of an identified Web service by a computer program or agent. For example, the user could request the purchase of an airline ticket from a particular site on a particular flight. Currently a user must go to the Web site offering that service, fill out a form, and click on a button to execute the service. Alternatively, the user might send an HTTP request directly to the service with the appropriate parameters in HTML. In either case, a human is necessary in the loop. Execution of a Web service can be thought of as a collection of function calls. DAML-S markup of Web services provides a declarative, computer-interpretable API for executing these function calls.

- **Automatic Web service Composition and Interoperation.** This task involves the automatic selection, composition, and

interoperation of Web services to perform some task, given a high-level description of an objective. For example, the user may want to make all the travel arrangements for a trip to a conference. Currently, the user must select the Web services, specify the composition manually, and make sure that any software needed for the interoperation is custom-created. With DAML-S markup of Web services, the information necessary to select and compose services will be encoded at the service Web sites.

- **Automatic Web service Execution Monitoring.** Individual services and, even more, compositions of services, will often require some time to execute completely. A user may want to know during this period what the status of his or her request is, or plans may have changed, thus requiring alterations in the actions the software agent takes.

3.5 Services Composition

Composition of Web services that have been previously annotated with semantics and discovered by a mediation platform is another benefit proposed by Semantic Web for Web services. Composition of services can be quite simple sequence of service calls passing outputs of one service to the next and much more complex, where *execution path* (service workflow) is not a sequence but more sophisticated structure, or intermediate data transformation is required to join outputs of one service with inputs of another. Within traditional approach such service composition can be created but with limitations: since semantics of inputs/outputs is not introduced explicitly, the only way to find matching service is to follow data types of its inputs and/or know exactly what service is required. This approach works for simple composition problem but fails for problems required for the future Web services for e-commerce. As an example of composition, suppose there are two Web services, an on-line language translator and a dictionary service, where the first one translates text between several language pairs and the second returns the meaning of English words.

If a user needs a *FinnishDictionary* service, neither of these can satisfy the requirement. However, together they can (the input can be translated from Finnish to English, fed through the English Dictionary, and then translated back to Finnish). The dynamic composition of such services is difficult using just the WSDL descriptions, since each description would designate strings as input and output, rather than the necessary concept for combining them (that is, some of these input strings must be the name of

languages, others must be the strings representing user inputs and the translator's outputs). To provide the semantic concepts, we can use the ontologies provided by the Semantic Web.

Service composition can also be used in linking Web (and Semantic Web) concepts to services provided in other network-based environments [31]. One example is the sensor network environment, which includes two types of services; basic sensor services and sensor processing services. Each sensor is related to one Web service, which returns the sensor data as the output. Sensor processing services combine the data coming from different sensors in some way and produce a new output. These sensors have properties that describe their capabilities, such as sensitivity, range, etc., as well as some non-functional attributes, such as name, location, etc. These attributes, taken together tell whether the sensor's service is relevant for some specific task. An example task in this environment would involve retrieving data from several sensors and using relevant fusion services to process them via SOAP calls. As an example, the data from several acoustic and infrared sensors can be combined together and after applying filters and special functions, this data may be used to identify the objects in the environment. In this setting, we need to describe the services that are available for combining sensors and the attributes of the sensors that are relevant to those services. More importantly, the user needs a flexible mechanism for filtering sensor services and combining only those that can realistically be fused.

In DAML-S *ServiceGrounding* part of service description provides knowledge required to access service (where, what data, in what sequence communication goes) and *ServiceProfile* part provides references to the *meaning* what service is used for. Both these pieces of information are enough (as it supposed by Semantic Web vision) to be used by intelligent mediator (intelligent agent, mediation platform, transaction manager etc.) for using this service directly or as a part of compound service. The implementation of service composer [31] have shown how to use semantic descriptions to aid in the composition of Web services-- it directly combines the DAML-S semantic service descriptions with actual invocations of the WSDL descriptions allowing us to execute the composed services on the Web. The prototype system can compose the actual Web services deployed on the Internet as well as providing filtering capabilities where a large number of similar services may be available.

3.6 Web Services and Security

The Industry view on Web services security [30, 34] is mostly focused on concerns such as data integrity and confidentiality, authentication, and non

repudiation of messages. The way they are ensured is by adapting general information security technologies (such as cryptography or digital signature) to XML data. The advantages is that these technologies have been extensively tested and improved for many years and that they are still a lively topic in the research community. Some of the most significant specifications in XML services security are [29]:

- *XML Encryption* (XML Enc): It describes how to en-code an XML document or some parts of it, so that its confidentiality can be preserved. The document's en-coding procedure is usually included in the file, so that a peer possessing the required secrets can find the way to decrypt it.
- *XML Digital Signature* (XML DSig): Describes how to attach a digital signature to some XML Data. This will ensure data integrity and non repudiation. The goal is to ensure that the data has been issued by a precise peer.
- *Web services Security* (WSS): This standard is based on SOAP, XML Enc and XML DSig and describes a procedure to exchange XML data between Web services in a secure way.
- *Security Assertion Markup Language* (SAML): Specifies how to exchange (using XML) authentication and authorization information about users or entities. Two services can use SAML to share authentication data in order not to ask a client to log again when it switches from one service to the another (Single Sign On procedure).

Considering that Active XML is a language that is certified pure XML, all these recommendations (or specifications) can be used to ensure its security during the transfer and the storage of information.

3.6.1 Typing and Pattern Matching

We need to control the services calls in order to avoid the ones that could execute malicious actions (for example, buy a house, propose to sell my car on ebay at a tiny price, and so on). Active XML [38] relies on a typing and function pattern matching algorithm that will compare the structure of the answer returned by the service with a "allowed structure" provided by the client. If the structures can match (using rewriting), then answer's call can be invoked. Details on the algorithm are given in [28], but this algorithm is k-depth limited, and its decidability is not proved when ignoring the k-depth limit. CDuce [7] is an example of language with powerful pattern matching features. It can easily compare structures and corresponding data, and its strong capability to handle types (and subtyping) makes it a good candidate

for defining structures precisely. However, CDuce is mostly oriented towards XML transformation, so Active XML is definitely more simple and adapted for Web services.

3.6.2 Trust in Web Services

There are many ways to consider the notion of "trust" in services. The most adopted vision of trust in services is based upon progressive requests and disclosures of credentials between the peers (according to a policy), that will gradually establish the trust relationship [6, 36]. The privacy of the peers can be preserved, and credentials do not have to be shown without a need for it, thus preventing the user to display some information that (s)he could want to keep from a non authorized peer [22]. In [19], the analysis of trust is based upon the basic beliefs that will lead to the decision of granting trust or not. This approach is much more sociological and context dependent from the previous one, but it is closer from the way a human being behaves when trusting or not another person. The conditions required for the final decision of trust granting are divided into two major parts :

- *internal attribution,* representing the conditions that depend on the trusting agent's personality and skills,
- *external attribution,* that represents conditions that are completely independent from the agent (opportunity, interferences, ...).

Depending on these factors, a value representing the "trustfulness" is computed using a fuzzy algorithm. This value will allow the agent to take the decision on trusting the peer or not.

4. CONCLUSION

One of Semantic Web promises is to provide intelligent access to the distributed and heterogeneous information and enable mediation via software products between user needs and the available information sources. Web Services technology resides on the edge of limitation of the current web and desperately needs advanced semantic provision oriented approach. At present, the Web is mainly a collection of information and does provide efficient support in its processing. Also the promising Web services idea to allow services to be automatically accessed and executed has no yet facilities to efficiently discover web services by those who need them. All service descriptions are based on semi-formal natural language descriptions and put limits to find them easily. Bringing Web services to their full potential requires their combination with approach proposed by Semantic Web

technology. It will provide automation in service discovery, configuration, matching client's needs and composition. Today there are much less doubts both in research and development world, than few months ago, whether Semantic Web approach is feasible. The importance of Web services has been recognized and widely accepted by industry and academic research. However, the two worlds have proposed solutions that progress along different dimensions. Academic research has been mostly concerned with expressiveness of service descriptions, while industry has focused on modularization of service layers for usability in the short term.

References

1. http://www.w3.org/2002/ws/
2. http://www.uddi.org/
3. http://www.w3.org/2002/ws/desc/
4. http://devresource.hp.com/drc/specifications/wsmf/index.jsp
5. A. Arkin: Business Process Modeling Language (BPML), Working Draft 0.4, 2001. http://www.bpmi.org/.
6. K. E. S. T. Barlow, A. Hess. Trust negotiation in electronic markets. In *Proceedings of the Eighth Research Symposium on Emerging Electronic Markets (RSEEM 01),* 2001.
7. V. Benzaken, G. Castagna, and A. Frisch. Cduce: An xml-centric general-purpose language. In *Proceedings of the ACM International Conference on Functional Programming,* Uppsala, SWEDEN, 2003.
8. A. Bernstein and M. Klein. Discovering Services: Towards High Precision Service Retrieval. In CaiSE workshop on Web services, e-Business, and the Semantic Web: Foundations, Models, Architecture, Engineering and Applications. Toronto, Canada ,May 2002.
9. C. Bussler: B2B Protocol Standards and their Role in Semantic B2B Integration Engines, *IEEE Data Engineering,* 24(1), 2001.
10. Fabio Casati and Ming-Chien Shan. Dynamic and adaptive composition of e-services. Information Systems ,26(3): 143 –163,May 2001.
11. D. Chakraborty, F. Perich, S. Avancha, and A. Joshi. DReggie: Semantic Service Discovery for M-Commerce Applications. In Workshop on Reliable and Secure Applications in Mobile Environment, 20th Symposium on Reliable Distributed System, pages 28 –31, Oct.2001.
12. E. Christensen, F. Curbera, G. Meredith, S. Weerawarana: Web Services Description Language (WSDL) 1.1,15 March 2001. http://www.w3.org/TR/wsdl.
13. The DAML Services Coalition. DAML-S: Web service Description for the Semantic Web. In The First International Semantic Web Conference (ISWC),pages 348 –363,Jun.2002.

14. The DAML Services Coalition (alphabetically Anupriya Ankolenkar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne and Katia Sycara), "DAML-S: Web service Description for the Semantic Web", *The First International Semantic Web Conference (ISWC),* Sardinia (Italy), June, 2002.
15. DAML Services Coalition (alphabetically A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, H. Zeng), "DAML-S: Semantic Markup for Web services", in *Proceedings of the International Semantic Web Working Symposium (SWWS),* July 30-August 1, 2001.
16. Data Engineering Bulletin: Special Issue on Infrastructure for Advanced E-Services.24(1), IEEE Computer Society, 2001.
17. K. Decker, K. Sycara, and M. Williamson. Middle-agents for the Internet. In IJCAI'97, 1997.
18. Y.Ding, D. Fensel and B. Omelayenko M.C.A. Klein. The semantic web: yet another hip? DKE ,6(2-3):205 –227,2002.
19. Rino Falcone, Giovanni Pezzulo, Cristiano Castelfranchi. A fuzzy approach to a belief-based trust computation. In Trust, Reputation, and Security: Theories and Practice, AAMAS 2002 International Workshop. LNCS 2631 Springer 2003. Pages 73-86.
20. D. Fensel and C. Bussler. The Web service Modeling Framework WSMF. http://www.cs.vu.nl/diete/wese/publications.html.
21. D. Fensel, C. Bussler, and A. Maedche. Semantic Web Enabled Web services. In International Semantic Web Conference,Sardinia,Italy ,pages 1 –2,Jun.2002.
22. K. E. S. J. Holt, R. Bradshaw and H. Orman. Hidden credentials. In *2nd ACM Workshop on Privacy in the Electronic Society (WPES'03),* Washington DC, USA, October 2003.
23. F. Leymann: Web Service Flow Language (WSFL 1.0), May 2001. http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.
24. D. Martin, A. Cheyer, and D. Moran. The Open Agent Architecture: A Framework for Building Distributed Software Systems. Applied Artificial Intelligence, 13(1-2):92-128, 1999.
25. S. McIlraith,T.C. Son,and H. Zeng.Semantic Web services. IEEE Intelligent Systems. Special Issue on the Semantic Web ,16(2):46 –53,March/April 2001.
26. S. McIlraith, T. C. Son, and H. Zeng. Mobilizing the Web with DAML-Enabled Web service. In Proceedings of the Second International Workshop Semantic Web (SemWeb'2001), 2001.
27. S. McIlraith, T. C. Son, and H. Zeng. Semantic Web service. IEEE Intelligent Systems, 16(2):46-53, 2001.

28. T. Milo, S. Abiteboul, B. Amann, O. Benjelloun, and F. D. Ngoc. Exchanging intensional xml data. In *Proc. of ACM SIGMOD 2003*, June 2003.
29. M. Naedele. Standards for xml and Web services security. *IEEE Computer,* pages 96–98, April 2003.
30. O. for the Advancement of Structured Information Standards (OASIS). http://www.oasis-open.org/
31. E. Sirin, J. Hendler, B. Parsia, Semi-Automatic Composition of Web services Using Semantic Descriptions, In proceedings of "Web services: Modeling, Architecture and Infrastructure" workshop in conjunction with *ICEIS2003*, 2003.
32. S. Thatte: XLANG: Web Services for Business Process Design, Microsoft Corporation, 2001. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.
33. The VLDB Journal: Special Issue on E-Services.10(1),Springer-Verlag Berlin Heidelberg, 2001.
34. W. W. W. C. (W3C). http://www.w3.org/
35. D. Waldt and R. Drummond: EBXML: The Global Standard for Electronic Business, http://www.ebxml.org/presentations/global_standard.htm.
36. M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, and L. Yu. Negotiating trust on the web. *IEEE Internet Computing,* 6(6):30–37, November/December 2002.
37. H.-C. Wong and K. Sycara. A Taxonomy of Middle-agents for the Internet. In ICMAS'2000, 2000.
38. http://www-rocq.inria.fr/gemo/Gemo/Projects/axml/