Chapter 3

WEB SERVICES MODELING ONTOLOGY

Michal Zaremba, Mick Kerrigan, Adrian Mocan and Matt Moran
Digital Enterprise Research Institute (DERI), Ireland, National University of Ireland, Galway, Ireland – <firstname.lastname>@deri.org

1. INTRODUCTION

Existing technologies enabling the integration of enterprise systems, use very few of the capabilities of modern computers. For example, the activity of finding services, which should deliver expected enterprise functionality, has to be driven by humans. The process of assembling pieces of functionality into complex business processes also involves human interaction. Finally translating between different message formats, which are exchanged between enterprises systems, cannot be done automatically. Computers and computer networks are used mainly for storing and sending information, but the interpretation of this information is done by software engineers and domain experts. It is currently a manager's responsibility, not a computer's, to find services and to make decisions about their suitability. A software programmer has the responsibility of assembling these services into a complex process block. Finally a domain expert is responsible for defining mappings between the message formats sent by one system and the formats expected by the second.

Web Services have promised to solve some of these problems, but because of their syntactical nature[1], they have failed in most of these cases

[1] existing specifications cannot formally specify what services provide and how they should be used, so these descriptions can not be automatically processed by machines

and humans must still be kept in the loop. According to Tidwell (Tidwell), Web Services are self-contained, self-describing, modular applications that can be published, located, and invoked over the Web. This definition, like any of many such definitions describing Web Services, makes no comment on *who* should publish, locate and invoke them. The hidden answer is that these are the humans, who are involved in almost every step of Web Services usage process. The unquestionable success of existing Web Service specifications lies in their ability to separate service interface from its implementation, based on standards which were accepted by all the major players of the IT industry. However these standards lack an appropriate semantic framework allowing for automation of many of the processes which are currently handled manually.

The application of semantics to Web Services can be used to remove humans from the integration jigsaw and substitute them with machines. There are many problems which Semantic Web Services (SWS) could be used to resolve. SWS will put in place an automated process for machine driven dynamic discovery, mediation and invocation. Work that will be presented in this chapter does not question the enormous success of Web Services, but rather this chapter recognizes the need to extend the existing Web Service standards with semantics to enable their full automation. The purpose of this chapter is to introduce and provide an overview of the Web Services Modeling Ontology (WSMO), a fully-fledged framework for SWS, showing a reader practical examples aimed at explaining the application of WSMO concepts to a real world scenario. First we present a very simply use case from the e-banking domain, which is used in an overview of WSMO concepts. One of the major intentions of this chapter is to present the technological framework for SWS development around WSMO. We discuss and present some of the key technologies related to the conceptual framework of WSMO, especially the Web Services Modeling Execution Environment (WSMX), which is its reference implementation.

The chapter is structured as follows: Section 2 presents a motivational use case for Semantic Web Services, Section 3 introduces WSMO and its top level concepts, Section 4 discusses selected technologies for WSMO, Section 5 compares competitive approaches, and Section 6 concludes the chapter.

2. CASE STUDY – APPLICATION FOR SEMANTIC WEB SERVICES

In this section we introduce an application from the banking industry as an example of how Semantic Web Services can be used to provide an

improved customer service. Our aim is to illustrate the benefits offered by Semantic Web Services in a familiar scenario. The application, for this use case, allows the comparison of the mortgage interest rates being offered by banks online. The emergence of internet banking has greatly increased the competitiveness of the market for services such as mortgage lending. Banks within the European Union (EU) can provide online banking facilities to any citizen of the EU. Many offer online tools allowing prospective bank customers to see, at a glance, current mortgage rates and the amount they could borrow. These tools are often constrained by being limited to the mortgage products offered by just one bank.

Third party websites are increasingly available that aggregate information from multiple banks allowing the comparison of the various mortgage products on offer. Different techniques can be used by these websites to retrieve data from the individual banks. In the next paragraphs, we describe three of the most common.

Manual population involves one or more humans researching the products offered by various banks based on telephone calls and investigation of marketing material – both print and internet based. This works best when interest rates are stable and the number of banks in the marketplace remains static. The reality is that neither of these conditions is likely to be true. Interest rates change and new online banks appear regularly.

Screen scraping is where a software application reads the HTML content of a Web page and extracts the required data. For example, the scraper may read the Web page used by a bank to publish details of the mortgage rates the bank is offering. The advantage is that, when it works, the information is always up-to-date. However, the technique tightly links the scraping application with the structure of the HTML page advertising the mortgage rates. These pages change frequently and each change requires the scraping application to be redesigned.

Web Services are where the banks themselves provide an online application using standard Web technology that allows their interest rates to be requested on demand. The advantage is that the interface to this application usually remains quite stable – requiring less ongoing maintenance at the client application side. Another advantage is that Web service technology is increasingly standards based. A drawback with Web Services is that the technology, by itself, does not help service requesters understand the meaning of the data or messages that they should exchange with the service. This must be determined by a human before the service is invoked for the first time.

Although Web Services provide the best solution of the three approaches described above, human intervention is still required to *find* services offered by banks online, *interpret* the data and the messages that the various banks'

services can support, and know how to *invoke* those services. Semantic Web Services address these problems by providing machine-understandable descriptions of what the service can do (*capability*) and how to communicate with it (*interface*). The use of ontologies as the basis for the descriptions guarantees that they are unambiguous and machine-understandable. In our banking example, an application would automatically discover new Semantic Web Services offering mortgage rate information as they became available. When such a service is located, the description of the interface would be examined automatically to determine how the application and service should communicate. Once data mismatches have been resolved, the application retrieves the information about mortgages as required. The whole operation is transparent to the customer and is always up-to-date.

3. THE WEB SERVICES MODELING ONTOLOGY

The Web Services Modelling Ontology (WSMO) initiative provides a complete framework enhancing syntactic description of Web Services with semantic metadata. The WSMO project[2] is an ongoing research and development initiative aiming to define a complete framework for SWS and consisting of three activities:

- WSMO, which provides formal specification of concepts for Semantic Web Services,
- WSML (Web Services Modelling Language), which defines the language for representing WSMO concepts;
- WSMX (Web Services Execution Environment), which defines and provides reference implementation allowing the execution of SWS

As depicted in Figure 3-1, there are four top level WSMO concepts: Ontologies, Goals, Web Services and Mediators.

In a nutshell, *Ontologies* provide formal terminologies which interweave human and machine understanding; *Goals* formally specify objectives, which clients would like to achieve by using Web Services; *Web Services* are the formal descriptions required to enable the automatic processing of Web Services, and finally *Mediators* enable handling any possible heterogeneity problems. More detailed explanation with the examples can be found in the following sections.

[2] http://www.wsmo.org

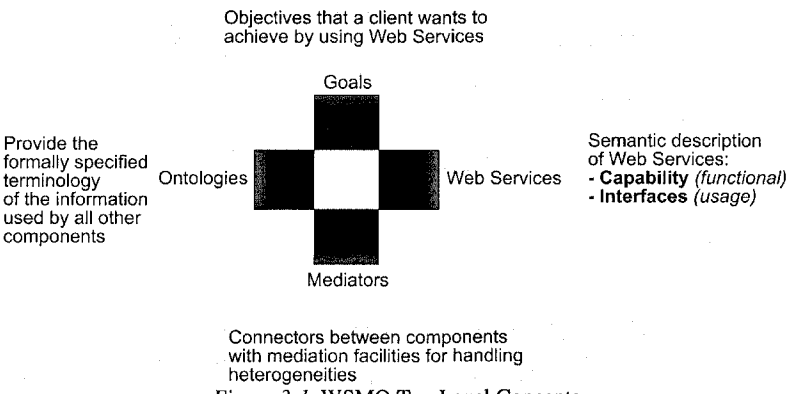

Figure 3-1. WSMO Top Level Concepts

3.1 Ontologies

The Web has revolutionised the publishing and sharing of information. The only obstacle to gaining access to this information is a communication link and simple software that can render and display HTML Web pages. The openness of the Web means the volume of published information is growing exponentially resulting in what is commonly termed 'information overload'. Finding specific data in this sea of information becomes increasingly difficult. Already, today's most valuable Web tools are search engines – the most popular of which accept keywords as input and get the results back fast. Each search engine uses its own proprietary, and usually secret, algorithm when determining what results to give back and in what order the results should be displayed.

It can often be difficult to extract relevant information from the retrieved search results. Sometimes, relevance can only be determined by sifting through the result, one by one. Although not difficult for a small number of search results this becomes impractical as the number of links increases. Ontologies provide a means to greatly help in querying for knowledge on the Web by enriching information with descriptions of its meaning. Significantly, these rich descriptions can be interpreted by computer systems allowing them to provide intelligently interpret the results of Web queries.

Ontology is a philosophical term meaning the study of things that actually exist. In the context of computer science, ontologies define formal shared descriptions of the things that exist in particular domains of interest as well as the relationships that exist between those things. Gruber (Gruber, 1993) defines an ontology as a formal specification of a shared conceptualization – formal because the descriptions it contains must have a

precise provable meaning, and shared as an ontology is only valid if its definitions are accepted by a community of users.

Ontologies by themselves are static sources of knowledge but become very powerful instruments when combined with logic and reasoning. Knowledge can be represented formally, using logical languages, as facts that can be interpreted and reasoned about by machines. Reasoning allows implicit knowledge to be inferred from existing knowledge and form an extremely powerful tool when combined with ontologies. In the case of a search engine returning results based on logical reasoning, the engine could also provide the user with the logical proof of where the results came from, if this was necessary.

In WSMO, the basic building blocks of an ontology are *concepts*, *relations*, *functions*, *instances*, and *axioms*. Concepts are descriptions of things that exist in the domain of the ontology. For example, a banking ontology would probably include concept definitions for bank, account, customer, deposit, loan, and so on. Here is an example of a simplified WSMO concept definition for a bank account:

```
concept bank_account
  accountNumber   ofType validAccountNumber
  owner           ofType customer
  balance         ofType currency
  overdraftLimit  ofType currency
```

Concepts may contain attributes with names and types. Relations describe interdependencies between multiple concepts. The relation married-to describes an interdependency between a man and a woman. Functions are special relations that result in a single typed value. For example, a function might be defined to return the amount of a monthly loan repayment based on the amount of the loan, its duration and the interest rate.

Where ontologies describe the conceptual model for a particular domain, instances are the actual facts described using these concepts. For example the details of each individual customer would be used to populate instances of the customer concept. Axioms are the logical expressions used in WSMO for various purposes including the definition of constraints of data, the definition of relations.

3.2 Goals

A service requester uses Goals to represent the type of service that they are seeking by specifying what capability they would like that service to offer and what public interface they would like it to provide. Where Web

Service descriptions are intended to provide detailed descriptions of the mechanics of how a service provides its capability and behaviour, Goal descriptions describe what capability and behaviour the requester would like to find. Importantly, the Goal is described in terms of ontologies used by the requester. The ability to model both Goals and Web Services provide a distinct conceptual separation between the points of view of service requesters and providers. This allows more flexibility in how service requesters and providers are brought together than is possible with current Web Service technology.

For example, the following steps would be needed to search for a Web Service offering mortgage interest rate comparisons. First, a suitable service must be located in a UDDI repository. The requester might try looking for services with the name 'mortgage'. If no services were located, they might try a search on 'home loan' or 'banking services'. If a service is located, its textual description can be checked to see if it fits the requirements. However, as service descriptions provided in UDDI are informal, the requester must assume that their understanding is the same as that intended by the service provider. If the requester is satisfied with the Web Service, the associated WSDL document provides the syntactic description of what messages the service accepts and what transport protocol to use when interacting with the service. The input and output messages are described in XML, in terms of an XML schema. To make an invocation of the Web Service, the requester may have to adjust their data to fit the service description. This example would require the interaction between service requester and service provider to be tightly coupled together. If the requester wants to use another banking service later, they will have to repeat the entire process of finding and binding to a suitable service again.

Describing both Goals and Web Services separately using the Web Service Modelling Ontology shifts the responsibility of matching service requests to service descriptions from the requester to Semantic Execution Environments, such as WSMX, which can interpret the requester's Goal and carry out whatever discovery, mediation and invocation mechanisms are required to connect the service requester to the service provider at run-time. This is distinct from the design-time binding required in the WSDL example described in the last paragraph. WSMO Goals comprise of the following sub concepts: *Capability*, *Interface*, *Imported Ontologies* and *Used Mediators*.

3.3 Web Services

Informally, in terms of current specification, the term "Web Service" is usually understood as a composition of three major elements: (1) interface descriptions captured by WSDL documents, (2) the communication protocol,

SOAP using XML to exchange messages and (3) UDDI repositories allowing potential users to find services that are offered by providers. In WSMO the *Web Services* concept is not directly related to WSDL, SOAP and UDDI. In the WSMO context, a Web Service is a formal description required to enable the automatic processing of Web Services. With WSDL, SOAP and UDDI anybody can use a Web Service regardless of the programming language, which has been used to implement the functionality of the service. Similarly, WSMO focuses on the external interface of the Web Service, while its internal implementation remains out of the scope of WSMO. The Web Service description in WSMO provides rich descriptions enabling not only humans, but also software entities "understand" the capabilities and interfaces of the service. Such an unambiguous description of a Web Service with well-defined semantics can be processed and interpreted by software agents without human intervention. This enables the automation of the tasks involved in the Web Service usage process such as discovery, selection, mediation, composition, execution and monitoring. Having appropriate information, software agents can provide automatic matching between Goals received from bank clients and Web Services offered by banks. While the interest rates from a particular bank would not be directly included in a Web Service definition, the capabilities of the service would be defined in a way, that the software agent can "draw" conclusions about the service and its suitability for obtaining information about interest rates.

All the information, stored in the WSMO Web Service description, contains certain aspects of the functionality and behavior of the actual service. The functional aspects are described by the *Capability* of the service. The behavioral aspects are addressed by the *Interface* of the service, which contains both the *Choreography*, which expresses the interface for consumption and the *Orchestration*, which defines how functionality can be achieved by aggregating other Web Services.

The *Capability* describes the functionality of a Web Services from the black box perspective allowing for automated Web Services discovery. This functionality is captured by conditions that need to hold before the Web Service can be executed and by the results that have been achieved after its execution. Web Service Capabilities are defined by four notions:

- *Preconditions* – conditions on the information space that have to hold before execution; For the e-banking Web Service these can be inputs, which have to be provided by a client e.g. in the following example these could be two inputs: (1) an amount of money, which client would like to borrow and (2) repayment period for a requested mortgage.

```
capability aibBankWSCapability
   precondition
      definedBy
         ?interestRateRequest[
             borrowedAmount hasValue ?amount,
             repaymentPeriod hasValue ?period
         ] memberOf aib#interestRateRequest.
```

- *Assumptions* – conditions on the world that have to hold before execution e.g. the fact that a client is coming from a member country of European Union would be an assumption,
- *Postconditions* – conditions on the information space after execution. There are no postconditions for the simple example of e-banking use case. But if after checking interest rates, the client would decide to go ahead and request a mortgage from one particular bank, as a result of Web Service execution (its postconditions) the mortgage money would become available to the client.
- *Effects* – conditions on the world that hold after service execution. Again there are no effects for a simple example of requesting interest rates. But in a complex scenario, as a result of Web Service execution, money would be transferred to client account.

WSMO differentiates two parts of the Web Service *Interface* that are concerned with the interaction behavior of the Web Service. WSMO *Choreography* specifies how the service achieves its capability by means of interactions with its user i.e. the communication with the user of the service. WSMO *Orchestration* specifies how the service achieves its capability by making use of other services - i.e. the coordination of other services. We provide some more details on choreography and orchestration in upcoming sections. Anyway WSMO Choreography and Orchestration are complicated topics and the reader is advised to consult the WSMO specifications for more information and the WSMO deliverables for practical examples of choreography and orchestration interfaces.

3.4 Mediators

For decades, the attempt to make machines or applications work together, interoperate with each other, exchange data and share functionality has been a great challenge both from the technological and efficiency point of view. The Web has pushed these problems to the extreme by offering an environment which adds to the practically infinite quantity of information available. That is, business entities willing to interact bring with them

completely independent applications with various ways of representing and structuring data. This drives the need for mediators[3], third-party systems able to deal with the potential mismatches that may appear both on the data and behaviour level between the interacting parties.

The techniques used in developing mediators have to be dynamic and scalable - hard-coded and one-scenario solutions are not feasible anymore. Mediators should be flexible systems and easy to extend, assuring loose coupling between various business entities.

WSMO provides the means of semantically describing mediator systems by introducing four classes of mediators able to cope with the heterogeneity problems that might occur between ontologies, web services and goals: *ontology-to-ontology mediators (ooMediators), goal-to-goal mediators (ggMediators), web services-to-goal mediators (wgMediators)* and *web service-to-web service mediators (wwMediators).*

ooMediators describe the class of mediators able to solve the heterogeneity problems between ontologies. Indeed, the ontologies could represent very helpful tools in classifying and describing the huge amount of data available on the Web, but they could also be developed in isolation, by different parties. As a consequence, one can find ontologies describing the same domain in different terms and, without mediators, applications using these kinds of ontologies would not be able to exchange data. Also the reuse of external ontologies might not be possible if the heterogeneity problems are solved in advance. For example, in our banking scenario, the bank can use a specific ontology for modelling the details related to mortgages and interest rates. If the application that aggregates mortgage information from different sources uses a different ontology to represent its data, an ooMediator can be used to solve the potential mismatches and conflicts. Such a mediator points to a concrete mediation solution (as the one described in Section 4.2) able to actually solve the heterogeneity problems between the specified source and target ontologies (i.e. the ontology used by the bank and the ontology used by the application, respectively).

ggMediators are used for coping with the differences and for exploiting the similarities that may exist between different goals. Constructing goal ontologies, or explicitly expressing the differences/similarities between different goals, might facilitate the entire process of discovering a Web service, or even the process of invoking a particular goal. Any ggMediator may use the services of ooMediators, in case the goals, between which it

[3] One of the first definitions of mediator systems appears in (Wiederhold, 1992) in 1992: "A mediator is a software module that exploits encoded knowledge about some sets or subsets of data to create information for a higher layer of applications."

mediates, are expressed using different ontologies. If a client has as goal to find the mortgage interest rate and there is an already defined goal that asks for mortgage interest rate and the eligibility of the inquiring client for this mortgage, a *ggMediator* can be defined to link these two goals. The *ggMediator* assures that any web service that can satisfy the second goal can satisfy the first one as well.

wgMediators are the class of mediators that address the heterogeneity problems between a goal and a Web service at two different levels: *functionality* (can the Web service completely satisfy the goal?) and *communication* (how can the two partners communicate?). The first level can be addressed in two steps:

- find a goal that is completely satisfied by the Web service
- use the services of a ggMediator that defines the relation between the initial goal and the newly discovered one.

The communication problem addresses the interface heterogeneity – each partner in a communication defines its own way of communicating (communication pattern) with the other one. In case the two patterns do not exactly match (for example, at some point in time one of them may expect something that the other one intends to send later), a communication mediator, also known as process mediator will have to accommodate these mismatches. In the online banking scenario a *wgMediator* can link the goal that asks for mortgage interest rate directly with the web service offering both the mortgage rates and the eligibility details of the client.

wwMediators are the most complex class of mediators in WSMO, addressing the heterogeneity problems between different Web services. These problems may occur when a Web service is invoking one or many other Web services in order to achieve certain functionality, and implies three levels of mediation: *functionality*, *communication* and *cooperation*. The first level can be address in the similar way as for the wgMediators: find goals that can be completely satisfied by the given Web services, and use ggMediators for expressing functional relations; the second level can be address by using wgMediators; the third level, which represents the most complex one, deals with how multiple Web services can be combined (that is, in what order should the Web services be combined). Also known as a problem of composing Web services, this particular level is investigated by different well-known researchers (Milanovic and Malek, 2004), but no truly automatic solutions are discovered so far. In our example, if the web service described above, achieves its functionality by using two other web services, one for retrieving the mortgage interest rates and the other one to check the eligibility of a given client for a particular mortgage type, it is the task of a

wwMediator to take care of how these two web services have to be combined.

4. SELECTED TECHNOLOGIES FOR WSMO

Creating ontologies and semantic descriptions for Web Services is only useful if these descriptions can ultimately be applied. Infrastructure is vital for a technology to be applied. Web servers and web browsers are the infrastructure that has lead to the success of HTML on the web. An execution environment for Semantic Web Services is the infrastructure required to enable automated discover, mediation, selection and invocation of these services. This section presents the Web Service Execution Environment (WSMX), by introducing the technologies used and solutions provided by it. WSMX is an execution environment for finding and using Semantic Web Services that are described using WSMO. WSMX is a reference implementation of WSMO and takes the full conceptual model of WSMO into consideration. Considering current Web Service technologies there is a large amount of human effort required in the process of finding and using Web Services. Firstly the user must browse a repository of Web Services to find a service that meets their requirements. Once the Web Service has been found the user needs to understand the interface of the service, the inputs it requires and outputs it provides. Finally the user would write some code that can interact with the Web Service in order to use it. The aim of WSMX is to automate as much of this process as is possible. The user provides WSMX with a WSMO Goal that formally describes what they would like to achieve. WSMX then uses the Discovery component to find Web Services, which have semantic descriptions registered with WSMX that can fulfill this Goal. During the discovery process the users Goal and the Web Services description may use different ontologies. If this occurs Data Mediation is needed to resolve heterogeneity issues. Data Mediation in WSMX is a semi-automatic process that requires a domain expert to create mappings between two ontologies that have an overlap in the domain that they describe. Once these mappings have been registered with WSMX the runtime data Mediation component can perform automatic mediation between the two ontologies. Once this mediation has occurred and a given service has been chosen that can fulfill the users Goal WSMX can begin the process of invoking the service. Every Semantic Web Service has a specific choreography that describes they way in which the user should interact with it. This choreography describes semantically the control and data flow of messages the Web Service can exchange. In cases where the choreography of the user and the choreography of the Web Service do not match process

mediation is required. The Process Mediation component in WSMX is responsible for resolving mismatches between the Choreographies (often referred to as public processes) of the user and Web Service. Running to the case study in section 2, an example of the sort of mismatches that the Process Mediator is likely to encounter is where the user wants to login to an online banking system using a Web Service, in this case the user may want to send the username and password together in one message where as the Web Service expects two messages, the first containing the username and the second containing the password. In this case the Process Mediator needs to take the message sent by the user and break it up into two messages, which are then sent in the correct order to the Web Service. At this point it is now possible to interact with the Web Service and the users Goal of logging into the system can be achieved.

More information on discovery can be found in section 4.1, mediation is described in section 4.2, choreographies of Web Services are presented in section 4.3 and a selection of front-end tools for use with WSMO and WSMX are shown in section 4.4.

4.1 Discovery

As already mentioned, with current Web Service technology the process of finding a Web Service is a manual one. The user must search by hand through a Web Service repository, which usually provides free-text descriptions of what the service does. This is a time consuming process and can be seen as a barrier to quick and efficient integration between potential business partners. With WSMX it is possible to perform automated discovery of Web Services on a semantic description of the service. When the user provides WSMX with a Goal that semantically describes what they want to achieve, WSMX can perform two types of discovery to find matching services. These two types of discovery will both return an ordered list of Web Services, ordered by how well they match the users Goal and are described in the following paragraphs.

Keyword Based Discovery. The keyword based discovery process involves matching keywords present in the user's Goal with keywords present in the Web Services semantic description. While this particular approach does not have well defined semantics and could suffer from natural language ambiguity issues it is useful to filter a large amount of Web Services down to a smaller more manageable set on which more advanced techniques can be used. There are a number of places that keywords can be found in the Web Service description, in the value sections of non-functional properties, for example title, subject and description, in the identifiers of the

concepts used in the Web Service description and in the logical expressions defining the capability of the Web Service.

Semantic Based Discovery. Semantic based discovery is a more formal mechanism for determining if a given Web Service can fulfill a users Goal. As described in section 3.2 a Web Service description is made up of a formal description of the capability of the Web Service and the interface of the Web Service. Performing discovery based on a Web Service involves matching the capability of the Web Service with the requested capability in the users Goal, by comparing the pre-conditions, post-conditions, assumptions and effects of both. When performing this discovery the relationship between the Goal and Web Service can be a number of different types:

- *Exact match*: where the Web Service can provide exactly what the Goal requires.
- *Subsumption match*: where the Web Service can provide part of what the Goal requires.
- *Plug-in match*: where the Web Service can provide what the Goal requires and provides other functionality also.
- *Intersection match*: where the Web Service can provide part of what the Goal requires and provides other functionality also.
- *Non-Match*: where the Web Service does not provide what the Goal requires.

Different levels of semantics can be provided in this matching, the richer the semantics the more time consuming the operation.

4.2 Data Mediation

One of the most important principles of WSMO and of the Web in general implies that resources are developed in isolation by various parties and than made available over the internet. In this context, the semantics meant to disambiguate and to describe data, Web Services or Goals is expressed in different terms. That is, different ontologies are developed to model the same domains of activity, this fact adding an additional level of complexity to all the operations related to Semantic Web Services.

Data mediation has the role of coping with the heterogeneity problems that may appear at the data level, for example between the requester and a provider of a Web Service. These problems appear when the application existing on one side uses a data format or representation unknown to the other party. In the context of WSMO and WSMX, we assume that both parties have described their data in terms of ontologies and the solution we propose tries to resolve the potential mismatches at the semantic level and to

apply the findings from this level to the actual data that is exchanged. The ontology mismatches are solved during design-time by an *Ontology Mapping Tool* and the results are applied during run-time by a *Runtime Mediation Component*. We describe each of these modules in more detail in the next subsections.

Ontology Mapping Tool. At this step of the mediation process, the mismatches existing between the ontologies used to describe the exchanged data have to be identified and captured in what it is called an *alignment* between these ontologies. In WSMX, the alignment consists of set of *mappings* that logically express the semantic relation between terms from one ontology and terms from the other ontology. As in most of the cases, the initial designers of one or both ontologies fails to completely capture the semantic of the domain in their model, the tool cannot determine the alignment in completely automatic and accurate manner[4]. As a consequence, the WSMX Ontology Mapping Tool is a design-time, graphical tool that provides support for semi-automatic mappings creation. The human user (i.e. the domain expert) is guided through the whole mapping process and they are asked to validate the suggestions offered by the tool.

The main advantage of this semi-automatic approach is that the tool transforms the mapping process from a laborious and error-prone task in to simple choices and validation using a graphical user interface. In particular, the mappings are expressed as logical rules and their manual editing would require domain experts with strong background in logics. With this approach the complexity of the mappings and the burdensome of logics are hidden under the system's hood: the domain expert places his inputs only through the graphical interfaces, while the underlying system automatically generates the corresponding mapping rules.

In the banking domain, the Ontology Mapping Tool can be used to create mappings between two ontologies that both model the mortgage concept. By such mappings it is stated that there is a semantic relationship between the two definitions of the concept; the mappings also describe what this semantic relationship means.

Runtime Mediation Component. The mappings created by using the Ontology Mapping Tool are saved in a persistent storage and made available to the Runtime Mediation Component for use during run-time. At this

[4] There are tools that automatically generate an alignment between two given ontologies, but they cannot guarantee the correctness and the accuracy of these alignments. As WSMX is a business oriented framework we consider these requirements a must.

second stage, the mappings are used for a specific mediation scenario, i.e. *instance transformation*[5]. This scenario requires that incoming data described in terms of one given ontology (i.e. source ontology) has to be transformed in order to comply with the definitions from another given ontology (i.e. target ontology). In other words, the source data represented as source ontology instances has to be transformed and expressed as target ontology instances.

In order to perform these transformations, the mapping rules generated during design-time are evaluated in a reasoner and applied on the source instances. The result consists of a set of target ontology instances, modelling exactly the same information as the source instances but conforming to the specifications in the target ontology.

It is worth mentioning that the run-time mediation process is a completely automatic one, no human intervention being necessary as long as the required mappings are available.

4.3 Choreography

An important part of Web Services interface is the *choreography*[6]. The choreography of a Web Service describes the way one can interact with the service in order to consume its functionality. In other words, the choreography defines the requester expected behaviour during the Web Service invocation. The requestors can also define their own choreographies as part of the goal they want to be accomplished – that is, the requested choreography, the behaviour they are able to comply with when invoking a Web Service.

WSMO choreography is expressed in terms of Abstract State Machine also formerly known as Evolving Algebra. This mechanism is used to describe systems in a precise manner using semantically well founded mathematical notations.

There are two main components in WSMX used to manage and to maintain the interaction between a requester and a provider of a Web Service

[5] Another well known mediation scenario (not required in WSMX) is *instance transformation*. By using a mediator that supports this scenario is possible to retrieve data expressed in terms of various ontologies by posting queries in terms of only one particular ontology.

[6] The other part of a WSMO Web Service's interface, not discussed in here, is the *Orchestration*. It describes the way that the web service functionality can be achieved by composing several other web services. It is very related as form of representation with choreography and it is strongly influenced the choreographies of the orchestrated web services.

in terms of their choreographies: the *Choreography Engine* and the *Process Mediator.*

Choreography Engine. The Choreography Engine has the role of managing all the operations regarding the choreographies of the two parties involved in a conversation: This implies:

- Identifying and loading the two choreographies;
- Creating a copy for each of the choreographies (i.e. choreography instances). These copies are used further as long as the communication session is maintained.
- Updating the choreography instances in respect with the incoming messages.

These messages might be sent by the communication partner provoking an update in the receiver's choreography instance. A response message could be generated and it will create in its turn an update in the target choreography instance.

Process Mediator. Choreography describes the behaviour of the service from the provider point of view, implying that all the requesters of that particular service should comply with that particular choreography. That is, the choreography of a requester should be compatible (but not necessarily equivalent) with the choreography of the service provider in order to enable communication. As one of the WSMO principles states that all entities involved in communication are equal partners, we should assume that none of them is willing to adjust its own choreography to match the other partner's choreography.

As a consequence there is a need for a Process Mediator, a component able to solve the communication mismatches that can appear during the conversation. It takes as inputs each party's choreography and analyses each incoming message to check if it is expected by the receiver choreography. If it is, it means that the message can be forwarded to the receiver; if it is not expected, the message can be transformed (as dictated by Data Mediator for example) or postponed for later stages of the conversation. The Process Mediator interacts directly with the Choreography Engine, acting as a middle layer between the choreographies of the requester and the provider. Such a process mediator (as well as the Data Mediator) is one of the technologies that can be used in realizing the types of mediators described by WSMO (i.e. ggMediators, wgMediators and wwMediators).

If we consider for example the service that checks the eligibility of an inquiring client for a particular type of mortgage, its choreography can specify that it expects first a message containing the incoming per year and

than a message containing the type of mortgage the client is interested in. Unfortunately, the client application is designed to send first the requested type of mortgage, to expect for a confirmation and only then to send annual income of the client. It is the role of the process mediator to inverse the order of messages and to generate a dummy acknowledgement to enable the interaction.

4.4 Front-end Tools

As with any emergent technology it is important that end-users can actually use the technology. Providing high quality front-end tools is a good way to get a technology adopted. To this end a number of software projects have emerged attempting to create tools for modeling and using WSMO and Semantic Web Services. From the case study in section 2, banks providing Semantic Web Services for obtaining mortgage quotes would use these tools to create ontologies that model the banking domain and use these ontologies to semantically describe the Web Services capabilities and interfaces, while users would use these tools to describe their requirements in the form of a Goal. Each of these tools is available for download; links are available in section 9.

Web Services Modeling Toolkit (WSMT)

The Web Services Modeling Toolkit (WSMT) is a framework for the rapid creation and deployment of homogeneous tools for Semantic Web Services. A homogeneous toolkit improves the users experience while using the toolkit, as the tools have a common look and feel. Usability is also improved as the user does not need to relearn how to use the application when switching between tools. The WSMT was designed to be the front-end of the WSMX system and provides a number of tools to users:

WSML Editor. The WSML Editor is used to create and manage WSML documents. It can be used to edit WSMO Ontologies, Mediators, Web Services and Goals. The first versions of the WSML Editor focused on the creation of semantic descriptions in WSMO and reading and writing these semantic descriptions to and from the local machine using the WSML syntax. Subsequent versions have looked at mechanisms for visualizing ontologies using directed graphs. These ontology visualizations make it easier for the domain expert to understand the relationships between entities in the WSML document.

WSMX Data Mediation Mapping Tool. As described in section 4.2, data mediation in WSMX is a semi automatic process. Mappings are

required where mediation between two ontologies is required. The WSMX Data Mediation Mapping Tool is used to create these mappings between two ontologies. These mappings can then be used by WSMX to transform instances of the source ontology into instances of the target ontology, thus resolving data mismatches between partners that use different ontologies to describe their web services.

WSMX Invoker. The WSMT contains a web service invocation component that can be used to send messages to and receive messages from web services. Messages can be received from the web services both synchronously (immediately following a sent message) and asynchronously (where the service calls the user back later with a response). The WSMX Invoker tool makes these components within the WSMT available to the end-user. The tool allows to user to send messages to a given service within the WSMX architecture, view the messages sent to services in the past and view responses received from these services.

Distributed Ontology Management Environment (DOME)

The DOME project aims to produce a suite of tools for the efficient and effective management of ontologies. DOME is implemented as a collection of Eclipse plug-ins that allows users to edit and manage WSMO Ontologies. These plugins include:

Editing and Browsing. The Editing and Browsing tool provides a tree structure for representing the concept and relation hierarchies within an ontology. Users can add new concepts and relations into these hierarchies as well as adding attributes and parameters to those already present. The tool also provides a real-time mechanism for switching between the graphical tree structure and the underlying file format. This allows users to make changes in one and see those changes reflected in the other.

Versioning and Evolution. The Versioning and Evolution tool allows users to mark the versions of a given ontologies. This is necessary as when an ontology reaches a stable position and individuals start using it, it becomes necessary to track which versions of a given ontology are being used by different individuals. Versions of a given ontology are tracked using the URI that identifies them; this URI is incrementally changed as the version of the ontology changes. This allows multiple versions of the same ontology to exist within the same knowledge base.

Mapping & Merging. The Mapping & Merging tool deals with cases where there are two ontologies that have an overlap in the domain that they describe. This tool is used to create mappings between these two ontologies so that execution environments, for example WSMX, can perform instance transformation, query rewriting and ontology merging. The mappings are created by opening two copies of the Editing and Browsing Tool and dragging items from one ontology to the other.

WSMO Studio

The aim of WSMO Studio is to create a collection of tools to assist potential users with ontology creation, service description, service discovery and service composition. These tools are implemented as a collection of plug-ins for the Eclipse framework. These tools include a WSMO Navigator for showing the entities in the WSMO description along with individual form-based editors for each of the WSMO entities. A syntax highlighting text editor is also available for editing the underlying WSML format for more advanced user. WSMO Studio also provides interfaces for interacting with WSMO repositories for storing and retrieving WSMO descriptions.

5. RELATED WORK – RELATIONSHIPS WITH COMPETITIVE APPROCHES

In addition to WSMO there are two major research initiatives in Semantic Web Services. The first and largest of these is OWL-S (Martin), a joint effort by BBN Technologies, Carnegie Mellon University, Nokia, Stanford University, SRI International and Yale University. OWL-S is an ontology for semantic markup of Web Services based on the Web Ontology Language (OWL) (Dean and Schreiber, 2004). The second effort is WSDL-S (Web Service Semantics) from the LSDIS Laboratory at the University of Georgia in co-operation with IBM. The next subsections describe these approaches in more detail using a small set of criteria, followed by a matrix that summarizes the comparison.

5.1 OWL-S

OWL-S is an OWL ontology for describing Web Services by annotating them with semantic information described in OWL (a W3C Recommendation, http://www.w3.org/TR/owl-semantics/). The top-most concept is Service and this in turn consists of three sub-concepts – ServiceProfile, ServiceModel and ServiceGrounding.

The ServiceProfile describes what the service does at a high level and provides the means by which the service can be advertised. It also provides the means by which a service requester can advertise a service that is required. Within the ServiceProfile, the capability description allows for the definition of preconditions, inputs, outputs and effects. There are also slots available in the ServiceProfile description for security parameters, quality rating and for descriptions based on standard business taxonomies.

The ServiceModel describes how a service works and, as a result, how to interact with the service. This part of the OWL-S description is responsible for specifying the service interaction protocol in terms of the messages that should be exchanged with the service and the control flow of that exchange.

The ServiceGrounding is where the abstract description of the service process model is grounded to operations in a WSDL document. Through the ServiceGrounding the actual communication protocols, transport mechanism and the communication languages used by the service are specified. The grounding provides the bridge that links the implementation of a Web Service with its semantic description

Both WSMO and OWL-S address the same problem space. After identifying fundamental drawbacks with the OWL-S approach, the WSMO working group was formed to devise a more complete conceptual model for describing Web Services. Conceptually, unlike WSMO, OWL-S does not explicitly model separate concepts for Goals and Web Services. Additionally OWL-S does not explicitly model mediators; rather they are as considered specific types of services. A detailed discussion of this rationale is provided in (Lara et al., 2004).

5.2 WSDL-S

WSDL-S is a lightweight approach for adding semantics to Web Services. It allows semantic representation of inputs, outputs, preconditions and effects of Web Service operations, by adding extensions to WSDL. WSDL-S allows semantic annotations using domain models, which are agnostic to the ontology used to describe the Web Services or its representation language. It means that ontologies can be used in the annotation process and be directly included in the WSDL documents. The annotations of the inputs and outputs in WSDL will be represented as concepts in an ontology. Additionally, the preconditions and effects associated with WSDL operations will be defined by the preconditions and effects of a specific Semantic Web Service description.

5.3 Matrix of Features and Approaches

The comparison is based on the following features:

- Viewpoint – provider vs. requester
- Mediation – handling heterogeneity between data and process models
- Non-functional properties – additional information about aspects that may affect service usage
- Grounding – how service descriptions relate to Web Service standards
- Availability of execution environments – how do SWS get used

Table 3-1. Comparison of WSMO, OWL-S and WSDL-S

Approach	Supported Viewpoints	Mediation	Non-funct. props.	Grounding	Execution Environment
OWL-S	Single modeling element for both views	Does not treat heterogeneity as a modeling issue.	Restricted to the Service Profile	Grounding of behaviour to WSDL and data to XML	Described but details of impl. are unavailable.
WSDL-S	Service provider view – same as with WSDL	Adopts the behaviour of the ontology used to describe annotations	Agnostic	WSDL-S is a legal extension to WSDL and, as such is directly grounded	Any WSDL compliant execution engine could be extended for WSDL-S
WSMO		Supports mediation of data and processes	Available to all WSMO elements	Grounding of behaviour to WSDL and data to XML	Open source provided by WSMX

6. CONCLUSIONS AND DISCUSSION

Web Services have become another milestone towards providing interoperability among distributed and independent software systems. But one major problem has remained unresolved. Although there is abundance of technologies which theoretically should enable interoperability for disperse systems, from the practical perspective the process of dynamic creation of ad-hoc interactions between companies, as envision by Web Services, is still a fiction. So it is the interoperability issue, not the communication, which has to be addressed next to enable dynamic collaboration of independent software entities on the Internet. Web Services specifications based on

commonly agreed standards and implemented in .NET and J2EE frameworks, are struggling to overcome existing limitations of Web architecture. Data that is exchanged between Web servers and Web browsers remains solely dedicated for human consumption, and cannot be readily processed by automatic software agents. Similarly Web Services and their underlying XML technology still deal mainly with infrastructure, syntax and basic representational issues, but not with the meaning of data and processes that are used by particular systems. Adding semantics to the existing Web Services technologies is a fundamental requirement if we want to deliver workable integration solutions for the next Web generation.

Commercial successes of Semantic Web Services are not yet apparent because the underlying technologies such as presented in this chapter are still in their infancy. Available specifications and technologies will have to go through the lengthy standardization process and real effort of consequent prototype developments, before first commercial solutions are available to the market. There is widespread agreement and recognition that dynamic interoperability on the Internet is only possible if resources are semantically described. WSMO and its related specifications and technologies are principal candidates to become the backbone on the next Web generation, enabling software entities to dynamically interoperate over the Internet.

7. ACKNOWLEDGEMENT

This work is supported by the SFI (Science Foundation Ireland) under the DERI-Lion project and by the European Commission under the projects DIP, Knowledge Web and ASG. The authors thank all members of the WSMO (cf. http://www.wsmo.org/) and WSMX (cf. http://www.wsmx.org/) working groups for fruitful discussions on this chapter.

8. QUESTIONS FOR DISCUSSION

Beginners:

1. Discuss different techniques used by automatic agents to retrieve data from existing computer systems.
2. Why screen scraping cannot scale?
3. Install WSMT and WSMX on your machine. Create ontologies, Web Services, Goal and Mediators. Register them with WSMX.

Intermediate:

1. Explain why existing Web Services specifications are not suitable to enable automated collaboration between distributed software systems.
2. Discuss each of four building blocks of WSMO. Which of them is the most important?

Advanced:

1. Thinking about some real use case scenario (different than presented in this chapter), please explain which elements of automation are the more important from the others. Why?
2. Imagine an interaction scenario similar with the one exemplified in Section 4.3 on Choreography. In which case you would require the usage of both the data and process mediators?
3. Discuss which of the mediation techniques described in this chapter (i.e. data mediation and process mediation) can be used in creating the four types of WSMO mediators? Hint: An ooMediator relay on data mediation for solving the heterogeneity problems between two ontologies.

9. SUGGESTED ADDITIONAL READING

Some key papers that provide more information on WSMO, WSML and WSMX are:

- D.Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler and D. Fensel: Web Service Modeling Ontology. Applied Ontology. Vol. 1, No. 1, 2005.
- H. Lausen, J. de Bruijn, A. Polleres, and D. Fensel: WSML - a Language Framework for Semantic Web Services. W3C Rules Workshop. In Proceedings of the W3C Workshop on Rule Languages for Interoperability, Washington DC, USA, April 2005. Position Paper: http://www.w3.org/2004/12/rules-ws/paper/44.
- M. Moran, M. Zaremba, A. Mocan and C. Bussler: Using WSMX to bind Requester & Provider at Runtime when Executing Semantic Web Services, In Proceedings of the 1st WSMO Implementation Workshop (WIW2004). Frankfurt, Germany, 2004.

For more information consider reading the following books:

- D. Fensel, Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce.
- H. Alesso and C. Smith, Developing Semantic Web Services.
- G. Antoniou and F. van Harmelen, A Semantic Web Primer.

10. ONLINE RESOURCES (INCLUDING OPEN SOURCE TOOLS)

Tool	URL
WSMX Execution Environment (WSMX)	http://www.wsmx.org
Web Services Modeling Toolkit (WSMT)	http://www.wsmx.org
Distributed Ontology Management Environment (DOME)	http://dome.sourceforge.net
WSMO Studio	http://www.wsmostudio.org

11. REFERENCES

Dean M. and Schreiber G. (eds.): OWL Web Ontology Language Reference. 2004. W3C Recommendation 10 February 2004

Gruber T. R., "A translation approach to portable ontology specifications, Knowledge," Knowledge Acquisition, vol. 5, pp. 199-220, 1993

Lara, R., Roman, D., Polleres, A. and Fensel, D., "A Conceptual Comparison of WSMO and OWL-S", Proceedings of The European Conference on Web Services, Erfurt, Germany, Sept 27-30, 2004, pp 254-269.

Martin D. (editor): OWL-S: Semantic Markup for Web Services, version 1.1 available at http://www.daml.org/services/owl-s/1.1/overview/

Milanovic N., Malek M., Current Solutions for Web Service Composition, IEEE Internet Computing, vol. 08, no. 6, pp. 51-59, November/December, 2004.

Tidwell D., "Web Services: the Web's next revolution", http://www-128.ibm.com/developerworks/edu/ws-dw-wsbasics-i.html

Web Service Semantics -- WSDL-S," A joint UGA-IBM Technical Note, version 1.0, April 18, 2005. http://lsdis.cs.uga.edu/library/download/WSDL-S-V1.pdf

Wiederhold G., Mediators in the architecture of future information systems, IEEE Computer, 25(3):38–49, March 1992