

## Semantic Web Services

Carlos Pedrinaci The Open University  
John Domingue The Open University  
Amit Sheth Wright State University

### Abstract

In recent years service-orientation has increasingly been adopted as one of the main approaches for developing complex distributed systems out of reusable components called services. Realizing the potential benefits of this software engineering approach requires semi-automated and automated techniques and tools for searching or locating services, selecting the suitable ones, composing them into complex processes, resolving heterogeneity issues through process and data mediation, and reduce other tedious yet recurrent tasks with minimal manual effort. Just as semantics has brought significant benefits to search, integration and analysis of data, semantics is also seen as a key to achieving a greater level of automation to service orientation. This has lead to research and development, as well as standardization efforts on semantic Web services. Activities related to semantic Web services have involved developing conceptual models or ontologies, algorithms and engines that could support machines in semi-automatically or automatically discovering, selecting, composing, orchestrating, mediating and executing services. This chapter provides an overview of the area after nearly a decade of research. The chapter presents the main principles and conceptual models proposed thus far including OWL-S, WSMO, SAWSDL/METEOR-S, as well as recent approaches that provide lighter solutions and bring support for the increasingly popular Web APIs and RESTful services, like SA-REST, WSMO-Lite and MicroWSMO. The chapter also describes the main engines and frameworks developed by the research community including discovery engines, composition engines, and even integrated frameworks that are able to use these semantic descriptions of services to support some of the typical activities related to services and service-based applications. Next, the ideas and techniques described are illustrated through two use cases that integrate semantic Web services technologies within real-world applications. Finally, a set of key resources that would allow the reader to reach a greater understanding of the field is provided, and the main issues that will drive the future of semantic Web services are outlined.

## 1 Introduction

Service-orientation is an approach to developing software by combining reusable and possibly distributed components called services [1]. Since it was first proposed, there has been much research and development around service-orientation principles, architectural models, languages providing means for describing components as reusable services, languages for defining service compositions and a plethora of software for supporting this vision have been implemented.

Although each major vendor promotes a distinct set of concrete technologies and solutions, all these approaches have in common the identification of a services registry where existing services can be located and the provisioning of means (e.g., languages, tools, and engines) to compose these services to achieve more complex functionalities, therefore supporting the seamless creation of complex distributed solutions out of pre-existing components [2]. This is essentially an architectural model for developing software out of reusable and distributed services which is often referred to as Service-Oriented Architecture (SOA).

Although technology-independent, SOA is typically implemented using Web service technologies such as the Web Service Description Language (WSDL) and SOAP [2]. WSDL was created specifically for this purpose, providing several useful constructs for describing services, including operations to describe service methods, parameter descriptions via XML schema, and information about the type of protocol needed to invoke the service (e.g., SOAP over HTTP). Many activities require additional functionalities not captured in the basic service specification supported by WSDL, and some use of multiple services to accomplish them. Correspondingly, the so-called WS-\* standards and composition languages [3] have been defined in order to provide further capabilities to service-oriented solutions [1, 2].

SOA is commonly lauded as a silver bullet for Enterprise Application Integration, implementation of inter-organizational business processes, and even as a general solution for the development of all complex distributed applications. However, their uptake on a Web-scale has been significantly less prominent than initially anticipated [4]. Instead, more recently, plain and simple Web technologies (HTTP, XML, and JSON), which underlie machine-oriented Web applications and APIs, are increasingly being used to provide added-value solutions that combine information from diverse sources seamlessly, constituting simple and “lightweight” service-oriented software. These recent services are commonly referred to as RESTful services—when they follow REST principles [5]—or Web APIs in general.

Independently from the technologies adopted and despite the appealing characteristics of service-orientation principles and technologies, the systematic development of service-oriented applications remains limited and effectively still far from truly benefiting from the promised simplicity for constructing agile and interoperable systems. The fundamental reason for this lies on the need for software developers to devote significant labour to discovering sets of suitable services, interpreting them, developing software that overcomes their inherent

data and process mismatches, and finally combining them into a complex composite process.

Semantic Web services (SWS) were proposed in order to pursue the vision of the semantic Web presented in [6] whereby intelligent agents would be able to exploit semantic descriptions in order to carry out complex tasks on behalf of humans [7]. This early work on SWS was the meeting point between semantic Web, Agents, and Web services technologies. Gradually, however, research focussed more prominently on combining Web services with semantic Web technologies in order to better support the discovery, composition, and execution of Web services, leaving aspects such as systems' autonomy more typical of agent-based systems somewhat aside.

Research on SWS has been active and fruitful over the years leading to a number of conceptual models, representation languages, as well as to a plethora of software components and even integrated execution frameworks that cover diverse tasks within the life-cycle of Web services and service-oriented applications. This chapter aims at providing an overview of the main results achieved so far, giving the reader pointers for gathering further insights and details on how these solutions have been devised. SWS research builds upon results and techniques from a wide-range of fields and therefore, for the sake of clarity and space, the focus is on the main approaches and techniques directly applied to SWS proposed thus far, leaving the more systematic review of related fields to the interested reader.

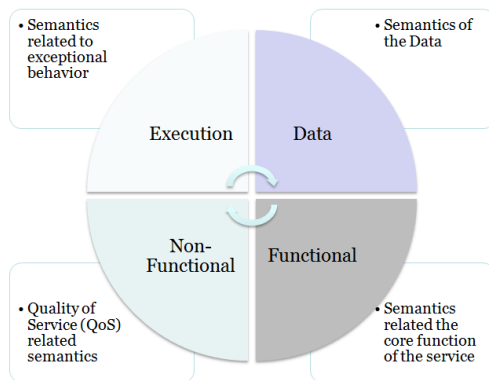
The remainder of this chapter is organised as follows. First, the types of semantics present in a service and the main concepts surrounding SWS are covered briefly so that the reader can understand better the subsequent sections. Next, the main conceptual models for describing SWS devised so far are presented. On the basis of these conceptual models the main frameworks and components that exploit SWS to achieve concrete tasks are introduced. After the technical details have been introduced, a couple of example applications are presented that aim at illustrating some of the concepts exposed in earlier sections of this chapter. Finally, a set of key references and related papers that are considered of particular relevance are included, and the chapter concludes introducing future issues that are expected to be at the centre of research on SWS in the forthcoming years.

## 2 Scientific Overview

Semantic Web services were first proposed by McIlraith et al. in [7] as an extension of Web services with semantic descriptions in order to provide formal declarative definitions of their interfaces as well as to capture declaratively what the services do. Early on, Sheth et al. [8, 9] introduced the four main types of semantics (see Figure 1), that corresponding semantic descriptions can capture:

1. Data semantics: The semantics pertaining to the data used and exposed by the service.
2. Functional semantics: Semantics pertaining to the functionality of the service.
3. Non-Functional semantics: Semantics related to the non-functional aspects of the service, e.g., Quality of Service (QoS), security or reliability.

4. Execution semantics: Semantics related to exceptional behaviours such as run time errors.



**Figure 1: Four types of semantics introduced in [8, 9].**

The essential characteristic of SWS is therefore the use of languages with well-defined semantics covering the subset of the mentioned categories that are amenable to automated reasoning. Several languages have been used so far including those from the semantic Web, e.g., RDF(S) and OWL, SWS-specific languages such as the Web Service Modeling Language (WSML), or others originating from research on Knowledge-Based Systems such as F-Logic and OCML.

On the basis of these semantic descriptions, SWS technologies seek to automate the tasks involved in the life-cycle of service-oriented applications which include the discovery and selection of services, their composition, their execution and their monitoring among others. Part of the research on SWS has been devoted precisely to identifying the requirements for SWS systems, the tasks involved and even to defining conceptual frameworks and architectures that cover the entire life-cycle of SWS [7, 10, 11, 12, 13].

The different conceptual frameworks proposed make particular emphasis on certain aspects such as better supporting the decoupling and scalability of the solutions, on supporting the interactions between agents, or on reaching a solution that appropriately fulfils humans expectations. Instead of adopting a concrete framework, it is herein distilled what are typically the essential features of all the frameworks proposed in order to give an overall view on the field and help the reader to better understand the remainder of the chapter. Later the concrete proposals will be covered in more detail.

All the frameworks take as a starting point service-orientation principles and are strongly based on Service-Oriented Architectures. Hence, they view the construction of systems as a process involving the encapsulation of reusable components as services, their publication into shared registries, the location of existing and suitable services from these shared repositories, their composition into executable workflows, and their eventual execution or enactment. SWS frameworks essentially propose the application of semantics to reach a higher-level of automation throughout these tasks.

The remainder of this section identifies the main tasks and concepts typically utilised and mentioned within in the SWS field in order to provide a common set of definitions. Some of the papers cited in this chapter may use a slightly different terminology often due to the evolution of the research in the area. Where appropriate, typical uses of different terminology that readers may encounter shall be identified. The definitions exposed herein are largely inline with those provided by the W3C Web Services Glossary [14].

**Crawling** is the task in charge of browsing the Web in order to locate existing services. This task is essentially identical to any other Web crawling endeavour with the difference that the information sought is not Web pages but rather WSDL files or more recently HTML pages describing Web APIs.

**Discovery** involves locating services that are able to fulfil certain user requirements. This task encompasses, starting from abstract definitions of users' needs, operationalising the requirements such that they can be used for identifying Web services that can subsequently be used for *discovering* whether they can provide the desired service. This definition of discovery, which is based on that given in [15], distinguishes the notion of service from a business perspective (e.g., booking a concrete flight), from the notion of Web service, which is the functional component able to provide a certain service (e.g., booking flights operated by a particular airline). This understanding of service discovery aims at providing functionality closer to human needs by clearly distinguishing between services that have effect on the real world from the Web services that can be invoked to achieve these changes. It therefore distinguishes the location of possibly suitable services from the actual discovery of those that can really provide the service sought for; after all not all the flight booking Web services will allow booking any flight from any company.

The term discovery is perhaps the most widely applied term in SWS research, however, in the large majority of the cases the task that it is referred to is what is here referred to as service matching or service matchmaking. It is worth noting that the definition just outlined involves but is not limited to service matching and may also require an additional activity called service crawling.

**Matching**, also referred to as matchmaking in several papers, is the task that given a request for some *kind of Web service* tries to identify Web service advertisements that match to a certain degree the request. Research in Web service matching has devoted substantial efforts to formalising Web services functionality in ways that can support automatic matching using reasoners. In general Web service functionality is specified in terms of inputs, outputs, pre-conditions and effects (in WSMO, see Section 2.1.1.2, assumptions and post-conditions are also considered). The set of known Web services advertisements are then matched against the functionality specifications sought for using reasoners and additional heuristics. The result is a restricted set of Web services according to different degrees of matching which most often contemplate at least exact, plugin—when the advertisement subsumes the request, subsumes—when the request subsumes the advertisement, and fail.

**Ranking** is the task that given a set of Web services obtained from the matching process, ranks the different matches according to a set of preferences. These preferences are usually given at invocation time and are specified in terms of the non-functional properties of Web services, e.g., price, quality of service. In this manner it is possible to order Web services that are able to provide the required functionality based on other kinds of criteria. It is worth noting in this respect, that the matching degree described earlier in one kind of simple criteria typically applied.

**Selection** is the task that having obtained a list of (possibly ranked) suitable Web services, selects one to be invoked. This task is often performed by humans but there also exist systems that carry it out automatically based on prior ranking criteria. Since in most cases there may exist data heterogeneities, most of the systems implementing automated selection also provide data mediation facilities so that invocation can take place.

**Composition** is the task in charge of combining Web services in order to achieve a complex task. Typically this task is triggered whenever the system is unable to find a Web service that fulfils all the requirements. The result of a composition task is an orchestration of Web services that, given some initial conditions and a set of Web services, would lead to the desired state when executed. Most of the work in automated semantic Web service composition has been approached as a planning task, which benefits from the formal specification of Web services inputs, outputs, pre-conditions and effects to generate suitable orchestrations.

**Orchestration** defines the sequence and conditions for the enactment of Web services in order to achieve a complex objective by appropriately combining the functionality provided by existing Web services. Orchestration definitions include data-flow (i.e., how the data is propagated and used throughout the process) and control-flow (i.e., when should a certain activity be executed). There exists a wide range of process specification languages that have been defined over the years. In this chapter those that have been used in SWS research are introduced. The reader is referred to [16, 17, 18] for further insights.

**Choreography** describes the interactions of services with their users, whereby any client of a Web service, may it be a human or a machine, is considered a user. A choreography defines the expected behaviour of a Web service, that is the exchanged messages, from a client's point of view. Choreography interpretation leads to a successful invocation of a Web service independently from how the execution of the Web service is performed internally.

**Mediation** is necessary in environments where having heterogeneous components is frequent. Heterogeneity is one of the main characteristics of the Web and therefore any Web-oriented solution must face this issue in one way or another. Mediation is a principle by which an intermediate element, a mediator, is introduced between two elements to resolve their heterogeneities without having to adapt one or the other. In a nutshell, heterogeneity in Web services can affect three main aspects: i) the terminology used; ii) the representation and network-level protocol used for communication; and iv) the application-level protocol expected by two interacting parties.

Issues concerning terminology and the representation of data are commonly referred to as data mediation. Data mediation aims at resolving the mismatches between the data handled by both services typically by approaching it as an ontology mapping/transformation problem. Conversely, protocol mediation aims at achieving a successful interaction between two processes (or Web services) by ensuring that the message exchanges between both processes are as they expect. Protocol mediation, which may not always be resolvable,

often involves buffering and the reordering of messages, or combining them so that each process can reach a successful end state.

**Invocation** is concerned with the actual call to an operation of a Web service. Invocation is therefore closely related to choreographies in that, the latter will specify an order for performing a set of invocations.

**Grounding** specifies how certain activities are mapped into low-level operations with Web services. The need for grounding specifications comes from the fact that semantic Web services are essentially handled at the semantic level where invocation details are often disregarded. In most of the approaches grounding definitions are basically pointers to existing operation definitions.

**Lifting** refers to the transformation of information from its representation at the syntactic level used by the Web service (typically XML) into its semantic counterpart. Lifting is usually expressed declaratively so that it can directly be interpreted by some engine in order to transform the data into some semantic representation, e.g., RDF, OWL, WSML. Given that many services exchange XML data, XSLT is often the language of choice.

**Lowering** refers to the task that takes information represented semantically and transforms it into some syntactic representation that can be used for communicating with the Web service. This task is the inverse of Lifting, and likewise is often approached with XSLT transformations.

## 2.1 Conceptual Models

Central to the work on SWS are the semantic descriptions or annotations of services that support automating to a greater extent tasks such as their discovery, selection and composition. Consequently, much effort has been devoted over the years to devising conceptual models able to support creating suitable semantic descriptions for services. In the remainder on this section the main approaches, which have been divided into top-down approaches and bottom-up approaches are introduced. Top-down approaches to the development of semantic Web services like the Web Service Modeling Ontology (WSMO) [19, 20] and OWL-S [21], are based on the definition of high-level ontologies providing expressive frameworks for describing Web services. On the other hand, bottom-up models, i.e., WSDL-S [22] and SAWSDL [23], adopted an incremental approach to attaching semantics to existing Web services standards by adding specific extensions that connect the syntactic definitions to their semantic annotations.

### 2.1.1 Top-Down Approaches

#### 2.1.1.1 OWL-S

OWL-S [21], formerly DAML-S, is an ontology for the description of semantic Web services expressed in the Web Ontology Language (OWL) [24]. OWL-S, which was submitted to W3C in 2004, defines an upper ontology for semantically describing Web services along three main aspects:

- The *Service Profile* describes ‘*what the service does*’ in terms of inputs, outputs, preconditions and effects (IOPEs);
- The *Service Model* describes ‘*how a service works*’ in terms of a process model that may describe a complex behaviour over underlying services; and
- The *Service Grounding* describes ‘*how the service can be accessed*’, usually by grounding to WSDL.

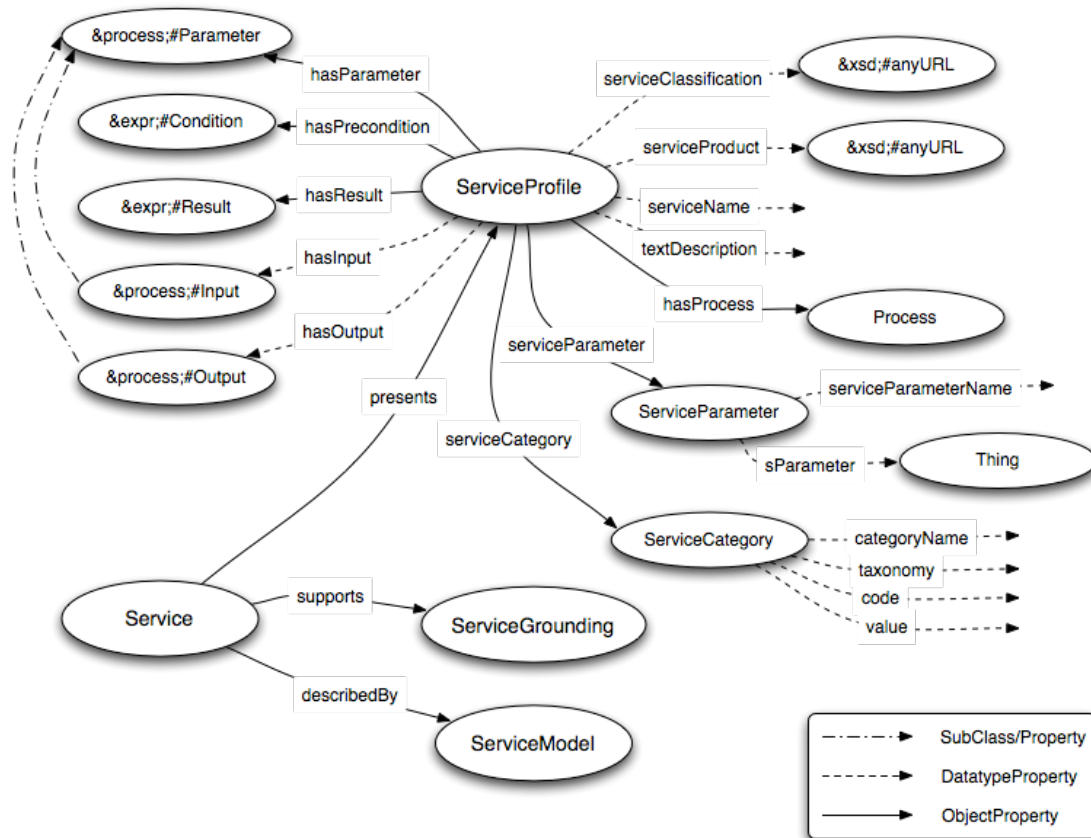


Figure 2. The OWL-S Ontology (figure adapted from [21]).

The Service Profile provides the core functional description of services which is used for advertising. This description provides a high-level representation on what the service does in a manner that is suitable for software agents to discover whether a service is adequate for their purposes or not. A service is described mainly in terms of its functional parameters: inputs, outputs, preconditions and effects (IOPEs), see Figure 2. Inputs and outputs basically specify semantically the kinds of parameters handled by the service. Preconditions are logical expressions that specify the conditions that are required for the service to be executed successfully (e.g., the customer has to be located in the USA). Effects, also referred to as Results in OWL-S, on the other hand specify changes in the world should the execution of the service be successful (e.g., the book is shipped to the given address). Additionally, the Service Profile includes support for capturing information such as classifications with respect to reference taxonomies, the name of the service, and textual descriptions.

The Service Model informs clients about how to use the service. It does so by specifying the semantic content of requests, replies, the conditions under which certain results hold, and how clients have to invoke the service. In order to tell clients how to interact with a service, OWL-S views services as processes with a set of inputs, outputs, preconditions and effects, as well as a process model specifying the ways in which a client may interact with the service. Reasoning support for OWL-S is provided primarily by OWL-DL reasoners. However, OWL-DL is often not sufficiently expressive or not suitable for defining preconditions and effects. For these cases, OWL-S supports the specification of

```

<process:AtomicProcess rdf:ID="Purchase">
  <process:hasInput>
    <process:Input rdf:ID="ObjectPurchased"/>
  </process:hasInput>
  <process:hasInput>
    <process:Input rdf:ID="PurchaseAmt"/>
  </process:hasInput>
  <process:hasInput>
    <process:Input rdf:ID="CreditCard"/>
  </process:hasInput>
  <process:hasOutput>
    <process:Output rdf:ID="ConfirmationNum"/>
  </process:hasOutput>
  <process:hasResult>
    <process:Result>
      <process:hasResultVar>
        <process:ResultVar rdf:ID="CreditLimH">
          <process:parameterType rdf:resource="#ecom;#Dollars"/>
        </process:ResultVar>
      </process:hasResultVar>
      <process:inCondition>
        <expr:KIF-Condition>
          <expr:expressionBody>
            (and (current-value (credit-limit ?CreditCard)
                               ?CreditLimH)
                 (>= ?CreditLimH ?purchaseAmt))
          </expr:expressionBody>
        </expr:KIF-Condition>
      </process:inCondition>
      <process:withOutput>
        <process:OutputBinding>
          <process:toParam rdf:resource="#ConfirmationNum"/>
          <process:valueFunction rdf:parseType="Literal">
            <cc:ConfirmationNum xsd:datatype="#xsd;#string"/>
          </process:valueFunction>
        </process:OutputBinding>
      </process:withOutput>
      <process:hasEffect>
        <expr:KIF-Condition>
          <expr:expressionBody>
            (and (confirmed (purchase ?purchaseAmt) ?ConfirmationNum)
                 (own ?objectPurchased)
                 (decrease (credit-limit ?CreditCard)
                           ?purchaseAmt))
          </expr:expressionBody>
        </expr:KIF-Condition>
      </process:hasEffect>
    </process:Result>
    ...
  </process:hasResult>
</process:AtomicProcess>

```

Listing 1. Example of an OWL-S Atomic Process (taken from [17]).

preconditions and effects by means of literals, either string literals or XML literals, that allow modellers to adopt arbitrary languages, such as SWRL [25], for including expressions that are beyond the expressivity of OWL.

OWL-S provides an advanced solution for conditioning outputs and effects so that one can express that a certain effect in the world would only occur if a certain condition held for the inputs (e.g., you get a voucher as a present if you spend more than a certain amount). Additionally, OWL-S includes additional modelling constructs in order to provide means for expressing complex processes. OWL-S distinguishes three main kinds of processes: *Atomic Processes*, *Composite Processes*, and *Simple Processes*. See Listing 1 for an example of an Atomic Process.

Atomic Processes are processes that are directly invocable and, as far as the service requester is concerned, they only require one single interaction. Atomic Processes therefore require a grounding indicating how invocation messages have to be constructed and the result messages parsed. More details about the grounding of Atomic Processes are provided later.

Composite Processes are processes that require several steps in the interaction and/or multi server actions. They are also decomposable into other processes (composite or not). In order to support the definition of Composite Processes, OWL-S provides a set of block-oriented control constructs, such as *Sequence*, *If-Then-Else* or *Repeat While*. It is worth noting, however, that this control flow definitions do not specify how the service will behave but rather what clients invoking the service could do.

Finally, Simple Processes provide an abstraction mechanism able to provide multiple views over existing Atomic and Composite Processes. Simple Processes are not directly invocable (they are not associated with a grounding), although they have single-step interactions much like Atomic Processes. Simple Processes are mostly used as suitable abstractions for simplifying tasks such as planning and reasoning. However, in order to support the eventual invocation of these processes OWL-S provides the *realizedBy* and *expandsTo* relations, connecting them to *Atomic Processes* and *Composite Processes* respectively.

The Service Grounding provides the details necessary for invoking the service. It is therefore concerned with aspects such as the protocol to be used, the message format, their serialization, the transport protocol and the address of the endpoint to be invoked. In a nutshell the Service Grounding is a mapping between the parameters handled by Atomic Processes and the messages that carry those parameters in some specific transmittable format. OWL-S does not predefine the language to be used for grounding, however, due to its wide adoption a reference grounding implementation is provided for WSDL.

### **2.1.1.2 WSMO**

WSMO [19, 20] is a member submission to W3C of an ontology that aims at describing all relevant aspects for the partial or complete automation of discovery, selection, composition, mediation, execution and monitoring of Web services. WSMO has its roots in the Web Service Modeling Framework (WSMF) [10] and in Problem-Solving Methods [26], notably the Unified Problem Solving Method Development Language UPML [27], which have been extended and adapted in order to support the automation of the aforementioned tasks for manipulating Web services.

WSMF provides an overall framework for describing Web services based on two essential principles for semantic descriptions of Web services:

- **Strict de-coupling** - the various components that realize an application are described in a unitary fashion without regard to the neighbouring components. This enables components to be easily linked and promotes scalability following the open and distributed nature of the Web.
- **Centrality of Mediation** – building on the concept of bridges within the UPML framework WSMF includes the notion of mediators to deal with mismatches which may occur between components. Heterogeneity can occur in terms of data, underlying ontology, protocol or process. WSMF recognizes the importance of mediation for the successful deployment of Web services by making mediation a first class component. A mediator provides a link to a mechanism which can resolve the identified mismatch.

WSMO provides an ontology for describing Web services based upon WSMF. WSMO identifies four top-level elements as the main concepts, namely *Ontologies*, *Web Services*, *Goals* and *Mediators* (in this chapter these terms will be used in capitals whenever WSMO elements are referred to). *Ontologies* provide the formal semantics for the terminology used within all other WSMO components. Essentially WSMO establishes that all resource descriptions and all data interchanged during service usage should be semantically described on the basis of ontologies. Web Services are computational entities that provide some value in a given domain. Goals, represent clients' perspective by supporting the representation of users' desires for certain functionality. Finally, Mediators represent elements that handle interoperability problems between any two WSMO elements. In fact, one core principle behind WSMO, is the centrality of mediation as a means to reduce the coupling and deal with the heterogeneity that characterises the Web.

Together with the ontology, research around WSMO has also produced a family of languages, the Web Service Modeling Language (WSML) [19, 28]. The remainder of this section covers each of the WSMO elements in more detail. However, for the sake of clarity and simplicity WSML itself is not introduced, instead Meta Object Facility (MOF) [29], which is the meta-meta-model language that has been used for representing the elements of the WSMO ontology is used.

Ontologies, see Listing 2, can be defined in a modular way by importing others. When importing ontologies in realistic scenarios, some steps for aligning, merging and transforming imported ontologies in order to resolve ontology mismatches are required. For this reason ontology mediators – *OO Mediators* in particular – are used. The other elements are as normally found within ontology definition languages. *Concepts* constitute the basic elements of the agreed terminology for some problem domain. *Relations* are used in order to model dependencies between several concepts (respectively instances of these concepts); *Functions* are special relations, with a unary range and an n-ary domain (parameters inherited from relation), where the range value is functionally dependent on the domain values, and instances are either defined explicitly or by a link to an instance store, i.e., an external storage of instances and their values.

**Class** Ontology

hasNonFunctionalProperties **type** nonFunctionalProperties  
importsOntology **type** Ontology  
usesMediator **type** OOMediator  
hasConcept **type** Concept  
hasRelation **type** Relation  
hasFunction **type** Function  
hasInstance **type** Instance  
hasAxiom **type** Axiom

**Listing 2. The Ontology element in WSMO.**

*Web Services*, see Listing 3, are online components that provide functionality. Web Services can make use of an extra type of mediator – *WW Mediator* – to deal with protocol and process related mismatches between Web services. The two core WSMO notions for semantically describing Web Services are a *capability* and *service interfaces*.

A *capability* defines the functionality offered by a service by means of the following four main items:

- *Pre-conditions* – a set of logical expressions which specify constraints over the inputs of a service. The focus here is on the data which can be accessed within the available reasoning system.
- *Assumptions* – within service usage scenarios one often needs to make statements about the world outside of the platform on which a service is executed. Assumptions provide the means for doing so. Although of course it is not always feasible to check the status value for the statements, the statements are still of value in terms of a formal description of a potentially important constraint.
- *Post-conditions* – a set of logical expressions which specify constraints over the outputs of a service. Like for pre-conditions, the focus here is on the data which can be accessed within the available reasoning system.
- *Effects* – statements which relate to the state of the world after the service has been executed. As with assumptions it may not always be feasible to check the absolute truth values of the statements but still they serve a useful formal documentation role and can facilitate verification and monitoring.

A *service interface* defines how the functionality of a service can be achieved by means of a *choreography* and an *orchestration*. A choreography describes the behaviour of a service from the client's point of view. WSMO provides a state-based mechanism for describing choreographies based on Abstract-State Machines (ASM) [30], whereby the choreography consists of a *state signature* (concepts and variables manipulated) and a set of *transition rules* that manipulate the data.

An *orchestration* captures the control and data-flow within a complex Web service by interacting with other Web services. Orchestrations are commonly used to: a) ensure behavioural congruence, i.e., that the orchestration of a service matches its declared choreography, b) facilitate the reuse of service

combinations, and c) enable client constraints to be checked. The definition of orchestrations in WSMO is still subject of research and is envisioned to be also based on ASM. Additionally, research has been carried out for providing transformations from more common workflow representation languages such as UML Activity Diagrams [31].

```
Class WebService
  hasNonFunctionalProperties type NonFunctionalProperties
  importsOntology type Ontology
  usesMediator type {OOMediator, WWMediator}
  hasCapability type Capability multiplicity = single-valued
  hasInterface type Interface
```

**Listing 3. The Web Service element in WSMO.**

*Goals*, see Listing 4, are derived from the notion of task prevalent in previous work including **Knowledge Acquisition and Documentation Structuring (KADS)** [26], UPML [32], and Generic Tasks [33]. Goals are used to represent the viewpoint of a service requester or client. Goals also reflect the structure of a Web service capturing aspects related to user desires with respect to the requested functionality and behaviour. Thus, the requested capability in the definition of a Goal represents the functionality of the services the user would like to have, and the requested interface represents the interface of the service the user would like to have and interact with. Goals therefore represent the starting point for service discovery in WSMO-based frameworks.

```
Class Goal
  hasNonFunctionalProperties type NonFunctionalProperties
  importsOntology type Ontology
  usesMediator type {OOMediator, GGMediator}
  requestsCapability type Capability multiplicity = single-valued
  requestsInterface type Interface
```

**Listing 4. The Goal element in WSMO.**

*Mediators* in WSMO handle heterogeneities which can occur when two software components are put together. Mediators, see Listing 5, are defined on the basis of a number of *sources*, a *target* and a *mediation service* that is in charge of performing the actual mediation. WSMO defines different types of mediators for connecting the distinct WSMO elements: *OO Mediators* connect and mediate between heterogeneous ontologies, *GG Mediators* connect Goals, *WG Mediators* link Web Services to Goals, and *WW Mediators* connect interoperating Web Services resolving mismatches between them. The mediation service may be specified as a service, a WW Mediator or as a Goal.

Following from the extensive use of metadata within the Web every WSMO element includes a non-functional properties attribute which extends the Dublin Core (DC) Metadata Set. Non-functional properties include basic information

such as the author and creation date and service-specific properties related to the quality of the described service.

```
Class Mediator
  hasNonFunctionalProperties type NonFunctionalProperties
  importsOntology type Ontology
  hasSource type {Ontology, Goal, WebService, Mediator}
  hasTarget type {Ontology, Goal, WebService, Mediator}
  hasMediationService type {Goal, WebService, WWMediator}
```

**Listing 5. Mediator element in WSMO.**

## 2.1.2 Bottom-up Approaches

### 2.1.2.1 WSDL-S & SAWSDL

WSDL-S was proposed as a member submission to the W3C in November 2005 between the LSDIS Laboratory at University of Georgia (the majority of the group has since moved to the Kno.e.sis Center, Wright State University, <http://knoesis.org>) and IBM [22]. WSDL-S is a light-weight approach to associating semantic annotations with Web services developed in the context of the Managing End-to-End Operations-Semantics (METEOR-S) project that will be presented in more detail in Section 2.2.4. The key innovation of WSDL-S lies in the use of extensibility in elements and attributes supported by WSDL specification [9]. Using the extensibility of WSDL, semantic annotations in the form of URI references to external models (which can be ontologies, and were broadly termed conceptual models) can be added to the interface, operation and message constructs. WSDL-S is independent from the language used for defining the semantic models and explicitly contemplates the possibility of using WSML, OWL and UML as potential candidates [22].

WSDL-S provides a set of extension attributes and elements for associating the semantic annotations. The extension attribute *modelReference* allows one to specify associations between a WSDL entity and a concept in a semantic model. This extension can be used for annotating XML Schema complex types and elements, WSDL operations and the extension elements *precondition* and *effect*. WSDL-S defines two new children elements for the WSDL operation element, namely *precondition* and *effect*. These elements facilitate the definition of the conditions that must hold before executing an operation and the effects the execution would have. This information is typically used for discovering suitable Web services.

The *schemaMapping* extension attribute can be used for specifying mechanisms for handling structural differences between XML Schema elements and complex types and their corresponding semantic model concepts. These annotations can then be used for what is referred to as the *lifting* and *lowering* of execution data (i.e., transforming syntactic data into its semantic counterpart and vice versa). The concept of using an intermediate model with lifting and lowering transformation is used as the standard mechanism for mediation by WSDL-S [34].

Finally, WSDL-S includes the *category* extension attribute on the interface element in order to define categorization information for publishing Web services in registries as defined by the Universal Description, Discovery and Integration (UDDI) specification [35] for example.

In April 2006 WSDL-S was adopted as the main input for a W3C working group whose task was to create the first W3C recommendation for enabling the semantic annotation of Web service descriptions. The working group produced the Semantic Annotations for WSDL and XML Schema (SAWSDL) specification [23], which was adopted as a W3C Recommendation in August 2007. SAWSDL is a restricted and homogenized version of WSDL-S including a few changes trying to give a greater level of genericity to the annotations and disregarding those issues for which there existed no agreement among the community at the time the specification was created.

There are essentially three main differences between SAWSDL and WSDL-S. The first one is the fact that *precondition* and *effect* are not directly contemplated since there was no agreement on how to model them within the semantic Web and semantic Web services community. It is worth noting however that SAWSDL does not preclude including this type of annotations as illustrated in the usage guide generated by the SAWSDL working group [36]. Secondly, the *category* annotation is replaced in SAWSDL by the more general *modelReference* extension attribute, which can be used to annotate XML Schema complex type definitions, simple type definitions, element declarations, and attribute declarations as well as WSDL interfaces, operations, and faults. Finally, WSDL-S' *schemaMapping* annotation was decomposed into two different extension attributes, namely *liftingSchemaMapping* and *loweringSchemaMapping*, so as to specifically identify the type of transformation performed. Figure 3 shows a high level architecture of a SAWSDL annotated Web service, and Listing 6 gives an example snippet.

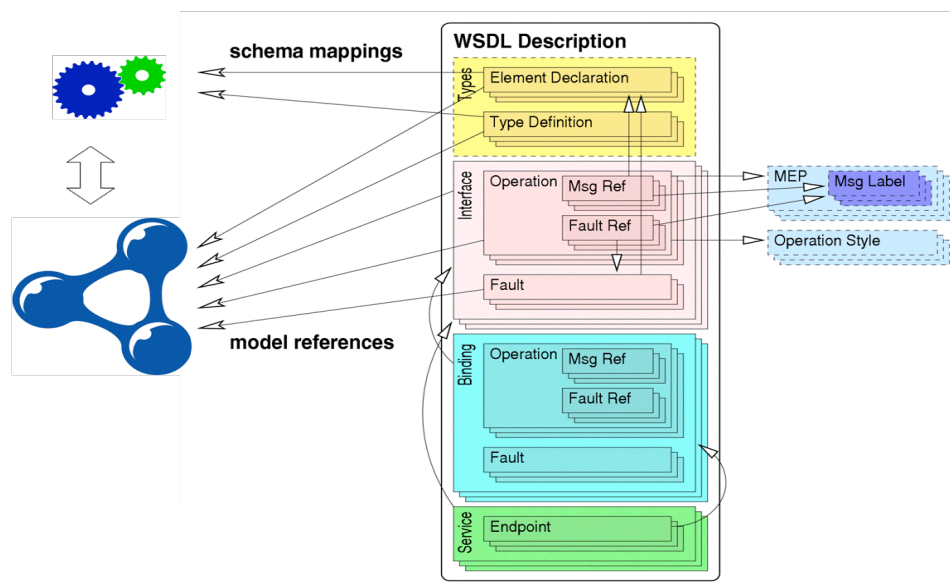


Figure 3. SAWSDL annotations (figure created by Jacek Kopecky).

SAWSDL therefore constitutes a light weight and incremental approach (compared to OWL-S and WSMO) to annotating WSDL services. In its inception, however, much care was devoted to ensuring its extensibility and on remaining

agnostic with respect to the ontologies used, and the languages in which these conceptual models as well as the transformations are defined. As a consequence, SAWSDL has so far been used to link to a variety of ontologies defined in different languages such as RDF(S) and WSM, as well as to point to diverse transformation languages among which XSLT and XSPARQL [37] are perhaps the most commonly applied.

```

<wsdl:description targetNamespace="http://www.w3.org/.../order#"
  xmlns:wsdl="http://www.w3.org/ns/wsdl"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:sawSDL="http://www.w3.org/ns/sawSDL">
  <wsdl:types>
    <xs:element name="processPurchaseOrderResponse" type="xs:string
      sawSDL:modelReference="http://www.w3.org/.../rosetta#PurchaseOrderResponse"
      sawSDL:liftingSchemaMapping="http://www.w3.org/.../POResponse2Ont.xslt"
      sawSDL:loweringSchemaMapping="http://www.w3.org/.../Ont2Response.xslt">
    </xs:element>
  </wsdl:types>
  <interface name="PurchaseOrder"
    sawSDL:modelReference="http://example.org/.../products/electronics />
  <operation name="order" pattern=wsdl:in-out
    sawSDL:modelReference="http://www.w3.org/.../rosetta#RequestPurchaseOrder">
    <input messageLabel = "processPurchaseOrderRequest"
      element="tns:processPurchaseOrderRequest" />
    <output messageLabel = "processPurchaseOrderResponse"
      element="processPurchaseOrderResponse" />
  </operation>
  <operation name="cancel" pattern=wsdl:in-out
    sawSDL:modelReference="http://www.w3.org/.../rosetta#CancelOrder" >
    <input messageLabel = "processCancelRequest"
      element="tns:processCancelRequest" />
    <output messageLabel = "processCancelResponse"
      element="processCancelResponse" />
  </operation>
</interface>
</wsdl:description>

```

Listing 6. A SAWSDL snippet describing (parts of) a service supporting the Purchase Order as defined in RosettaNet (URIs have been shortened for reasons of space).

### 2.1.2.2 WSMO-Lite

As described in the previous section SAWSDL provides simple hooks for pointing to semantic descriptions from WSDL and XML elements. In particular, it supports three kinds of annotations, namely *modelReference*, *liftingSchemaMapping* and *loweringSchemaMapping* which can point to semantic elements described elsewhere on the Web, or to specifications of data transformations from a syntactic representation to the semantic counterpart and back respectively. SAWSDL does not advocate a particular representation language for these documents nor does it provide any specific vocabulary that users should adopt. This characteristic is a means to support extensibility but also forces users to choose their own ontologies for describing services semantically.

WSMO-Lite continues this incremental construction of a stack of technologies for semantic Web services by precisely addressing this lack [38]. WSMO-Lite identifies four main types of semantic annotations for services which are a variant of those in [8]:

- **Functional semantics** defines service functionality, that is, the function a service offers to its clients when it is invoked. This information is of particular relevance when finding services and when composing them.

- **Nonfunctional semantics** defines any specific details concerning the implementation or running environment of a service, such as its price or quality of service. Nonfunctional semantics provide additional information about services that can help rank and select the most appropriate one.
- **Behavioral semantics** specifies the protocol (i.e., ordering of operations) that a client needs to follow when invoking a service.
- **Information model** defines the semantics of input, output and fault messages.

WSMO-Lite provides a minimal RDFS ontology and provides a simple methodology for expressing these four types of semantic annotations for WSDL services using SAWSDL hooks. In particular, to specify the annotations over a concrete WSDL service, WSMO-Lite relies on the SAWSDL *modelReference* attribute for all four semantic annotations, except for information models where *liftingSchemaMapping* and *loweringSchemaMapping* might also be necessary.

WSMO-Lite offers two mechanisms for representing functional semantics, namely simple taxonomies and more expressive preconditions and effects. Functionality taxonomies provide a simple means by which one can define service functionalities through a hierarchy of categories (e.g., eCl@ss [39]). This is essentially the typical cataloguing approach used in traditional systems such as UDDI, although it is enhanced with reasoning support. In order to distinguish functional classifications from other types of *modelReference* annotations, WSMO-Lite offers the RDFS class *wsl:FunctionalClassificationRoot*, where *wsl* identifies the namespace for WSMO-Lite.

Whenever more expressivity is necessary, WSMO-Lite offers the possibility to enrich functional classification with logical expressions defining conditions that need to hold prior to service execution, and capturing changes that the service will carry out on the world. In particular, WSMO-Lite supports defining both by means of the classes *wsl:Condition* and *wsl:Effect* respectively.

Nonfunctional semantics in WSMO-Lite are represented using external ontologies capturing nonfunctional properties such as security aspects, the quality of service and price. To do so WSMO-Lite includes the class *wsl:NonfunctionalParameter* so that ontologies defining concrete non-functional properties for a service, can refer to this class and let the machine know what kind of information it contains.

Behavioral semantics describe how the client should communicate with a service. This kind of description is necessary in order for clients to know, given a particular goal to be achieved, in which order it should invoke the different operations provided by the service. WSMO-Lite, as opposed to its heavyweight counterpart – WSMO –, does not include explicit behavioural descriptions. Instead, clients should use the existing functional annotations (classifications, conditions and effects) over the entire service and the internal operations in order to figure out in which order to invoke them. For instance, planning-based techniques could be applied locally to compute the order for invoking operations. This approach gives total flexibility to both clients and service annotators.

Finally, the information model captures the semantics of the data exchanged between a service and its clients. Indeed, understanding the data is crucial for automated invocation as well as for Web service compositions where data mediation may be necessary. WSMO-Lite relies on external ontologies for

capturing information models. Information models are distinguished from other models such as functional classifications by using *wsl:Ontology*. Additionally, because Web services generally work with application-specific XML data, WSMO-Lite advocates the use of *liftingSchemaMapping* and *loweringSchemaMapping* for pointing to transformation specifications alongside the model references on the appropriate XML Schema components.

### 2.1.2.3 MicroWSMO

Web sites are increasingly offering services and data through Web APIs and RESTful services [40], that is services adopting REST principles [5]. These services are combined by Web developers into what is usually referred to as mashups, which obtain data from various sources and process the data in order to generate all sorts of enriched data visualizations like annotated maps, for supporting the integration of social Web sites, etc.

This type of service is generally described using plain, unstructured HTML, except for a few that use the XML-based format WADL [40]. As a consequence, despite their popularity, the development of Web applications that integrate disparate services in this manner suffers from a number of limitations similar to those previously outlined for Web services with the increased complexity that most often no machine-processable description is available. Discovering services, handling heterogeneous data, and creating service compositions are largely manual and tedious tasks that end up in the development of custom tailored solutions that use these services.

In the light of the popularity and limitations of this technology, research on semantic Web services has recently focused on trying to support further automation within the life-cycle of applications based on Web APIs and RESTful services. MicroWSMO is a microformat supporting the semantic annotation of RESTful services and Web APIs in order to better support their discovery, composition and invocation. Microformats offer means for annotating human-oriented Web pages in order to make key information machine-processable [41].

```
<div class="service" id="s1"><h1>happenr API</h1>
  <span class="label">Happenr </span>has two main methods to call
    "getEvents" and . .
  <p>All operations should be directed at http://happenr.3scale.net/</p>
  <h2>Example usage</h2>
  <span class="address">
    http://happenr.3scale.ws/.../getEvents.php?user_key=xxx
  </span>
  <p>where the userkey is the key issues with the signup you made.</p>
  <div class="operation" id="op1">
    <h2><span class="label">getEvents </span>Method</h2>
    <span class="input"><h3>username</h3>
      <p>Your username that you received from Happenr ...</p>
      <h3>password</h3>
      <p>Your password that you received from Happenr ...</p>
      <h3>eventid</h3>
      <p>The id of the event.</p>
    </span>
  </div>
</div>
```

Listing 7. Example of an hRESTS annotation.

MicroWSMO builds upon hRESTS (HTML for RESTful services) [42]. hRESTS enables the creation of machine-processable Web API descriptions based on available HTML documentation. hRESTS provides a number of HTML classes that allow one to structure APIs descriptions by identifying services, operations, methods, inputs, outputs, and addresses. It therefore supports, by simple injections of HTML code within Web pages, the transformation of unstructured HTML-based APIs descriptions into structured service descriptions similar to those provided by WSDL. Listing 7 shows an example of an hRESTS description.

With the hRESTS structure in place, HTML service descriptions can be annotated further by including pointers to the semantics of the service, operations and data manipulated. To this end MicroWSMO extends hRESTS with three additional properties, namely *model*, *lifting* and *lowering* that are borrowed from SAWSDL and have the same semantics explained earlier. Although, not strictly necessary, MicroWSMO adopts the WSMO-Lite ontology as the reference ontology for annotating RESTful services semantically. By doing so, both WSDL services and RESTful services annotated with WSMO-Lite and MicroWSMO respectively, can be treated homogeneously. See Listing 8 for an example of a MicroWSMO annotation.

Like WSMO-Lite, MicroWSMO incorporates four main types of semantic annotations for services: functional semantics, nonfunctional semantics, behavioural semantics and information model semantics. Functional and nonfunctional semantics are to be provided through model references on the service. Behavioral semantics are included as model references on the operations. Finally, information model semantics are captured on the input and output messages of operations. The reader is referred to the WSMO-Lite description for further details [38].

```

<div class="service" id="s1"><h1>happenr API</h1>
  <a rel="model" href="http://example.com/events/getEvents">
    <span class="label">Happenr </span>has two main methods to call
    "getEvents" and . .
  </a>
  <p>All operations should be directed at http://happenr.3scale.net/</p>
  <h2>Example usage</h2>
  <span class="address">
    http://happenr.3scale.ws/webservices/getEvents.php?user_key=xxx
  </span>
  <p>where the userkey is the key issues with the signup you made.</p>
  <div class="operation" id="op1"><h2>
    <span class="label">getEvents </span>Method</h2>
    <span class="input">
      <h3>
        <a rel="model"
          href="http://example.com/data/onto.owl#Username">username</a>
        (<a rel="lowering"
          href="http://example.com/data/event.xsparql">lowering</a>)
      </h3>
      <p>Your username that you received from Happenr ...</p>
      <h3><a rel="model"
        href="http://example.com/data/onto.owl#Password">password<a
        (<a rel="lowering"
          href="http://example.com/data/event.xsparql">lowering</a>)
      </h3>

```

Listing 8. Example of a MicroWSMO annotation.

### 2.1.2.4 SA-REST

SA-REST [43, 44] is an open, flexible, and standards-based approach to adding semantic annotations to RESTful services and Web APIs. SA-REST essentially borrows the idea of grounding service descriptions to semantic metamodels by using model reference type of annotations from SAWSDL. However, this is not all there needs to be since, like MicroWSMO, SA-REST does not start from a machine processable description of a RESTful service or Web APIs similar to WSDL files. In most of the cases, Web APIs are solely described in human-oriented HTML Web pages, which do not include elements that a machine could directly use for identifying services and their elements.

Consequently, SA-REST uses microformats as a means to embed semantic annotations within the Web pages describing RESTful services. In particular, it supports the use of GRDDL [45] and RDFa [46] which are both W3C Recommendations. The former offers a way for choosing any microformat and specifying a translation of the Web page into machine-processable text. The latter supports embedding RDF within XML, XHTML and HTML. The SA-REST specification, however, recommends using RDFa since it supports keeping the entire description of the service, may it be the human-readable text or the machine-processable RDF, within one single document.

SA-REST leaves up to the user as to how to embed the RDF triples within the text. They could therefore be spread across the document or clustered together. The triples embedded are such that the subject should be the URL at which the service is invoked. The predicate should be one of *sarest:input*, *sarest:output*, *sarest:operation*, *sarest:lifting*, *sarest:lowering*, *sarest:action* whereby *sarest* corresponds to the SA-REST namespace. The triple's object should be either a URI or a URL pointing to a resource, which in the case of *sarest:lifting* and *sarest:lowering* will be a transformation and for the others it will be an element from an ontology. Example annotations for Craigslist search service are illustrated in Listing 9.

```
<html xmlns:sarest="http://lsdis.cs.uga.edu/SAREST#">
...
<p about="http://craigslist.org/search/">
  The logical input of this service is an
  <span property="sarest:input">
    http://lsdis.cs.uga.edu/ont.owl#Location_Query
  </span>
  object. The logical output of this service is a list of
  <span property="sarest:output">
    http://lsdis.cs.uga.edu/ont.owl#Location
  </span>
  objects. This service should be invoked using an
  <span property="sarest:action">
    HTTP GET
  </span>
  <meta property="sarest:lifting"
content="http://craigslist.org/api/lifting.xsl" />
  <meta property="sarest:lowering"
content="http://craigslist.org/api/lowering.xsl" />
  <meta property="sarest:operation"
content="http://lsdis.cs.uga.edu/ont.owl#Location_Search" />
</p>
```

Listing 9. SA-REST annotations for Craigslist search service.

Pertinent research that builds upon SA-REST includes the faceted search of API documents [47] including a ranking algorithm called ServiUt where annotations are used to build a searchable, comprehensive classification of API documents.

## **2.2 Semantic Web Services infrastructure**

Alongside the conceptual models described in the previous section, there has been extensive development trying to exploit the conceptual models to automate some of the core tasks for handling Web services. In the remainder of this section some of the main systems are covered organised by framework—when there exist several tools and engines belonging to the same framework—and by the conceptual model they build upon. OWL-S related tools are covered first, then the main WSMO platforms are presented and finally the work around WSDL-S and SAWSDL is described.

### **2.2.1 OWL-S Tools and Engines**

Research focussing on tool development for OWL-S has largely taken place as separate elements or components developed by different research groups in a stand-alone basis rather than as a fully-fledged framework covering the entire life-cycle of semantic Web services. Among the main research and development around OWL-S, there has been effort devoted to implementing matchmaking engines, editors, execution environments or even (semi) automated composition engines. A number of these is described in the remainder of this section.

#### **2.2.1.1 Annotation**

One of the very first OWL-S editors is a plug-in for Protégé [48] which extended the ontology editor towards supporting the definition of semantic Web services based on OWL-S. It therefore benefits from the general ontology editing features Protégé provides and extends and includes OWL-S specific panes structured around the OWL-S subontologies capturing the service profile, the service model, and the service grounding. The editor includes additional features for supporting the management of Inputs, Outputs, Preconditions and Effects, as well as graphical support for defining the control-flow of processes based on OWL-S in-built constructs.

A second editor that is worth mentioning is ASSAM, which stands for Automated Semantic Service Annotation with Machine learning. ASSAM as opposed to most semantic Web services editors is a tool for the semi-automatic annotation of Web services [49]. In a nutshell, the tool assists the user in annotating Web services relying on two machine learning algorithms. The first algorithm aids in annotating datatypes in WSDL files by classifying services, their operations and the messages they exchange given an initial training set of previously annotated services. The second algorithm helps the user in mapping schemas for different yet related Web services. The resulting annotations can eventually be exported in OWL-S. Currently ASSAM constitutes one of the most advanced semantic Web services annotation tools, together with the research carried out by Sabou [50].

### 2.2.1.2 Matching

Work on service matching based on OWL-S has largely been based on the use of subsumption reasoning supported by OWL and related reasoners. Among the first service matchmakers that were developed in the area the DAML-S Matchmaker [51, 52] developed at Carnegie Mellon could be cited, which was used to develop an enhanced UDDI registry. The registry maps OWL-S profiles to UDDI defining specialised T-Models for OWL-S specific elements that cannot be mapped to UDDI directly such as inputs, outputs, for example. To support advanced matching, each advertisement is pre-processed in order to derive the corresponding UDDI advertisements out of OWL-S profile definitions and also to define the degree of matching with respect to ontology concepts using RACER [53] as the OWL reasoning engine. Through this simple approach the enhanced UDDI registry is able to support more advanced matching contemplating situations where requests are less specific than advertisements (plug in match), where requests are more specific than advertisements (subsume match) thanks to the use of subsumption reasoning. At querying time, the degree of matching is established depending on the number of intervening classes between requests and advertisements.

The OWL-S Web service Matcher [54] (OWLS-M), is another matchmaking engine. In this case, the engine carries out the service matchmaking in a four step process. The first two steps consider the input and output types of a service and performs subsumption reasoning between the requests and the known services. The third step takes into account the categorisation of the service itself and finally OWLS-M facilitates the use of custom elements addressing individual constraints or requirements (e.g., quality of service) to filter the results.

The OWL-S Service Matchmaker (OWLS-MX) [55] is a hybrid OWL-S service matchmaker that exploits logic-based techniques and Information Retrieval syntactic similarity measures for approximate matching. In a nutshell the OWLS-MX takes any OWL-S service as a matchmaking query, and returns an ordered set of relevant services including the degree of match and syntactic similarity. Matchmaking is carried out by processing every input and output of the services contemplated, doing subsumption checking and similarity comparisons with the inputs and outputs specified in the request.

OWLS-MX provides five different matching filters, three of which are logic-based namely *Exact*, *Plug in*, and *Subsumes*, and two of which are hybrid, namely *Subsumed-by*, and *Nearest-neighbour*. Combined, they allow users to carry out approximate service matchmaking with varying degrees of match ranging from exact logic-based matching to approximate neighbourhood based on syntactic similarity. The user is provided the means for specifying a threshold for syntactic similarity that establishes that any failure in subsumption checking will be tolerated if the syntactic similarity obtained is beyond the threshold.

In addition to the work mentioned so far, some researchers have approached service matchmaking in OWL-S purely from a Description Logics (DL) perspective, perhaps the most notable example being [56]. In a nutshell the idea is to treat both service advertisements and queries as concepts and then consider the subsumption relation between both. Based on this approach, Li and Horrocks [56] have been able to develop a generic service matchmaking engine for DAML-S using RACER [53]—a general purpose DL reasoner. Despite the fact that treating advertisements as concepts is somewhat counterintuitive, the

authors showed that there is no noticeable expressivity impact and conversely there is a clear processing advantage. The matchmaking engine developed in this manner is able to provide the typical range of matching degrees namely exact, plug-in, subsume, and fail (called disjoint in this case) and an additional one—intersection—when the advertisement and the query are almost but not completely incompatible.

### **2.2.1.3 Orchestration**

At the core of OWL-S framework is the OWL-Virtual Machine (former DAML-S VM) [52, 57], a general purpose Web service client which relies on the OWL-S process model and grounding to support interaction between OWL-S Web services. At the core of the OWL-S Virtual Machine lies an OWL-S Processor which is supported by an OWL Inference Engine that combines Jena [58] and the Jess engine [59] for including rules support. The OWL-S Processor is in charge of driving the interaction with services and therefore implements the operational semantics of the OWL-S Process Model by means of rules and relying on a Web service invocation module for interacting with remote services.

### **2.2.1.4 Composition**

The OWL-S Virtual Machine was also utilised to explore the possibility for automatically composing and executing OWL-S processes. The research which was carried out by extending an existing planning engine, although preliminary, highlighted to an extent the potential and also the outstanding challenges that had to be tackled in this area. In the light of these issues, and the inherent computational complexity of planning algorithms, Sirin et al. developed a semi-automated service composition tool [60]. Essentially the tool allows humans to search and filter services based on the DAML-S Matchmaker techniques allowing them to choose the most appropriate services to compose new processes. Thus, by delegating the core decisions to humans the tool reduces the inherent computational complexity of process composition while it leverages semantic descriptions to better support the creation of processes. The resulting processes can subsequently be exported as OWL-S Composite Processes.

In [61] Traverso and Pistore present one of the main planning solutions devised for supporting the automated composition of services described in OWL-S. Their approach is able to deal with nondeterminism, e.g., the fact that a service invocation may return a result or an error, partial observability accounting for the fact that one can only observe communications with services and not their internal variables, as well as complex goals. The result of the composition is an executable BPEL process. In a nutshell, the solution they propose is based on the translation of OWL-S process models into state transitions systems and subsequently applying knowledge level planning algorithms that aim to fulfil the requested composition goal. In addition to the actual formalisation and application of AI planning algorithms to OWL-S, perhaps the most notable outcome of their research is the fact that they illustrate how the use of semantic annotations can help improve the performance of Web service composition.

## **2.2.2 WSMX**

Research and development on WSMO has dedicated substantial efforts to developing engines and frameworks able to interpret WSMO descriptions and use them for

achieving a greater level of automation during the life-cycle of service-oriented applications. The Web Service Execution Environment (WSMX) [15] is, together with IRS-III described in the next section, one of the reference implementations of WSMO and also of Semantic Execution Environments (SEE) being standardised within OASIS [13].

WSMX is a component-based execution environment for WSMO that aims to support the discovery, composition, mediation, selection, and invocation of Web services based on a set of user's requirements. The main effort associated with WSMX has been on defining the mandatory functional services that need to be used within each of these particular tasks as well as their interfaces.

WSMX has been defined as a layered architecture depicted in Figure 4, whereby at the bottom layer reside formal languages for representing semantic information (WSML in this case) and the machinery for reasoning and storing this knowledge. The middle layer provides brokering facilities necessary for manipulating semantic Web services descriptions. This layer therefore comprises components for discovery, selection, data and process mediation, choreography, orchestration, grounding, and transport handling. The top layer is the interface to external applications and users and is therefore the layer where domain ontologies, applications and developer tools reside. Finally, a vertical layer across all three layers is in charge of managing the execution management as well as security and authentication aspects.

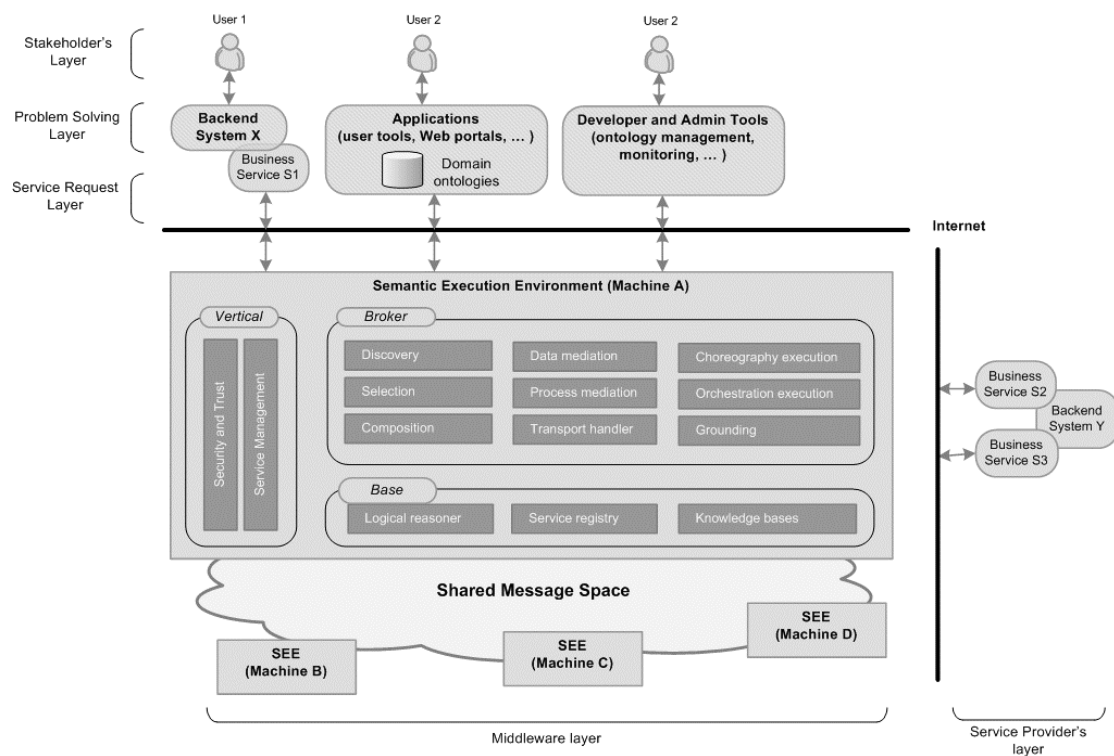


Figure 4. WSMX Architecture (taken from the OASIS Semantic Execution Environment Technical Committee).

The degree of development and refinement for each of these components is quite heterogeneous since the main focus was put on the definition of a generic architecture and the different execution scenarios, such as goal achievement, service selection, etc. In the remainder of this section, the main aspects behind

some of the core components are introduced. The reader is referred to [15] for more concrete details on each of these and the underlying approaches adopted.

### 2.2.2.1 Annotation

As previously introduced WSMX is based on WSMO for which a specific family of representation languages, WSML, was defined. SWS descriptions processed by WSMX, as well as the respective domain ontologies are expressed in WSML terms. As a consequence substantial efforts have been devoted to providing fully-fledged development environments able to support users in defining WSMO entities including Ontologies.

The two main frameworks developed, were WSMO Studio [62] and the Web Service Modeling Toolkit (WSMT) [63]. The features provided by both eclipse-based frameworks are quite similar. They include support for modelling all four top-level elements of WSMO as well as visualization support for ontologies. Also, they both integrate WSML reasoners so that developers can validate ontologies or evaluate queries over knowledge bases. Finally, they include support for interacting with Ranking and Selection engines and there is also support for creating mapping definitions in order to define Mediators.

### 2.2.2.2 Discovery and Matching

In Section 2 a somewhat singular definition for *discovery* was introduced, which contrasts with the functionality-oriented view often adopted by researchers in the field. This definition is based on an understanding of Discovery anchored on the distinction between services as business entities and Web services as computational entities that allow clients to interact with providers in order to benefit from (business) services. This distinction, which has been highlighted in several occasions by other researchers [11, 64], was investigated in WSMX.

Driven by this understanding of discovery and by the core notions of WSMO, WSMX proposes a conceptual model for service discovery (as opposed to Web service discovery) that approaches this endeavor as a *heuristic classification* task [65]. Heuristic classification is a problem solving method that was identified as being widely applied for carrying out tasks such as *classification* and *diagnosis*. It essentially separates the task into three main steps: *abstraction*, *matching* and *refinement*. Abstraction is in charge of abstracting the essential features of a given case or situation. The second step is in charge of *matching* the abstracted features with prototypical cases. Finally, refinement proceeds towards a final solution by trying to explain all the particularities of the given case given all the potential matches identified.

WSMX views discovery as a process starting from some desires out of which a set of Goals are *abstracted*. Once the Goals have been identified, they are matched against known Web Services (what they call Web service discovery) and finally each of the Web services matched is checked to determine whether they can fulfill the original desires (what they call service discovery).

Most of the implementation work on discovery in WSMX has focused on one of the steps of the conceptual model described above, namely Web service discovery, that is the matching of abstract Goals to Web Services, leaving the abstraction and refinement processes somewhat aside. Among the work so far it is worth mentioning the theoretical work exposed in [15] tackling the problem

with different expressivity ranging from simple keywords up to advanced Web service discovery based on rich Web services descriptions and transaction logic.

It is also worth mentioning the research carried out Stollberg et al. [66], which, as opposed to most of the work around SWS matching, focused on achieving performance and scalability at runtime rather than on improving accuracy. This work is based upon an extension of the WSMO model that distinguishes between the notion of *Goal Templates* and that of *Goal instances*, which is inspired by the treatment of Goals as meta-classes in IRS-III, which is described in more detail in Section 2.2.3. Goal Templates are used for generating a Semantic Discovery Caching graph that organises Goal Templates in a hierarchy based on their similarity from a functional perspective. Then, at runtime this caching graph is exploited in order to minimise the discovery time even for large repositories of Web services.

Finally, Glue [67] is another implementation of Web service discovery for WSMX. Glue makes use of a number of extensions to WSMO to support its objectives. First, Glue's model defines the class of Goals and the class of Web services in a similar fashion to that of IRS-III. Second, the GG-Mediators (see Section 2.1.1.2) are used for automatically generating a set of goals semantically equivalent to the one expressed by the requester but expressed with a different form or using different ontologies. Third, WG-Mediators are used for evaluating the matching between Goals and Web Services and OO-Mediators are explicitly incorporated in the discovery process in order to deal with semantic mismatches. Finally, the discovery process is extended in order to include both the matchmaking and also the discovery of Mediators that could resolve heterogeneities.

### **2.2.2.3 Ranking and Selection**

As part of WSMX a ranking and selection engine has been developed. The engine is able to take into account non-functional annotations associated with Web Services in order to, once they have been selected as matching a Goal, rank them accordingly. The engine contemplates general annotations such as the creator of the annotation and the publisher as well as non-functional properties of the service such as its availability or its price.

Together with the engine, there are a set of ontologies enabling the capture of non-functional properties. Among the ontologies provided, the system includes conceptualisations covering location, time, price, availability, trust, and quality of service. On the basis of these ontologies, one can define non-functional properties for services using logical expressions and at selection time the engine can check the values for the matching services in order to rank them. The current implementation takes as input the user preferences expressed as logical expressions as well as the matching services and makes use of a multi-criteria algorithm to rank the services accordingly.

### **2.2.2.4 Orchestration**

Research on orchestration definition in WSMO and consequently on its interpretation has essentially focussed on two orthogonal problems. On the one hand effort has been devoted to minimising the complexity for modelling workflows while supporting the mapping into formal representations that can support reasoning about process behaviour. In [31] Norton et al. propose a 3-

level approach to orchestration definition which provides a layer based on UML Activity Diagrams for supporting process modelling using state of the art editors that are well known by software developers. The process modelling constructs are defined by the Cashew workflow language which provides support for block-oriented compositions largely inspired by OWL-S. Processes represented in Cashew can be interpreted directly or they can be transformed into Abstract State Machines for this purpose.

On the other hand other research has focussed on providing a solution with existing workflow standards while supporting the application of semantics to support process adaptability among others. The work carried out in this respect is based on the use of BPEL for defining workflows that can directly refer to Web services or that can support the inclusion of Goals so that at runtime and given the concrete situation, the most promising service can be used [68].

### **2.2.2.5 Mediation**

Previously, the fact that one of the core features of WSMO is the central role played by mediation was introduced. Mediation is brought into WSMO models by means of four main constructs, namely OO-Mediators, GG-Mediators, WG-Mediators and WW-Mediators.

Data mediation in WSMX takes place at two main stages [15]. The first one concerns data representation and is supported by the inclusion of lifting and lowering mechanisms. On the basis of these definitions, at runtime WSMX is able to transform data from its internal semantic representation in WSML to and from the XML-base syntactic representation used by Web services. Doing so therefore avoids issues between heterogeneous XML representations by using an intermediate transformation into WSML. The second stage takes care of semantic heterogeneity by means of an abstract mapping language supporting the declarative specification of mappings between two different ontologies. The language is supported by the WSMT modelling environment through a semi-automated editor. The resulting mappings can be interpreted at runtime by an engine in order to perform the appropriate transformations.

Work on process mediation aims at mediating between two communicating Web services so that differences in the kinds of messages and their order does not prevent them from communicating effectively. Research on process mediation in WSMX has produced a categorisation of the different mismatches that can be found and a process mediator that can resolve those that are deemed solvable. The categorisation includes solvable mismatches such as stopping unexpected messages, splitting a message, combining messages or inverting messages to name a few. The unsolvable mismatches occur when some piece of information expected by one of the actors cannot be sent by the other or when both actors expect data from the other. In a nutshell, the prototype developed solves these process mediation problems by keeping track of both Web service choreographies, checking the messages expected and the data available in both sides so as to forward the necessary information to the other Web service when appropriate. Indeed the process mediator relies on existing data mediation information in order to solve mismatches like the splitting or merging of messages.

### 2.2.3 IRS-III

The Internet Reasoning Service (IRS) project carried out at the Knowledge Media Institute of The Open University has the overall aim of supporting the automated or semi-automated construction of semantically enhanced systems over the Internet. IRS-I supported the creation of knowledge intensive systems structured according to the Unified Problem-solving Method Development Language (UPML) [27]; IRS-II [69] integrated the UPML framework with Web service technologies. IRS-III extended this framework incorporating the WSMO conceptual model, and providing additionally a set of tools to support the SWS developer at design time in creating, editing and managing a library of semantic descriptions as well as publishing and invoking semantic Web services [70].

IRS-III is a broker-based platform that mediates between clients and service providers allowing each of them to adopt the most convenient means for representing their perspective, while supporting an effective interaction. To this end IRS-III is entirely underpinned by ontological representations facilitating knowledge sharing by machines and humans.

IRS-III uses its own ontology representation language, OCML [71]. The OCML language combines a frame system with a tightly integrated forward and backward chaining rule system and includes constructs for defining: classes, instances, relations, functions, procedures and rules. Additionally, procedures and functions can be attached to Lisp code. This feature allows ontologies related to service descriptions to be attached to the IRS service invocation mechanism thus enabling inferred values to reflect the state of a deployed service (e.g. to retrieve a current exchange rate). In order to ensure its interoperability, OCML contains import/export facilities to RDF(S) and WSML and import facilities for OWL [70].

IRS-III is based on a service ontology which influenced as well as it was informed by Web Services Modelling Ontology presented earlier in this chapter, and is therefore largely aligned with it. Hence, the IRS-III service ontology contemplates *Goals*, *Web Services* and *Mediators* as the main concepts, and Web Services have *Choreographies* and *Orchestrations* specifying how to communicate with them and how they combine other services to achieve their functionality respectively. Still, there exist fundamental differences between both ontologies that are worth outlining.

First and foremost, the IRS-III service ontology uses meta-classes for the top-level SWS concepts (Goal, Mediator and Web Service). As a consequence, IRS-III components can reason over the top-level concepts within the service ontology as first class entities, which is something not directly possible in WSML-based representations of WSMO where special keywords as opposed to ontological entities are used. On the basis of these concepts, IRS-III allows users to define the required Goals, Mediators and Web services as subclasses of the corresponding WSMO concepts rather than as instances, supporting the creation of reusable service descriptions and taxonomic structures that can be exploited for reasoning and indexing purposes.

Additionally and based on the previous distinction, the invocation of semantic Web services are kept as instances. When IRS-III receives a client request, instances of relevant goals, mediators and Web services are created to capture the current invocation context. Doing so paves the way for carrying out

thorough analyses and monitoring of the invocations, as well as allowing the implementation of caching mechanisms.

Finally, in the interest of simplifying the definition of Goals and Web Services the service ontology incorporates explicit input and output role declarations. The declared input and output types are imported from domain ontologies. This feature enables developers to view Goals and Web Services as 'one-shot' invocable entities thus minimising the need to consider complex interactions when appropriate.

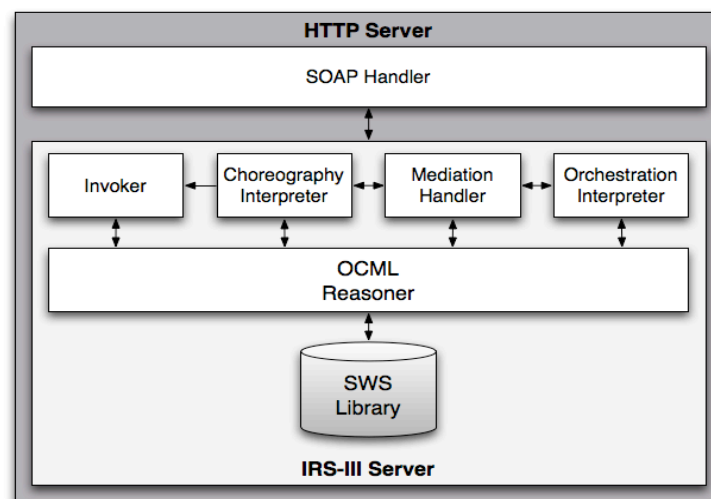


Figure 5. Architecture of IRS-III.

One distinctive feature of IRS is the support it provides for the seamless publishing of services. Indeed, users quite often will have an existing system functionality, which they would like to be made available as a service, but have no knowledge of the tools and processes involved in turning a stand-alone program into a Web service. To cater for this IRS-III provides support for 'one click' publishing of stand-alone code (currently Java and Lisp), as well as Web services from a WSDL description or from the URI (i.e., a HTTP GET request) of a web application.

Significant effort has been devoted towards supporting the interoperability between IRS-III and other semantic Web services frameworks. In particular, IRS-III has an OWL-S import mechanism [72] and is interoperable with WSMO implementations (e.g., WSMO Studio [62] and WSMX [15]) through a common API [13] inline with the standardisation within OASIS of the Semantic Execution Environment.

From a technical perspective, as can be seen in Figure 5, the *IRS-III Server* builds upon an *HTTP Server* written in Lisp, which has been extended with a SOAP handler. At the heart of the server is the *SWS Library*, where the semantic descriptions associated with Web services are stored using OCML as the representation language. The library is structured into domain ontologies and knowledge models for Goals, Web Services and Mediators as described previously.

### 2.2.3.1 Annotation

IRS-III provides its own development environment called the IRS Browser which supports users in defining WSMO entities and uploading them into the IRS

directly. The browser additionally supports the direct invocation of Web Services or Goals. Furthermore, based on the work carried out on interoperability between Semantic Execution Environments, IRS-III implements the APIs defined which therefore ensures its interoperability with existing environments like WSMT and WSMO Studio introduced earlier. It is therefore possible to use both editing environments to retrieve and store any WSMO entity to and from the IRS, and to use it for invoking Goals directly.

### 2.2.3.2 Ranking and Selection

IRS-III provides capability-based invocation allowing users to request a Goal to be achieved by instantiating a Goal definition with concrete values (e.g., Book transportation from *London* to *Manchester* for *next Tuesday morning*). Upon reception of a Goal request, IRS-III explores its library of known services in order to identify all the Web services that are suitable, and then select the one that is considered to be the best.

Research on IRS-III has therefore not focussed on Web services discovery by crawling the Web or existing UDDI registries. Instead research has been concentrated on supporting an effective ranking and selection of suitable services based on a variety of criteria. To this end, IRS-III leverages the WG-Mediator definitions that link Goals to Web Services to identify suitable and directly invocable services and the existing hierarchies of Goals to provide more refined or relaxed solutions should a perfectly fitting service not be available or known.

This simple selection mechanism has been extended towards supporting more refined mechanisms based on arbitrary criteria over non-functional parameters. This work is based on the application of a domain-independent library of Heuristic Classification [65] Problem-Solving Methods implemented in OCML [73]. As mentioned earlier, Heuristic Classification is a method for classifying entities that “*relates data to a pre-enumerated set of solutions by abstraction, heuristic association, and refinement*” [65]

The library provides a task ontology that defines the kinds of knowledge necessary to classify things according to a *taxonomy of classes* given a set of *observed features*. Based on raw information, classification criteria as well as additional domain-specific knowledge including heuristics the library is able to automatically classify the situation at hand with respect to the target taxonomy.

This library has been used for developing a trust-aware service selection mechanism [74] that takes into account *user preferences*, *trust requirements* and *trust guarantees* promised by service providers in order to choose those services that best meet the trust requirements out of those that are functionality suitable. To this end, the solution implemented provides two main classification methods: a *single solution classification* method based on a hill-climbing algorithm; and an *optimal classification* method based on exhaustive search.

Although this selection mechanism is solely based on trust information, work has been carried out more recently in order to benefit from the genericity of the approach [75]. In particular, the aforementioned techniques have been extended towards supporting a context-aware selection of services that can take into account observed/monitored features of services and users across arbitrary and composable dimensions (e.g., time, trust, location), so as to refine the set of functionally suitable services into the most contextually relevant subset.

### 2.2.3.3 Choreography

IRS-III offers capability-based invocation, that is the ability to invoke services based on the client request expressed as a Goal. IRS-III then acts as a broker: finding, composing and invoking appropriate Web services in order to fulfil the request. In this way, IRS maintains a clean separation between users and services but still supports the invocation of services. Choreography addresses the problem of communication between a client and a Web service. Since the IRS-III acts as a broker the choreography work in IRS-III is focused on the communication between the IRS-III and the relevant deployed Web services.

Goal achievement consists of a number of discrete steps, in which, at any given point of the execution, the next action performed will depend upon the current state. IRS-III, adopts the Abstract State Machine (ASMs) formalism [30] to represent the IRS-III interaction with a client (choreography) or providing Web services (orchestration).

A choreography is described in IRS-III by the declaration of a *grounding* and a set of *guarded transitions*. The grounding specifies the conceptual representation of the operations involved in the invocation of a Web service and their mapping to the implementation level. More specifically, the grounding definitions include the operation name and bindings for input and output data. Guarded transitions can be seen as ASM transition rules. These represent the interaction between IRS-III and the Web service and are applied when executing the choreography. This model is executed at a semantic level when IRS-III receives a request to achieve a Goal.

The IRS-III service ontology defines a set of choreography specific primitives, which can be used in transition rules. The primitives provided include *init-choreography*, *send-message*, *receive-message*, *send-suspend*, *receive-suspend*, *receive-error* and *end-choreography*. These primitives provide an easy to use interface to control a conversation between IRS-III and a Web service. Developers are also able to include any relation defined within the imported ontologies in guarded transition specifications.

The IRS-III uses the OCML forward-chaining-rule engine to execute a choreography. This means that rules belonging to a choreography are fired according to the state taking into account inferences at the ontological level. During communication with a Web service, IRS-III provides lifting and lowering mechanisms mapping the ontological level descriptions to the XML-based representations used by the specific Web service invoked. The actual lifting and lowering definitions are based on Lisp macros mapping XPath expressions to ontological elements.

### 2.2.3.4 Orchestration

Research and development on orchestration in the IRS has focussed on supporting the creation and visualization of orchestrations by developers [31] and on executing them. In IRS-III, the orchestration is used to describe the model of a composed Web service. At the semantic level the orchestration is represented by a workflow model expressed in OCML. The main characteristics of the IRS-III orchestration model are:

- **Goal centric composition** - the basic unit within compositions is a Goal. The orchestration model thus provides control and data flow constructs over sets of Goals.

- **Invocation is one-shot** – it is assumed that when a Goal is invoked the result is returned and there is no direct interaction between any of the underlying Web Services involved. IRS-III acts as a broker and therefore a dialog between two Web Services becomes a pair of IRS-III to Web Service choreographies.
- **Orchestration is layered** - within the IRS-III there are a number of layers each of which support a specific set of activities [31].

Based on the aforementioned principles, IRS-III provides a set of control flow primitives, namely *orch-sequence*, *orch-if*, *orch-repeat*, *orch-get-goal-value*, *orch-return*, which have been implemented and mapped into ASMs. At runtime IRS-III provides the capability to interpret orchestrations defined using these constructs by resolving Goals to concrete Web Services and dealing with dataflow mismatches on the basis of Mediators that are described in more detail next.

### 2.2.3.5 Mediation

IRS-III provides support for data mediation by supporting the modelling of specialized mediators which provide a mediation service or declarative mappings for solving different types of conceptual mismatches. The mediation handler interprets each type of mediator accordingly during selection, invocation and orchestration. The mediator models are created at design time and used at runtime.

In IRS-III application developers can associate a relation mapping to OO-Mediators (i.e., Mediators between ontologies) in order to map between instances of different ontologies using declarative expressions in OCML. Additionally, the remaining kinds of mediators (WW-Mediator, WG-Mediator and GG-Mediator) can be associated with a mediation function encapsulated as a standard semantic Web service in order to perform transformations between inputs and outputs. These mediators can provide mediated data-flow between a Goal and a Web Service, and between Web Services or Goals by relying in turn on specific OO-Mediators that declaratively specify the mappings.

Mappings specifications are expressed in OCML based on three main primitives [70, 71]:

- *Maps-to*: a relation created internally for every mapped instance.
- *Def-concept-mapping*: generates the mappings, specified with the *maps-to* relation, between two ontological concepts.
- *Def-relation-mapping*: generates a mapping between two relations using a rule definition within an ontology. As OCML represents concept attributes as relations, this primitive can be used to map between input and output descriptions.

### 2.2.4 METEOR-S

The METEOR-S project [76] was carried out at the LSDIS Laboratory at the University of Georgia, with follow on work at the Kno.e.sis Center at Wright State University. METEOR-S supports and leverages semantics through the complete life-cycle of semantic Web processes, encompassing the annotation, publication, discovery, dynamic binding or composition, and enactment of Web services. The distinguishing characteristic of the research undertaken in the METEOR-S project is

the strong coupling with existing Web services standards [72]. In fact, the philosophy of METEOR-S is to incrementally extend pre-existing standards with semantics so as to better support the discovery, composition and enactment of Web services. This contrasts with the top-down approaches based on OWL-S and WSMO.

The METEOR-S project has tackled the semantic annotation of Web services, the semantics-based discovery of Web services and their composition, which also encompasses data mediation. The remainder of this section will focus on the specific approaches adopted in METEOR-S for each of these research topics. First, the METEOR-S Web service Annotation Framework (MWSAF) [77] which contributed to a big extent to the definition of WSDL-S is presented. Second, the focus is on the METEOR-S Web Service Discovery Infrastructure (MWSDI) [78]. Next, the METEOR-S approach to data mediation [79] is described and finally the METEOR-S Web Service Composition Framework (MWSCF) is characterised [80].

#### **2.2.4.1 Service Annotation**

At the centre of METEOR-S project is MWSAF [77], a framework for the semi-automatic annotation of Web services. These annotations address four different aspects of Web services' semantics. First of all, MWSAF supports the inclusion of annotations about the semantics of the inputs and the outputs of Web services. Secondly, the annotation framework supports the definition of functional semantics, i.e., what the service does. Thirdly, MWSAF enables the inclusion of execution semantics to support verifying the correctness of the execution of Web services. Finally, the framework incorporates information regarding the quality of service, such as performance or costs associated to the execution of Web services.

Initial research on the framework was devoted to supporting the semi-automatic annotation of XML Schemas part of Web services definitions. This work is based on the transformation of both XML Schemas and ontologies into a common representation format called SchemaGraph [77] in order to facilitate the matching between both models. Once the Ontologies and XML Schema are translated into this common representation, a set of matching algorithms can be applied to (semi) automatically enhance the syntactic definitions with semantic annotations.

In a nutshell, the matching algorithm computes a *match score* between each element of the WSDL SchemaGraph and the ontology SchemaGraph. This score takes into account the linguistic and the structural similarity. After all the match scores have been computed, the “best” matching element is chosen by taking into account both the match score and the specificity of the concepts. Finally, a global matching average is computed to help in selecting the best overall match between the Web services and ontologies. Further details about the algorithm can be found in [77].

MWSAF is composed of three main components: an ontology store, the matcher library and a translator library. The first component stores the ontologies that will be used for annotating the Web services. The matcher library provides different algorithm implementations for linguistic and structural matching between concepts and Web services elements. Finally the translator library consists of the programs used for generating the SchemaGraph representation for ontologies and Web services.

MWSAF assists users in annotating Web services by browsing and computing the concordance between domain models and the Web service elements. The last step in the annotation process is their representation for future reuse and automated processing. To cater for this the METEOR-S project makes use of SAWSDL, which was presented in Section 2.1.2.1.

#### **2.2.4.2 Matchmaking**

UDDI and the Universal Business Registry (UBR) are the main industrial efforts supporting the automation of Web services matchmaking. The METEOR-S Web Services Discovery Infrastructure (MWSDI) attempts to enhance existing Web services discovery infrastructure by using semantics [78]. MWSDI is a scalable infrastructure for the semantics-based publication and discovery of Web services.

MWSDI aims to provide unified access to a large number of third party registries. Thus, in order to provide a scalable and flexible infrastructure it has been implemented using Peer-to-Peer (P2P) computing techniques. It is based on a four-layered architecture which includes a Data layer, a Communications layer, an Operator Services layer, and a Semantic Specification layer. The Data layer consists of the Web services registries and is based on UDDI. The Communications layer is the P2P infrastructure which is based on JXTA. The Operator Services layer provides the semantic discovery and publication of Web services. Finally, the Semantic Specification layer enhances the framework with semantics.

MWSDI uses semantics for two purposes. Firstly, it uses the so-called *Registries Ontology* which stores registries information, maintains relationships between domains within MWSDI and associates registries to them. This ontology stores mappings between registries and domains so that finding Web services for a specific domain can be directed to the appropriate registries. Additionally the *Registries Ontology* captures relationships between registries so that searches can be made more selective on the basis of these relationships.

Secondly, MWSDI envisions including domain specific ontologies for registries, so that Web services can be annotated by mapping inputs and outputs to existing domain ontologies. The purpose of defining these mappings is to enable semantic discovery by allowing users to express their requirements as *Service Templates* which are expressed using concepts from the same ontology.

The semantic publication of services in MWSDI registries uses UDDI *tModels* for registering the domain ontologies and *CategoryBags* for categorizing WSDL entities according to one or more *tModels*. MWSDI provides both a manual and a semi-automatic mechanism for defining the mappings between WSDL elements and the concepts in the domain ontologies [78].

#### **2.2.4.3 Data Mediation**

The METEOR-S data mediation technique introduced the *lifting* and *lowering* mechanism for transforming from one representation to another. An overview of Service heterogeneities and the mediation challenges are outlined in Table 1. The *lifting* mapping indicates the conversion from the existing format to the common model whereas *lowering* mapping indicates the reverse conversion. An extensive discussion of the mediation mechanism is available in [75,81].

**Table 1. An outline of service heterogeneities and potential methods to overcome them.**

Heterogeneities / Conflicts	Examples - conflicted elements shown in color	Suggestions / Issues in Resolving Heterogeneities
<b>Domain Incompatibilities – attribute level differences that arise because of using different descriptions for semantically similar attributes</b>		
<b>Naming conflicts</b> Two attributes that are semantically alike might have different names (synonyms) Two attributes that are semantically unrelated might have the same names (homonyms)	<b>Web service 1</b> Student(Id#, Name) <b>Web service 1</b> Student(Id#, Name)	<b>Web service 2</b> Student(SSN, Name) <b>Web service 2</b> Book (Id#, Name)
<b>Data representation conflicts</b> Two attributes that are semantically similar might have different data types or representations	<b>Web service 1</b> Student(Id#, Name) Id# defined as a 4 digit number	<b>Web service 2</b> Student(Id#, Name) Id# defined as a 9 digit number
<b>Data scaling conflicts</b> Two attributes that are semantically similar might be represented using different precisions	<b>Web service 1</b> Marks 1-100	<b>Web service 2</b> Grades A-F
<b>Entity Definition – entity level differences that arise because of using different descriptions for semantically similar entities</b>		
<b>Naming conflicts</b> Semantically alike entities might have different names (synonyms) Semantically unrelated entities might have the same names (homonyms)	<b>Web service 1</b> EMPLOYEE (Id#, Name) <b>Web service 1</b> TICKET (TicketNo, MovieName)	<b>Web service 2</b> WORKER (Id#, Name) <b>Web service 2</b> TICKET(FlightNo, Arr. Airport, Dep. Airport)
<b>Schema Isomorphism conflicts</b> Semantically similar entities may have different number of attributes	<b>Web service 1</b> PERSON (Name, Address, HomePhone, WorkPhone)	<b>Web service 2</b> PERSON (Name, Address, Phone)
<b>Abstraction Level Incompatibility – Entity and attribute level differences that arise because two semantically similar entities or attributes are represented at different levels of abstraction</b>		
<b>Generalization conflicts</b> Semantically similar entities are represented at different levels of generalization in two Web services	<b>Web service 1</b> GRAD-STUDENT (ID, Name, Major)	<b>Web service 2</b> STUDENT(ID, Name, Major, Type)
<b>Aggregation conflicts</b> Semantically similar entities are represented at different levels of generalization in two Web services	<b>Web service 1</b> PROFESSOR (ID, Name, Dept)	<b>Web service 2</b> FACULTY (ID, ProfID, Dept)
<b>Attribute Entity conflicts</b> Semantically similar entity modeled as an attribute in one service and as an entity in the other	<b>Web service 1</b> COURSE (ID, Name, Semester)	<b>Web service 2</b> DEPT( Course, Sem, ... ..)

\* Interoperation between services needs transformation rules (mapping) in addition to annotation of the entities and/or attributes indicating their semantic similarity (matching).

#### 2.2.4.4 Composition

Semantic Composition of Web services in METEOR-S is supported by the METEOR-S Web Service Composition Framework (MWSCF) [80]. In a nutshell, the composition framework aims to increase the flexibility of Web services composition by making use of *Semantic Process Templates*. *Semantic Process Templates* define processes in terms of semantically defined activities. Using these *Semantic Process Templates* executable processes can be generated by binding the semantically defined activities to concrete Web services that conform to the activity specification.

MWSCF is composed of four components: the process builder, the discovery infrastructure (see previous section), XML repositories and the process execution engine. The process builder includes a graphical user interface for defining Semantic Process Templates and a process generator. The process generator retrieves ontologies, activity interfaces and process templates from

the XML repositories and uses MWSDI for discovering suitable Web services, in order to transform the templates into executable processes. The executable process definitions can then be handed to the process execution engine for the actual execution of the Web services composition.

In MWSCF *Semantic Process Templates* [82] are basically a set of Activities connected by means of Business Process Execution Language (BPEL) [18] control flow constructs. Activities can be defined with a varying degree of flexibility by using a specific Web service implementation, a Web service interface or a *Semantic Activity Template*. Specific Web service implementations can be specified for static compositions. Web service interfaces can be applied to gain some flexibility allowing diverse implementations of the same interface to be interchangeably executed. Finally, *Semantic Activity Templates* provide a greater degree of flexibility by defining activities semantically in terms of their inputs, outputs and functional semantics, e.g. preconditions and effects.

The creation of an executable process is a semi-automated activity performed at design-time where the user is assisted in refining the template with concrete Web services and dataflow. In order to do so, Web services that implement the specified Web service interfaces are retrieved from the XML Repository and the MWSDI is used for discovering suitable services when *Semantic Activity Templates* have been specified. After all the activities have been replaced by concrete Web services, the user can map Web service outputs to other service inputs in order to define the process dataflow. Once the explicit dataflow has been defined, the process generator creates the executable representation, which is a BPEL4WS process that can be executed in any BPEL execution engine.

METEOR-S research also addressed dynamic process composition [82], WS-agreement based partner selection [83] optimal process adaptation with constraints [84], and event identification [85].

### 3 Example Applications

So far the main principles underlying SWS were discussed along with the main conceptual models devised so far and the main frameworks able to deal SWS descriptions to achieve some automation for some of the tasks involved in the life-cycle of Web services. In this section a couple of examples illustrating some of the features provided by SWS and also showing how SWS applications can be developed are provided. This section is mostly for explanatory purposes and is therefore not to be taken as a proof of the capabilities of SWS nor as a recipe for constructing applications since this would require a level of detail and complexity that is beyond the scope of this chapter.

Although applications always present particular architectural characteristics and constraints inherited from the environment where they are deployed or the concrete requirements that need to be fulfilled, SWS applications typically are structured around the following four layers shown in Figure 6:

- *Legacy System layer*: consists of the existing data sources and IT systems available from each of the organizations involved in the integrated application.

- *Service Abstraction layer:* exposes (micro-) functionalities of the legacy systems as Web services, abstracting from the hardware and software platforms.
- *Semantic Web Service layer:* to set up an application, a set of application-specific SWS descriptions has to be provided. These descriptions are typically centrally stored within some repository for easy querying and retrieval through a more or less advanced discovery or service matchmaking machinery.
- *Presentation layer:* consist of a Web application accessible through a standard Web browser. The possible actions presented depend on the underlying services available. Typically input and output data is directly gathered from and presented to the user based on some semantic representation such as RDF(S), OWL or WSMML.

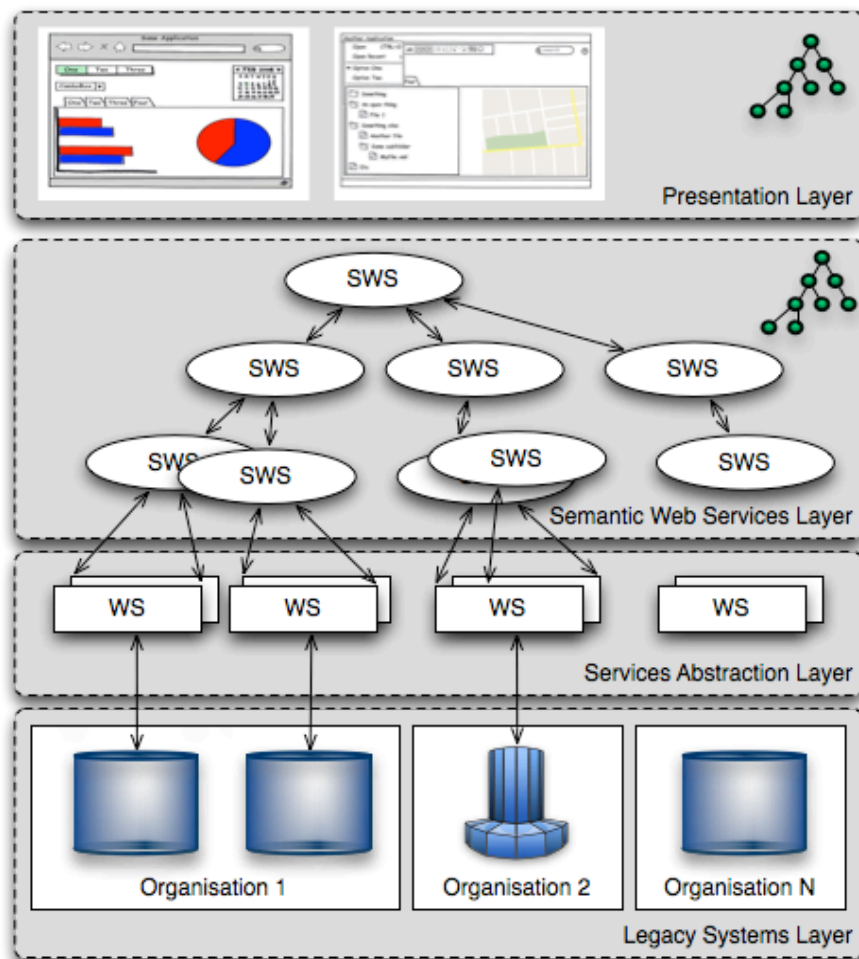


Figure 6. Typical architecture of SWS-based applications.

### 3.1.1 Applying Semantic Web Services in eGovernment

In the context of the European project DIP<sup>1</sup>, a close collaboration was established with the Essex County Council (ECC) - a large local authority in South East England (UK) comprised of 13 boroughs and containing a population of 1.3M - to

<sup>1</sup> <http://dip.semanticweb.org/>

deploy real-world applications in the e-Government domain based on SWS technologies [86]. During this collaboration, a framework designed around IRS-III was developed, tested and refined. The use cases show how SWS technology provides an infrastructure in which new services can be added, discovered and composed continually, and the organization processes automatically updated to reflect new forms of cooperation.

This section illustrates the development and application of SWS by describing two compelling use cases in the e-Government domain: Change of Circumstances and the Emergency Management System. The first application illustrates how Web services and ontologies can be leveraged in order to support a seamless integration and interaction within and between governmental administrative organisations to automatically handle the change of a citizen situation. In the second one, the developed application supports emergency planning and management personnel by retrieving, filtering, and presenting data from a variety of legacy systems to deal with a specified hazardous situation. This second scenario puts more emphasis on the dynamics of SWS technologies illustrating how systems can dynamically and transparently chose and orchestrate specific services in order to better deal with the situation at hand. Although the two examples are based on WSMO and IRS-III as the specific execution environments, most of the techniques and solutions illustrated herein could be supported using different modelling and execution frameworks.

### **3.1.1.1 Change of Circumstances**

Tiers of government – such as national, county, and district – largely operate autonomously, without central control of service provision. Additionally, they each have distinct viewpoints that may differ from that of general citizens. Therefore, integration and interoperability are significant requirements in the development of applications in the e-Government domain encompassing diverse administrative bodies.

Integration requires the assembly and transformation of processes needed to support specific user tasks into a single service with the corresponding back-office practices by integrating multiple governmental entities at different levels. Interoperability is a key issue in order to allow for data and information to be exchanged and processed seamlessly across the agencies involved. Interoperability is not only a technical issue but also a fundamental requirement to share and re-use knowledge between networks and administrations which often calls for the re-organisation of administrative processes. Interoperability problems therefore span technical issues such as interfaces, data formats and protocols; semantic issues concerning the exchange of information in an understandable way; and finally organizational issues regarding the modelling of business processes suited to how governmental agencies work internally.

The application was developed to solve a specific use case problem at Essex County Council (ECC). Whenever the circumstances in which a given citizen lives change, he/she might be eligible for a set of services and benefits provided by ECC and other governmental agencies together with public service providers. An example of such a change of circumstances is, if an elderly, partly disabled woman moves in together with her daughter. This is a change of circumstances for both, the mother and the daughter. For instance, the mother might no longer receive a “meals on wheels” service, whereas the daughter might get financial

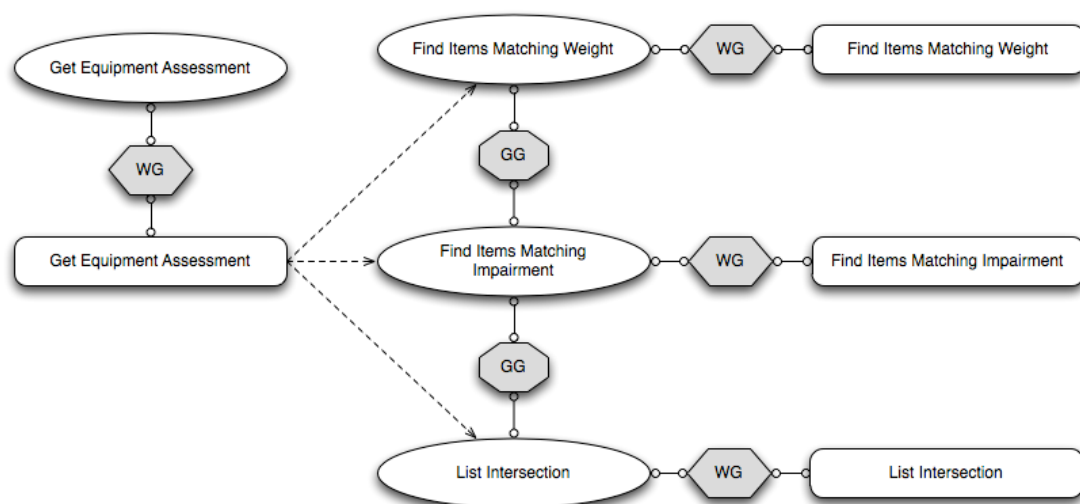
supporting for caring her mother. Starting from existing legacy systems, the aim is to provide integrated functionalities, such as: change of patient details within multiple legacy systems, change of patient pending equipment orders, a list of all services for a patient, the cessation of some services, and patient equipment assessment.

Generally, even very simple processes in a change of circumstances require the interaction of many different government agencies. Each agency has different legacy systems in place to keep track of citizen records, provide services, third-party service providers, etc. In the application discussed herein, the following two data sources provided by two different departments (at two distinct governmental levels) were considered:

- *Citizen Assessment (Community Care Department of the ECC)*: this relates to information about citizens registered in ECC for assessment of services and benefits (e.g. meals on wheels; house cleaning; in situ patient car, etc). This information is stored in the SWIFT database.
- *Order Equipment (Housing department of the Chelmsford District Council)*: this relates to information about equipment (e.g. stair lift, wheel chair, crutch, etc) which is provided to citizens registered in Essex. This information is stored in the ELMS database.

On top of the two legacy systems, a set of Web services that perform SQL queries and carried out certain basic operations for dealing with changes of circumstances were developed. In particular, eight Web services that interact with the SWIFT database and nineteen Web services interacting with the ELMS database were created.

A number of ontologies were defined covering the whole application domain and also providing the SWS descriptions required to achieve the goals. In particular, three domain ontologies were integrated with two ontologies specific for each of the back-end systems, and other three including the SWS descriptions which, being based on WSMO (see Section 2.1.1.2) therefore comprises Goals, Web Services and Mediators definitions.



**Figure 7. Structure of the SWS descriptions created for the Find Item ELMS by Impairment and Weight functionality.**

Each ellipse in Figure 7 represents a Goal that has to be accomplished by simple or integrated services—represented as rectangles. Specifically, the three

goals in the middle are accomplished by functionalities provided by Web services available. Such goals are automatically orchestrated to accomplish the main goal on the left, namely *Get Equipment Assessment*. The goal defines two inputs (weight and impairment) and one output (equipment list) expressed in terms of the domain ontologies for the application. At runtime – when the goal is invoked to be accomplished – the instances of the input classes are selected through the user interface of the application, while the result instances of the catalogue-data class are created on-the-fly by lifting them from the syntactic representation obtained from the results of Web service invocations.

Each Goal, includes a set of Web service(s) that can achieve the required objectives. In this case there is only one Web service per Goal so there is no need to apply any advanced matchmaking functionalities in order to choose the most suitable Web service to use (the next use case illustrates how these situations can be handled). Additionally, as explained earlier in Section 2.2.3, one WG Mediator connects every Goal to suitable Web services and takes care of data transformations between heterogeneous ontologies if necessary.

The Goal *Get Equipment Assessment*, shown in Figure 7, is a composite Goal whose objective is achieved by orchestrating three sub-goals. The actual orchestration definition for this example is a simple sequence as illustrated in Listing 10. Although the definition described in the listing is in OCML and based on the WSMO conceptual model the reader should note that a similar approach could be followed in OWL-S by defining a Service Process Model or using BPEL. Each sub-goal, invoked through the orchestration, is conversely directly accomplished by invoking the respective Web service.

```
(DEF-CLASS GET-EQUIPMENT-ASSESSMENT (GOAL) ?GOAL
  ((HAS-INPUT-ROLE :VALUE HAS-CITIZEN-WEIGHT
                   :VALUE HAS-CITIZEN-DISEASE
                   :VALUE HAS-CASE-WORKER-CODE)
   (HAS-OUTPUT-ROLE:VALUE HAS-SUITABLE-ITEMS-LIST)
   (HAS-CITIZEN-WEIGHT :TYPE NUMBER)
   (HAS-CITIZEN-DISEASE :TYPE DISEASE)
   (HAS-CASE-WORKER-CODE :TYPE NUMBER)
   (HAS-SUITABLE-ITEM-LIST :TYPE ITEM-LIST)))

(DEF-CLASS GET-EQUIPMENT-ASSESSMENT-INTERFACE-ORCHESTRATION
  ((HAS-BODY
    :VALUE ((ORCH-SEQUENCE
             GET-EQUIPMENT-ASSESSMENT-GOAL
             CHECK-EQUIPMENT-CASE-WORKER-GOAL)
           (ORCH-RETURN (ORCH-GET-GOAL-VALUE CHECK-EQUIPMENT-CASE-WORKER-GOAL))))))
```

**Listing 10.** Get Equipment Assessment goal definition.

This example application has illustrated how one can define simple semantic Web services by means of WSMO to implement cross-organizational integrated functionality, abstracting from the underlying legacy systems, keeping the autonomy of involved parties and covering multiple heterogeneous domains. If new systems need to be integrated, one can simply introduce the appropriate SWS descriptions and mediation facilities when mismatches occur. This example application contrasts with traditional database and Web services techniques that would necessitate that the different parties involved harmonise their database schemas and agree upon concrete service interfaces. The next application illustrates a slightly more complex use SWS technologies that shows how dynamic Web service matchmaking can provide additional flexibility to systems.

### 3.1.1.2 eMerges

In an emergency situation, multiple agencies need to collaborate, sharing data and information about actions to be performed. However, many emergency relevant resources are not available on the network and interactions among agencies or emergency corps usually occur on a personal/phone/fax basis. The resulting interaction is therefore limited in scope and slower in response time, contrary to the nature of the need for information access in an emergency situation.

Emergency relevant data is often spatial-related. *Spatial-Related Data (SRD)* is traditionally managed with the help of *Geographical Information Systems (GIS)*, which allow access to different layers of SRD such as highways, transportation, postal addresses index, land use, etc. GIS support decision making by facilitating the integration, storage, querying, analysis, modeling, reporting, and mapping of this data.

The prototype, called eMerges [86], is in effect a decision support system, which assists the end-user – currently the Emergency Planning Officer – in assembling information related to a certain type of event, more quickly and accurately. The application's user interface, shown in Figure 8, is based on Web standards. XHTML and CSS are used for presentation, while JavaScript is used to handle user interaction together with AJAX techniques to communicate with IRS-III. One of the main components of the interface is a map, which uses the Google Maps API to display polygons and objects (custom images) at specific coordinates and zoom level. Each time an object is displayed by a user at a particular location, the user interface provides a set of contextually relevant actions that can be carried out and the corresponding invocation form, should the user wish to invoke a particular action.

A number of ontologies were defined covering the application domain and also providing the SWS descriptions required to support emergency officers in handling extreme situations. In particular, three service-oriented ontologies were developed for describing the domains of the services provided by the back-end systems. These ontologies were integrated with user-oriented domain ontologies providing concepts for describing spatial aspects and context among other things. Finally, three ontologies captured the SWS descriptions, i.e., the Goals, Web Services and Mediators, for meteorological information SWS, emergency planning SWS and for dealing with distributed teams through a messaging system.

The purpose of the application ontologies is the aggregation of different data sources on, respectively, a representation, a cognitive and a spatial level. They allow the different data sources to be handled and presented in a similar way. Inversely to the lifting operations, *lowering operations* transform instances of aggregation ontologies into syntactic documents to be used by the server and client applications.

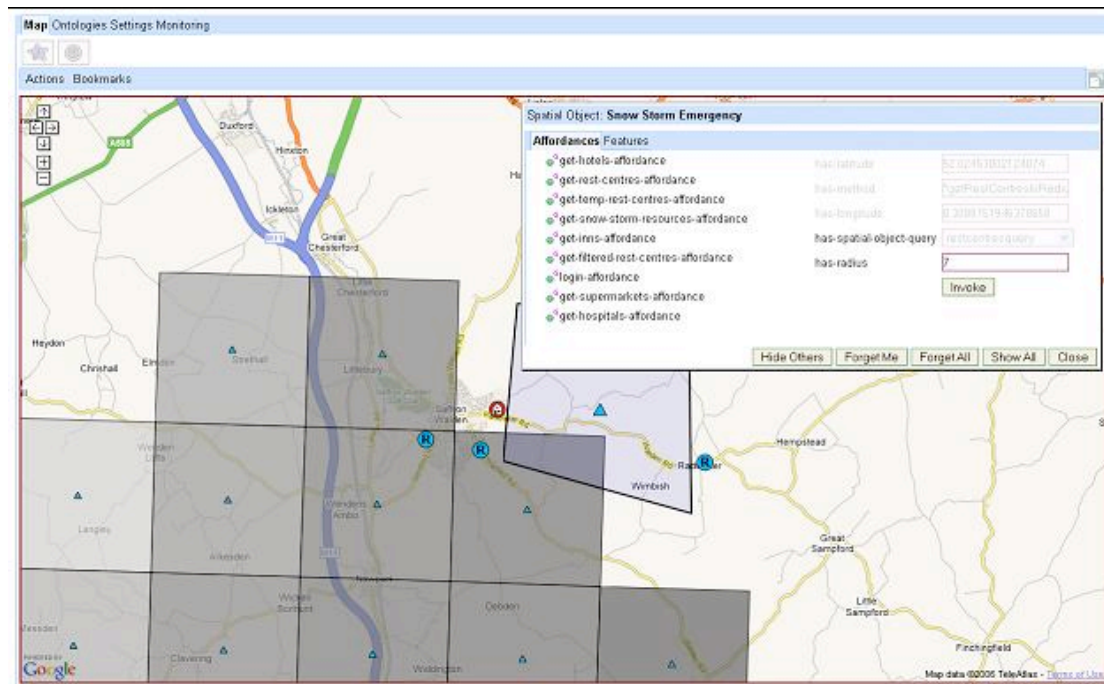


Figure 8. Screenshot of eMerges.

The emergency management system aggregates data and functionalities from three different sources:

- *Meteorological Office*: is a national UK organization that provides environmental resources, such as weather forecast, snow and pollution data.
- *ViewEssex*: is a collaboration between ECC and British Telecommunications (BT) which has created a single corporate spatial data warehouse. As can be expected ViewEssex contains a wide range of data including data for roads, administrative boundaries, buildings and Ordnance survey maps, as well as environmental and social care data.
- *BuddySpace*: is an Instant Messaging client facilitating lightweight communication, collaboration, and presence management built on top of the instant messaging protocol Jabber<sup>2</sup>. The BuddySpace client can be accessed on standard PCs, as well as on PDAs and mobile phones which in an emergency situation may be the only hardware device available.

eMerges distinguishes between two classes of services: *data* and *smart*. The former refer to the three data sources introduced above, and they are exposed by means of standard Web services:

- *Meteorological services*: provide weather information, e.g., snowfall level over a given rectangular spatial area.
- *ViewEssex services*: return detailed information on specific types of rest centre. For example, *Get Hospitals* is a Web service that returns a list of relevant hospitals within a given circular area.
- *BuddySpace services*: allow the presence information of online users to be accessed.

*Smart services* represent specific emergency planning reasoning and operations on the data provided by the data services. They are implemented in a

<sup>2</sup> Jabber. <http://www.jabber.org/>

mixture of Common Lisp and OCML and make use of the emergency management system ontologies. In particular, the application includes a number of services that filter the data retrieved from ViewEssex according to emergency-specific requirements: e.g., rest centres with heating system, hotels with at least 40 beds, easy accessible hospital, etc.

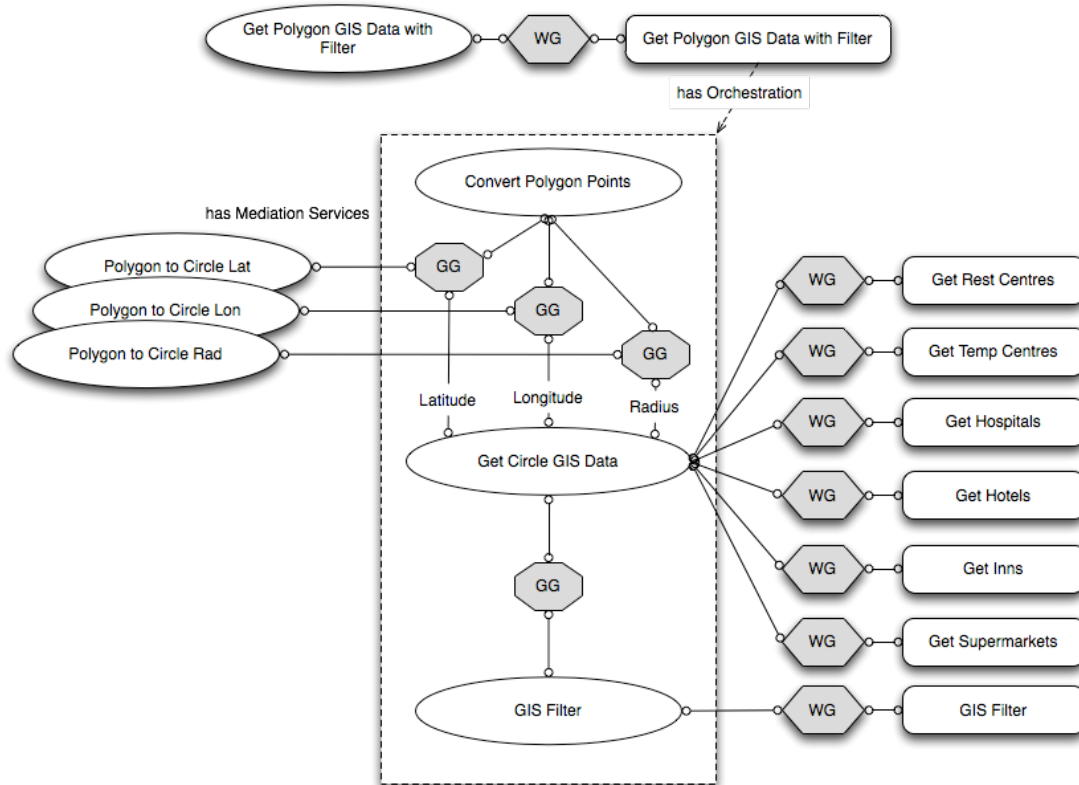


Figure 9. A portion of WSMO descriptions for eMerges.

Figure 9 shows an example of the created SWS descriptions: *Get Polygon GIS Data with Filter* represents a request for available shelters within a given area. The user specifies a polygon area and the shelter type (e.g. hospitals, inns, hotels). The results obtained by querying ViewEssex need to be filtered in order to return shelters correlated to emergency-specific requirements only. This scenario involves i) *selecting* the appropriate ViewEssex Web service; (ii) *mediating* the difference in area representations (polygon vs. circular) between the user goal and available Web services; (iii) *orchestrating* the retrieve and filter data operations to eventually achieve the user's goal.

```
(DEF-CLASS GET-ECC-HOSPITALS-WEB-SERVICE-CAPABILITY (CAPABILITY) ?CAPABILITY
  ((USED-MEDIATOR :VALUE GET-GIS-DATA-MEDIATOR)
   (HAS-ASSUMPTION:VALUE
    (KAPPA(?WEB-SERVICE)
      (= (WSMO-ROLE-VALUE ?WEB-SERVICE 'HAS-SPATIAL-OBJECT-QUERY)
         'HOSPITALSQUERY))))))
```

Listing 11. Example of a capability definition in OCML.

The SWS representations address the Web service selection problems as follows. When a user wants to achieve the *Get Circle GIS Data* Goal, IRS-III gets all the possible Web services that can solve it by means of the WG-mediators. Each semantic description of ViewEssex Web service defines the Web Service

capability (see Section 2.1.1.2), i.e., the functionality it provides, based on the class of shelter provided by the Web service. Listing 11 reports an example in OCML on how the assumption of a capability can be defined. If the Web service provides the class of shelters defined in one of the inputs of the goal, IRS-III selects it. In the example above, the Web service is selected if the request class of shelters are hospitals.

This use case therefore provides, as in the previous case, the ability to aggregate and reuse diverse information resources relevant to a given situation in a cost-effective way and to make this available as a basis for transparent interaction between community partner organizations and individual users. Additionally, this use case highlights the following aspects:

- Complex SWS descriptions were created on top of three distinct kinds of legacy system: database, GIS and instance messaging. The use of Web services allows one to abstract from the underlying technologies and ease thus their integration.
- A given Goal, e.g., *Get Circle GIS Data*, might be achieved by several Web services. The most appropriate one is selected on the basis of the specific situation. The actual workflow, i.e., the sequence of service invocations, is established at run-time only and it is therefore possible to better adapt the execution to the concrete context and situation at invocation time. In existing Web service-based approaches the functionalities are mapped at design-time, when the actual context is not known and therefore one can only reach this level of adaptability by complicating unnecessarily the workflow definition.
- The use of WG and GG mediators allows goal and process mediation and thus a smoothly crossing among services of distinct domains in the same workflow. Like for the previous case, the appropriate mediation service is selected at run-time, according to the specific situation.
- If new Web services are integrated, for instance providing data from further GIS, new Web Service descriptions can be simply introduced and linked to the Goals by means of mediators. In the same way, new filter services, e.g. more efficient ones, may be introduced with minimal changes.

## 4 Related Resources and Key Papers

This section briefly revises some of the main papers in the area and also provides a set of related resources that are considered of particular importance for understanding the research results produced up to date. For space reasons this is a distilled set of resources limited to those that are considered have been more influential in a given area of research within SWS. The resources have been ordered by year of publication.

### 4.1 Key papers

McIlraith et al. present in [7] what is perhaps the first comprehensive attempt to describe semantic Web services, the principles underlying them and the possible approaches for achieving this ambitious vision. The paper suggests how Web services can be annotated with DAML-S and outlines the main requirements for supporting the automation of Web services discovery, composition and execution. Finally the authors present a proof of concept implementation based

on agents technology. This paper established the main notions underlying semantic Web services and subsequently triggered and motivated a good part of the research around SWS described herein.

In [10], Fensel and Bussler introduce the Web Service Modeling Framework (WSMF), a conceptual model for developing and describing Web services and their compositions, which subsequently gave birth to the Web Service Modeling Ontology (WSMO) and Web Service Execution Environment (WSMX). The distinctive characteristics of WSMF are the two complementary principles it builds upon: an extreme decoupling of components supported by making mediation between components a central aspect. In this paper, the authors outline the core issues related to achieving the vision promoted by Web services and outline the main aspects that need to be described in a formal manner to cater for further automation.

In [8, 9] Sheth and Sivashanmugam et al. present the first attempt to SWS based on extending existing SOA standards and introduced the four main types of semantics that semantic Web service descriptions can capture. In particular the authors identify data semantics capturing the data exposed and used by services, functional semantics formalising the functionality of the service, non-functional semantics such as the Quality of Service), and execution semantics (e.g., execution errors).

In [52] Sycara et al. present in detail the work they have carried out on semantic Web services using DAML-S which then became OWL-S. This paper describes in detail their approach to service matchmaking as well as the DAML-S Virtual Machine that is able to interpret DAML-S processes and support their execution. Finally, they briefly introduce their work on using DAML-S descriptions for Web services composition by using an existing planner. The framework described in this paper was probably the most advanced SWS environment at that point on time and is still considered today as a reference for subsequent research.

In [11] Preist carries out a thorough analysis of a number of use cases in order to highlight the main requirements for SWS and execution environments. Based on this analysis, Preist presents a conceptual architecture for SWS based largely based on the W3C Web Services Architecture, which is extended to cater for the additional features he identifies. The conceptual architecture defined by the author interleaves conceptual aspects that are of central relevance for formalisms for SWS with procedural concerns aimed at highlighting the activities execution environments need to support and how the conceptual model defined covers these. Much of the work presented in this paper subsequently informed the development of SWS models and execution environments.

In [56] Li and Horrocks present an advanced approach to service matchmaking that treats service advertisements and requests as whole concepts as opposed to most of the approaches devised thus far that treat inputs, outputs and in the most advanced cases, also preconditions and effects separately. Thus, this work provides a somewhat higher-level of abstraction closer to the problem faced by developers of service-oriented systems that need activities to be fulfilled rather than specific services fitting certain inputs and outputs schemas. Additionally, this approach has also the merit of being entirely based on Description Logics which therefore makes it particularly suitable for existing semantic Web languages.

In [61] Traverso and Pistore propose a planning technique for supporting the automated composition of services described in OWL-S. The authors propose an approach that is able to deal with nondeterminism, partial observability as well as complex goals and generates executable compositions in BPEL4WS. Their planner makes use of semantic annotations in order to enhance the performance of the composition processes as compared to automating the composition by dealing directly with the syntactic descriptions of processes. One fundamental outcome of this research is precisely the fact that they illustrate how the use of semantic annotations can help improve the performance of Web service composition, therefore showing some of the potential benefits semantics can bring to Web services.

[21] describes OWL-S, formerly DAML-S, one of the main ontologies for the description of semantic web services expressed in OWL. OWL-S defines an upper ontology for semantically describing Web services along their *Profile*, i.e., 'what the service does', their *Model*, i.e., 'how the service works', and their *Grounding*, i.e., 'how the service can be accessed'.

In [12] Burstein et al. describe the main outcomes of the Semantic Web Services Initiative Architecture committee, which are a set of architectural and protocol abstractions that define the foundations for semantic Web service technologies. This work although essentially inline with prior work on conceptual models and architectures like [11] and [10], provides a more agent-oriented representation defining an interoperability model that can underpin diverse implementations.

[87] is a quite extensive compilation of diverse work carried out in semantic Web services. This book, edited by Cardoso and Sheth, presents some of the different approaches to annotating semantic Web services. It also tackles aspects such as the matchmaking of services, the choreography of services, and the use of temporal reasoning for supporting reactive processes. Finally, the book presents a number of real-world applications that exploit semantic Web services technologies thus illustrating but also motivating the techniques described. All in all this book presents a very valuable overview of some of the main aspects around semantic Web services and therefore complements this chapter with more detailed information about most of the topics exposed herein.

[23] provides the specification of SAWSDL, which is, at the time of this writing, the only Web standard for semantic Web services. SAWSDL is a lightweight bottom-up approach to bringing semantics to Web services described in WSDL and provides also a means for annotating XML schema. The specification identifies three kinds of hooks, namely *Model Reference*, *Lifting Schema Mapping* and *Lowering Schema Mapping*. The former provides means for linking certain WSDL and XML elements to semantic elements through URIs. The other two on the other hand provide the means for specifying how syntactic data is transformed into their semantic counter part and vice versa.

[19] compiles a good deal of the body of research carried out around WSMO. The book therefore describes the ontology in detail explaining the rationale behind its core elements Goal, Web Service, Mediator and Ontology and how they help to support the automation of typical activities involved when using Web services. The book additionally provides details on the Web Service Modeling Language (WSML) that supports describing WSMO entities and

provide examples on how these descriptions can be utilised to support the discovery, composition and execution of services among others.

[15] complements [19] providing a more architectural perspective over the construction of systems that can make use of semantic Web services descriptions to cover the entire life-cycle of SWS-based applications. The book presents a conceptual architecture for Semantic Execution Environments and provides details for how these components shall be integrated and how they internally achieve their particular goals.

In [70] Domingue et al. thoroughly describe IRS-III one of the main frameworks for SWS, which is covered in Section 2.2.3. IRS-III is a broker-based platform based on research on Problem-Solving Methods that mediates between clients and service providers allowing each of them to adopt the most convenient means for representing their perspective, while supporting an effective interaction. It is largely based on WSMO and provides support for ranking and selecting services, as well as orchestrating and invoking them by carrying out the appropriate data mediation and choreography of message exchanges.

In [44] Sheth et al. present SA-REST an open, flexible, and standards-based approach to adding semantic annotations to RESTful services and Web APIs. SA-REST is perhaps the first serious approach to tackling the annotation of the emerging RESTful services and Web APIs thus bringing them into the semantic Web services spectrum. SA-REST uses GRDDL and RDFa as a means to structure Web pages and support the embedding of semantic annotations within them. On top of these microformats, SA-REST supports the linking of service descriptions to semantic metamodels by using model reference type of annotations from SAWSDL. The use of SA-REST and semantically annotated Web APIs for rapidly creating smart or semantic mashups and for supporting semantic search and ranking of Web services are discussed in [47] and [88] respectively.

WSMO-Lite [38] is a recent approach to bring semantics to SAWSDL. WSMO-Lite identifies four main types of semantic annotations for services, namely functional semantics, nonfunctional semantics, behavioral semantics and the information model and how these can be expressed by means of SAWSDL annotations. Additionally, WSMO-Lite provides a simple RDF Schema that allows one to refine the semantics of the SAWSDL links in an incremental and compatible manner.

## 4.2 Related Papers

[26] describes in detail the CommonKADS methodology for constructing Knowledge Based Systems. This book accounts for a large body of research carried out in the context of Problem Solving Methods which aims at providing systematic means including a conceptual framework as well as a methodology for the development of systems that solve knowledge intensive tasks by reusing general purpose yet task specific components. Although the notion of service is not directly contemplated in this research, certain approaches to semantic Web services notably WSMO and IRS-III are largely based on many of the notions from research on Problem Solving Methods that this book addresses.

[89, 90] provide the specifications for WSDL 1.1 and WSDL 2.0 respectively. Although, WSDL solely provides syntactic means for describing Web services, it is one of the most widely applied technologies for describing

services on the Web and indeed most of the approaches to describing services semantically are compatible with it.

[18, 91] provide the initial specification of the Business Process Execution Language (BPEL) and subsequent refinement. BPEL is the de-facto standard for the specification of executable compositions of Web services in the industry and has therefore been utilised within semantic Web service applications to support the orchestration of services. Some researchers have additionally worked on providing semantic extensions [61, 68, 92].

[93] presents an overview of the main issues faced by Web services both from an industrial perspective as well as from the stand point of researchers focussing on semantic Web services. This series of short articles presents a quick yet remarkable comparison of existing process modelling languages, highlighting their features and drawbacks. Additionally, it also covers conceptual concerns that highlight the lessons that can be learnt from prior work on Problem Solving Methods, as well as conceptual models for describing services semantically. All in all, it provides a brief yet valuable overview of issues and approaches that can help better understand the research and development carried out so far in the area of Web services and semantic Web services.

Most of the research on attaching semantics to services has focussed mainly on the technical aspects. In [64] Akkermans et al. provide a different perspective focussing on the essential features of services from a business stand point. This research, together with other complementary work by the authors on a suite of business oriented ontologies and tools, is complementary to that focussing on technical aspects. It provides an advanced framework for the development of business solutions that are able to adequately fulfil customer needs and support the analysis and combination of business services into added-value solutions.

[35, 94, 95, 96, 97, 98] are a few out of the growing body of specifications being defined for Web services. These specifications cover aspects such as service registries, the brokering of messages through queues, the definition of policies and the establishment of trustworthy and secure communications. Listing them all is out of the scope of this chapter, however, the reader should note that these new specifications are likely to trigger the need for semantic enrichment and alignment with initiatives coming from the semantic Web.

## 5 Future Issues

After almost a decade, research on SWS has produced a wealth of conceptual models, languages, architectures, algorithms and engines that highlight the potential of these technologies both for enterprise settings and for the Web. This chapter has introduced some of the main results and has also provided examples of applications that have explored and showcased the use of SWS in real-world settings. Despite the advances, however, the vision initially proposed in [6] and later on highlighted in [7] when SWS were first proposed is still to be achieved. In order to outline what could be the future issues to be addressed and what the future could bring to this field it is therefore necessary to analyse the current situation and try to identify what the reasons for this reduced take up are. An example of critical analysis appears in [99].

Research on SWS so far has focussed mostly on extending existing Web services technologies with semantics. Yet, recent analyses estimate that the number of publicly available Web services on the Web is around 27,000 which contrasts with the number of Web pages available and with the number of services that big companies have internally (e.g., approximately 1,500 for Verizon) [4]. Hence, despite their name, Web services seem to be essentially enclosed within enterprises and it has been argued that indeed Web services are not really thought for the Web [100]. The application of SWS technologies mostly concerns enterprises and in this respect the appearance of SAWSDL as the first W3C standard for annotating Web services has had a positive impact in the take up of SWS technologies. On the Web the use of SWS is even scarcer and it seems that the appearance of intelligent Web agents that act of behalf of users remains an elusive target. Still, the demand for services on the Web exists as indicated by the proliferation and popularity of publicly available Web APIs and RESTful services.

Additionally, SWS are particularly demanding from a knowledge acquisition perspective. Creating a rich semantic description of a Web service requires the use of domain ontologies, the use of services taxonomies, the definition of lifting and lowering mechanisms, and in some cases the inclusion of complicated logical expressions. This tedious and complex annotation process has arguably hampered the adoption of SWS technologies especially in the early days of the semantic Web when publicly available ontologies and semantic information were scarce. As a consequence, the application of SWS technologies within real applications is usually limited to simple annotations that simplify the retrieval of services. This leaves aside more advanced features such as the automated selection of services and therefore reduces the potential benefit that can be obtained.

In the short term the future of services will continue and exacerbate current trends that are giving birth to a dichotomy between services for the enterprise (i.e., Web services and the WS-\* stack) and services for the Web (i.e., Web APIs). In the long run, with the wider use of services on the Web, it is expected that both technologies will gradually fuse driven by the interest of companies to also address the long tail of customers on the Web. The future of SWS research is undeniably going to be closely related to the evolution of services technologies. Consequently research on SWS will be affected by this evolution and there will be two main application areas, one driven by the use of services in the enterprise and the other by the application of services on the Web.

One application area will therefore focus on providing semantic annotations to the growing body of WS-\* standards in order to properly describe, communicate and reason about processes, policies, trust, and security. This area will work on addressing some of the main research challenges in the area such as the self-management, self-adaptation and the governance of service-oriented systems, aspects that have indeed traditionally been of concern for SWS [101]. The other application area will bring in, extend, and adapt the results gathered so far, to deal with the intricacies of Web APIs. In this area major efforts will have to be devoted to adequately dealing with an open environment like the Web, thus truly assuming its heterogeneity and dynamicity but also exploiting extensively the benefits that can be obtained by using semantics. Also, the fact

that Web applications are turning towards handcrafted Web APIs rather than WSDL Web services will drive the current focus of semantic Web services in this second area of application putting the Web back at the centre of the research.

Alongside these major trends, impelled by the technical evolution of services, research on SWS needs to focus on the main pending issue: its take up. This will necessarily have to put the emphasis on reducing the annotation bottleneck and on properly accommodating Web trends above other aspects. One shall see further solutions based on lightweight annotations (e.g., SAWSDL and WSMO-Lite) as has happened in the semantic Web in general. Still, SWS will necessarily have to move towards a consensus on the languages used and possibly SAWSDL and RDF(S) are good starting points.

Additionally, work on services will need to integrate with the Linked Data phenomenon. Linked Data will on the one hand provide a wealth of data able to reduce to a certain extent the semantic annotation bottleneck. On the other hand the Linked Data community and the semantic Web in general will require the development of more and more complex distributed solutions for which services are an adequate software engineering abstraction. In essence, a significant step toward a greater adoption and use of SWS technologies lies on viewing semantic Web services as services for the semantic Web rather than as semantics on top Web services.

## 6 Reference

1. T. Erl, SOA Principles of Service Design. Prentice Hall (2007)
2. M. Papazoglou. Web Services: Principles and Technology. Prentice Hall (2007)
3. W. Van Der Aalst, Don't Go with the Flow: Web Services Composition Standards Exposed. IEEE Intelligent Systems. **18**, 72--76 (2003)
4. J. Davies, J. Domingue, C. Pedrinaci, D. Fensel, R. Gonzalez-Cabero, M. Potter, M. Richardson, S. Stincic. Towards the Open Service Web. BT Technology Journal. **26**, (2009)
5. R. T. Fielding. Architectural Styles and the Design of Network-based Software Architectures. (2000)
6. T. Berners-Lee, J. Hendler, O. Lassila. The Semantic Web. Scientific American. pp 34-43 (2001)
7. S. McIlraith, T. Son, H. Zeng. Semantic Web Services. IEEE Intelligent Systems. **16**, 46--53 (2001)
8. A. Sheth, Semantic Web Process Lifecycle: Role of Semantics in Annotation, Discovery, Composition And Orchestration. (2003)
9. K. Sivashanmugam, K. Verma, A. Sheth, J. Miller. Adding Semantics to Web Services Standards. Proceedings of the 2003 International Conference on Web Services (ICWS'03) (2003)

10. D. Fensel, C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*. **1**, 113--137 (2002)
11. C. Preist. A Conceptual Architecture for Semantic Web Services. *International Semantic Web Conference* (2004)
12. M. Burstein, C. Bussler, M. Zaremba, T. Finin, M. N. Huhns, M. Paolucci, A. P. Sheth, S. Williams. A Semantic Web Services Architecture. *IEEE Internet Computing*. **9**, 72-81 (2005)
13. B. Norton, C. Pedrinaci, J. Domingue, M. Zaremba. Semantic Execution Environments for Semantics-Enabled SOA. *IT - Methods and Applications of Informatics and Information Technology*. **Special Issue in Service-Oriented Architectures**, 118--121 (2008)
14. H. Haas, A. Brown. *Web Services Glossary*. (2004)
15. *Implementing Semantic Web Services: the SESA Framework*. Springer D. Fensel, M. Kerrigan, M. Zaremba (eds.) (2008)
16. W. Van Der Aalst, M. Dumas, A. Ter Hofstede. *Web Service Composition Languages: Old Wine in New Bottles?* *Proceedings of EUROMICRO'03* (2003)
17. W. Van Der Aalst, A. Ter Hofstede, B. Kiepuszewski, A. Barros. *Workflow Patterns. Distributed and parallel databases*. **14**, 5--51 (2003)
18. T. Andrews, F. Curbera, H. Dholakia, Y. Golland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana. *Business Process Execution Language for Web Services version 1.1*. (2003)
19. D. Fensel, H. Lausen, A. Polleres, J. De Bruijn, M. Stollberg, D. Roman, J. Domingue. *Enabling Semantic Web Services: the Web Service Modeling Ontology*. Springer (2007)
20. D. Roman, H. Lausen, U. Keller. *Web Service Modeling Ontology (WSMO)*. WSMO Working Draft. <http://www.wsmo.org/TR/d2/v1.3/> (2006)
21. D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, K. Sycara. *OWL-S: Semantic Markup for Web Services*. W3C Member Submission. <http://www.w3.org/Submission/OWL-S> (2004)
22. R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M.T. Schmidt, A. Sheth, K. Verma. *Web Service Semantics - WSDL-S*. W3C Member Submission. <http://www.w3.org/Submission/WSDL-S/> (2005)
23. J. Farrell, H. Lausen *Semantic Annotations for WSDL and XML Schema*. W3C Recommendation. <http://www.w3.org/TR/sawSDL/> (2007)

24. P. F. Patel-Schneider, P. Hayes, I. Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation. <http://www.w3.org/TR/owl-semantics/> (2004)
25. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosz, M. Dean. SWRL: a Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission. <http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/> (2004)
26. G. Schreiber, H. Akkermans, A. Anjewierden, R. De Hoog, N. Shadbolt, W. v. De Velde, B. Wielinga. Knowledge Engineering and Management: the CommonKADS Methodology. Mit Press. (1999)
27. D. Fensel, E. Motta, F. Van Harmelen, V. R. Benjamins, M. Crubezy, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, M. Musen, E. Plaza, G. Schreiber, R. Studer, B. Wielinga. The Unified Problem-Solving Method Development Language UPML. Knowledge and Information Systems. **5**, 83--131 (2003)
28. The Web Service Modeling Language WSML. J. De Bruijn (ed.). WSML Working Draft D16.1v0.2. <http://www.wsmo.org/TR/d16/> (2008)
29. The Object Management Group: Meta-Object Facility, version 1.4. <http://www.omg.org/technology/documents/formal/mof.htm>. (2002)
30. Y. Gurevich. Evolving Algebras: an Attempt to Discover Semantics. Current Trends in Theoretical Computer Science, eds. G. Rozenberg and A. Salomaa, World Scientific, 266--292 (1993)
31. B. Norton, C. Pedrinaci, L. Henocque, M. Kleiner. 3-Level Behavioural Models for Semantic Web Services. International Transactions on Systems Science and Applications. **4**, 340-355 (2008)
32. B. Omelayenko, M. Crubézy, D. Fensel, V. R. Benjamins, B. Wielinga, E. Motta, M. Musen, Y. Ding. UPML: the Language and Tool Support for Making the Semantic Web Alive. Spinning the Semantic Web. D. Fensel, J. Hendler, H. Lieberman, W. Wahlster (eds.). MIT Press. (2003)
33. B. Chandrasekaran. Generic Tasks in Knowledge Based Reasoning: High-level Building Blocks for Expert System Design. IEEE Expert. 23--30 (1986)
34. K. Verma, A. Sheth, Semantically annotating a web service. IEEE Internet Computing. **11**, 83-85 (2007)
35. L. Clement, A. Hatley, C. Von Riegen T. Rogers. UDDI Specification Version 3.0.2. (2004)
36. R. Akkiraju, B. Sapkota. Semantic Annotations for WSDL and XML Schema -- Usage Guide. Working Group note. <http://www.w3.org/TR/sawSDL-guide/> (2007)
37. W. Akhtar, J. Kopecký, T. Krennwallner, A. Polleres. XSPARQL: Traveling

Between the XML and RDF Worlds -- and Avoiding the XSLT Pilgrimage. *The Semantic Web: Research and Applications (ESWC 2008)* (2008)

38. T. Vitvar, J. Kopecky, J. Viskova, D. Fensel. WSMO-Lite Annotations for Web Services. *Proceedings of the 5th European Semantic Web Conference* (2008)

39. M. Hepp. Products and Services Ontologies: a Methodology for Deriving OWL Ontologies from Industrial Categorization Standards. *International Journal on Semantic Web and Information Systems (IJSWIS)*. **2**, 72-99 (2006)

40. L. Richardson, S. Ruby. *RESTful Web Services*. O'reilly Media, Inc. (2007)

41. M. Maleshkova, J. Kopecký, C. Pedrinaci. Adapting SAWSDL for Semantic Annotations of RESTful Services. *Workshop: Beyond SAWSDL at OnTheMove Federated Conferences & Workshops* (2009)

42. J. Kopecky, K. Gomadam, T. Vitvar. hRESTS: an HTML Microformat for Describing RESTful Web Services. *The 2008 IEEE/WIC/ACM International Conference on Web Intelligence (WI2008)* (2008)

43. J. Lathem, K. Gomadam, A. Sheth. SA-REST and (S)mashups: Adding Semantics to RESTful Services. *ICSC '07: Proceedings of the International Conference on Semantic Computing* (2007)

44. A. Sheth, K. Gomadam, J. Lathem. SA-REST: Semantically Interoperable and Easier-to-use Services and Mashups. *IEEE Internet Computing*. **11**, 91--94 (2007)

45. Gleaning resource descriptions from dialects of languages (GRDDL). D. Connolly (ed.). W3C Recommendation. <http://www.w3.org/TR/grddl/> (2007)

46. RDFa in XHTML: Syntax and Processing. B. Adida, M. Birbeck, S. Mccarron, S. Pemberton (eds.). W3C Recommendation. <http://www.w3.org/TR/rdfa-syntax/> (2008)

47. K. Gomadam, A. Ranabahu, M. Nagarajan, A. Sheth, K. Verma. A Faceted Classification Based Approach to Search and Rank Web APIs. *ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services* (2008)

48. D. Elenius, G. Denker, D. Martin, F. Gilham, J. Khouri, S. Sadaati, R. Senanayake. The OWL-S Editor--a Development Tool for Semantic Web Services. *The Semantic Web: Research and Applications*. 78--92 (2005)

49. A. Heß, E. Johnston, N. Kushmerick. ASSAM: a Tool for Semi-Automatically Annotating Semantic Web Services. *International Semantic Web Conference* (2004)

50. M. Sabou, *Building Web Service Ontologies*. PhD Thesis. Vrije Universiteit Amsterdam. (2006)

51. N. Srinivasan, M. Paolucci, K. Sycara. Adding OWL-S to UDDI: Implementation and Throughput. *Proceedings of 1st International Conference on Semantic Web Services and Web Process Composition (SWSWPC 2004)* (2004)

52. K. Sycara, M. Paolucci, A. Ankolekar, N. Srinivasan. Automated Discovery, Interaction and Composition of Semantic Web Services. *Web Semantics: Science, Services and Agents on the World Wide Web*. **1**, 27 - 46 (2003)
53. V. Haarslev, R. Moller. RACER: A Core Inference Engine for the Semantic Web. *Second International Semantic Web Conference ISWC 2003* (2003)
54. M. Jaeger, G. Rojec-Goldmann, C. Liebetrueth, G. Mühl, K. Geihs. Ranked Matching for Service Descriptions using OWL-S. *Kommunikation in verteilten Systemen (KiVS 2005)* (2005)
55. M. Klusch, B. Fries, K. Sycara. Automated Semantic Web Service Discovery with OWLS-MX. *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems* (2006)
56. L. Li, I. Horrocks. A Software Framework for Matchmaking based on Semantic Web Technology. *International Journal of Electronic Commerce*. **8**, 39 (2004)
57. M. Paolucci, A. Ankolekar, N. Srinivasan, K. Sycara. The DAML-S Virtual Machine. *Proceedings of 2nd International Semantic Web Conference (ISWC 2003)* (2003)
58. HP Labs. Jena: A Semantic Web Framework for Java. <http://jena.sourceforge.net/> (2004)
59. E. Friedman-Hill. *JESS in Action*. Manning Publications Co. Taylor, T.(ed.): (2003)
60. E. Sirin, B. Parsia, J. Hendler. Filtering and Selecting Semantic Web Services with Interactive Composition Techniques. *IEEE Intelligent Systems*. **19**, 42--49 (2004)
61. P. Traverso, M. Pistore. Automated Composition of Semantic Web Services into Executable Processes. *3rd International Semantic Web Conference (ISWC 2004)* (2004)
62. M. Dimitrov, A. Simov, M. Konstantinov, V. Momtchev. WSMO Studio - a Semantic Web Services Modelling Environment for WSMO (System Description). *Proceedings of the 4th European Semantic Web Conference (ESWC) (2007)*
63. M. Kerrigan, A. Mocan, M. Tanler, D. Fensel. The Web Service Modeling Toolkit - an Integrated Development Semantic Web Services. *ESWC '07: Proceedings of the 4th European conference on The Semantic Web* (2007)
64. H. Akkermans, Z. Baida, J. Gordijn, N. Peña, A. Altuna, I. Laresgoiti, Value Webs: Ontology-Based Bundling of Real-World Services. *IEEE Intelligent Systems*. **19**, 57--66 (2004)
65. W. J. Clancey, Heuristic classification. *Artificial intelligence*. **27**, 289--350 (1985)

66. M. Stollberg, Scalable Semantic Web Service Discovery for Goal-driven Service-Oriented Architectures. PhD thesis, University Innsbruck, Austria(2008)
67. A. Turati, E. D. Valle, D. Cerizza, F. M. Facca. Using GLUE to Solve the Discovery Scenarios of the SWS-Challenge. Semantic Web Services Challenge. 185--197 (2009)
68. J. Nitzsche, T. Van Lessen, D. Karastoyanova, F. Leymann. BPEL for Semantic Web Services (BPEL4SWS). On the Move to Meaningful Internet Systems 2007: OTM 2007 workshops. 179--188 (2007)
69. E. Motta, J. Domingue, L. Cabral, M. Gaspari. IRS-II: a Framework and Infrastructure for Semantic Web Services. 2nd International Semantic Web Conference (ISWC) (2003)
70. J. Domingue, L. Cabral, S. Galizia, V. Tanasescu, A. Gugliotta, B. Norton, C. Pedrinaci, IRS-III: A Broker-based Approach to Semantic Web Services. Web Semantics: Science, Services and Agents on the World Wide Web. **6**, 109--132 (2008)
71. E. Motta. Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving. IOS Press (1999)
72. F. Hakimpour, J. Domingue, E. Motta, L. Cabral, Y. Lei. Integration of OWL-S into IRS-III. First AKT Workshop on Semantic Web Services (2004)
73. E. Motta, W. Lu. A Library of Components for Classification Problem Solving. IBrow (ist-1999-19005) deliverable. [http://projects.kmi.open.ac.uk/ibrow/Publications/Motta\\_pkaw00.pdf](http://projects.kmi.open.ac.uk/ibrow/Publications/Motta_pkaw00.pdf) (2001)
74. S. Galizia, A. Gugliotta, C. Pedrinaci. A Formal Model for Classifying Trusted Semantic Web Services. 3rd Asian Semantic Web Conference (ASWC 2008) (2008)
75. C. Pedrinaci, P. Grenon, S. Galizia, A. Gugliotta, J. Domingue. A Knowledge-Based Framework for Web Service Adaptation to Context. Enabling Context-aware Web Services: Methods, Architectures, and Technologies. Chapman and Hall/CRC. (2010)
76. A. Sheth, K. Gomadam, A. Ranabahu. Semantics Enhanced Services: METEOR-S, SAWSDL and SA-REST. IEEE Data Engineering bull. **31**, 8--12 (2008)
77. A. Patil, S. Oundhakar, A. Sheth, K. Verma. METEOR-S Web Service Annotation Framework. WWW '04: Proceedings of the 13th international conference on World Wide Web (2004)
78. K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, J. Miller. METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. International Journal of Information Technologies and Management. **6**, 17--39 (2005)

79. M. Nagarajan, K. Verma, A. Sheth, J. Miller, J. Lathem. Semantic Interoperability of Web Services - Challenges and Experiences. ICWS '06: Proceedings of the IEEE International Conference on Web Services (ICWS'06) (2006)
80. K. Sivashanmugam, J. A. Miller, A. Sheth, K. Verma. Framework for Semantic Web Process Composition. International Journal of Electronic Commerce. **9**, 71--106 (2005)
81. M. Nagarajan, K. Verma, A. P. Sheth, J. A. Miller. Ontology Driven Data Mediation in Web Services. International Journal of Web Service Research. **4**, 104-126 (2007)
82. K. Sivashanmugam, J. A. Miller, A. P. Sheth, K. Verma. Framework for Semantic Web Process Composition. International Journal on Electronic Commerce. **9**, 71--106 (2004-2005)
83. N. Oldham, K. Verma, A. Sheth, F. Hakimpour. Semantic WS-Agreement Partner Selection. WWW '06: Proceedings of the 15th international conference on World Wide Web (2006)
84. K. Verma, P. Doshi, K. Gomadam, J. Miller, A. Sheth. Optimal Adaptation in Web Processes with Coordination Constraints. Web Services, 2006. ICWS '06. International Conference on (2006)
85. K. Gomadam, K. Verma, A. Sheth, J. Miller, Demonstrating Dynamic Configuration and Execution of Web Processes. Service-Oriented Computing - ICSOC 2005. 502--507 (2005)
86. A. Gugliotta, J. Domingue, L. Cabral, V. Tanasescu, S. Galizia, R. Davies, L. Gutiérrez-Villariás, M. Rowlatt, M. Richardson, S. Stincic. Deploying Semantic Web Services-based Applications in the e-Government Domain. Journal of Data Semantics. **10**, 96-132 (2008)
87. Semantic Web Services, Processes and Applications. J. Cardoso, A. Sheth (eds.). Springer (2006)
88. A. Sheth, K. Verma, K. Gomadam. Semantics to Energize the Full Services Spectrum. Communications of the ACM. **49**, 55--61 (2006)
89. Web Services Description Language (WSDL) 1.1. E. Christensen, F. Curbera, G. Meredith, S. Weerawarana (eds.). W3C Note (2001)
90. Web Services Description Language (WSDL) version 2.0 part 0: Primer. D. Booth, C. K. Liu (eds.). W3C Recommendation. <http://www.w3.org/TR/wsd20-primer/> (2007)
91. OASIS Web Services Business Process Execution Language (WSBPEL) TC. Web Services Business Process Execution Language version 2.0. Committee specification. <http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.pdf> (2007)

92. D. Mandell, S. Mcilraith. Adapting BPEL4WS for the Semantic Web: the Bottom-up Approach to Web Service Interoperation. International Semantic Web Conference (2003)
93. S. Staab, W. Van Der Aalst, V. R. Benjamins, A. Sheth, J. A. Miller, C. Bussler, A. Maedche, D. Fensel, D. Gannon. Web services: been there, done that? IEEE Intelligent Systems. **18**, 72-85 (2003)
94. OASIS Web Services Notification TC. Web Services Topics (WS-Topics) 1.3. [http://docs.oasis-open.org/wsn/wsn-ws\\_topics-1.3-spec-os.pdf](http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf) (2006)
95. OASIS Web Services Notification TC. Web Services Brokered Notification (WS-BrokeredNotification) 1.3. [http://docs.oasis-open.org/wsn/wsn-ws\\_brokered\\_notification-1.3-spec-os.pdf](http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf) (2006)
96. OASIS Web Services Notification TC. Web Services Base Notification (WS-BaseNotification) 1.3. [http://docs.oasis-open.org/wsn/wsn-ws\\_base\\_notification-1.3-spec-os.pdf](http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf) (2006)
97. OASIS Web Services Notification TC. Web Services Notification (WSN) 1.3. [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wsn](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn)(2006)
98. A. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez, Ü. Yalçinalp. Web Services Policy 1.5 - framework. W3C Recommendation. <http://www.w3.org/TR/ws-policy/> (2007)
99. A. Sheth, Beyond SAWSDL: a Game Plan for Broader Adoption of Semantic Web Services. IEEE Intelligent Systems Trends & Controversies. **22**, (2007)
100. S. Vinoski, Putting the "Web" into Web Services: Interaction Models, part 2. IEEE Internet Computing. **6**, 90-92 (2002)
101. M. Papazoglou, P. Traverso, S. Dustdar, F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. Computer. **40**, 38--45 (2007)