Semantic Web Services

OWL-S and others

© Copyright 2010: Dieter Fensel and Srdjan Komazec

Where are we?

<table>
<thead>
<tr>
<th>#</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Web Science</td>
</tr>
<tr>
<td>3</td>
<td>Service Science</td>
</tr>
<tr>
<td>4</td>
<td>Web services</td>
</tr>
<tr>
<td>5</td>
<td>Web2.0 services</td>
</tr>
<tr>
<td>6</td>
<td>Semantic Web</td>
</tr>
<tr>
<td>7</td>
<td>Web Service Modeling Ontology (WSMO)</td>
</tr>
<tr>
<td>8</td>
<td>Web Service Modeling Language (WSML)</td>
</tr>
<tr>
<td>9</td>
<td>Web Service Execution Environment (WSMX)</td>
</tr>
<tr>
<td>10</td>
<td>OWL-S and others</td>
</tr>
<tr>
<td>11</td>
<td>Light-weight Annotations</td>
</tr>
<tr>
<td>12</td>
<td>Applications</td>
</tr>
<tr>
<td>13</td>
<td>Mobile Services</td>
</tr>
</tbody>
</table>

Outline

- Motivation
  - Technical solution
    - Comparison approach
    - Framework description
    - Comparison criteria
  - Evaluated Semantic Web Service (SWS) frameworks
    - OWL-S
    - METEOR-S
    - SWSF
    - Evaluated Semantic Execution Environment (SEE) implementation
  - WS – 8
- Illustration by larger example
- Summary
- References

Motivation
Motivation

- WSMO is not the only initiative aimed towards Semantic Web Services
- Other major approaches in the area are documented by recent W3C member submissions:
  - OWL-S,
  - METEOR-S, and
  - SWSF.
- Other implementations of WSMO
  - IRS-III.

Goals of this lecture:
- Explain principal characteristics of the approaches,
- Exemplify their usage,
- Give overview of the tools support,
- Compare approach to WSMO (for IRS-III to WSMO), and
- Evaluate approach in context to the SWS framework requirements.

Technical Solution
Framework description

Comparison Approach
Framework Description

- Overall approach
  - Describes principal characteristics of evaluated framework
- Conceptual model
  - Explains underlying model adopted by the framework
- Example
  - Illustrates the framework usage through an example
- Tool support
  - Overview existing tools developed to support the framework
- Relation to WSMO
  - Draws parallel between WSMO and evaluated framework

Technical solution
Comparison criteria
Comparison Approach

Comparison Criteria

- SWS are basically combining Semantic Web technologies and Web services.
- Thus SWS frameworks need to integrate
  - basic Web design principles,
  - design principles defined for the Semantic Web, and
  - design principles for distributed, service-oriented computing on the Web.
- A SWS framework should answer to the following requirements\(^1\):
  - Web compliance
  - Ontology-based
  - Strict decoupling
  - Centrality of mediation
  - Ontological role separation
  - Description vs. implementation
  - Execution semantics
  - Service vs. Web service

\(^1\)Fensel, Dieter; Kerrigan, Mick; Zaremba, Michal (Eds.). Implementing Semantic Web Services: The SESA Framework. Springer, 2008

### Comparison Approach

#### Comparison Criteria – Web Compliance

- What is it?
  - A set of simple and flexible concepts forming the foundation of Web such as Universal Resource Identifier (URI), namespaces, etc.
- Why is it important?
  - Reliance on the already established concepts such as URLs enables seamless integration of pre-existing Web resources and application of proven and tested tools and libraries.
- What is expected from the framework?
  - A SWS framework should integrate with the current Web, thus it should reuse/exchange the basic Web concepts.

#### Comparison Criteria – Ontology-based

- What is it?
  - Ontologies are formal and explicit specifications of shared conceptualizations.
  - An ontology-based system is a system which adopts ontologies as a founding format to express data and processes which the system deals with.
- Why is it important?
  - Ontologies are allowing enhanced information processing and automatic resolution of heterogeneity issues between resources.
- What is expected from the framework?
  - Usage of ontologies as the data model throughout framework.
  - All resource descriptions and data exchanged are ontologically described.

#### Comparison Criteria – Strict Decoupling

- What is it?
  - Decoupling represents separation of resources which should not depend on each other.
  - Size, diversity and unanticipated usage of Web resources are advocating towards their strict decoupling.
- Why is it important?
  - On the Web resources are developed in isolation from one another.
- What is expected from the framework?
  - Resources should be strictly decoupled from one another.
  - Semantic descriptions are developed in isolation without regard for their possible usage or interaction with other resources.
### Comparison Approach - Centrality of Mediation

**What is it?**
- Mediation is an activity which reconciles heterogeneities between parties.

**Why is it important?**
- Decoupling of resources requires existence of a mechanism for resolving heterogeneity issues between them.
- Mediation performs the role of resolving potential mismatches between resources that occur at the data, protocol and process level.

**What is expected from the framework?**
- Mediators should be top-level elements of the framework.
- It should be easy to connect mediators to the other elements.

### Comparison Approach - Ontological Role Separation

**What is it?**
- Consumers and providers of Web Services are loosely coupled.
- Consumption of a Web Service functionality can be neither expected nor enforced.

**Why is it important?**
- The contexts within which requesters and providers of services exist can be very different from one another.

**What is expected from the framework?**
- A framework should differentiate between the requirements that a given requester has and the functionality that service providers are willing to offer.
- The framework should stress the difference between the goals and Web Services.

### Comparison Approach - Description vs. Implementation

**What is it?**
- Difference between description of functionality and its implementation.
- Service description is separated from the way the service is implemented and offered.

**Why is it important?**
- A firm distinction between the description of elements of Semantic Web Services and executable technologies should be present.
- It allows decoupling from the current implementation technologies and possibility to ground descriptions to different underlying standards.

**What is expected from the framework?**
- A framework should aim at providing ontological description model for the elements and to be compliant with the executable technologies.

### Comparison Approach - Execution Semantics

**What is it?**
- It is a formal description of the operational behavior of the system.
- It is a formal description of the operational behavior of the system.
- A formal description of the operational behavior of the system.

**Why is it important?**
- It provides a mechanism to test, simulate and verify the SWS specification, identify anomalies such as deadlocks, livelocks and tasks that are never reached.
- It provides a sound foundation to build an engine being able to execute created models.
- It improves or even rules out ambiguities in model understanding by humans.

**What is expected from the framework?**
- Framework should provide means to specify and run formal execution semantics.
Comparison Approach

Comparison Criteria – Service vs. Web Service

• What is it?
  – A clear distinction between Services and Web Services.
  – A Web Service is a computational entity that is able to provide some functionality to
    the user by invoking it.
  – A service is the actual value provided by the invocation.

• Why is it important?
  – Although services are what counts for the consumers their value may be subjective
    and context-dependent.
  – A SWS framework should provides means to describe Web Services.

• What is expected from the framework?
  – A framework should be designed as a means to describe Web Services and not to
    replace services.
  – Actual value is provided by Web Service invocations.

Technical solution

OWL – S

OWL-S

Overall approach

• OWL-S represents an upper ontology for the description of Semantic
  Web Services expressed in OWL.
  – It has its roots in the DAML Service Ontology (DAML-S).
  – It adopts existing Semantic Web recommendations (i.e. OWL).
  – It maintains bindings to the Web Services world by linking to WSDL descriptions.

• W3C Member Submission

OWL-S Conceptual Model – Service Class

• The class Service
  – provides an organizational point of reference for a declared Web service.

• The class has three properties:
  – presents
    – Defines what service does.
    – Points to the ServiceProfile instance.
  – supports
    – Defines how to access the described
      service.
    – Points to the ServiceGrounding instance.
  – describedBy
    – Defines how the service works.
    – Points to the ServiceModel instance.

Figure 1 – OWL-S Conceptual Model

Figure taken from David Martin et al. OWL-S: Semantic Markup for Web Services, W3C Member Submission 22 November 2004
**OWL-S Conceptual Model – Service Profile Class**

- **Expresses “what a service does”** for
  - Advertising purposes - used to express the service functional and non functional properties, and
  - Template for service requests.

- **Specification of functionality** provided by a service is given through:
  - Requirements that the service requester must satisfy to use the service successfully.
  - Inputs and outputs (OWL classes) and
  - Preconditions and results (format not fixed).

- **Non-functional properties of a service** are defined through:
  - Instances of the ServiceCategory class
  - an entry in some ontology or taxonomy of services.
  - Instances of the ServiceParameter class
  - an expandable list of arbitrary properties that may accompany a profile description.
  - A set of simple value attributes such as serviceName, textDescription, serviceProduct, serviceClassification (explained later).

**Figure 2 – OWL-S Service Profile**

Figure taken from David Martin et al. OWL-S: Semantic Markup for Web Services, W3C Member Submission 22 November 2004

---

**Functional properties**

- **hasParameter** - ranges over a Parameter instance from the Service Model ontology.
  - Note that the Parameter class models our intuition that Inputs and Outputs (which are kinds of Parameters) are both involved in information transformation and therefore they are different from Preconditions and Effects. As a consequence, we do not expect this class to be instantiated. Its role is solely making domain knowledge explicit.

- **hasInput** - ranges over instances of Inputs as defined in the Service Model ontology.

- **hasOutput** - ranges over instances of type Output, as defined in the Service Model ontology.

- **hasPrecondition** - specifies one of the preconditions of the service and ranges over a Precondition instance defined according to the schema in the Service Model ontology.

- **hasResult** - specifies one of the results of the service, as defined by the Result class in the Service Model ontology. It specifies under what conditions the outputs are generated. Furthermore, the result specifies what domain changes are produced during the execution of the service.

---

**Non-functional properties**

- **serviceParameter** - is an expandable list of properties that may accompany a profile description. The value of the property is an instance of the class ServiceParameter (next slide).

- **serviceCategory** - refers to an entry in some ontology or taxonomy of services. The value of the property is an instance of the class ServiceCategory (next slide).

- **serviceName** - refers to the name of the service that is being offered. It can be used as an identifier of the service.

- **textDescription** - provides a brief description of the service. It summarizes what the service offers, describes what the service requires to work, and indicates any additional information that the compiler of the profile wants to share with the receiver.

- **serviceClassification** - defines a mapping from a Profile to an OWL ontology of services, such as an OWL specification of NAICS.

- **serviceProduct** - defines a mapping from a Profile to an OWL ontology of products, such as an OWL specification of UNSPSC.


OWL-S Conceptual Model – Service Profile Class

- Non-functional properties (cont’d)
  - ServiceParameter class - additional attributes include the quality guarantees that are provided by the service, possible classification of the service, and additional parameters that the service may want to specify
    - serviceParameterName - is the name of the actual parameter, which could be just a literal, or perhaps the URI of the process parameter (a property)
    - sParameter - points to the value of the parameter within some OWL ontology
  - ServiceCategory class - describes categories of services on the bases of some classification
    - categoryName - is the name of the actual category, which could be just a literal, or perhaps the URI of the process parameter (a property)
    - taxonomy - stores a reference to the taxonomy scheme. It can be either a URI of the taxonomy, or a URL where the taxonomy resides, or the name of the taxonomy or anything else
    - value - points to the value in a specific taxonomy. There may be more than one value for each taxonomy, so no restriction is added here
    - code - to each type of service stores the code associated to a taxonomy

OWL-S Conceptual Model – Service Model Class

- Exposes “how a service works” to enable invocation, composition, monitoring, recovery, etc.
- The Service Model views interaction with the service as a process
  - The Process class represents the central class in the model:
    - Process generates and returns some new information based on the given information and the world state (inputs and outputs - slide # 23). 
    - Process produces a change in the world (preconditions and results - slide # 23).
  - A Process instance is characterized by the number of properties:
    - Zero or more inputs - representing the information that is, under some conditions, required for the performance of the process
    - Any number of outputs - the information that the process provides to the requester
    - Any number of preconditions - which must all hold in order for the process to be successfully invoked
    - Any number of effects - which must all hold after the process invocation
  - The Process class is further specialized into the:
    - Atomic process - a description of a service that expects one (possibly complex) message and returns one (possibly complex) message
    - Simple process - used for abstraction and simplification purposes
    - Composite process - maintains some state; each message the client sends advances it through the process

OWL-S Conceptual Model – Service Model Class – Process Classification

- Atomic processes
  - directly invocable (by passing them the appropriate messages)
  - no subprocesses and execute in a single step
  - they take an input message, do something, and then return their output message
  - provided a grounding that enables a service requester to construct messages to the process from its inputs and deconstruct replies
- Simple processes
  - not invocable and are not associated with a grounding
  - conceived of as having single-step executions
  - used as elements of abstraction
  - used to provide a view of (a specialized way of using) some atomic process
- Composite processes
  - decomposes into other (non-composite or composite) processes
  - their decomposition can be specified by using control constructs such as Sequence and If-Then-Else
  - a composite process is not a behavior a service will do, but a behavior (or set of behaviors) the client can perform by sending and receiving a series of messages
### Sequence
- A list of control constructs to be done in order

### Split
- The components of a Split process are a bag of process components to be executed concurrently. Split completes as soon as all of its component processes have been scheduled for execution.

### Split+Join
- Here the process consists of concurrent execution of a bunch of process components with barrier synchronization. That is, Split+Join completes when all of its component processes have completed. With Split and Split+Join, we can define processes that have partial synchronization (e.g. split all and join some sub-bag).

### Any-Order
- Allows the process components (specified as a bag) to be executed in some unspecified order but not concurrently. Execution and completion of all components is required. The execution of processes in an Any-Order construct cannot overlap; i.e. atomic processes cannot be executed concurrently and composite processes cannot be interleaved. All components must be executed. As with Split+Join, completion of all components is required.

### Choice
- Calls for the execution of a single control construct from a given bag of control constructs (given by the components property). Any of the given control constructs may be chosen for execution.

### If-Then-Else
- Control construct that has properties IfCondition, then and else holding different aspects of the If-Then-Else. Its semantics is intended as “Test If-condition; if True do Then, if False do Else.” (Note that the class Condition, which is a placeholder for further work, will be defined as a class of logical expressions.)

### Iterate
- Makes no assumption about how many iterations are made or when to initiate, terminate, or resume. The initiation, termination or maintenance condition could be specified with a whileCondition or an untilCondition.

### Repeat-While and Repeat-Until
- Both of these iterate until a condition becomes false or true, following the familiar programming language conventions.

### Maps the constructs of the process model to detailed specifications of message formats, protocols and so forth.

### Mapping of the atomic processes to WSDL operations and their I/Os to WSDL messages is supported.

### Mappings might have XSLT transformations attached to solve the lifting/lowering problem between OWL and XML Schema.

### Other grounding mechanisms can be supported

![Figure 4 - Mapping between OWL-S and WSDL](http://www.daml.org/services/owl-s/1.0/BravoAirGrounding.owl)

---

### Bravo Air - fictitious airline site
- The site is providing a service which enables reservation of trips as a composite process consisting of following steps
  - Retrieval of the desired flight details
  - Selection of the available flights, and
  - Booking of the selected flights, which includes following substeps
    - Login and
    - Confirmation of reservation.

### The example consists of the following artifacts
- BravoAirService.owl – OWL-S service,
- BravoAirProfile.owl – OWL-S ServiceProfile description,
- BravoAirProcess.owl – OWL-S ServiceModel description,
- BravoAirGrounding.owl – OWL-S ServiceGrounding description, and
- BravoAirGrounding.wsdl – annotated WSDL which points to the specifics OWL-S descriptions used during the processing (not exemplified here).

### Presented examples are lacking full details due to the space constrains.

---

---
This ontology represents the OWL-S service description for the BravoAir web service example.

```xml
<owl:Ontology rdf:about="">
  <rdfs:comment>This ontology represents the OWL-S service description for the BravoAir web service example.
  </rdfs:comment>
  <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.0/Service.owl"/>
  <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.0/BravoAirProfile.owl"/>
  <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl"/>
</owl:Ontology>
```

```xml
<service:Service rdf:ID="BravoAir_ReservationAgent">
  <!-- Reference to the BravoAir Profile -->
  <service:presents rdf:resource="http://www.daml.org/services/owl-s/1.0/BravoAirProfile.owl#Profile_BravoAir_ReservationAgent"/>
  <!-- Reference to the BravoAir Process Model -->
  <service:describedBy rdf:resource="http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl#BravoAir_ReservationAgent_ServiceModel"/>
  <!-- Reference to the BravoAir Grounding -->
  <service:supports rdf:resource="http://www.daml.org/services/owl-s/1.0/BravoAirGrounding.owl#Grounding_BravoAir_ReservationAgent"/>
</service:Service>
```

```xml
<owl:Ontology rdf:about="">
  <rdfs:comment>BravoAir Example for OWL-S Profile description
  </rdfs:comment>
  <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.0/Service.owl"/>
  ...<profile:serviceParameter>
    <addParam:GeographicRadius rdf:ID="BravoAir-geographicRadius">
      <profile:serviceParameterName>BravoAir Geographic Radius</profile:serviceParameterName>
      <profile:sParameter rdf:resource="http://www.daml.org/services/owl-s/1.0/Country.owl#UnitedStates"/>
    </addParam:GeographicRadius>
  </profile:serviceParameter>
  <profile:serviceCategory>
    <addParam:NAICS rdf:ID="NAICS-category">
      <profile:value>Airline reservation services</profile:value>
      <profile:code>561599</profile:code>
    </addParam:NAICS>
  </profile:serviceCategory>
  ...
  <profile:hasInput rdf:resource="http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl#DepartureAirport_In"/>
  <profile:hasOutput rdf:resource="http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl#AvailableFlightItineraryList_Out"/>
</owl:Ontology>
```

```xml
<process:ProcessModel rdf:ID="BravoAir_ReservationAgent_ProcessModel">
  <process:hasProcess rdf:resource="#BravoAir_Process"/>
  <service:describes rdf:resource="http://www.daml.org/services/owl-s/1.0/BravoAirService.owl#BravoAir_ReservationAgent"/>
</process:ProcessModel>

<process:CompositeProcess rdf:ID="BravoAir_Process">
  <process:composedOf>
    <process:Sequence>
      <process:components rdf:parseType="Collection">
        <process:AtomicProcess rdf:about="#GetDesiredFlightDetails"/>
        <process:AtomicProcess rdf:about="#SelectAvailableFlight"/>
        <process:CompositeProcess rdf:about="#BookFlight"/>
      </process:components>
    </process:Sequence>
  </process:composedOf>
</process:CompositeProcess>

<process:AtomicProcess rdf:ID="LogIn">
  <process:hasInput rdf:resource="#AcctName_In"/>
  <process:hasInput rdf:resource="#Password_In"/>
</process:AtomicProcess>
```

```xml
<process:Input rdf:ID="AcctName_In">
  <process:parameterType rdf:resource="http://www.daml.org/services/owl-s/1.0/Concepts.owl#AcctName"/>
</process:Input>

<process:Input rdf:ID="Password_In">
  <process:parameterType rdf:resource="http://www.daml.org/services/owl-s/1.0/Concepts.owl#Password"/>
</process:Input>
```
OWL-S
Example – Bravo Air Service Grounding Ontology

OWL-S
Example – Bravo Air Service Grounding Ontology

OWL-S
Tool support

• Reasoning
  – OWL-DL reasoners like Pellet and FaCT++/FaCT+
  – Rule-oriented extensions and embedded SWRL are used to overcome limitations of
    OWL-DL, as well as reasoners Pellet and FaCT++, and
  – Possibility to use metareasoning.

• Discovery
  – Match-making relies on the explicit specification of inputs and outputs.
  – Ontological relationships between the I/Os of "template" and candidate services are
    computed.
  – A set of five filters classify structural relationships between the template and candidate
    I/Os.
  – OWLS-MX matchmaker implements this classification and combines it with syntactic
    models.
  – There is no full usage of ICPS-based matchmaking, neither discovery approach which
    takes into account behavioral aspects of the service.

OWL-S
Tool support

• Choreography and Orchestration
  – OWL-S assumes invocation of services as atomic actions (choreography is not
    modeled).
  – Process-model allows services to be attached to the atomic but also composite
    processes thus allowing hierarchical control-flow oriented decomposition (i.e. ordering
    over atomic processes).
  – OWL-S VM executes the atomic processes up to the completion of the composite
    process description.
  – OWL-S VM fulfills the basic role of an orchestration engine.

• Mediation
  – No first-class support for data and process mediation.

• Composition
  – Abstraction of service interactions to atomic ones and its simple model of composite
    processes are good fit to existing (AI planning-based) techniques.
OWL-S Relationship to WSMO

- Underlying specification
  - OWL-S is specified using OWL, and WSMO uses abstract MOF model.
- Language unification
  - OWL-S needs to combine OWL with more expressive languages (e.g., for expressing conditions and workflow constructs) while WSMO provides a single unified language framework.
- Conceptual model similarities
  - OWL-S Service Profile is close to the capability of a service/goal in WSMO.
  - WSMO distinguishes requester’s and provider’s view.
- OWL-S lacks mediation facilities
- Grounding similarities
  - Both approaches adopt similar ideas with respect to binding to WSDL.

OWL-S Criteria discussion

- Web compliance
  - Supports URI, namespaces, XML
- Ontology-based
  - Based on OWL
- Strict decoupling
  - Ontologies and service descriptions can be specified independently
- Centrality of mediation
  - No support for mediation
- Ontological role separation
  - Doesn’t have notion of user desires
- Description vs. implementation
  - OWL-S is a description framework based on appropriate formalisms in order to provide concise semantic descriptions
- Execution semantics
  - Doesn’t have it
- Service vs. Web services

Technical solution

METEOR-S Overall approach

- METEOR-S project defines semantics for the complete lifecycle of Semantic Web processes including annotation, discovery, composition and enactments of Web Services.
- It is strongly coupled with existing Web Services standards, thus extending them with semantics.
- As opposed to the top-down approach embraced by WSMO and OWL-S, the project is promoting bottom-up approach.
- Conceptual model is simple and quite generic
  - Relying on Semantically Annotated WSDL (explained in next slide).
- Based on this simple model the project is providing a set of tools
  - Web Service Annotation Framework.
  - Web Service Discovery Infrastructure.
  - Web Service Composition Framework.

3 http://lsdis.cs.uga.edu/projects/meteor-s
The central point of METEOR-S approach is Semantically Annotated WSDL (SAWSDL).

- It is a way to add semantic annotations to various parts of a WSDL document:
  - Input and output message structures, interfaces, and operations.
  - Relying on simple extension attributes
    - Compliant to WSDL v2.0 and v1.1, and XML Schema.
- Annotations can be used for various purposes:
  - WSDL interfaces and operations with categorization information used to advertise Web services.
  - XML Schema types to foster discovery and composition.
  - Specifying the data mapping of XML Schema to/from an ontology used during invocation (possible mediation).
- Independent on the ontology expression language and mapping languages.

Extension attributes of SAWSDL are:
- modelReference
  - Pointers to a concept in some semantic model.
  - Annotates XML Schema type definitions, element declarations, and attribute declarations, WSDL interfaces, operations, and faults.
- liftingSchemaMapping
  - Added to XML Schema element declarations and type definitions for specifying lifting mappings between semantic data and XML.
- loweringSchemaMapping
  - Added to XML Schema element declarations and type definitions for specifying lowering mappings between semantic data and XML.

METEOR-S Tools

- Web Service Annotation Framework for semiautomatic annotation of Web Services’ semantics addressing four different aspects:
  - Semantic of the inputs and outputs of Web Services.
  - Functional semantics (“what service does”),
  - Execution semantics to support verification of the correctness of the Web Service executions, and
  - Inclusion of information regarding the quality of service (performance, costs, ...).
- Semiautomatic annotation is based on
  - Transformation of both XML Schema part of Web Service definitions and ontologies into a common representation – SchemaGraph, and
  - Matching algorithms which compute “match scores” between the SchemaGraph elements.

The framework comprises:
- Ontology store – ontologies to be used during Web Service annotation.
- Matcher library – algorithm implementations for linguistic and structural matching and
- Translator library – SchemaGraph generation procedures.
**METEOR-S**

**Tools – Annotation Framework - Example**

- **Web Service Discovery Infrastructure**
  - Attempts to enhance Web Service discovery infrastructure by using semantics.
  - Relies strongly on the existing UDDI Web Service registries.
  - Unifies access to the Web Service registries and ontology registries over P2P infrastructure.
  - User is expressing his requirements through the Service Templates which specify the inputs and outputs using concepts from the registered domain specific ontologies.
  - Semantic matchmaking is performed between the SAWSDL-enriched service descriptions and the Service Templates.

- **Web Service Composition Framework**
  - Web Service Composition Framework (MWSCF) aims to increase the flexibility of WS compositions by relying on semantic process templates.
  - Template defines process in terms of semantically defined activities (BPEL control-flow constructs with activities).
  - Template is used to generate executable processes by binding the semantically defined activities to concrete WS.

**METEOR-S**

**Tools – Composition Framework - Example**

**Relationship to WSMO**

- With its minimalistic approach can be viewed as orthogonal to WSMO.
- It can reference WSMO descriptions from its annotations.
- All the SAWSDL referenced concepts can be annotated in WSML/WSMO.
- SAWSDL doesn’t clearly establish how requests should be defined.
  - Usage of different languages hinder possibility to formally define requests, queries or notions of a “match” between service requests and service descriptions.
- Other WSMO supported notions such as nonfunctional properties and behavioral/interface descriptions should be addressed by the initiative in the same manner (i.e. extending WS-Policy, BPEL or WS-CDL with lightweight notions).
- SAWSDL has a strong industrial point.
  - WSMO/WS-L workgroups are looking at closer alignment with industry standards and adoption of SAWSDL as a possible future grounding formalism for WSMO.
METEOR-S
Criteria discussion

- Web compliance
  - Supports URI, namespaces, XML
- Ontology-based
  - Based on OWL and RDF-S
- Strict decoupling
  - Ontologies and service descriptions can be specified independently, connected through SAWSDL annotations
- Centrality of mediation
  - No support for mediation
- Ontological role separation
  - Doesn’t have notion of user desires
- Description vs. implementation
  - Follows a much more technology centered approach, not providing a conceptual model for the description of services and their related aspects
- Execution semantics
  - Doesn’t have it
- Service vs. Web services

Technical solution
Semantic Web Services Framework (SWSF)

SWSF
Overall Approach

- Semantic Web Services Framework (SWSF) has its roots in OWL-S and the Process Specification Language (PSL).
- It is based on two major components:
  - Ontology (Conceptual Model)
    - First-Order Logic Ontology for Web Services (FLOWS), and
  - Rules Ontology for Web Services (ROWS)
  - Language
    - SWL-FOL, and
    - SWL-Rules.
- It is a complex solution which hasn’t received much of the attention in the community
- General lack of tool support
- Comprehensive examples are missing

SWSF
Conceptual Model

- Semantic Web Service Ontology (SWSO)
  - Influenced by OWL-S (shares the same 3 concepts),
  - Extension/refinement of OWL-S,
  - Underlying language (SWL-S) is more expressive, and
  - Richer behavioral process model based on PSL.
- Two independent formalizations of the conceptual model
  - First-order Logic Ontology for Web Services (FLOWS), and
  - Rule Ontology for Web Services (ROWS)
  - Relies on the semantics of SWL-FOL
  - Relies on the semantics of SWL-Rules.
- In order to describe behavior of a service based on PSL approach two fundamental elements are added:
  - Structured notion of atomic processes, and
  - Infrastructure for specifying various forms of data flow.
SWSF Relationship to WSMO

- From the conceptual model point of view the same consideration apply like in the case of OWL-S.
- Many of the deficiencies of OWL-S have been overcome in SWSL:
  - Not being bound to the restricted expressivity of DL,
  - Providing rigid definitions of semantic aspects of conditions and dynamic aspects of SWS by reuse of PSL, and
  - Providing an single semantic language framework.

SWSF Criteria discussion

- Web compliance
  - Supports URI, namespaces, XML
- Ontology-based
- Strict decoupling
  - Not supported
- Centrality of mediation
  - No support for mediation
- Ontological role separation
  - Doesn’t have direct notion of user desires
- Description vs. implementation
  - Provides serialization/deserialization mechanisms to/from WSDL
- Execution semantics
  - Doesn’t have it
- Service vs. Web services

Technical solution
Internet Reasoning Service - III

Internet Reasoning Service (IRS-III) Overall Approach

- IRS III is a framework and implemented infrastructure which supports the creation of Semantic Web Services according to the WSMO ontology
- Builds upon the previous version (IRS-II) by incorporating and extending the WSMO ontology within the IRS-III server, browser and API.
Internet Reasoning Service (IRS-III)

Principal Characteristics

- Ontological separation of User and Web Service Contexts
  - Consumer and service exist in their own contexts which should be modeled with distinct ontological structures.

- Capability Based Invocation
  - Users can invoke a Web Service by specifying a concrete desired capability. Semantic descriptions in IRS-III are operational.

- Ease of Use
  - Interfaces are designed to hide the complexities surrounding the creation of SWS-based applications.

- One Click Publishing
  - Standalone code written in a standard programming language (Java or LISP) can be turned into a Web Service by only one-click.

- Agnostic to Service Implementation Platform
  - IRS-III is not making strict assumptions about the underlying service implementation platform.

  - It accepts the current dominance of Web Service stack of standards.

- Connected to External Environment
  - Functions and relations can make extra logical calls (e.g., invoking a Web Service) which results can be integrated smoothly with the internal reasoning.

- Open
  - IRS-III clients are based on publicly accessible Java APIs, components of the IRS-III server are Semantic Web Services which fosters flexibility.

- Complete Descriptions
  - Data which can be represented is represented. It is not possible a priori to know which specific data will be required for SWS related reasoning.

-Inspectable
  - Making the semantic descriptions accessible in a human readable form. The key is that the content and form are easily understandable by SWS application builders.

- Interoperable with SWS Frameworks and Platforms
  - IRS-III has an OWL-S import mechanism. It is interoperable with the WSMO reference implementation WSMX.

Internet Reasoning Service (IRS-III)

Features

- Based on SOAP messaging standard
- Provides Java API for client applications
- Provides built-in brokering and service discovery support
- Provides capability-centred service invocation
- Publishing support for variety of platforms
  - Java, Lisp, Web Applications, Java Web Services
- Enables publication of 'standard code'
  - Provides clever wrappers
- One-click publishing of web services
- Integrated with standard Web Services world
  - Semantic web service to IRS
  - 'Ordinary' web service

Internet Reasoning Service (IRS-III)

Architecture

Internet Reasoning Service (IRS-III)  
Comparison with WSMX

• Underlying language
  – In WSMX underlying language is WSML, while in IRS-III it is OCML.

• Goals
  – In IRS-III goals have inputs and outputs while in WSMX they have pre- and post-conditions.

• Discovery mechanism
  – IRS-III broker is finding applicable Web Services via mediators
  – Used mediator within IRS-capability
  – Mediator source = goal

WSMX doesn’t provide One-click-publishing

Summary

<table>
<thead>
<tr>
<th>WSMO</th>
<th>OWL-S</th>
<th>METEOR-S</th>
<th>SWSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

References

• Mandatory reading:
  – Dieter Fensel, Mike Kargl, Vitali Zuevsky (Eds.), Implementing Semantic Web Services: The SESA Framework. Springer-Verlag, 2008. (Chapter 13)

• Further reading:

• Wikipedia and links
<table>
<thead>
<tr>
<th>#</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Web Science</td>
</tr>
<tr>
<td>3</td>
<td>Service Science</td>
</tr>
<tr>
<td>4</td>
<td>Web services</td>
</tr>
<tr>
<td>5</td>
<td>Web 2 services</td>
</tr>
<tr>
<td>6</td>
<td>Semantic Web</td>
</tr>
<tr>
<td>7</td>
<td>Web Service Modeling Ontology (WSMO)</td>
</tr>
<tr>
<td>8</td>
<td>Web Service Modeling Language (WSML)</td>
</tr>
<tr>
<td>9</td>
<td>Web Service Execution Environment (WSX)</td>
</tr>
<tr>
<td>10</td>
<td>OWL-S and others</td>
</tr>
<tr>
<td>11</td>
<td>Light-weight Annotations</td>
</tr>
<tr>
<td>12</td>
<td>Applications</td>
</tr>
<tr>
<td>13</td>
<td>Mobile Services</td>
</tr>
</tbody>
</table>