Table of contents

(I) Introduction to Semantic Web Services (SWS)

(II) Semantic Web Services
  • OWL-S & WSMO
  • OWL-S and WSMO - Design decisions and trade-offs

#Q&A, Coffee break#

(III) Semantic Web Services implementations
  • OWL-S
  • WSMX
  • IRS – III – bridge implementation between OWL-S & WSMO

(IV) Summary, Conclusions & Future Work
Introduction to Semantic Web Services

Laurentiu Vasiliu

Contributors: Sinuhe Arroyo, Christoph Bussler
Semantic Web Services = Semantic Web Technology + Web Service Technology
Semantic Web Services

Web Services: [Stencil Group]
- loosely coupled, reusable components
- semantically encapsulate discrete functionality
- distributed
- programmatically accessible over standard internet protocols
- add new level of functionality on top of the current web
Semantic Web Services (2)

Semantic Web:
• ontologies - basic building block
• allow machine supported data interpretation

Semantic Web Services:
• will allow the automatic publication, discovery, selection, composition, mediation and execution of inter-organization business logic
• Internet to become a global common platform to support SWS applications
Semantic Web Services

Usage Process:
• **Publication**: Make available the description of the capability of a service
• **Discovery**: Locate different services suitable for a given task
• **Selection**: Choose the most appropriate services among the available ones
• **Composition**: Combine services to achieve a goal
• **Mediation**: Solve mismatches (data, protocol, process) among the combined
• **Execution**: Invoke services following programmatic conventions
Semantic Web Services

Usage Process – execution support

- Monitoring: Control the execution process
- Compensation: Provide transactional support and undo or mitigate unwanted effects
- Replacement: Facilitate the substitution of services by equivalent ones
- Auditing: Verify that service execution occurred in the expected way
Lack of SWS standards
Lack of SWS standards
Lack of SWS standards

Current technology does not allow realization of any of the parts of the Web Services’ usage process:

- Only syntactical standards available
- Lack of fully developed markup languages
- Lack of marked up content and services
- Lack of semantically enhanced repositories
- Lack of frameworks that facilitate discovery, composition and execution
- Lack of tools and platforms that allow to semantically enrich current Web content
# Table of contents

(I) Introduction to Semantic Web Services (SWS)

(II) Semantic Web Services
- OWL-S & WSMO
- OWL-S and WSMO - Design decisions and trade-offs

#Q&A, Coffee break#

(III) Semantic Web Services implementations
- OWL-S
- WSMX
- IRS – III – bridge implementation between OWL-S & WSMO

(IV) Summary, Conclusions & Future Work
OWL-S & WSMO (II)

Semantic Web Services Concepts
OWL-S Ontology

Katia Sycara
Massimo Paolucci
David Martin
OWL-S Ontology

• OWL-S is an OWL ontology to describe Web services
• OWL-S leverages on OWL to
  – Support capability based discovery of Web services
  – Support automatic composition of Web Services
  – Support automatic invocation of Web services

*Complete do not compete*
  – OWL-S does not aim to replace the Web services standards
    rather OWL-S attempts to provide a semantic layer
  • OWL-S relies on WSDL for Web service invocation (see Grounding)
  • OWL-s Expands UDDI for Web service discovery (OWL-S/UDDI mapping)
OWL-S Upper Ontology

- Mapping to WSDL
  - communication protocol (RPC, HTTP, ...)
  - marshalling/serialization
  - transformation to and from XSD to OWL
- Control flow of the service
  - Black/Grey/Glass Box view
  - Protocol Specification
  - Abstract Messages

- Capability specification
- General features of the Service
  - Quality of Service
  - Classification in Service taxonomies
Service Profiles

- Presented by a service.
- Represents what the service provides
- Two main uses:
  1. Advertisements of Web Services capabilities
  2. Request of Web services with a given set of capabilities
OWL-S Profile in a Nutshell

- Describes Web service
  - What capabilities it provides:
    - What transformation the service computes
    - Type of service and products
  - General features such as
    - Agent providing the service
    - Security requirements
    - Quality guarantees of service
- Primary role: to assist discovery
  - Allows capability based search
  - Allows selection based on requirements of the requester
- Profile does not specify use/invocation
OWL-S Service Profile
Capability Description

- **Preconditions**
  - Set of conditions that should hold prior to service invocation
- **Inputs**
  - Set of necessary inputs that the requester should provide to invoke the service
- **Outputs**
  - Results that the requester should expect after interaction with the service provider is completed
- **Effects**
  - Set of statements that should hold true if the service is invoked successfully.
- **Service type**
  - What kind of service is provided (eg selling vs distribution)
- **Product**
  - Product associated with the service (eg travel vs books vs auto parts)
OWL-S Service Profile

Additional Properties

- **Security Parameters**
  - Specify the security capabilities of a Web service (e.g., support X509 Encryption)
  - Specify the security requirements of a Web service (e.g., a client should be able to provide X509 Encryption)

- **Quality rating**
  - What level of service quality does the Web service provide?

- **Description with standard business taxonomies**
  - How would the service be classified in standard taxonomies such as UNSPSC or NAICS?

This is not a closed set, new properties can be added using existing ontologies
Process Model

- Process Model
  - Describes how a service works: internal processes of the service
  - Specifies service interaction protocol
  - Specifies abstract messages: ontological type of information transmitted

Facilitates
- Web service invocation
- Composition of Web services
- Monitoring of interaction
Viewpoints of Process Model

- Three viewpoints of a Web service
  - **Glass Box:**
    - The Web service reveals all its internal structure
    - Which parts of the service it performs in-house, which one it subcontracts, etc
  - **Black Box:**
    - The Web service model does not reveal anything about the internal working of the service
    - It just specifies what data it gathers and what data it sends back
  - **Grey Box:**
    - The Web service selectively hides some parts of its Process Model, while it publicizes others
Definition of Process

- A Process represents a transformation (function). It is characterized by four parameters
  - **Inputs**: the inputs that the process requires
  - **Preconditions**: the conditions that are required for the process to run correctly
  - **Outputs**: the information that results from (and is returned from) the execution of the process
  - **Results**: a process may have different outcomes depending on some condition
    - **Condition**: under what condition the result occurs
    - **Constraints on Outputs**
    - **Effects**: real world changes resulting from the execution of the process
Motivation for Results

• Processes may terminate in exceptional states:
  – The credit company may fail to charge the credit card
  – The book may be out of stock
  – The deliver of the goods may fail

• Results support modeling of non-deterministic outcomes of Web services
  – The condition specifies when an outcome is generated
  – Each outcome is characterized by
    • a set of constraints on outputs
    • a set of effects
Example of Process

<pre>Example of Process</pre>

```xml
<process:AtomicProcess rdf:ID="LogIn">
  <process:hasInput rdf:resource="#AcctName"/>
  <process:hasInput rdf:resource="#Password"/>
  <process:hasOutput rdf:resource="#Ack"/>
  <process:hasPrecondition isMember(AccName)/>
  <process:hasResult>
    <process:Result>
      <process:inCondition>
        <expr:SWRL-Condition>
          correctLoginInfo(AccName,Password)
        </expr:SWRL-Condition>
      </process:inCondition>
      <process:withOutput rdf:resource="#Ack">
        <valueType rdf:resource="#LoginAcceptMsg"/>
      </process:withOutput>
      <process:hasEffect>
        <expr:SWRL-Condition>
          loggedIn(AccName,Password)
        </expr:SWRL-Condition>
      </process:hasEffect>
    </process:Result>
  </process:hasResult>
</process:AtomicProcess>
```
Ontology of Processes

Process
  Atomic
    Invokable bound to grounding
  Simple
    Provides abstraction, encapsulation etc.
  Composite
    Defines a workflow composed of process performs
Process Model Organization

• **Process Model is described as a tree structure**
  – Composite processes are internal nodes
  – Simple and Atomic Processes are the leaves

• **Simple processes represent an abstraction**
  – Placeholders of processes that aren’t specified
  – Or that may be expressed in many different ways

• **Atomic Processes correspond to the basic actions that the Web service performs**
  – Hide the details of how the process is implemented
  – Correspond to WSDL operations
Composite Processes

• Composite Processes specify how processes work together to compute a complex function
• Composite processes define

1. Control Flow
   Specify the temporal relations between the executions of the different sub-processes

2. Data Flow
   Specify how the data produced by one process is transferred to another process
Example of Composite Process

Control Flow Links
Specify order of execution

Data-Flow Links
Specify transfer of data

Perform statements
Specify the execution of a process

Sequence
BookFlight

Perform
Get Flights

Perform
Select Flight

Airline

Flights

Arrive

Depart

Flights
Perform Construct

• *Perform* provides invocation mechanism
  – Specify context of process execution
    • input data flow
    • hooks for output data flow
• Distinction between definition and invocation of a process
  – Definition specifies the process’ I/P/R
  – Perform specify when the process is invoked and with what parameters
Control Flow

- Processes can be chained to form a workflow
- OWL-S supports the following control flow constructs
  - **Sequence/Any-Order**: represents a list of processes that are executed in sequence or arbitrary order
  - **Conditionals**: if-then-else statements
  - **Loops**: while and repeat-until statements
  - **Multithreading and synchronization**: split process in multiple threads, and rendezvous (joint) points
  - **Non-deterministic choices**: (arbitrarily) select one process of a set
Data Flow

Dataflow allows information that is transferred from process to process.

Output → Input:
The information produced by one process is transferred to another in the same control construct.

Input → Input:
The information received by a composite process is transferred to the sub-processes.

Output → Output:
The information produced by a subprocess is transferred to a super-process.
Process Model: take home lesson

• Service Model describes
  – Set of processes that define the operations performed by the Web service
  – Control flow describing the temporal flow of processes
  – Data flow describing the transfer of information between sub-processes
Service Grounding

- Service Grounding
  - Provides a specification of service access information.
  - Service Model + Grounding give everything needed for using the service.
  - Builds upon WSDL to define message structure and physical binding layer.

- Specifies:
  - communication protocols, transport mechanisms, communication languages, etc.
Rationale of Service Grounding

• Provides a specification of service access information.

• Service Model + Grounding give everything needed for using the service
  – Service description is for reasoning about the service
    • Decide what information to send and what to expect
  – Service Grounding is for message passing
    • Generate outgoing messages, and get incoming messages
    • Mapping XML Schemata to OWL concepts

• Builds upon WSDL to define message structure and physical binding layer
Mapping OWL-S / WSDL 1.1

- Operations correspond to Atomic Processes
- Input/Output messages correspond to Inputs/Outputs of processes
Example of Grounding

Sequence
BookFlight

Perform
Airline
Get Flights
Depart
Arrive

Perform
Select Flight
Flights

WSDL
Get Flights Op
Depart
Arrive
Airline
Flights

Select Flight op
Flights

Airline
Flight

Flight
Perform
Get Flights
Flights

Select Flight op
Flight
Result of using the Grounding

• **Invocation mechanism for OWL-S**
  – Invocation based on WSDL
  – Different types of invocation supported by WSDL can be used with OWL-S

• **Clear separation between service description and invocation/implementation**
  – Service description is needed to reason about the service
    • Decide how to use it
    • Decide how what information to send and what to expect
  – Service implementation may be based on SOAP an XSD types
  – The crucial point is that the information that travels on the wires and the information used in the ontologies is the same

• **Allows any web service to be represented using OWL-S**
  – For example: Amazon.com
Handling stateful vs stateless Web services

1. Stateless Web services
   • The server does not maintain the state of the computation
   • Dataflow links specify how the client communicate the state to the service

2. Stateful Web services
   • The service does maintain the state
   • No need of dataflow links since transfer of information is opaque to the client
Representing Stateful Web services

Stateless: no information is transferred between the two operations
Representing Stateless Web services

1. **Client**
   - Airline 
   - Sequence: BookFlight 
   - Perform: Get Flights
   - Perform: Select Flight

2. **Server**
   - Get Flights Op
   - Select Flight op
   - Stateful: information is recorded by the server, no need of transfer between the two operations
Conclusion  OWL-S section

• OWL-S provides a language for the description of Web services
  – Service Profile provides description of capabilities of Web Service
    • Allows capability-based discovery
  – Process Model provides the description of how to use a Web service
    • Allows automatic invocation of Web service
  – Service Grounding maps Atomic Processes into WSDL operations
    • Allows separation between description and implementation
    • Supports description of arbitrary Web services
WSMO

Web Service Modeling Ontology

Michael Stollberg

Contributors: Dumitru Roman, Holger Lausen, Rubén Lara, Axel Pollerers
Features

- WSMO is a complete conceptual model for Semantic Web Services and related aspects
- WSMO is derived from and based on the Web Service Modeling Framework WSMF
- WSMO is a SDK-Cluster Working Group
Outline

• WSMO Working Groups
• WSMO Design Principles
• WSMO Top Level Notions
  – Ontologies
  – Goals
  – Web Services
  – Mediators
• Walk-Thru Example
WSMO Working Groups

A Conceptual Model for SWS

A Formal Language for WSMO

A Rule-based Language for SWS

Execution Environment for WSMO

SdK
WSMO Design Principles

Strong Decoupling & Strong Mediation

*autonomous components with mediators for interoperability*

Interface vs. Implementation

*distinguish interface (= description) from implementation (= program)*

Peer to Peer

*interaction between equal partners (in terms of control)*

Execution Semantics

*reference implementation (WSMX)*
WSMO Top Level Notions

Objectives that a client may have when consulting a Web Service

Provide the formally specified terminology of the information used by all other components

Semantic description of Web Services:
- Capability (functional)
- Interfaces (usage)

Connectors between components with mediation facilities for handling heterogeneities

WSMO D2, version 1.0, 20 September 2004
Non-Functional Properties

- Every WSMO elements is described by properties that contain relevant, non-functional aspects of the item
- used for management and element overall description
- **Core Properties:**
  - Dublin Core Metadata Element Set plus version (evolution support)
  - W3C-recommendations for description type
- **Web Service Specific Properties:**
  - quality aspects and other non-functional information of Web Services
  - used for Service Selection
Non-Functional Properties

ontology <http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/dt.wsml>

dc:title "Date and Time Ontology"
dc:creator "DERI International"
dc:subject "Date", "Time", "Date and Time Algebra"
dc:description "generic representation of data and time including basic algebra"
dc:publisher "DERI International"
dc:contributor "Holger Lausen", "Axel Polleres", "Ruben Lara"
dc:date 2004-06-28
dc:type http://www.wsmo.org/2004/d2/v0.3/20040329/#ontos
dc:format "text/plain"
dc:language "en-US"
dc:relation <http://www.isi.edu/~pan/damltime/time-entry.owl>,
<http://www.w3.org/TR/xmlschema-2/>
dc:coverage "World"
dc:rights <http://www.deri.org/privacy.html>
version 1.21
WSMO Ontologies

Objectives that a client may have when consulting a Web Service

- Provide the formally specified terminology of the information used by all other components

Semantic description of Web Services:
- **Capability** (functional)
- **Interfaces** (usage)

Connectors between components with mediation facilities for handling heterogeneities
Ontology Specification

- **Non functional properties** (see before)
- **Imported Ontologies** importing existing ontologies where no heterogeneities arise
- **Used mediators:** OO Mediators (ontology import with terminology mismatch handling)

- **‘Standard’ Ontology Notions:**
  - **Concepts** set of concepts that belong to the ontology, incl.
  - **Attributes** set of attributes that belong to a concept
  - **Relations:** define interrelations between several concepts
  - **Functions:** special type of relation (unary range = return value)
  - **Instances:** set of instances that belong to the represented ontology
  - **Axioms** axiomatic expressions in ontology (logical statement)
WSMO Goals

Objectives that a client may have when consulting a Web Service

Provide the formally specified terminology of the information used by all other components

Connectors between components with mediation facilities for handling heterogeneities

Semantic description of Web Services:
- **Capability** (functional)
- **Interfaces** (usage)
Goals

• **De-coupling of Request and Service**
  - **Goal-driven Approach**, derived from AI rational agent approach
    - Requester formulates objective independent / without regard to services for resolution
    - ‘Intelligent’ mechanisms detect suitable services for solving the Goal
    - Allows re-use of Services for different purposes

• **Usage of Goals within Semantic Web Services**
  - A Requester, that is an agent (human or machine), defines a Goal to be resolved
  - Web Service Discovery detects suitable Web Services for solving the Goal automatically
  - Goal Resolution Management is realized in implementations
Goal Specification

- Non functional properties
- Imported Ontologies
- Used mediators
  - **OO Mediators:** for importing ontologies with integration
  - **GG Mediator:**
    - Goal definition by reusing an already existing goal
    - Allows specification of **Goal Ontologies**

- Post-conditions
  Describe the state of the information space that is desired.
  - The result expected from execution a Web Service
  - Expressed as an axiom (unambiguous, based on ontology)

- Effects
  Describe the state of the world that is desired.
  - Expected changes in the world that shall hold after a service execution
  - Expressed as an axiom (unambiguous, based on ontology)
WSMO Standard

WSMO Web Services

Objectives that a client may have when consulting a Web Service

Provide the formally specified terminology of the information used by all other components

Semantic description of Web Services:
- **Capability** *(functional)*
- **Interfaces** *(usage)*

Connectors between components with mediation facilities for handling heterogeneities
WSMO Web Service Description

- complete item description
- quality aspects
- Web Service Management

- Advertising of Web Service
- Support for WS Discovery

Non-functional Properties

Core + WS-specific

Capability

functional description

Interaction Interface for consuming WS
- Messages
- External Visible Behavior
- ‘Grounding’

Web Service Implementation
(not of interest in Web Service Description)

Realization of WS by using other Web Services
- Functional decomposition
- WS Composition

Choreography --- Interfaces --- Orchestration
Web Service specific Properties

- non-functional information of Web Services:
  
  Accuracy  Robustness
  Availability  Scalability
  Financial  Security
  Network-related QoS  Transactional
  Performance  Trust
  Reliability
Capability Specification

- Non functional properties
- Imported Ontologies
- Used mediators
  - **OO Mediator**: importing ontologies as terminology definition
  - **WG Mediator**: link to a Goal that is solved by the Web Service
- Pre-conditions
  What a web service expects in order to be able to provide its service. They define conditions over the input.
- Assumptions
  Conditions on the state of the world that has to hold before the Web Service can be executed
- Post-conditions
  describes the result of the Web Service in relation to the input, and conditions on it
- Effects
  Conditions on the state of the world that hold after execution of the Web Service (i.e. changes in the state of the world)
Choreography in WSMO

“Interface of Web Service for client-service interaction when consuming the Web Service”

• External Visible Behavior
  – those aspects of the workflow of a Web Service where User Interaction is required
  – described by process / workflow constructs

• Communication Structure
  – messages sent and received
  – their order (messages are related to activities)
Choreography in WSMO (2)

• **Grounding**
  – concrete communication technology for interaction
  – choreography related errors (e.g. input wrong, message timeout, etc.)

• **Formal Model**
  – allow operations / mediation on Choreographies
  – Formal Basis: Abstract State Machines (ASM)
Choreography & Mediation

**Aim:** support collaboration of multiple Web Services

**Future Work:**
- Language and Formal Model for multi-party Choreographies
  - Specification of **Global Interaction Protocols**
  - related: **WS-CDL** (W3C WS Choreography Working Group)
- Protocol and Process Mediation Facilities
  - formal model for operations on Choreography Interfaces
  - related: Process Algebra, PI Calculus, Petri Nets
WSMO Orchestration

“Achieve Web Service Functionality by aggregation of other Web Services”

- **Orchestration Language**
  - decomposition of Web Service functionality
  - control structure for aggregation of Web Services

- **Web Service Composition**
  - Combine Web Services into higher-level functionality
  - Resolve mismatches occurring between composed Web Services

- **Proxy Technology**
  - Placeholders for used Web Services
  - Facility for applying the Choreography of used Web Services
WSMO Orchestration Overview

**decomposition** of the Web Service functionality into sub-functionalities

**Proxies** as placeholders for used Web Services

**Control Structure for aggregation of other Web Services**
WSMO Mediators

Objectives that a client may have when consulting a Web Service

- Semantic description of Web Services:
  - Capability (functional)
  - Interfaces (usage)

Connectors between components with mediation facilities for handling heterogeneities

Provide the formally specified terminology of the information used by all other components

Goals

Ontologies

Mediators

Web Services
Mediation

• **Heterogeneity** …
  – Mismatches on structural / semantic / conceptual / level
  – Occur between different components that shall interoperate
  – Especially in distributed & open environments like the Internet

• **Concept of Mediation** (Wiederhold, 94):
  – *Mediators* as components that resolve mismatches
  – **Declarative Approach:**
    • Semantic description of resources
    • ‘Intelligent’ mechanisms that resolve mismatches independent of content
  – Mediation cannot be fully automated (integration decision)

• **Levels of Mediation within Semantic Web Services** (WSMF):
  (1) **Data Level:** mediate heterogeneous *Data Sources*
  (2) **Protocol Level:** mediate heterogeneous *Communication Patterns*
  (3) **Process Level:** mediate heterogeneous *Business Processes*
WSMO Mediators Overview
Mediator Structure

WSMO Mediator
uses a Mediation Service via

Mediation Services

Source Component

Target Component

Source Component

- as a Goal
- directly
- optionally incl. Mediation

1 .. n
OO Mediator - Example

Merging 2 ontologies

Train Connection Ontology (s1)

Purchase Ontology (s2)

OO Mediator Mediation Service

Goal:
“merge s1, s2 and s1.ticket subclassof s2.product”

Discovery

Mediation Services

Train Ticket Purchase Ontology
GG Mediators

• **Aim:**
  – Support specification of Goals by re-using existing Goals
  – Allow definition of *Goal Ontologies* (collection of pre-defined Goals)
  – Terminology mismatches handled by OO Mediators

• **Example: Goal Refinement**

  ![Diagram](Image)

  - **Source Goal**
    
    “Buy a ticket”

  - **Target Goal**
    
    “Buy a Train Ticket”

  - **postcondition:**
    
    “aTicket memberof trainticket”
WG & WW Mediators

• WG Mediators:
  – link a Web Service to a Goal and resolve occurring mismatches
  – match Web Service and Goals that do not match a priori
  – handle terminology mismatches between Web Services and Goals
    ⇒ broader range of Goals solvable by a Web Service

• WW Mediators:
  – enable interoperability of heterogeneous Web Services
    ⇒ support automated collaboration between Web Services
  – **OO Mediators** for terminology import with data level mediation
  – Protocol Mediation for establishing valid multi-party collaborations
  – Process Mediation for making Business Processes interoperable
WSMO Walk-Thru Example

• Use Case
  – Buy a train ticket to travel from Innsbruck, Austria to Frankfurt, Germany.
  – Departure Date: 11 November 2004
  – Departure Time: after 6 p.m.

• Show:
  – modeling of WSMO components
  – service usage process
Use Case Overview

how does the interplay of the Customer, VTA, and the other Web Services look like?
Goal Specification - Example

Goal Postcondition
„I want to buy a train ticket from Innsbruck to Frankfurt on 11/11/04, after 6 p.m.”

postcondition
axiom buyATicketForItinerary

nonFunctionalProperties
dc:description “defines the desire expressed in the Goal"
definedBy

?Ticket[
  trip hasValue someTrip[
    start hasValue innsbruck
    end hasValue frankfurt
    departure hasValue myDeparture[
      date hasValue 2004-11-11,
      time hasValue 18-00] memberOf dt:dateandtime
    ] memberOf tc:trainTrip,
  passenger hasValue aPassenger memberOf loc:person,
  ] memberOf tc:ticket .
Capability - Example

Postcondition (returns a ticket for a train trip with constraints)

```
postcondition
nonFunctionalProperties
  dc:description "the output of the service with constraints"
definedBy
?Ticket[
  trip hasValue ?Trip[
    start hasValue ?Start,
    end hasValue ?End,
    departure hasValue ?Departure
  ]memberOf tc:trainTrip and
  passenger hasValue ?Passenger memberOf tc:ticket and
]memberOf tc:ticket and
(?Start.locatedIn = austria or ?Start.locatedIn = germany) and
(?End.locatedIn = austria or ?End.locatedIn = germany) and
?Departure > currentDate().
```
Step 1: Goal Definition and Web Service Discovery

Goal: “I want to buy a train ticket from Innsbruck to Frankfurt on 11th November 2004, departure later than 6 p.m.”
Web Service Interfaces

Choreography
- request: buyer information, itinerary
- input not valid
- no valid connection
- set of valid itineraries
- itinerary
- purchase proposition
- option selection OR accept OR not accept
- request payment information
- payment information
- payment information incorrect
- successful purchase

invocation

connection choice

contract of purchase

payment & delivery

Orchestration

connection choice

payment & delivery

TimeTable

Composition

Payment

Delivery
Service Usage I: “Invocation”

Customer

Invocation Message
incl. Input-Information (Buyer, Itinerary)

CI

VTA

Choreography Side Orchestration Side
Service Usage II: “Connection Choice”

- Customer
- TimeTable
- VTA
- P
- CI
- Choreography Side
- Orchestration Side

- REQ: valid itineraries
- RES: set of itineraries
- INF: set of itineraries
- INF: itineraries
Service Usage III: “Contract of Purchase”

Customer

INF: Purchase Proposition
incl. all purchase contract information
INF: Proposition Option Selection
INF: Purchase Offer Acceptance

repeat until acceptance

Choreography Side

VTA

Orchestration Side

time
Service Usage IV: “Payment & Delivery”

Customer

Choreography Side

CI

ERR: creditcard invalid

REQ: creditcard info

RES: creditcard info

INF: successful purchase

time

Orchestration Side

VTA

CI

ERR: creditcard invalid

REQ: payment incl. item, creditcard

RES: payment OK

REQ: delivery incl. item, ship-address

ACK: delivery OK

CI

Payment

Delivery

REQ: creditcard info

RES: creditcard info

REQ: payment incl. item, creditcard

RES: payment OK

ACK: delivery OK
Table of contents

(I) Introduction to Semantic Web Services (SWS)
(II) Semantic Web Services
   • OWL-S & WSMO
   • OWL-S and WSMO - Design decisions and trade-offs

#Q&A, Coffee break#

(III) Semantic Web Services implementations
   • OWL-S
   • WSMX
   • IRS – III – bridge implementation between OWL-S & WSMO

(IV) Summary, Conclusions & Future Work
OWL-S and WSMO

Design decisions and tradeoffs

Katia Sycara    Ruben Lara

David Martin (presenter for Katia)
Contributors: Massimo Paolucci, WSMO team
OWL-S vs WSMO Perspective

- OWL-S is an ontology and a language to describe Web services
  - The guiding lines for the development of OWL-S have been
    - Strong relation to Web Services standards
      - Rather than proposing another WS* standard, OWL-S aims at enriching existing standards
      - OWL-S is grounded in WSDL and it has been mapped into UDDI
    - Based on the Semantic Web
      - Ontologies provide conceptual framework to describe the domain of Web services and an inference engine to reason about the domain
      - Ontologies are essential elements of interoperation between Web services
    - Build upon 30 years of AI research on Knowledge Representation and Planning
OWL-S vs WSMO Perspective

- WSMO provides a **conceptual model** for Web Services and related aspects
  - WSMO separates the different **language specifications layers** (MOF style)
    - Language for defining WSMO is the meta – meta - model in MOF
    - WSMO and WSML are the meta - models in MOF
    - Actual goals, web services, etc. are the model layer in MOF
    - Actual data described by ontologies and exchanged is the information layer in MOF
  - Stress on **solving the integration problem**
    - Mediation as a key element
  - Languages to cover wide range of scenarios and improve **interoperability**
  - Relation to industry **WS standards**
  - All the way from conceptual modelling to usable **implementation**
Web Services Problems

• Web services as loosely coupled components that work through collaboration
• WS interaction requires:
  – Discovery
    • How are Web services found and selected?
  – Composition
    • How to make different Web services work together?
  – Invocation
    • How is data transformed to fit the requirement of the partner Web service?
  – Guaranteeing Security and Policies
    • How are the partners requirements satisfied?
  – Mediation and Interoperation
    • How are data and protocol mismatches resolved?
Problems of Discovery

• Discovery requires
  – Infrastructure that allows storage and retrieval of information about Web services
    • For example a UDDI server
  – Description of capabilities of Web services
  – Description of requests or goals
  – Algorithms for matching requesters for capabilities with the corresponding providers
OWL-S vs WSMO Discovery

• OWL-S Profile provides capability description & request
  – Functional capabilities (what the Web services does)
  – Quality parameters (how the Web service does it)
  – Capability description & request are both Profile-based
  – OWL-S reliance on OWL provides (one type of) matching
  – It can be mapped to UDDI or used in other architectures such as brokering or P2P

• WSMO separates requester and provider viewpoints
  – WSMO goals describe requester objectives
  – WSMO capabilities describe WS functionality
  – Non-functional properties used for security, trust, etc.
  – Different steps in service discovery
  – Different approaches to web service discovery
    • Cover a wide range of scenarios!
Differences between OWL-S and WSMO

OWL-S profile ≈ WSMO capability + goal + non-functional properties

• Request
  – OWL-S uses Profiles to express existing capabilities (advertisements) and desired capabilities (requests)
  – WSMO separates provider (capabilities) and requester points of view (goals)
• Conceptually, OWL-S requested profile and WSMO goal are not exactly the same
  – Requested service profile vs requester objectives
Problems of Composition

• No single Web service may achieve all goals of an agent
  – Composition is the process of chaining results from different Web services automatically

• Planning problem
  – How do the Web services fit together?

• Interoperation problem
  – How does the information exchanged fit together?
  – How is this information interpreted by the end points?
OWL-S vs WSMO Composition

• OWL-S WS composition based on Process Model
  – Processes are modeled as planning operators
  – OWL-S does not provide a “pure” choreography language, but Process Model can be used as a highly flexible choreography language for the description of WS protocols
  – Multi party orchestration is not modeled directly; it results from a planning process driven by the goals of the main actor and that involves the Process Models of all the participants

• WSMO enables automatic, semiautomatic and fixed composition
  – Automatic composition based on planning and use of WS capabilities and choreographies
  – Orchestration can define the use of other WSs
    • Fixed WSs
    • Proxies (goals) to be resolved at run-time
Differences between OWL-S and WSMO

OWL-S Process Model ≈ WSMO Choreography

• Differences:
  – WSMO provides choreography + orchestration while OWL-S provides only choreography and facilitates automatic orchestration
  – WSMO allows multiple choreographies
  – WSMO choreography will come with ASM-based formal semantics
  – OWL-S formal semantics has been developed in very different frameworks such as Situation Calculus, Petri Nets, Pi-calculus
  – OWL-S Process Model more mature than WSMO choreography
Problem of Invocation

- Invocation requires the mapping of abstract “semantically based” descriptions into data-exchanges with partner services
  - Specification of what information is required
  - Transformation into a data-format that the server understands
  - Resolve process and protocol heterogeneity
  - Accommodate a different granularity of description
  - Interpretation of the information received using the available ontologies
  - Dealing with server failures
OWL-S vs WSMO Grounding

OWL-SGrounding ≈ WSMO Grounding

- OWL-S and WSMO provide default mapping to WSDL
  - Clear separation between WS description and interface implementation
  - Other mappings could be used
  - OWL-S Grounding is more mature than WSMO’s
## Relation to Web Services Technology

<table>
<thead>
<tr>
<th></th>
<th>OWL-S</th>
<th>WSMO</th>
<th>Web Services Infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Discovery</strong></td>
<td>Profile</td>
<td>Web Services (capability)</td>
<td>UDDI API</td>
</tr>
<tr>
<td><em>What it does</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Choreography</strong></td>
<td>Process Model</td>
<td>Orchestration + choreography</td>
<td>BPEL4WS</td>
</tr>
<tr>
<td><em>How is done</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Invocation</strong></td>
<td>Grounding+ WSDL/SOAP</td>
<td>Grounding</td>
<td>WSDL/SOAP</td>
</tr>
<tr>
<td><em>How to invoke</em></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- OWL-S and WSMO map to UDDI API adding semantic annotation
- OWL-S and WSMO share a default WSDL/SOAP Grounding
- BPEL4WS could be mapped into WSMO orchestration and choreography
- Mapping still unclear at the level of choreography/orchestration
  - In OWL-S, multi-party interaction is obtained through automatic composition and invocation of multiple parties
  - BPEL allows hardcoded representation of many Web services in the same specification.
  - Trade-off: OWL-S support substitution of Web services at run time, such substitution is virtually impossible in BPEL.
Mediation and Interoperation

- Interaction of Web services is bound to produce many forms of mismatch
  - **Data mismatch**: the interacting parties do not agree on the data format that they are using
  - **Ontology mismatch**: the interacting parties refer to different ontologies
  - **Protocols mismatch**: the interacting parties expect information at different times
  - **Goals Mismatch**: the interacting parties attempt to achieve very different goals
  - **Interpretations Mismatch**: The interacting parties interpret the same information in very different ways

- These mismatches need to be reconciled for the interoperation to succeed.
- **Mediators are the components that reconcile these mismatches**
Mediators in OWL-S and WSMO

• OWL-S does not have an explicit notion of mediator
  – Mediation is a by-product of the orchestration process
    • For example protocol mismatches are resolved by constructing a plan that coordinates the activity of the Web services
  – …or it results from translation axioms that are available to the Web services
    • It is not the mission of OWL-S to generate these axioms

• WSMO regards mediators as key conceptual elements
  – Different kinds of mediators modelled
    • Refiners and bridges
  – Reusable mediators
  – Mediation mechanism not dictated
    • E.g. Rules or WS invocation
Differences between OWL-S and WSMO

• There is no clear mapping between OWL-S and WSMO approach to mediation
  – OWL-S adopts the view that mediators emerge
    • as infrastructure elements
    • or as by product of the reasoning capabilities of the Web service (for example through matchmaking or planning)
  – WSMO views mediators as fundamental conceptual elements…
    • But they can also be located as the result of matchmaking or composition
Semantic Representation

- OWL-S and WSMO adopt a similar view on the need of ontologies and explicit semantics but they rely on different logics
  - OWL-S is based on OWL/SWRL
    - OWL represents taxonomical knowledge
    - SWRL provides inference rules
  - WSMO is based on WSML a family of languages with a common basis for compatibility and extensions in the direction of Description Logics and Logic Programming
• The relation between WSML and OWL+SWRL is still to be completely worked out
  - For some languages it is known
    • WSML-Core is an interesting subset of OWL Lite
    • WSML-DL is equivalent to OWL DL
  but for other languages the relation is still unknown
Using OWL-S to address Web Services problems

Katia Sycara
David Martin
Overview

OWL-S, as Web services description language needs to support

- Discovery
- Composition
- Invocation
- Guaranteeing Security and Policies
- Mediation and Interoperation

In this section we will discuss these issues in more detail
Discovery with OWL-S

Expressing capabilities in OWL-S

• OWL-S Profile describes capabilities of Web services
• Three types of representations:
  1. Functional representation
     – Input/Output specify the information transformation produced by the Web service
     – Precondition/Effect specify the domain transformation produced by the Web service
  2. Non-functional properties
  3. Type of service and product information
• Many capability matching algorithms have been proposed, here we discuss three.
Discovery with OWL-S

CMU’s Matchmaker

- Matching of I/O of the request with I/O of the advertisement
- **Efficient implementation** given correct indexing of advertisements
  - Match within ms
  - Linear complexity on the size of the query
- **Current work** aims at generalizing matching process to include preconditions/effects service and product types and service parameters

Proposed by Paolucci et al, ISWC 2002
Discovery with OWL-S

Using Subsumption

- Use subsumption relation between advertisement and request
- Five degrees of match
  - Exact
  - PlugIn $R \subseteq A$
  - Subsumed $A \subseteq R$
  - Intersection $\neg (A \not\subseteq R \subseteq \perp)$
  - Fail when disjoint $A \not\subseteq R \subseteq \perp$
- It shows that pure subsumption is inadequate for discovery in OWL-S
  - But problem is much deeper: subsumption is inadequate for discovery of Web services because
    - It is inherently difficult to specify partial descriptions of services which would allow the requester to say which are the features of the WS it really care about
    - Most of the matches reduce to intersection which is not really informative

Proposed by Li et al, WWW 2003
Discovery with OWL-S

Integration of OWL-S and UDDI

• OWL-S Profile has been mapped to UDDI data structure
  • OWL-S Web services can be advertised in UDDI as any other Web service
    (see Paolucci et al 2002)

• CMU OWL-S Matching engine has been integrated within UDDI server
• CMU UDDI server provides
  • Normal UDDI Publish/Inquiry ports
    • Complete interoperability with any UDDI Client
  • Capability Port provides OWL-S based capability requests
    (see Srinivasan et al 2004)

CMU UDDI is publicly available at
www.daml.ri.cmu.edu/matchmaker
or on SemWebCentral
www.semwebcentral.org
A variant of the CMU UDDI is in use at the NTT UDDI Business Registry (The main public UDDI in Japan) (see Kawamura et al 2003, 2004)
Composition with OWL-S

MindSwap’s Web Service Composer

• WS composition environment
  - Uses SHOP2, a well established planner
  - Contains an OWL-S execution environment

• Used for many applications of WS composition ranging from
  - Information gathering
  - Language translation
  - etc…

• Generates a composition that is directly executable through WSDL groundings.
Composition with OWL-S

KSL Automated WS Composition Tool

Approach:

I. Plan a sequences of services that realize user’s objective, using Golog & sit’n calculus.
   (NP complete or worse)

II. Customize reusable generic procedures
    - Define and archive reusable **generic procedures**
    - Customize with **user’s constraints**.
    (NP complete or worse in a reduced search space)

**Advantages:** efficiency, ease of use, customization
Composition with OWL-S

CMU Composition Architecture

• It integrates discovery and composition
  - OWL-S/UDDI Matchmaker for discovery
  - Retsina planner to control the agent
    • Interleaving of planning and execution to allow communication while planning
  - OWL Reasoner
  - OWL-S Virtual Machine to communicate with other Web Services

• Used in a number of applications: travel domain, supply chain management

• Connection with autonomous agent technology

in collaboration with Toshiba
Mapping OWL-S to WSDL

- OWL-S invocation is based on the Grounding
  - Map atomic processes into WSDL operations
  - Use XSLT to map between XML Schema data structures and Ontological Information
  - Invocation procedure totally separated from semantic description of Web service
    - Invocation may be modified without changing semantic description
    - Any Web service can be described in OWL-S without modifying the WSDL description of the service
      - Amazon’s Web service has been described in OWL-S maintaining Amazon’s XML-Schema data types
Invocation with OWL-S

OWL-S Virtual Machine

- OWL-S VM a generic processor for the OWL-S Process Model
  - It can interact with any OWL-S Web service
  - Based on the Process Model formal semantics (Ankolekar et al 2002)
  - Implement grounding mapping to WSDL
  - Exploits Web services technology such as Axis and WSIF for actual invocation and message exchange
Security and Policies

• No standard OWL-S representation for Security and Policies has been published yet
  – But experimentation already underway
  – Adoption of a solution will depend on WS security standards

• Security Experiments with
  • representing security capability/requirements for discovery
  • Representing security information in Process Model.
    (See Denker et al 2003)

• Policies:
  – Experiments combining OWL-S and Rei
  – Rei statements included in Process Model to constrain the use of a Web service (see Kagal 2004)

• Recent work on Formal Verification of OWL-S Process Models provides a way to certify adherence to a policy
  (see Ankolekar et al “Spinning the OWL-S Process Model” In Semantic Web Services Workshop at ISWC '04)
Mediation with OWL-S

• **OWL-S is orthogonal to mediation**
  – Mediators are architecture components
  – OWL-S is a language for the description of Web services
    • It works with *any* architecture that supports ontology specification

• **To the extent that WSMO mediators are Web services, they can be described in OWL-S.**

(See Paolucci et al. “Expressing WSMO Mediators in OWL-S” In Semantic Web Services Workshop at ISWC ’04)
Mediation with OWL-S (2)

• General schema to represent WSMO mediators:
  – any xy-mediator is represented by a Web service that takes input x and reports output y
• …but the mediation is more complex than asserting the need for mappings
  – Discovery maps advertisements and requests
  – Planning systems to reconcile discrepancies between Web services
  – Data type Mapping rules are used in the OWL-S Groundings
• OWL-S assumes all these technologies for interoperation and mediation
Conclusion: How OWL-S Addresses WS problems

• **Discovery**
  - Provide formal representation of capabilities of WSs
  - Many different types of inferences possible to find Web services using OWL/OWL-S

• **Composition**
  - Support formal representation of WS Process Model of Web services
  - Process Model can be integrated into Planning systems for automatic composition

• **Invocation**
  - Support any type of WS invocation mechanism
  - Clear separation between WS description and implementation

• **Guaranteeing Security and Policies**
  - No explicit policy and security specification yet
  - Proposed solution will interoperate with WS security standards

• **Mediation and Interoperation**
  - Mediation services can be directly described
  - Interoperation allowed by ontology-based description of WS descriptions and data

• The solutions are envisioned maintaining a strong relation with existing WS standards
Using WSMO to address Web Services problems

Rubén Lara
WSMO discovery
Steps of discovery

• GOAL DISCOVERY
  – Abstracting user goal and producing a suitable representation of the goal
    • Tool support (delegation to the user)
    • (Semi)automatic
    • Parameterized pre-defined goals ([Kifer et al., to appear], Semantic Web Fred)
Steps of discovery

• WEB SERVICE Vs SERVICE

  – Want to buy a book
    • Look for a Web Service which sells books
    • Consult the Web Service to check whether the book is in stock, price, delivery conditions, etc.
      – Web Service: interface to database or “actions”
      – Service: the database or “actions” themselves

  – Finding services based on the semantic annotation of Web Services requires COMPLETE AND CORRECT descriptions
    • In practical terms, DUPLICATION OF SERVICES!
      – Unrealistic assumption
      – Difficult to scale (in terms of complexity of reasoning & human resources)
Web Service Discovery

• 1) Keyword-based search
• 2) Characterization of the service results
• 3) Precise description of Web Service functionality
Web Service discovery

• [Kifer et al., 2004] uses FLORA-2 to do discovery and contracting.
  – First approach using relation input-output/effects + mediation (see SWSs workshop tomorrow)

• [Keller et al., 2004] uses FOL in the context of the Semantic Web Fred for discovery (see demo sessions)

• WSMX discovery (see implementation slides)

• Subsumption-based approaches can be equally applied
  – Good indexing technique for discovery based on characterization of results

• Consideration of preferences and non-functional properties will be included
WSMO composition

• 3 levels of dynamism

  – Fixed orchestration
    • Appropriate in some real world cases

  – Orchestration with proxies
    • Provides dynamic resolution of activities with a single service
      (multiple invocations possible)

  – Automatic composition
    • Planning-based
    • Necessary at the functionality and at the process level!

  – Heterogeneity is to be resolved by mediators
WSMO composition

- Knowledge Web: integration of discovery and composition
Service Grounding – WSMO

- Deal with existing WSDL services
  - Map from XML Schema used in WSDL to WSML
  - Use existing tools to mediate from WSML to WSML
- Also investigating
  - Using XSLT to map from XML-S of WSDL directly to WSML/XML of ontology used by WSMO description
- Ultimate aim to have Semantic description of interface grounding in the Choreography
Service Grounding – WSMO

1. Create WSMO description
2. Map XML schema to WSML
3. Create Mapping Rules
4. Add mapping rules to WSMO choreography
Perspective on Security and Policies

- WSMO distinguishes capabilities, constraints and preferences on both sides [Arroyo et al., 2004]
  - Functional and non-functional
  - Extensions to WSMO required
  - Policies at WSDL level?
  - Must be ensured at execution time
    - Extend WSDL (and others) to include policies and control execution

- Experiments with the representation of policies in WSMO using Peertrust [Lara et al., 2004]
  - Different scope to WS-Policy (trust negotiation)
  - Link to WS-Policy feasible
Conclusion: How WSMO Addresses WS problems

- **Discovery**
  - Provide formal representation of capabilities and goal
  - Conceptual model for service discovery
  - Different approaches to web service discovery
- **Composition**
  - Provide formal representation of capabilities and choreographies
  - 3 levels of automatization: full, partial, none
- **Invocation**
  - Support any type of WS invocation mechanism
  - Clear separation between WS description and implementation
- **Guaranteeing Security and Policies**
  - No explicit policy and security specification yet
  - Proposed solution will interoperate with WS standards
- **Mediation and Interoperation**
  - Mediators as a key conceptual element
  - Mediation mechanism not dictated
  - (Multiple) formal choreographies + mediation enable interoperation
- The solutions are envisioned maintaining a strong relation with existing WS standards
Questions and Answers

# Coffee Break #
Table of contents

(I) Introduction to Semantic Web Services (SWS)
(II) Semantic Web Services
   • OWL-S & WSMO
   • OWL-S and WSMO - Design decisions and trade-offs

#Q&A, Coffee break#

(III) Semantic Web Services implementations
   • OWL-S
   • WSMX
   • IRS – III – bridge implementation between OWL-S & WSMO

(IV) Summary, Conclusions & Future Work
SWS
Implementations (III)

OWL-S  WSMX
OWL-S

Tools and applications

Katia Sycara
Massimo Paolucci
David Martin
OWL-S Tools

• The OWL-S community is heavily engaged to produce tools that facilitate the use and adoption of OWL-S

• Three tools presented here
  – CMU Eclipse-based OWL-S IDE
  – SRI Protégé-based OWL Editor
  – MindSwap Swoop: an Editor and verifier for OWL and OWL-S
CMU OWL-S IDE

- CMU OWL-S IDE is an Eclipse based tool that integrates the generation of OWL-S representation with the generation of the WS Java code

- Tools targeted to Web service developers
  - Main idea is to allow developers to generate their code and OWL-S description within the same environment

*Demo available at Conference Demo Session*
OWL-S Production cycle

1. Developer creates Java code
2. IDE transforms Java into partial OWL description
   1. WSDL is generated as by-product
3. Easy to use OWL-S editor is used to complete the OWL-S description
4. UDDI client can be used for automatic advertisement in UDDI
5. Verification tools are available for correctness checking
6. Automatic client generation
7. Extension to SWeDE OWL Editor
Architecture OWL-S IDE

Legend:
- Tools integrated in the OWL-S IDE
- Data Files

Tools integrated in the OWL-S IDE:
- OWL-S Editor (eclipse)
- Java Code
- Apache's Java2WSDL Converter
- WSDL2OWL-S Converter
- BBN's SWeDE OWL Editor

Data Files:
- OWL-S Files
- Profile
- Process
- Grounding
- OWL-S VM
- OWL-S S/2UDDI Converter
- UDDI Client
- UDDI-data structure

Spin Based Verification

Java Code

Java2WSDL Converter
WSDL Code
WSDL2OWL-S Converter
BBN's SWeDE OWL Editor

Spin Based Verification

Java Code
OWL-S Editor for Protégé

• Easy, intuitive OWL-S service development environment
• Based on popular Protégé/OWL ontology editor
• Open-source, with code available at http://owlseditor.projects.semwebcentral.org
• It provides
  – IOPR Manager
    • Input/Output/Precondition/Result
    • Maintain IOPR correspondences between OWL-S sub-ontologies
    • Perform consistency checks
  • Graph Overview
  • Visualize & navigate relationships between OWL-S sub-ontologies
• Generate & import skeletal OWL-S from WSDL

Demo session: Wed., 17.00 - 18.30

Thanks to Daniel Elenius, Grit Denker and David Martin
Sample Functionalities

- Full control of OWL-S properties with customized widgets
- Toolbar provides WSDL import, graphical overview, and more
- Instance panes for Services, Profiles, Processes, and Groundings

Thanks to SRI
Additional Features

- **Control Flow** (shown at right)
  - View and edit as a tree
  - Also visualize as a graph

- **Work in progress**
  - Data Flow
  - Customized OWL-S code generation
  - Search the Semantic Web for OWL-S services

Edit details of control constructs
SWOOP

SWOOP is meant for rapid and easy browsing and development of OWL ontologies

Features

– **Web Browser like look & feel:**
  - hyperlink based navigation
  - history buttons (Back, Next etc) for traversal;
  - bookmarks that can be saved for later reference

– **Inline Editing**
  - Color coding to emphasize ontology changes,
  - Undo/redo options are provided with an ontology change log and a rollback option

– **Verification tools highlighting logical problems**
SWOOP and OWL-S

- Swoop can be used to display OWL-S ontologies
  - It provides validation of correctness of OWL code
  - It will provide visualization of both XML syntax and human readable syntax
Applications

• OWL-S has been used in a number of applications ranging from e-commerce to mobile computing, to robotics.

• Here we briefly discuss...
  – Task Computing
    • Use OWL-S in pervasive computing
  – OWL-S for Robots
    • OWL-S used to describe behavior of agents and robots
Task Computing

Problem

• User wants to do “Tasks” while on the run
  - email – printing – sharing documents – complex tasks

• Services to perform those tasks may be offered in the environment

• But the user may not be able to access them
  - She may not know what is available
  - How to use the services
  - She will likely need some configuration to use those services

(see http://taskcomputing.org/)
Task Computing

The Objective

Task Computing fills the gap between a user’s desires and the available means.

Task computing helps the user to
- Discover the services that are available
- Use those services
- Combine those services to fit the needs of the user
Task Computing Technology

• Help users access Services (Web based and not) and
  – Discovery using UPnP
  – Composition produced at *execution time*, not at the *design time*

• Use:
  – OWL-S based representation of services and devices

Thanks to Fujitsu
Beyond Web Services: OWL-S for Robotic Applications

• Objective:
  – To develop a common, implementation-independent, extendable knowledge source for researchers and developers in the intelligent vehicle community that will:
    • Provide a standard set of domain concepts along with their attributes and inter-relations
    • Allow for knowledge capture and reuse
    • Facilitate systems specification, design, and integration
    • Accelerate research in the field.
Interchange Formats and Upper Ontologies

• OWL
  – Neutral (W3C) interchange format
  – XML base enables use of XSLT transforms
  – Provides access to emerging semantic web technologies

• OWL-S
  – Rich semantics for describing complex processes (without being too complicated)
  – Well suited to agent architectures

• Pieces of SUMO (Suggested Upper Merged Ontology)
  – Class structure and properties provide a good starting point for developing domain specific ontology
  – Native KIF format too complex for target community and not necessary for requirements capture

• Namespaces
  – Used quite a bit to make ontology more manageable
IGV Ontology

- Intelligent Ground Vehicle (IGV) Ontology based on OWL-S
- Upper ontology based on three concepts
  - Agent
  - The service that the agent can perform,
  - The procedures that the agent follows to perform the services

- OWL-S used to model Agents and Services
Tactical Behaviors
Plan State-Table Selection

Example of representation of vehicle operation where
• The first column represents the condition of the IGV
• The first column also represents preconditions,
• The second column the processes that are invoked

<table>
<thead>
<tr>
<th>StartUpAndOperate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Conditions</strong></td>
</tr>
<tr>
<td>New StartupAndOperateCommand</td>
</tr>
<tr>
<td>S1 EngineStarted</td>
</tr>
<tr>
<td>S2 GearChangeRequired</td>
</tr>
<tr>
<td>S3 GearChanged</td>
</tr>
<tr>
<td>S2 NewCommandedVelocity</td>
</tr>
<tr>
<td>S4 EngineThrottleAdjusted</td>
</tr>
<tr>
<td>S2 ShutDownRequested</td>
</tr>
<tr>
<td>S5 GearInPark</td>
</tr>
<tr>
<td>S6 EngineShutDown</td>
</tr>
</tbody>
</table>
WSMO-WSMX

Introduction

Michal Zaremba

Contributors: WSMX team
WSMO-WSMX Introduction

- WSMX is a software framework that allows runtime binding of service requester and provider
- Requester provides semantic description of goal
- WSMX interprets the goal to:
  - Discover matching services
  - Select the service that best fits
  - Provide data mediation if required
  - Make the service invocation
- Based on the conceptual model provided by WSMO
  - Add-ons required for WSMXGoal, BusinessPartner, Preferences
- WSMX has a formal execution semantics
  - Describes how WSMX gets from requester goal to service invocation
WSMX Execution Semantics

• What is it?
  – Description of the operation of a system using a formal language

• What are the benefits?
  – Precise system description based on a formal mathematical language
  – Can run simulations to test for potential problems
    • Live-lock
    • Dead-lock or
    • Unreachable states in the system

• Petri-Nets
  – Have a formal semantics
  – Allow simulations – test for deadlocks etc.
  – Other methodologies – Abstract State Machines, UML …
The WSMX Manager is the central component for managing WSMO descriptions. It has interfaces for WSMX Manager Listener, WSMX Manager Core, Compiler, Parser, Discovery, Selector, Mediator, Invoker, Data Adapter, Resource Manager, WSMO Repository, and Reasoner APIs.

Adapters connect to external WSMO Repository, such as RDBMS or e.g., UDDI, and external WSDL Adapter, e.g., UDDI, to provide interoperability.

WSMO Editor and WSMO Monitoring provide tools for creating and managing WSMO descriptions.

WSMO editor is an independent tool for creating & managing WSMO descriptions.
Architecture: Compilation

WSMX Manager

WSDL

WSMX Manager Core

WSMO Editor
WSMO Monitoring

Adapter 1
WSML for Goal, WS, Mediator or Ontology

Adapter 2

Adapter n

Back-end application

Agent

Another WSMX

Resource Manager

Parser
Discovery
Selector
Mediator
Invoker

Data Mediator
Data Adapter

Reasoner APIs
API 1
API 2
API n

Reasoner
(e.g. Flora/XSB)

External WSMO Repository
(e.g. UDDI)

Adapter 1
Adapter 2
Adapter n

API 1
API 2
API n

...
Architecture: Get Goal

WSMX Manager
- WSMX Manager Listener
  - Message Scanner
  - Event Scanner
- Parser Listener
- Discovery Listener
- Selector Listener
- Mediator Listener
- Invoker Listener
- WSMX Manager Core
  - Compiler
  - Discovery
  - Selector
  - Data Mediator
  - Invoker
  - Data Adapter
- Resource Manager
  - Events Repository
    - RDBMS
  - Ontology Repository
    - RDBMS
  - WSMO Repository
    - RDBMS
- Reasoner APIs
  - API 1
  - API 2
  - ... (n)
- Reasoner
  - (e.g. Flora/XSB)
- External WSMO Repository
  - e.g. UDDI

WSMO Editor
WSMO Monitoring

Back-end application
- Adapter 1
- Adapter 2
- Adapter n

Agent
Another WSMX

Web Service 1
Web Service 2
Another WSMX

Internet

Architecture: Get Goal

- WSMX Manager
- WSMX Manager Core
- WSDL
- WSDL
- Resource Manager
- Reasoner APIs
- Reasoner
- (e.g. Flora/XSB)
- External WSMO Repository
  - e.g. UDDI
- Back-end application
- Adapter 1
- Adapter 2
- Adapter n
- WSDL
- Agent
- Another WSMX
- Internet
- Web Service 1
- Web Service 2
- Another WSMX
- WSMO Editor
- WSMO Monitoring
- Internet
- Web Service 1
- Web Service 2
- Another WSMX

WSML Message is persistently stored
Architecture: New Message

WSMX Manager

- WSMX Manager Listener
- Parser Listener
- Discovery Listener
- Selector Listener
- Mediator Listener
- Invoker Listener

WSMX Manager Core

- Compiler
- Resource Manager
- Reasoner APIs
- Reasoner (e.g. Flora/XSB)

RDBMS

- WSDL
- WSMO Repository
- Ontology Repository
- Events Repository
- WSMO Editor
- WSMO Monitoring
- External WSMO Repository (e.g. UDDI)

Adapter 1
Adapter 2
Adapter n

Web Service 1
Web Service 2
Another WSMX

Back-end application
Agent
Another WSMX

Internet

WEB SERVICE 1
WEB SERVICE 2

Parser
Discovery
Selector
Mediator
Invoker

Msg scanner picks up a new WSML Message

Page dimensions: 595.0x842.0
Architecture: Event Raised

Event scanner sends event to the WSMX Manager which broadcasts the event to registered components.
Architecture: Parse

Parser listener picks up events for the Parser component, then retrieves WSML using Resource Mgr.

Parser interface takes the WSML message as input.

External WSMO Repository
- e.g. UDDI

Reasoner
- (e.g. Flora/XSB)

API 1
- API 2
- API n

Web Service 1

Web Service 2

Another WSMX

Agent

Another back-end application

Parser
- Discovery
- Selector
- Mediator Listener
- Invoker Listener

Resource Manager
- Events Repository
- Ontology Repository
- WSMO Repository
- WSDL

Adapter 1
- Adapter 2
- Adapter n

WSMX Manager
- WSMX Manager Core
- WSDL

WSMO Editor
- WSMO Monitoring

Internet

Back-end application

Web Service 1

Web Service 2

Another WSMX

Agent

Another back-end application

Parser
- Discovery
- Selector
- Mediator Listener
- Invoker Listener

Resource Manager
- Events Repository
- Ontology Repository
- WSMO Repository
- WSDL

Adapter 1
- Adapter 2
- Adapter n

WSMX Manager
- WSMX Manager Core
- WSDL

WSMO Editor
- WSMO Monitoring

Internet
Architecture: Discovery

WSMX Manager
- Discovery listener picks up events for the Discovery component
- Discovery interface takes WSML representation of requester goal

Discovery Listener
Selector Listener
Mediator Listener
Invoker Listener

WSMX Manager Core

Parser
Discovery
Adapter

Resource Manager
- Events Repository
  - RDBMS
- Ontology Repository
  - RDBMS
- WSMO Repository
  - RDBMS
- Reasoner APIs
  - API 1
  - API 2
  - API n

Reasoner
- (e.g. Flora/XSB)

External WSMO Repository
- e.g. UDDI

Agent
Another WSMX

Web Service 1
Web Service 2
Another WSMX

Back-end application
Adapter 1
Adapter 2
Adapter n

WSM Editor
WSMO Monitoring

WSDL

API 1
API 2
API n

 Parser
 Discovery
 Selector
 Adapter

Compiler

WSDL

Web Service 1
Web Service 2
WSMX Discovery

- Based on matching of logical Goals with WS Capabilities
- Goals and capabilities have postconditions and effects.
- Capabilities additionally have preconditions and assumptions
- WSMX adds concept of conditional Web Service to capability

![Diagram of WSMX Discovery process]

1. **Step 1**
   - Goal

2. **Step 2**
   - Possible Matches

3. **Step 3**
   - Collection of WS

4. **Step 4**
   - Match requester
Architecture: Selection

WSMX Manager

WSMX Manager Core

Selector listener picks up events for the Selector component

Selector interface takes collection of WS and returns one WS

Resource Manager

Events Repository
  RDBMS

Ontology Repository
  RDBMS

WSMO Repository
  RDBMS

Reasoner APIs
  API 1
  API 2
  ... API n

Reasoner
  (e.g. Flora/XSB)

External WSMO Repository
  e.g. UDDI

Parser

Discovery

Selector

Mediator Listener

Invoker Listener

Adapter

Web Service 1

Web Service 2

Another WSMX

Agent

Another end application

API 1

API 2

API n

Parser

Discovery

Selector

Mediator

Invoker

Adapter

Back-end application

Network
Architecture: Mediation

WSMX Manager

WSMX Manager Core

WSMX Manager Listener

Msg Scanner

Events Scanner

Parser

Selector

Mediator

Resource Manager

Events Repository

Ontology Repository

WSMO Repository

External WSMO Repository

RDBMS

RDBMS

RDBMS

RDBMS

RDBMS

RDBMS

WSDL

API 1

API 2

API n

Reasoner

(e.g. Flora/XSB)

e.g. UDDI

WSDL

Adapter 1

Adapter 2

Adapter n

Back-end application

Agent

Another WSMX

Web Service 1

Web Service 2

Another WSMX

Web

Internet

Mediator listener picks up events for the Mediator component and gets the IDs for the source and target ontologies as well as the data for mediation.

Mediator takes source & target ontologies as input as well as WSML data to mediate.

WSMX Mediation
Architecture: Invocation

WSMX Manager
- WSMX Manager Listener
- WSMX Manager Core
  - Parser
  - Discovery
  - Selector
  - Mediator
  - Invoker
- Invoker interface takes WS to be invoked and the mediated data as input.
- Adapter
- Invoker

WSMX Manager Core
- WSDL
- WSDL Adapter

Resource Manager
- Events Repository
  - RDBMS
- Ontology Repository
  - RDBMS
- WSMO Repository
  - RDBMS

Reasoner APIs
- API 1
- API 2
- API n
- Reasoner
  (e.g. Flora/XSB)

External WSMO Repository
- e.g. UDDI

Agent

Another WSMX

Adapter 1
- WSDL

Adapter 2
- WSDL

Adapter n
- WSDL

Back-end application

Internet

Web Service 1

Web Service 2

Another WSMX

WSMO Editor

WSMO Monitoring
WSMX Summary

- Event based component architecture
- Conceptual model is WSMO with some add-ons
- End to end functionality for executing SWS
- Has a formal execution semantics
- Real implementation
- Open source code base at SourceForge
- Event driven component architecture
- Developers welcome
WSMX Useful Links

- Home
  - http://www.wsmx.org/
- Overview
  - http://www.wsmo.org/2004/d13/d13.0/v0.1/
- Architecture
  - http://www.wsmo.org/2004/d13/d13.4/v0.2/
- Mediation
  - http://www.wsmo.org/2004/d13/d13.3/v0.2/
- Execution Semantics
  - http://www.wsmo.org/2004/d13/d13.2/v0.1/
- Open source code base at SourceForge
  - https://sourceforge.net/projects/wsmx
IRS III

Bridge implementation between OWL-S & WSMO

John Domingue

Contributors: Liliana Cabral
IRS-III: A framework and platform for building Semantic Web Services
The Internet Reasoning Service is an infrastructure for publishing, locating, executing and composing Semantic Web Services, organized according to the WSMO framework.
Design Principles

- Compatible with WSMO
- OWL-S import
- Tight integration
- Open
- Inspectable
- Backward compatible
- Research platform for semantic web services
Features of IRS-III (1/2)

• Based on Soap messaging standard
• Provides Java API for client applications
• Provides built-in brokering and service discovery support
• Provides capability-centred service invocation
Features of IRS-III (2/2)

• Publishing support for variety of platforms
  – Java, Lisp, Web Applications, Java Web Services

• Enables publication of ‘standard code’
  – Provides clever wrappers automatically, which turn code into web services
  – One-click publishing of web services

• Integrated with standard Web Services world
  – Published code appears as
    • Semantic web service to IRS
    • ‘Ordinary’ web service to web service world
IRS-III Framework

IRS-3 Server
- Domain Models
- Web Service Specifications + Registry of Implementors
- Goal Specifications + SOAP Binding

IRS Publisher
- Lisp

IRS Publisher
- Java

IRS Publisher
- Java WS

IRS Client

SOAP
IRS-III Architecture

Publishing Platforms

Web Service
Java Code
Web Application

WSMX
Browser
Publishing Clients
Invocation Client

WS Publisher Registry
OCML
WSMO Library

LispWeb Server

SOAP
Browser Handler
Publisher Handler
Invocation Handler

SOAP Handler

OWL(-S) Handler

OWL(-S)
Publishing Platform Architecture

Publishing Clients

SOAP

IRS-III Server

WS Service Registry

Invocation Client

SOAP

HTTP Server

Web Service 1

Web Service 2

Web Service 3

IRS-III Publishing Platform

SOAP Handler

Service Registrar

Service Invoker
IRS-III/WSMO differences

- Underlying language OCML
- Goals have inputs and outputs
- IRS-III broker finds applicable web services via mediators
  - Used mediator within WS capability
  - Mediator source = goal
- Web services have inputs and outputs ‘inherited’ from goal descriptions
- Web service selected via assumption (in capability)
OWL Process to Web Service

• IOPEs are translated to:
  has-input, has-output, has-precondition and has-postcondition
  in the capability of a Web service.
• The type and condition definitions at the range of the above roles are translated by the OWL to OCML translator.
• Simple goal and mediators can be generated (optional) as template for later development.
IRS-III Demo

(including OWL-S Import)
Multiple WS for goal

- Each WS has a mediator for used-mediator slot of capability
  - Some WS may share a mediator
- Define a kappa expression for assumption slot of WS capability
- Kappa expression format
  - (kappa (?goal) <ocml relations>)
- Getting the value of an input role
  - (wsmo-role-value ?goal <role-name>)
Defining a Mediation Service

• Define a wg-mediator
• Source = goal
• Mediation-service = goal for mediation service
• Mediation goal
  – Mediation goal input roles are a subset of goal input roles
• Define mediator and WS as normal
Goal Based Invocation

Solve Goal
Goal -> WG Mediator -> WS/Capability/Used-mediator

Instantiate Goal Description
Exchange-rate-goal
Has-source-currency: us-dollars
Has-target-currency: pound

Web Service Discovery
European-exchange-rate-ws
Non-european-exchange-rate-ws
European-bank-exchange-rate-ws

Web service selection
European-exchange-rate

Mediate input values
‘$’ -> us-dollar

Invocation
Invoke selected web service
European-exchange-rate
# Table of contents#

(I) Introduction to Semantic Web Services (SWS)

(II) Semantic Web Services

- OWL-S & WSMO
- OWL-S and WSMO - Design decisions and trade-offs

#Q&A, Coffee break#

(III) Semantic Web Services implementations

- OWL-S
- WSMX
- IRS – III – bridge implementation between OWL-S & WSMO

(IV) Summary, Conclusions & Future Work
Summary, Conclusions & Future Work

Laurentiu Vasiliu
Other SWS implementations

SELF-SERV

- Bottom-up approach to service composition
- Aim is scalable and decentralized middleware
- Services are registered & grouped by capability
- Registered services can be declaratively composed
- Not directly Semantic Web Services
- Has a formal execution semantics
- Prototype graphical tool implemented
Other SWS implementations

Meteor-S

- Web service annotation framework
- Provides a mechanism to add data, functional and QoS semantics to WSDL files
- Semi-automatically annotate WSDL descriptions
- Implements algorithms for semantic annotation and categorisation of Web services
- Empirical testing of semantic annotation of Web services
Tutorial Wrap-up

• The **targets** of the presented tutorial were to:
  – understand aims & challenges within Semantic Web Services
  – understand the main technologies of OWL-S and WSMO
  – be able to correctly **assess emerging technologies & products** for Semantic Web Services

• Given an **overview of ‘hot topics’** within the Semantic Web and Semantic Web Services

• Provided a **detailed introduction** into OWL-S and WSMO:
  – design principles & paradigms
  – building blocks of OWL-S and WSMO
  – technologies & OWLS+WSMO implementations
OWL-S and WSMO

- North-American and European initiatives with converging aims
- Offer a SWS platforms to be used by B2C and B2B applications
- Provide a backbone for advanced integration and automation of industrial and business processes
- Are the most developed SWS technologies up to now available to be used in commercial and industrial applications
- Developments towards refining and interconnecting them
OWL-S and WSMO technologies

• In spite of some existing scepticism, *logic formalism* and elements of logic are needed for *advanced B2C and B2B applications*.

• **Rules** (based on logic) are compulsory in automating the selection and composition of processes.
OWL-S and WSMO technologies

• SWS designed to allow automatic
  – publication
  – discovery
  – selection
  – composition
  – mediation
  – execution

  of intra / inter-organization business processes
Future work – OWL-S

• OWL-S is close to conclusion, but a few issues still need to be addressed
  – An exception mechanism is still missing
  – There is a need of an exec instruction for loading and executing Process Models dynamically
  – A new Grounding for WSDL 2 should be developed
• Additional issues that OWL-S does not address
  – Security and Policies are not directly expressed in OWL-S yet
  – There are no facilities for Contracting and agreement
  – There are no facilities for Web service management
Future work – OWL-S (2)

• Standardization
  – The OWL-S coalition is planning to submit a W3C note to
draw attention and create momentum for W3C
standardization activities on Semantic Web services
  – Members of the OWL-S coalition are already active in
standardization committee such as UDDI, WSDL 2 and
WS Coordination

• The Future of OWL-S
  – OWL-S is nearing its completion and it will converge in the
results of the SWSI working group or future
standardization activities
  – The OWL-S coalition plans to remain in existence to
maintain and further develop the language if needed
Future work - WSMO

• Further develop and consolidate concepts and implementation aspects of WSMO, WSML and WSMX
  – Choreography and orchestration
  – Business process execution
  – Web services composition
  – Process and protocol mediation
• Open to new ideas, contributions and suggestions
Future Work WSMO (2)

- WSMO & WSMX – applied in several case studies within EU funded projects
- WSMX v2 to be release in November
- IRS III new release at the beginning of 2005
- Following on during the conference: WSMX demo and poster, IRS III demo
Beyond OWL-S and WSMO

• Although OWL-S and WSMO are the main initiatives on Semantic Web services, they are not the only activities

• Semantic Web Services Interest Group
  – Interest group founded at W3C to discuss issues related to Semantic Web Services (http://www.w3.org/2002/ws/swsig/)

• SWSI: International initiative to push toward a standardization of SWS (http://www.swsi.org)

• Semantic Web services are entering the main stream
  – UDDI is adopting OWL for semantic search
  – WSDL 2 will contain a mapping to RDF
  – The use of semantics is also discussed in the context of standards for WS Policies
SWSI (www.swsi.org)

- SWSI (Semantic Web Services Initiative) is becoming the point of synthesis of the SWS activity around the World
- SWSI includes many participants belonging to both academy and industry from the US and Europe
- SWSI is composed of two committees
  - SWSL which is expected to produce a language for Semantic Web services
  - SWSA which is expected to describe the architectural requirements for Semantic Web services
- OWL-S and WSMO are two main inputs, but contributions include IRS, Meteor-S
Semantics in the Main Stream

- Many WS standardization groups are realizing that they need to add semantic representation
- **UDDI v.next**
  - UDDI v.next is the new version of UDDI
  - UDDI TC has decided to use OWL as a standard language for the representation of business taxonomies
  - OWL-based inference will be used to improve WS search
- **Web Service Description Language v2**
  - The WSDL working group at W3C has decided to add an RDF mapping to WSDL 2
  - The RDF mapping may effectively provide a standard grounding mechanism for OWL-S and WSMO
- **Web Services policies proposals** require a significant amount of inference
  - There have been proposals to use OWL or SWRL as basic languages
  - Or to provide a mapping to semantic Web languages
References OWL-S

• The main repository of papers on OWL-S is at http://www.daml.org/services/owl-s/pub-archive.html that contains many papers produced by the coalition as well as from the community at large.

• The main source of information on OWL-S is the Web site http://www.daml.org/services/owl-s

• The rest of this section will report what we believe to be the most influential papers on OWL-S as well as paper referred in this tutorial.
References OWL-S

• Fundamental


References OWL-S

• Discovery


References OWL-S

• Composition and Invocation

Evren Sirin, Bijan Parsia, Dan Wu, James Hendler, and Dana Nau. **HTN planning for web service composition using SHOP2.** In *Journal of Web Semantics*, To appear, 2004


References OWL-S

• Formal Models and Verification

Anupriya Ankolekar, Massimo Paolucci, and Katia Sycara


References OWL-S

• Policies and Security


References OWL-S

• Applications


Aabhas V Paliwal, Nabil Adam, Christof Bornhövd, and Joachim Schaper Semantic Discovery and Composition of Web Services for RFID Applications in Border Control In Proceedings of Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications (ISWC 2004)


References WSMO

• The central location where WSMO work and papers can be found is WSMO Working Group: http://www.wsmo.org
• In regard of WSMO languages: WSML Working Group: http://www.wsml.org
• WSMO implementation: WSMX working group can be found at: http://www.wsmx.org
• WSMX open source can be found at: https://sourceforge.net/projects/wsmx/
References WSMO


References WSMO

References WSMO

References WSMO

References IRS III tutorial


• J. Domingue and S. Galizia: Towards a Choreography for IRS-III.

• Approaches to Semantic Web Services: An Overview and Comparisons. In proceedings of the First European Semantic Web Symposium (ESWS2004); 10-12 May 2004, Heraklion, Crete, Greece.

Acknowledgements

We would like to acknowledge the contribution of the past and present members of the OWL-S coalition for their hard work in the development of the language. Furthermore, we would like to thank the community at large for contributing to tools and ideas.

Furthermore, we would like to thank to all the members of the WSMO, WSML, and WSMX working groups for their advice and input into this tutorial.

Special thanks to Sheila McIlraith, Craig Schlenoff, Daniel Elenius and Naveen Srinivasan for providing slides and suggestions on this tutorial.

Slide design by Harriett Cornish, Knowledge Media Insitute, The Open University
Acknowledgements

The development of OWL-S has been funded almost exclusively by the DAML DARPA program.

The WSMO work is funded by the European Commission under the projects DIP, Knowledge Web, SEKT, SWWS, AKT and Esperonto; by Science Foundation Ireland under the DERI-Lion project; and by the Vienna city government under the CoOperate program.