Semantic Web Services with the Web Service Modeling Ontology WSMO

Tutorial at the 7th edition of the EuroLAN Summer School

Cluj-Napoca, Romania,
August 2, 2005

Dieter Fensel
Dumitru Roman
Ioan Toma

{firstname.lastname}@deri.org
The **aims** of this tutorial

- Introduce the aims & challenges of Semantic Web Services (SWS) to the EuroLAN 2005 audience
- Present a general overview of a fully fledged framework for SWS: a conceptual model, a language, and an execution environment
- Investigate and discuss possible collaborations between SWS and EuroLAN communities
But first a few words about us…

- We are members of Digital Enterprise Research Institute (DERI) - DERI’s vision is to make the Semantic Web and Semantic Web Services a reality enabling fully flexible eCommerce for small, medium-sized and large enterprises.
  - Semantic Web Services have the potential to become a key-enabling infrastructure for Knowledge Management and eWork, Enterprise Application Integration, and eCommerce
  ➔ In consequence, Semantic Web Services are one of the key areas of applied computer science
DERI International (1)

DERI Stanford (CA, USA)
DERI Galway
DERI Innsbruck
DERI Seoul (S. Korea)
DERI International (2)

• We work on a new infrastructure for eWork and eCommerce on a global scale.
  – Therefore, the mission of DERI is truly international.
  – Therefore, we look for cooperation at a global scale.

• Areas of synergies:
  – Standardization
  – Open source Tool development
  – Research Cooperation
  – Funding acquisition
  – Outreach
  – Business Development
Major technologies currently developed by DERI

- **WSMO** - an ontology for Semantic Web Services
- **WSML** - Semantic Web Services and Semantic Web languages
- **WSMX** - an execution environment for Semantic Web Services compliant with WSMO/L
- **Triple Space Computing** - communication platform for Semantic Web services based on Web principles: “Persistently publish and read semantic data that is denoted by unique identifiers”
We are many but we need more!
Feel invited to join our efforts.
<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>14:00 – 14:30</td>
<td><strong>Part I: Introduction to Semantic Web Services</strong></td>
</tr>
<tr>
<td>14:30 – 15:30</td>
<td><strong>Part II: Web Service Modeling Ontology (WSMO) and the Web Services Modelling Language (WSML)</strong></td>
</tr>
<tr>
<td>15:30 – 16:00</td>
<td>Coffee Break</td>
</tr>
<tr>
<td>16:00 – 16:30</td>
<td><strong>Part III: WSMO Discovery</strong></td>
</tr>
<tr>
<td>16:30 – 17:20</td>
<td><strong>Part IV: Web Service Modeling Execution Environment (WSMX)</strong></td>
</tr>
<tr>
<td>17:20 – 17:30</td>
<td><strong>Summary, Conclusions, Questions &amp; Answers</strong></td>
</tr>
</tbody>
</table>
PART I – Outline

~ Introduction to Semantic Web Services ~

• Introduction to Semantic Web

• Introduction to Web services

⇒ Semantic Web Services
Semantic Web - The Vision

- 500 million users
- more than 3 billion pages

Dynamic

Static

WWW .......... URI, HTML, HTTP

Syntax

Semantics
Serious Problems in
- information finding,
- information extracting,
- information representing,
- information interpreting and
- and information maintaining.

Semantic Web - The Vision

Static
WWW
URI, HTML, HTTP

Dynamic

Syntax
Semantics

Semantic Web
RDF, RDF(S), OWL
Semantic Web - The Vision

Dynamic

Web Services
UDDI, WSDL, SOAP

Static

WWW
URI, HTML, HTTP

Semantic Web
RDF, RDF(S), OWL

Syntax

Semantics

Bringing the computer back as a device for computation
Semantic Web - The Vision

Bringing the web to its full potential

Dynamic

Web Services
UDDI, WSDL, SOAP

Intelligent Web Services

Static

WWW
URI, HTML, HTTP

Semantic Web
RDF, RDF(S), OWL

Syntax

Semantics
Ontology Definition

- formal, explicit specification of a shared conceptualization
- unambiguous definition of all concepts, attributes and relationships
- machine-readability
- commonly accepted understanding
- conceptual model of a domain (ontological theory)
Ontology Example

Concept
conceptual entity of the domain

Property
attribute describing a concept

Relation
relationship between concepts or properties

Axiom
coherent description between Concepts / Properties / Relations via logical expressions

holds(Professor, Lecture) :-
Lecture.topic € Professor.researchField
Ontology Languages

• Requirements:
  – ”expressivity“
    • knowledge representation
    • ontology theory support
  – ”reasoning support“
    • sound (unambiguous, decidable)
    • support reasoners / inference engines

• Semantic Web languages:
  – web compatibility
  – Existing W3C Recommendations:
    • XML, RDF, OWL
“Semantic Web Language Layer Cake”
Web Services

Web Services: [Stencil Group]
• loosely coupled, reusable components
• encapsulate discrete functionality
• distributed
• programmatically accessible over standard internet protocols
• add new level of functionality on top of the current web
Web Services Problems

- UDDI Registry
  - Points to Description
  - Points to Service
  - Finds Service

- Service Consumer
  - Communicates with XML Messages

- SOAP

- WSDL
  - Describes Service

- Web Service
Web Services Problems

UDDI Registry

Points to Description

WSDL

Describes Service

Service Consumer

Finds Service

Syntax only

SOAP

Communicates with XML Messages

Web Service
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
Semantic Web Services

• Define exhaustive description frameworks for describing Web Services and related aspects (Web Service Description Ontologies)

• Support ontologies as underlying data model to allow machine supported data interpretation (Semantic Web aspect)

• Define semantically driven technologies for automation of the Web Service usage process (Web Service aspect)
Semantic Web Services (2)

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, process) among the combined
• **Execution:** Invoke services following programmatic conventions
Semantic Web Services (3)

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
Conclusion

Semantic Web Services

= Semantic Web Technology + Web Service Technology
PART II – Outline

~ WSMO and WSML ~

• Overview of WSMO: mission and working groups
• WSMO building blocks:
  – Ontologies
  – Web services
  – Goals
  – Mediators
• Web Service Modeling Language WSML
• WSMO & WSML Conclusions
WSMO is…

• A conceptual model for Semantic Web Services:
  – Ontology of core elements for Semantic Web Services
  – a formal description language (WSML)
  – execution environment (WSMX)

• … derived from and based on the Web Service Modeling Framework WSMF

• an SDK-Cluster Working Group
  (joint European research and development initiative)
WSMO Working Groups

- A Conceptual Model for SWS
- A Formal Language for WSMO
- A Rule-based Language for SWS
- Execution Environment for WSMO
WSMO Design Principles

- Web Compliance
- Ontology-Based
- Strict Decoupling
- Ontological Role Separation
- Centrality of Mediation
- Execution Semantics
- Description versus Implementation
Objectives that a client wants to achieve by using Web Services

- 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)*

**WSMO D2, version 1.2, 13 April 2005 (W3C submission)**
Non-Functional Properties

every WSMO elements is described by properties that contain relevant, non-functional aspects

- Dublin Core Metadata Set:
  - complete item description
  - used for resource management
- Versioning Information
  - evolution support
- Quality of Service Information
  - availability, stability
- Other
  - Owner, financial
### Non-Functional Properties List

<table>
<thead>
<tr>
<th>Dublin Core Metadata</th>
<th>Quality of Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contributor</td>
<td>Accuracy</td>
</tr>
<tr>
<td>Coverage</td>
<td>NetworkRelatedQoS</td>
</tr>
<tr>
<td>Creator</td>
<td>Performance</td>
</tr>
<tr>
<td>Description</td>
<td>Reliability</td>
</tr>
<tr>
<td>Format</td>
<td>Robustness</td>
</tr>
<tr>
<td>Identifier</td>
<td>Scalability</td>
</tr>
<tr>
<td>Language</td>
<td>Security</td>
</tr>
<tr>
<td>Publisher</td>
<td>Transactional</td>
</tr>
<tr>
<td>Relation</td>
<td>Trust</td>
</tr>
<tr>
<td>Rights</td>
<td>Other</td>
</tr>
<tr>
<td>Source</td>
<td>Financial</td>
</tr>
<tr>
<td>Subject</td>
<td>Owner</td>
</tr>
<tr>
<td>Title</td>
<td>TypeOfMatch</td>
</tr>
<tr>
<td>Type</td>
<td>Version</td>
</tr>
</tbody>
</table>
WSMO Ontologies

Objectives that a client wants to achieve by using Web Services

- 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 Usage & Principles

• Ontologies are used as the ‘data model’ throughout WSMO
  – all WSMO element descriptions rely on ontologies
  – all data interchanged in Web Service usage are ontologies
  – Semantic information processing & ontology reasoning

• WSMO Ontology Language WSML
  – conceptual syntax for describing WSMO elements
  – logical language for axiomatic expressions (WSML Layering)

• WSMO Ontology Design
  – Modularization: import / re-using ontologies, modular approach for ontology design
  – De-Coupling: heterogeneity handled by OO Mediators
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)

Ontology Elements:

- 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 Web services

Objectives that a client wants to achieve by using Web Services

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)*
WSMO Web service description

- complete item description
- quality aspects
- Web Service Management

Non-functional Properties

DC + QoS + Version + financial

Capability

functional description

client-service interaction interface for consuming WS
- External Visible Behavior
- Communication Structure
- ‘Grounding’

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

realization of functionality by aggregating other Web Services
- functional decomposition
- WS composition

Choreography --- Service Interfaces --- Orchestration

EuroLAN 2005, Cluj-Napoca, Romania
Capability Specification

- **Non functional properties**
- **Imported Ontologies**
- **Used mediators**
  - *OO Mediator*: importing ontologies with mismatch resolution
  - *WG Mediator*: link to a Goal wherefore service is not usable a priori
- **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 & Orchestration

• VTA example:

- **Choreography** = how to interact with the service to consume its functionality
- **Orchestration** = how service functionality is achieved by aggregating other Web services
Choreography Aspects

**Interface for consuming Web Service**

- **External Visible Behavior**
  - those aspects of the workflow of a Web Service where Interaction is required
  - described by workflow constructs: sequence, split, loop, parallel

- **Communication Structure**
  - messages sent and received
  - their order (communicative behavior for service consumption)

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

- **Formal Model**
  - reasoning on Web Service interfaces (service interoperability)
  - allow mediation support on Web Service interfaces
Orchestration Aspects

Control Structure for aggregation of other Web Services

- decomposition of service functionality
- all service interaction via choreographies

EuroLAN 2005, Cluj-Napoca, Romania
Orchestration Aspects

• Service interfaces are concerned with service consumption and interaction
• Choreography and Orchestration as sub-concepts of Service Interface
• Common requirements for service interface description:
  1. represent the dynamics of information interchange during service consumption and interaction
  2. support ontologies as the underlying data model
  3. appropriate communication technology for information interchange
  4. sound formal model / semantics of service interface specifications in order to allow operations on them.
Future Directions

Choreography:
- interaction of services / service and client
- a „choreography interface“ describes the behavior of a Web Service for client-service interaction for consuming the service

Orchestration:
- how the functionality of a Web Service is achieved by aggregating other Web Services
- extends Choreography descriptions by control & data flow constructs between orchestrating WS and orchestrated WSs.

User language
- based on UML2 activity diagrams
- graphical Tool for Editing & Browsing Service Interface Description

Workflow constructs as basis for describing service interfaces:
- workflow based process models for describing behavior
- on basis of generic workflow constructs (e.g. van der Aalst)

Formal description of service interfaces:
- ASM-based approach
- allows reasoning & mediation

Ontologies as data model:
- every resource description based on ontologies
- every data element interchanged is ontology instance

Grounding:
- making service interfaces executable
- currently grounding to WSDL
WSMO Goals

Objectives that a client wants to achieve by using Web Services

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

• **Ontological De-coupling of Requester and Provider**

• **Goal-driven Approach**, derived from AI rational agent approach
  - Requester formulates objective independently
  - ‘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*: importing ontologies with heterogeneity resolution
  - *GG Mediator*:
    - Goal definition by reusing an already existing goal
    - allows definition of Goal Ontologies
- Requested Capability
  - describes service functionality expected to resolve the objective
  - defined as capability description from the requester perspective
- Requested Interface
  - describes communication behaviour supported by the requester for consuming a Web Service (Choreography)
  - Restrictions / preferences on orchestrations of acceptable Web Services
WSMO Mediators

Objectives that a client wants to achieve by using Web Services

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
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
WSMO Mediator uses a **Mediation Service** via **Mediation Services** - as a Goal - directly - optionally incl. Mediation
OO Mediator - Example

Merging 2 ontologies

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

OO Mediator
Mediation Service

Train Connection
Ontology (s1)

Purchase
Ontology (s2)

Train Ticket
Purchase Ontology

Discovery

Mediation
Services

EuroLAN 2005, Cluj-Napoca, Romania
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**

  **Source Goal**
  "Buy a ticket"

  **GG Mediator**
  Mediation Service

  **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
Web Service Modeling Language

• Aim – to provide a language (or a set of interoperable languages) for representing the elements of WSMO:
  – Ontologies, Web services, Goals, Mediators

• WSML provides a formal grounding for the conceptual elements of WSMO, based on:
  – Description Logics
  – Logic Programming
  – First-Order Logic
  – Frame Logic
Rationale of WSML

• Provide a Web Service Modeling Language based on the WSMO conceptual model
  – Concrete syntax
  – Semantics
• Provide a Rule Language for the Semantic Web
• Many current Semantic Web languages have
  – undesirable computational properties
  – unintuitive conceptual modeling features
  – inappropriate language layering
    • RDFS/OWL
    • OWL Lite/DL/Full
    • OWL/SWRL
Variants of WSML

WSML-DL → First-Order Logic (with nonmonotonic extensions) → WSML-Full

WSML-Core → WSML-Flight → WSML-Rule

Description Logics

Logic Programming
WSML-Core

- Basic interoperability layer between Description Logics and Logic Programming paradigms
- Based on Description Logic Programs
  - Expressive intersection of Description Logic $SHIQ$ and Datalog
  - Allows to take advantage of many years of established research in Databases and Logic Programming
  - Allows reuse of existing efficient Deductive Database and Logic programming reasoners
- Some limitations in conceptual modeling of Ontologies
  - No cardinality constraints
  - Only “inferring” range of attributes
  - No meta-modeling
WSML-DL

- Extension of WSML-Core
- Based on the Description Logic SHIQ
  - Entailment is decidable
  - Close to DL species of Web Ontology Language OWL
  - Many efficient subsumption reasoners
- Some limitations in conceptual modeling of Ontologies
  - No cardinality constraints
  - Only “inferring” range of attributes
  - No meta-modeling
- Limitations in logical expressions
  - From Logic Programming point-of-view, there is a lack of:
    - N-ary predicates
    - Chaining variables over predicates
    - (Default) negation
WSML-Flight

• Extension of WSML-Core
• Based on the Datalog,
  – Ground entailment is decidable
  – Allows to take advantage of many years of established research in Databases and Logic Programming
  – Allows reuse of existing efficient Deductive Database and Logic programming reasoners
• No limitations in conceptual modeling of Ontologies
  – Cardinality constraints
  – Value constraints for attributes
  – Meta-modeling
WSML-Rule

• Extension of WSML-Flight; based on Horn fragment of F-Logic
  – Ground entailment is undecidable
  – Turing complete
  – Allows to take advantage of many years of established research in Logic Programming
  – Allows reuse of existing efficient Logic programming reasoners

• Extends WSML-Flight logical expressions with:
  – Function symbols
  – Unsafe rules

• From Description Logic point-of-view, there is a lack of:
  – Existentials
  – Disjunction
  – (Classical) negation
  – Equality
WSML-Full

- Extension of WSML-Rule and WSML-DL
- Based on First Order Logic with nonmonotonic extensions
  - Entailment is undecidable
  - Very expressive
- Extends WSML-DL logical expressions with:
  - Chaining variables over predicates
  - Function symbols
  - Nonmonotonic negation
  - N-ary predicates
- Extends WSML-Rule with:
  - Existentials
  - Disjunction
  - Classical negation
  - Equality
- Specification of WSML-Full is open research issue
WSML - example

wsmlVariant _”http://www.wsmo.org/wsml/wsml-syntax/wsml-flight”

namespace {_”http://www.example.org/example#”, dc _”http://purl.org/dc/elements/1.1/”}

ontology _”http://www.example.org/exampleOntology”
  […]

goal _”http://www.example.org/exampleGoal”
  […]

etc…
WSML Syntax

• WSML human-readable syntax
• WSML exchange syntaxes:
  – XML syntax:
    • Syntax for exchange over the Web
    • Translation between human-readable and XML syntax
    • XML Schema for WSML has been defined
  – RDF syntax:
    • Interoperability with RDF applications
    • Maximal reuse of RDF and RDFS vocabulary
    • WSML RDF includes most of RDF
    • Translation between human-readable and RDF syntax
    • For logical expressions, XML literals are used
WSMO & WSML - conclusions

• WSMO - a conceptual model for SWS and a basis for SWS languages and SWS execution environments
  – More needs to be done with respect to Web service behavior modeling
• WSML is a language for modeling of Semantic Web Services; based on the WSMO
• WSML is a Web language:
  – IRIs for object identification
  – XML datatypes
• WSML is based on well-known logical formalisms: Description Logics, Logic Programming, and Frame Logic
• WSML - syntax has two parts:
  – Conceptual modeling
  – Arbitrary logical expressions
• WSML - XML and RDF syntaxes for exchange over the Web
PART III – Outline
~ WSMO Discovery ~

- Web Service vs. Service
- Automated WS discovery
- Descriptions and Discovery
- WSMO Discovery process
Web Service vs. Service

• Notions of **Web Service & Service** are often interpreted in various ways in the literature
• We use the following **terminology & interpretation** here
  – **Service**
    • A **provision of value in some domain** (not necessarily monetary, independent of how service provider and requestor interact)
  – **Web Service**
    • **Computational entity** accessible over the Internet (using Web Service Standards & Protocols), **provides access to (concrete) services** for the clients.

• Thus, we have the following **relation between the notions:**
  – **Service** corresponds to a **concrete execution of a Web service** (with given input values)
  – **Web Service** provides a **set of services** to its client; one service for each possible input value tuple
Automated WS discovery

• The task
  – Identify possible web services \( W \) which are able to provide the requested service \( S \) for ist clients

• An important issue ...
  – „being able to provide a service“ has to be determined based on given descriptions only \((WS, \ Goal, \ Ontos)\)
  – Discovery can only be as good as these descriptions

  • Very detailed WS descriptions: are precise, enable highly accurate results, are more difficult to provide; in general, requires interaction with the provider (outside the pure logics framework)
  • Less detailed WS descriptions: are easy to provide for humans, but usually less precise and provide less accurate results
Descriptions and Discovery (I)

• We aim at supporting a wide-variety of clients and applications
  – Support different description techniques for clients
  – Support a wide-variety of applications wrt. needed accuracy
  – Main focus here: Capability – What does the service deliver?

• Basic possibilities for the description of web services:
  – Syntactic approaches
    • Keyword-based search, natural language processing techniques, Controlled vocabularies
  – Lightweight semantic approaches
    • Ontologies, What does W provide (not how)?, Action-Object-Modelling, Coarse-grained semantic description of a service
  – Heavyweight semantic approaches
    • Describes the service capability in detail, Pre/Post-Cond, takes „in-out“ relationship into account, Fine-grained web service description

Æ WS as a set of keywords
Æ WS as a set of objects
Æ WS as a set of state-changes
Descriptions and Discovery (II)

- **Service provider side:**
  - Capability description & levels of abstraction

What do I provide?
(Syntactically)

What do I provide?
(Semantically)

What do I provide &
When (for what input)?
(Semantically)

<table>
<thead>
<tr>
<th>Level of Abstraction</th>
<th>Syntactic</th>
<th>Semantic (&quot;Light&quot;)</th>
<th>Semantic (&quot;Heavy&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1 … WL</td>
<td>W1 … WL</td>
<td>W1 … WL</td>
<td>W1 … WL</td>
</tr>
</tbody>
</table>

{Keyword}
Descriptions and Discovery (III)

- **Service requester side**: Goal description

  - What do I want? (Syntactically)
  - What do I want? (Semantically)
  - What do I want & What (input) can I provide? (Semant.)

  **Level of Abstraction**
  - Syntactic
  - Semantic (“Light“)
  - Semantic (“Heavy“)
Descriptions and Discovery (IV)

• Basic idea for Matching on the single levels

Common keywords

Set-theoretic relationship

Adequate (common) execution/state-transition

{Keyword}

W1 … WL K1 … Kn

WS

Syntactic

Semantic („Light“)

Semantic („Heavy“)
Descriptions and Discovery (V)

- Capability descriptions: Layers of Capabilities
  - How to combine various levels of abstraction?

Abstraction (manual/automated)

What? (Syntactically)
- Syntactic capability

What? (Semantically)
- Abstract capability

What & When? (Semant.)
- Concrete capability

{Keyword}

Level of Abstraction

WS
Descriptions and Discovery (VI)

- Capability descriptions:
  - Levels of abstraction & possible accuracy?

  What? (Syntactically)
  - Syntactic capability
    - perhaps complete & perhaps correct

  What? (Semantically)
  - Abstract capability
    - complete & perhaps correct

  What & When? (Semant.)
  - Concrete capability
    - complete & correct
      (if user input known & interaction)
Descriptions and Discovery (VII)

- Possible approaches for checking matches and their assumed costs

Information Retrieval: efficient

DL-based reasoning/deductive databases: more or less efficient

Deductive databases with TA-Logic support/
Theorem-Proving: less efficient/no guarantees

Semantic ("Light")

Semantic ("Heavy")

Syntactic
Keyword-based description and discovery

- Service descriptions and user request: bag of keywords
- Simple syntactic matching
- Uses relevant keywords for matching: NFP values, etc.
“Lightweight” descriptions and discovery

• Service providing a value in some domain:
  – Goal describes the desired post state as a set of objects
  – Service describes the state after its execution

• Intentions:
  – Describe if the Requester/Provider requests/provides all objects or just one of the objects in the set
“Heavyweight” descriptions and discovery

- Web Service as a computational entity
  - Takes input values $I_1, \ldots, I_n$ that fulfill certain properties (precondition)
  - Input values determine Outputs $O(I_1, \ldots, I_n)$ and Effects $E(I_1, \ldots, I_n)$

- Semantics
  - Web Service as a state-relation (transformation)
  - Captured by:
    - Precondition/Assumptions
    - Postcondition/Effects
WSMO Discovery Process (I)

• Distinguish further between
  – Web Service Discovery
  – Service Discovery
• Web Service Discovery
  – No interaction with the provider, matches are only based on static capability descriptions
  – Matching is less accurate (we can only return web services which might be able to deliver a requested service)
  – Possibly ignore preconditions and inputs in service capabilities
  – Most likely with abstract capabilities
• Service Discovery
  – Interaction with the provider with concrete input from user (dynamic capabilities)
  – Only with heavyweight descriptions of service capabilities possible (Input has to be considered!)
  – Matching is can be as accurate as possible
  – The more interaction, the less efficient becomes checking a match
WSMO Discovery Process (II)

The process envisioned at present …

- Requester Desire
- Predefined formal Goal
- Goal Discovery
- Requester Goal
- Web Service Discovery
- Concrete Capability (possibly dynamic)
- Still relevant WS
- Web Service (Service Discovery)
- Service to be returned

Goals, Repositories, Available WS, Abstract Capability, Concrete Capability, Still relevant WS.

- Ease of description
- Efficient Filtering
- Accuracy

EuroLAN 2005, Cluj-Napoca, Romania
PART IV – Outline
～ Web Service Modeling Execution Environment ～

• Web Service Execution Environment

• Demo
Overview

• WSMX Development
• System Architecture and its Components
• Demo of WSMT/Data Mediation
• Step through Architecture
• System dissemination
WSMO Working Groups - WSMX

A Conceptual Model for SWS

A Formal Language for WSMO

A Rule-based Language for SWS

Execution Environment for WSMO
WSMX Introduction

- WSMX is a software framework that allows runtime binding of service requesters and service providers
- WSMX interprets service requester goal to
  - Discover matching services
  - Select the service that best fits
  - Provide data mediation if required
  - Make the service invocation
- WSMX is based on the conceptual model provided by WSMO
- WSMX has a formal execution semantics
- WSMX has service oriented and event-based architecture based on microkernel design using such enterprise technologies as J2EE, Hibernate, Spring, JMX, etc.
WSMX 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)

WSMO Design Principles == WSMX Design Principles
== SOA Design Principles
WSMX Development Process and Releases

- The development process for WSMX includes:
  - Establishing its conceptual model
  - Defining its execution semantics
  - Develop the architecture
  - Design the software
  - Building a working implementation

- Planned releases
  - January 2005 (WSMX 0.1.5)
  - June 2005 (WSMX 0.2)
  - November 2005 (WSMX 0.3)
  - November 2004 (WSMX 0.1.5)

Current status of components
Scope of WSMX Development

• Reference implementation for WSMO
• Complete architecture for SWS discovery, mediation, selection and invocation
• Example of implemented functionality - achieving a user-specified goal by invoking WS described with the semantic markup
WSMX Components

• Selected components
  – Data Mediator
  – Parser
  – Invoker
  – Resource Manager
Overview on WSMX Data Mediation Approach

• Objectives
  – To mediate the interchanged messages part of a communication process
  – To keep the communication process transparent from data representation point of view
  – To have a semi-automatic mediation process

• Assumptions:
  – Ontological approach to Data Mediation
  – Communicating parties express data in terms of an ontology
  – Interchanged messages $\rightarrow$ ontology instances
  – Ontologies conform to WSMO conceptual model for ontologies
Scenario

- Ontological Approach
  - Ontology Merging
  - Ontology Alignment
    » Instances Transformation
Data Mediation Prototype
Design Time

- Design Time Component → Ontology Mapping System
- Inputs
  - Source Ontology and Target Ontology
- Features
  - Graphical interface to the human user
  - Set of mechanism towards semi-automatic creation of mappings
  - Capturing the semantic relationships identified in the process
  - Storing these mappings in a persistent storage
- Output
  - Abstract representation of the mappings
    - DIP/SEKT mapping language
Design Time Component – Features

Graphical Interface

- Browsing the ontologies
  - Guide the human user
  - Views based approach

- Top-down vs Bottom-up Approach
  - Top-Down
    - Start from a specific problem to solve
    - Determine all the related elements for that mapping
  - Bottom-Up
    - Identify the minimal common subset

- Contexts
  - Not all the information in the ontology are relevant for a specific mapping creation step
  - Updated by applying decomposition
Design Time Component – Features

Views (I)

- **View**
  - Covers only certain aspects of the ontology
  - Associate roles to the elements part of that view
  - Strategies and algorithms are applied on roles
  - The same algorithms and strategies can be used across the views

- **Roles**
  - Primitive Item
  - Compound Item
  - Description Item
    - Has a *Successor* which could be either a *Primitive* or *Compound* item

- Two views are maintained: one for the source and one for the target ontology
Design Time Component – Features

Views (II)

- primitive_item1
- compound_item1
  - hasDescription1 → compound_item2
  - hasDescription2 → primitive_item1
- primitive_item2
- primitive_item3
- compound_item2
  - hasDescription1 → primitive_item3
  - hasDescription2 → compound_item3
- compound_item3
  - hasDescription1 → primitive_item2
  - hasDescription2 → primitive_item1
  - hasDescription3 → primitive_item3
Design Time Component – Features

Views - Example

concept Person
  name ofDataType {0 1} xsd:string
  age ofDataType {0 1} xsd:integer
  hasGender ofDataType {0 1} gender
  hasChild ofType Person
  marriedTo ofType Person

datatype gender
  range ofDataType xsd:string
  definedBy
    ?x = "Male"^*xsd:string or ?x = "Female"^*xsd:string.

---

EuroLAN 2005, Cluj-Napoca, Romania
Design Time Component – Features

Decomposition

- **primitive_item1**
- **compound_item1**
  - hasDescription1 → compound_item2
  - hasDescription2 → primitive_item1
- **primitive_item2**
- **primitive_item3**
- **compound_item2**
  - hasDescription1 → primitive_item3
  - hasDescription2 → compound_item3
- **compound_item3**
  - hasDescription1 → primitive_item2
  - hasDescription2 → primitive_item1
  - hasDescription2 → primitive_item3
Design Time Component – Features

Decomposition

- primitive_item1
- compound_item1
  - hasDescription1
  - hasDescription2 → primitive_item1
- primitive_item2
- primitive_item3
- compound_item2
  - hasDescription1 → primitive_item3
  - hasDescription2 → compound_item3
- compound_item3
  - hasDescription1 → primitive_item2
  - hasDescription2 → primitive_item1
  - hasDescription2 → primitive_item3
Design Time Component – Features

Decomposition → Contexts

- **primitive_item1**
- **compound_item1**
  - hasDescription1 → **compound_item2**
  - hasDescription2 → **primitive_item1**
- **primitive_item2**
- **primitive_item3**
- **compound_item2**
  - hasDescription1 → **primitive_item3**
  - hasDescription2 → **compound_item3**
- **compound_item3**
  - hasDescription1 → **primitive_item2**
  - hasDescription2 → **primitive_item1**
  - hasDescription2 → **primitive_item3**

- Reveals the description of the decompose item
- Updates the contexts
  - Description of the decomposed item
- Ends when there no compound items to decompose
Design Time Component – Features

Capturing the semantic relationships

- The semantic relationships → mappings
- Critical step
  - Links the graphical representation of mappings with their abstract representation
  - Hides the complexity of the mappings from the human user
- Each time a pair of items is chosen a set of mappings are created
- Different views imply different mappings for the same action
Design Time Component – Features

Towards Semi-automatic Mappings

• Domain expert role:
  – To choose pairs of items from the source and target to be decomposed
  – Each pair denotes a semantic relationship

• Suggest semantic relationships
  – Lexical Algorithms
    • Applicability: primitive/compound items & description
    • Uses lexical relationships between items names
    • WordNet + string analyze algorithms
  – Structural Algorithms
    • Applicability: compound items
    • Based on
      – the already done mappings
      – items descriptions
    • Uses the decomposition algorithm
Design Time Component – Features

Capturing the semantic relationships – PartOf View

- primitive_item
- compound_item
  - hasDescription1 → primitive_item
  - hasDescription2 → compound_item

- primitive_concept (data type)
- compound_concept
  - attribute1 → primitive_concept
  - attribute2 → compound_concept

- primitive_item
- compound_item
  - hasDescription1 → primitive_item
  - hasDescription2 → compound_item

- primitive_concept (data type)
- compound_concept
  - attribute1 → primitive_concept
  - attribute2 → compound_concept
Design Time Component – Features

Capturing the semantic relationships – PartOf View

- **primitive_item**
- **compound_item**
  - hasDescription1 → **primitive_item**
  - hasDescription2 → **compound_item**

- **primitive_concept** (data type)
- **compound_concept**
  - attribute1 → **primitive_concept**
  - attribute2 → **compound_concept**

- class to class mapping (classMapping)
Design Time Component – Features

Capturing the semantic relationships – PartOf View

- primitive_item
- compound_item
  - hasDescription1 → primitive_item
  - hasDescription2 → compound_item

- primitive_item
  - hasDescription1 → primitive_item
  - hasDescription2 → compound_item

- primitive_concept (data type)
- compound_concept
  - attribute1 → primitive_concept
  - attribute2 → compound_concept

- primitive_concept (data type)
  - attribute1 → primitive_concept
  - attribute2 → compound_concept

- class to class mapping (classMapping)
- attribute to attribute mapping (attributeMapping)
- class to class mapping (classMapping)
Design Time Component – Features

Capturing the semantic relationships – PartOf View

- primitive_concept (data type)
- compound_concept
  - attribute1
  - attribute2

- primitive_item
  - compound_item
    - hasDescription1
    - hasDescription2

- primitive_concept (data type)
- compound_concept
  - attribute1
  - attribute2

- class to class mapping (classMapping)
- class to class mapping (classMapping)
- class to attribute mapping (classAttributeMapping)
Design Time Component – Features

Capturing the semantic relationships – PartOf View

- primitive_item
- compound_item
  - hasDescription1 → primitive_item
  - hasDescription2 → compound_item

- primitive_item
- compound_item
  - hasDescription1 → primitive_item
  - hasDescription2 → compound_item

- primitive_concept (data type)
- compound_concept
  - attribute1 → primitive_concept
  - attribute2 → compound_concept

- primitive_concept (data type)
- compound_concept
  - attribute1 → primitive_concept
  - attribute2 → compound_concept

- class to class mapping (classMapping)
- class to class mapping (classMapping)
- attribute to class mapping (classAttributeMapping)
Design Time Component – Features

Capturing the semantic relationships – PartOf View

• primitive_item
• compound_item
  hasDescription1 → primitive_item
  hasDescription2 → compound_item

• primitive_item
• compound_item
  hasDescription1 → primitive_item
  hasDescription2 → compound_item

• primitive_concept (data type)
• compound_concept
  attribute1 → primitive_concept
  attribute2 → compound_concept

• primitive_concept (data type)
• compound_concept
  attribute1 → primitive_concept
  attribute2 → compound_concept
Design Time Component – Features

Capturing the semantic relationships – *PartOf View*

- primitive_item
- compound_item
  - hasDescription1 → primitive_item
  - hasDescription2 → compound_item

- primitive_concept (data type)
- compound_concept
  - attribute1 → primitive_concept
  - attribute2 → compound_concept

- primitive_item
- compound_item
  - hasDescription1 → primitive_item
  - hasDescription2 → compound_item

- primitive_concept (data type)
- compound_concept
  - attribute1 → primitive_concept
  - attribute2 → compound_concept
Run Time

• Run Time Component – *Data Mediator*

• Inputs
  – Incoming data
    • Source ontology instances

• Features
  – *Completely automatic process*
  – Grounding of the abstract mappings to a concrete language
    • Flora2
  – Uses the services of a reasoner to evaluate the mapping rules
    • Flora2 reasoner

• Outputs
  – Mediated data
    • Target ontology instances
Run Time Component - Architecture


Instance
Run Time Component – Features (I)

- **Grounding** the abstract mappings
- Associate a formal semantics to the mappings
  - Obtain rules in a concrete language
- Why not during design time?
  - Offers a greater flexibility
  - Different groundings for the same mappings set
  - Different execution environments for the grounded mappings
  - Easier to maintain the abstract mappings
  - Important point of alignment
- Cashing mechanism can be used
Run Time Component – Features (II)

• Reasoning on the mapping rules
  – Flora2/XSB as an execution environment for the mapping rules

• Java Wrapper around Flora2 reasoner
  – Declarativa Interprolog http://www.declarativa.com/interprolog/

• Steps:
  – Abstract mapping loaded from the storage
    • Relational database
  – Grounded to Flora2 -> Flora2 rules
  – Flora2 Rules loaded in Flora2 reasoner
  – Incoming instances loaded in Flora2 reasoner
  – The result (mediated instances) retrieve from the reasoner
WSMX Parser

- WSML 1.0 compliant parser
  - Code handed over to wsmo4j initiative
- Validates WSML description files
- Compiles WSML description into internal memory model
- Stores WSML description persistently (using Resource Manager)
WSMX Invoker

- WSMX V0.1 used the SOAP implementation from Apache AXIS
- Web Service interfaces were provided to WSMX as WSDL
- Both RPC and Document style invocations possible
- Input parameters for the Web Services were translated from WSML to XML using an additional XML Converter component.
WSMX Resource Manager

- Stores internal memory model to a data store
- Decouples storage mechanism from the rest of WSMX
- Data model is compliant to WSMO API
- Independent of any specific data store implementation i.e. database and storage mechanism
Dynamic Execution Semantics

- WSMX consists of loosely coupled components
- Components might be dynamically plug-in or plug-out
- Execution Semantics - invocation order of components
- Event-based implementation
- New execution semantics can appear in the future including new components
- We need a flexible way to create new execution semantics and deploy them in the system
- Ultimate goal is to execute workflow definition describing interactions between system components
Define “Business” Process

1. Discover Web Services
2. Discover Services
3. Mediate Data
4. Mediate Data
5. Return Mediated Data
6. Return Mediated Data
7. Return Web Services
8. Call Invoker
9. Check Choreography
10. Confirmed
11. Confirmed
12. End

- Start
- Create Choreography
- Created
- Data Mediator Wrapper
- Choreography Wrapper

- Communication Manager Wrapper
- Registry of known components
Event-based Implementation

"Business" Process – Internal Workflow

Event and Notification Distribution/Delivery Mechanism

- Core – Manager
- "Business" Process – Internal Workflow
- Event and Notification Distribution/Delivery Mechanism
- Discovery Wrapper implements Mediator Interface
- Data Mediator Wrapper
- Communication Manager Wrapper
- Choreography Wrapper
- Discovery
- Mediator
- Communication Manager

Choreography

Discovery

Mediator

Communication Manager
System Architecture
Request to discover Web services. May be sent to adapter or adapter may extract from backend app.
System Architecture

<table>
<thead>
<tr>
<th>Administration Framework</th>
<th>WSMT – Web Services Modelling Toolkit</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSMX Management</td>
<td>WSML Editor</td>
</tr>
<tr>
<td></td>
<td>Choreography Editor</td>
</tr>
</tbody>
</table>

Goal expressed in WSML sent to WSMX System Interface
System Architecture

Comm Manager component implements the interface to receive WSML goals
System Architecture

Comm Manager tells core
Goal has been received
System Architecture

Choreography wrapper
Picks up event for Choreography component
System Architecture

A new choreography instance is created.
Core is notified that choreography instance has been created.
System Architecture

Administration Framework
- WSMX Monitoring
- WSMX Management

WSMT – Web Services Modelling Toolkit
- WSML Editor
- Choreography Editor

Service Requesters
- Back-end application 1
- Back-end application 2
- ... (Adapters)

Service Providers
- Web Service 1
- Web Service 2
- ... (Adapters)

WSMX Manager Core

WSMX Manager
- CM Wrapper
- RM Wrapper
- Parser Wrapper
- Discovers Wrapper
- Selector Wrapper
- DM Wrapper
- PM Wrapper
- Choreography Wrapper
- Orchestrator Wrapper

Data and Communication Protocols Adapters
- Agent 1 acting on behalf of user a
- Agent 2 acting on behalf of user b
- ... (Adapters)

Agent 3 acting on behalf of user m

Resource Manager Interface
- WSMO Objects Datastore
- NonWSMO Object Datastore

Reasoner Interface
- Reasoner
- Reasoner
- Reasoner
- Reasoner

EuroLAN 2005, Cluj-Napoca, Romania 128
System Architecture

Discovery is invoked for parsed goal
System Architecture

EuroLAN 2005, Cluj-Napoca, Romania
System Architecture

Administration Framework
- WSMX Monitoring
- WSMX Management

WSMT – Web Services Modelling Toolkit
- WSML Editor
- Choreography Editor

Service Requesters
- Back-end application 1
- Back-end application 2
- ... 
- Back-end application n

Agent 1 acting on behalf of user a
Agent 2 acting on behalf of user b
... 
Agent 3 acting on behalf of user m

WSMX
- WSMX Manager
- WSMX Manager Core
- CM Wrapper
- RM Wrapper
- Parser Wrapper
- Discover Wrapper
- Selector Wrapper
- DIME Wrapper
- PM Wrapper
- Choreography Wrapper
- Orchestration Wrapper

WSMX Manager Interfaces
- Resource Manager Interface
- Reasoner Interface
- Datastore Interface
- Data Mediator Interface
- Process Mediator Interface
- Choreography Interface
- Orchestration Interface

WSMO Objects Datestore
- WSMO Reasoner

NonWSMO Object Datestore
- Datastore

Datastore

FloraXSB

WSMO Reasoner

EuroLAN 2005, Cluj-Napoca, Romania
System Architecture

Discovery component requires data mediation.
System Architecture
After data mediation, discovery component completes its task.
System Architecture

EuroLAN 2005, Cluj-Napoca, Romania
After discovery, the choreography instance for goal requester is checked for next step in interaction.
System Architecture
System Architecture

EuroLAN 2005, Cluj-Napoca, Romania
Next step in choreography is to return set of discovered Web services to goal requester.
Set of Web Service descriptions expressed in WSML sent to appropriate adapter
System Architecture

Set of Web Service descriptions expressed in requester's own format returned to goal requester
WSMX Uptake

• Interoperability
  – With IRS3 from Open University, UK
  – Ongoing work on Meteor-S interoperability

• DIP
  – WSMX as reference implementation of DIP architecture

• Cocoon

• Business development
  – Vehicle for projects and partnerships
Open Source WSMX at Sourceforge

Project: Web Services Execution Environment: Summary

The Web Services Execution Environment (WSMX) is an execution environment for dynamic matchmaking, selection, mediation, invocation and interoperation of Semantic Web Services.

Donate to Web Services Execution Environment

- Development Status: 9 - Alpha
- Intended Audience: Developers, Science/Research
- License: MIT License
- Programming Language: Java
- Topic: Distributed Computing

Project UNIX name: wsmx
Registered: 2004-08-09 13:46
Activity Percentile (last week): 37.86%
View project activity statistics
View list of RSS feeds available for this project

Latest File Releases

<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
<th>Date</th>
<th>Notes / Monitor</th>
<th>Download</th>
</tr>
</thead>
<tbody>
<tr>
<td>toolkit</td>
<td>WSMT v0.1</td>
<td>March 16, 2005</td>
<td></td>
<td>Download</td>
</tr>
<tr>
<td>wsmx-components</td>
<td>WSMX Components 0.1.6</td>
<td>January 31, 2005</td>
<td></td>
<td>Download</td>
</tr>
<tr>
<td>wsmx-core</td>
<td>WSMX Core 0.01</td>
<td>July 26, 2004</td>
<td></td>
<td>Download</td>
</tr>
</tbody>
</table>

EuroLAN 2005, Cluj-Napoca, Romania
WSMX Summary

• Event based component architecture
• Conceptual model is WSMO
• End to end functionality for executing SWS
• Has a formal execution semantics
• Open source code base at sourceforge
• Developers welcome
WSMX – Future Work

• Connect components developed by other consortia and partners
• Dynamic Execution Semantics
• Triple Space as Communication Mechanism
• Orchestration and Choreography modules
• Finalize process mediation, refine data mediation
• Further work on Communication Manager
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
WSMO Tools (in development)

http://www.wsmo.org/wsmo_tools.html

1. WSMX Server - http://sourceforge.net/projects/wsmx
   Java API for WSMO / WSML
4. WSMT – Web Services Modelling Toolkit
5. WSMO Studio - http://www.wsmostudio.org/
   (currently: SWWS Studio)
   Creation and editing of WSMO specifications
   WSML Editor
   Ontology Management System OMS
   Open for Plug-Ins for SWS tools (discovery, composer, …)
6. WSML Validator and Parser
   validates WSMO specifications in WSML
   parsing into intermediary FOL format (every FOL compliant syntax can be derived from this)
7. OWL Lite Reasoner for WSML-OWL variant
   OWL Lite Reasoner based on TRIPLE
Conclusions

• This tutorial should enable you to:
  – understand aims & challenges within Semantic Web Services
  – understand the objectives and features of WSMO
  – model Semantic Web Services with WSMO
  – correctly assess emerging technologies & products for Semantic Web Services
  – start using implemented tools to create SWS
WSMO, WSML, WSMX – useful links

• The central location where WSMO work and papers can be found, is WSMO Working Group: http://www.wsmo.org
• Most of the WSMO/WSML/WSMX deliverables can be accessed from http://www.wsmo.org/TR/
• In regard of WSMO languages: WSML Working Group: http://www.wsmo.org/wsml
• WSMO implementation: WSMX working group can be found at: http://www.wsmx.org
• WSMX open source code can be found at: https://sourceforge.net/projects/wsmx/
• WSMO tools: http://www.wsmo.org/wsmo_tools.html
Questions & Answers
Acknowledgements

The WSMO work is funded by the European Commission under the projects ASG, 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.