Applying Semantics to Service Oriented Architectures

Oasis Symposium 2006
The Meaning of Interoperability
9-12 May, San Francisco

Presenters:
Adrian Mocan
Mick Kerrigan
Michal Zaremba

Special Thanks to:
Emilia Cimpian
Thomas Haselwanter
Brahmananda Sapkota
The Aims of this Tutorial

- Introduce the aims & challenges of Semantic Web Services (SWS) - the WSMO approach
- Describe how SOA can be used with Semantic Web Services – WSMX Approach
- Semantic SOA enables interoperability
Overview

- Introduction to SWS
  - WSMO
- Introduction to SOA
  - WSMX
- Means of Interoperability
- Web Service Modeling Toolkit (WSMT)
- Conclusions
Overview

• **Introduction to SWS**
  - WSMO
• **Introduction to SOA**
  - WSMX
• **Means of Interoperability**
• **WSMT**
• **Conclusions**
Introduction to Semantic Web Services

• Introduction to Semantic Web

• Introduction to Web services

⇒ Semantic Web Services
500 million user
more than 3 billion pages

Static

WWW
URI, HTML, HTTP
Semantic Web and Web Services

Static

WWW
URI, HTML, HTTP

→ Semantic Web
RDF, RDF(S), OWL

Serious Problems in
information finding,
information extracting,
Information representing,
information interpreting and
information maintaining.
Semantic Web and Web Services – The Vision

Static

WWW
URI, HTML, HTTP

Dynamic

Web Services
UDDI, WSDL, SOAP

Semantic Web
RDF, RDF(S), OWL

Bringing the computer back as a device for computation
Semantic Web and Web Services – The Vision

Dynamic

Web Services
UDDI, WSDL, SOAP

Intelligent Web Services

Static

WWW
URI, HTML, HTTP

Semantic Web
RDF, RDF(S), OWL

Bringing the Web to its full potential
Ontology Definition

Formal, explicit specification of a shared conceptualization

- unambiguous terminology definitions
- conceptual model of a domain (ontological theory)
- machine-readability with computational semantics
- commonly accepted understanding
Ontology Example

Concept
cancelual entity of the domain

Attribute
property of a concept

Relation
relationship between concepts or properties

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

\[
\text{holds(Professor, Lecture)} \Rightarrow \text{Lecture.topic} \in \text{Professor.researchField}
\]
Ontology Languages

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

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

- Trust
  - Digital Signature

- Proof
  - Logic
  - Ontology vocabulary
  - RDF + rdfschema
  - XML + NS + xsmschema

- Unicode
- URI
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
Using Web Services

- UDDI Registry
  - Points to Description
  - Points to Service
  - Finds Service
  - Service Consumer
  - Communicates with XML Messages

- WSDL
  - Describes Service

Web Service
Using Web Services

- UDDI Registry
  - Points to Description
  - Communicates with XML Messages
  - Finds Service
- Service Consumer
- Syntax Only
- Describes Service
- Web Service
- WSDL
Lack of SWS standards

- Current technology does not allow realization of any of the parts of the Web Service usage process:
  - Only syntactical standards available
  - Lack of fully developed semantic markup languages
  - Lack of semantically 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 capabilities 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 (in data or process) among the combined services
• **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
Summary

Semantic Web Services  
=  
Semantic Web Technology  
+  
Web Service Technology
Overview

• Introduction to SWS
  – WSMO
• Introduction to SOA
  – WSMX
• Means of Interoperability
• WSMT
• Conclusions
Web Service Modeling Ontology (WSMO)

- 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 European Semantic System Initiative
  - “ESSI 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
- Strict Decoupling Of Modeling Elements
- Centrality of Mediation

Ontology-Based
- Ontological Role Separation
- Execution Semantics

Description versus Implementation
Objectives that a client wants to achieve by using Web Services

- Capability (functional)
- Interfaces (usage)

Ontologies

Goals

Web Services

Mediators

Connectors between components with mediation facilities for handling heterogeneities

Provide the formally specified terminology of the information used by all other components
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
<table>
<thead>
<tr>
<th>Dublin Core Metadata</th>
<th>Quality of Service</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contributor</td>
<td>Accuracy</td>
<td>Financial</td>
</tr>
<tr>
<td>Coverage</td>
<td>NetworkRelatedQoS</td>
<td>Owner</td>
</tr>
<tr>
<td>Creator</td>
<td>Performance</td>
<td>TypeOfMatch</td>
</tr>
<tr>
<td>Description</td>
<td>Reliability</td>
<td>Version</td>
</tr>
<tr>
<td>Format</td>
<td>Robustness</td>
<td></td>
</tr>
<tr>
<td>Identifier</td>
<td>Scalability</td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td>Security</td>
<td></td>
</tr>
<tr>
<td>Publisher</td>
<td>Transactional</td>
<td></td>
</tr>
<tr>
<td>Relation</td>
<td>Trust</td>
<td></td>
</tr>
<tr>
<td>Rights</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subject</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Non-Functional Properties List
WSMO Ontologies

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

Objectives that a client wants to achieve by using Web Services

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

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

Connectors between components with mediation facilities for handling heterogeneities
WSMO Web service description

Non-functional Properties
- complete item description
- quality aspects
- Web Service Management

DC + QoS + Version + financial

Capability
- Advertising of Web Service
- Support for WS Discovery

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

Choreography --- Service Interfaces --- Orchestration

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

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

- complete item description
- quality aspects
- Web Service Management
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
  - 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 WS 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)
    - choreography related errors (e.g. input wrong, message timeout, etc.)
  - Grounding
    - concrete communication technology for interaction
  - Formal Model
    - reasoning on Web Service interfaces (service interoperability)
    - allow mediation support on Web Service interfaces
Control Structure for aggregation of other Web Services

- decomposition of service functionality
- all service interaction via choreographies
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:
  – represent the dynamics of information interchange during service consumption and interaction
  – support ontologies as the underlying data model
  – appropriate communication technology for information interchange
  – sound formal model / semantics of service interface specifications in order to allow operations on them.
Ontologies as data model:
- every resource description based on ontologies
- every data element interchanged is ontology instance

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

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.

Choreography and Orchestration - Overview

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

workflows 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)

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

Connectors between components with mediation facilities for handling heterogeneities
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 / functional / 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. **Functional Level:** mediate mismatches between Web Service/Goal and Web Service/Goals functionalities
  3. **Process/Protocol Level:** mediate heterogeneous Business Processes/Communication Patterns

- **Layers of Mediators**
  - **Specification Layer** – WSMO Mediators
  - **Implementation Layer** – Levels of Mediation
WSMO Mediators Overview
WSMO Mediator uses a Mediation Service via

Source Component

1..n

Target Component

- as a Goal
- directly
- optionally incl. Mediation

Mediation Services

Specification layer

Implementation layer
Merging 2 ontologies

Goal:

"merge s1, s2 and s1.ticket subclassof s2.product"

OO Mediator
Mediation Service

Train Connection
Ontology (s1)

Purchase
Ontology (s2)

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]

  - **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
Data Level Mediation

• **Scope**
  – Solving terminological mismatches

• **Related Aspects / Techniques:**
  – Ontology Integration (Mapping, Merging, Alignment)
  – Data Lifting & Lowering
  – Transformation between Languages / Formalisms

• **Terminology Mismatches Classification**
  – Conceptualization Mismatches
    • same domain concepts, but different conceptualization
    • different levels of abstraction
    • different ontological structure
    • => resolution only includes human intervention
  – Explication Mismatches
    • mismatches between:
      – T (Term used), D (definition of concepts), C (real world concept)
    • => automated resolution partially possible
Functional Level Mediation

• **Scope**
  – Solving functional mismatches between goals and/or ws

• **Related Aspects/Techniques**
  – Discovery
  – Semantic Matchmaking

• **Matchmaking Mismatches**

\[
\begin{align*}
\text{Exact Match} & \quad \text{Plugin Match} & \quad \text{Subsumption Match} & \quad \text{Intersection Match} & \quad \text{No Match}
\end{align*}
\]

\(\circ = \text{G/WS} \quad \bullet = \text{G/WS}\)
Process Level Mediation

- **Scope**
  - Resolves communication mismatches and establish behavior compatibility
- **Related Aspects/Techniques**
  - Data and control flow composition
- **Process Mismatches**
  - Signature terminology mismatches (need for data level mediation)
  - Communication/behavior mismatches
WSMO Mediators and Mediation Levels

- ooMediator
  - Data Level Mediation

- ggMediator
  - Data Level Mediation
  - Functional Level Mediation

- wgMediator
  - Data Level Mediation
  - Functional Level Mediation
  - Process Level Mediation

- wwMediator
  - Data Level Mediation
  - Functional Level Mediation
  - Process Level Mediation

Ex:

```
internal business logic of Web Service
(not of interest in Service Interface Description)
```

```
internal business logic of Web Service
(not of interest in Service Interface Description)
```
Overview

• Introduction to SWS
  – WSMO
• **Introduction to SOA**
  – WSMX
• Means of Interoperability
• WSMT
• Conclusions
Key Enablers

- People (e.g., organization structure, human capital)
- Business Processes
- IT (e.g., systems)
- Physical Infrastructure (e.g., facilities, workplace environment)

Capabilities

- Strategy selection
- Value Proposition development
- Long term vision alignment

Strategies

- Goals
- Objectives

Mission

- Critical success factors for customers and service offerings
- Specific definition functional performance

OASIS Symposium 2006
Existing IT architectures cannot support changing needs

Existing Architectures do not scale

<table>
<thead>
<tr>
<th>Agility</th>
<th>Process Heterogeneity</th>
<th>Data Heterogeneity</th>
<th>Costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT assets cannot be easily repositioned in response to changing requirements</td>
<td>Systems, organizations units and network of partners duplicate the same work</td>
<td>Difficult to determine “what data means” fosters duplicate applications and data</td>
<td>Duplicate data entry and manual data reunion require extra man power</td>
</tr>
<tr>
<td>No solution for efficient intra- and inter-organization information sharing</td>
<td>Information stovepipes require point-to-point integrations that limit flexibility and create maintenance overhead</td>
<td>Inability of applications to interoperate due to platform incompatibility</td>
<td>Point-to-point integration shifts IT professionals towards repetitive employees to time consuming tasks</td>
</tr>
<tr>
<td>Decision cycles are unnecessarily lengthened by data stovepipes</td>
<td>More efforts spent connecting systems together than adding mission critical capabilities</td>
<td>Data used across an organization is often inconsistent and potentially inaccurate</td>
<td>Integrating data stovepipes is expensive and wasteful</td>
</tr>
</tbody>
</table>

No agility, processes redundant, lack of system interactions, and everything is very costly

Operations and maintenance costs are a rising percentage of the budget
SOA - A paradigm that encourages organizations to re-think how their IT capabilities are organized.

SOA is an approach to organizing and using IT to match and combine needs with capabilities in support of the overall mission of an enterprise.

Service
Capabilities performed by one for another to achieve a desired outcome.

Oriented
Functionally aligning architecture to enable a collection of independent services to be linked together to solve a business problem.

Architecture
The fundamental organization of a system embodied in its capabilities, their interactions, and the environment.
Analogy - traditional software architecture versus SOA

**Traditional approach to software architecture**

"Separate Specialist" model

- No *Agility* to repair your car even for trivial tasks
- A *Process* that is duplicative and inefficient
- *Costly* to operate and maintain – keep many people

- In garage every mechanic specialize only in one type of car so it does not matter what you want to repair you always have to wait for a mechanic who knows your type of car; if he/she is sick or on holiday you cannot repair your car at all

**Service-Oriented Architecture**

"Service-Oriented" model

- *Agility* to repair cars quickly (next available mechanic takes care)
- A *Process* that is efficient
- *Cost* effective to operate and maintain

- Mechanic does job himself or asks other mechanics to take care of tasks he is not capable to do

- You ask any mechanic in a garage to repair your car – model of your car does not matter
## SOA Benefits

**SOA allows to align IT with mission of the organization**

**Better agility, no redundancy, system interactions, and reduces overall costs of system maintenance**

<table>
<thead>
<tr>
<th><strong>Agility</strong></th>
<th><strong>Process Heterogeneity</strong></th>
<th><strong>Data Heterogeneity</strong></th>
<th><strong>Costs</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus more on core competencies</td>
<td>Allow interoperation with other systems without time consuming customization and point-to-point integration</td>
<td>Improve semantics of data exchanged during business process execution</td>
<td>Leverage existing IT infrastructure</td>
</tr>
<tr>
<td>Creates a network of service requesters (consumers) and service providers (producers)</td>
<td>Ensure system change is not a constraint on business or mission change</td>
<td>Maintain consistency of data across different systems</td>
<td>Reduce costs of development of new functionalities by acquiring pre-built components/services</td>
</tr>
<tr>
<td>Enable enterprises to be more agile and respond quickly to changing requirements</td>
<td>Facilitate integration with multiple solutions via open IT standards</td>
<td>Remain platform, language, and vendor independent to remove IT barriers for using best-of-breed software packages</td>
<td>Lower maintenance costs</td>
</tr>
<tr>
<td>Increase business flexibility through plug-and-play architecture and re-use of existing services</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SOA 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)
Benefits of SOA

• Better reuse
  – Build new functionality (new execution semantics) on top of existing Business Services

• Well defined interfaces
  – Manage changes without affecting the Core System

• Easier Maintainability
  – Changes/Versions are not all-or-nothing

• Better Flexibility
Currently, computer science is in a new period of abstraction. A generation ago we learnt to abstract from hardware and currently we learn to abstract from software in terms of SERVICE oriented architectures (SOA).

It is the service that counts for a customer and not the specific software or hardware that is used to implement the service.

In a later stage, we may even talk in terms of problem-oriented architectures (or more positively expressed in terms of problem-solving oriented architectures) because SOAs are biased towards the service provider and not towards the customer that has a problem that needs to be solved.
Semantically Empowered Service-oriented Architecture (SESA)

- Service-oriented architectures will become quickly the leading software paradigm
- However, SOAs will not scale without significant mechanization of
  - Service discovery, service adaptation, negotiation, service composition, service invocation, and service monitoring; and
  - Data and process mediation
- Therefore, machine processable semantics needs to be added to bring SOAs to their full potential
- Development of open standards (languages) and open source architectures and tools that add semantics to service descriptions
Semantic Web Services Infrastructure

- A service oriented architecture.
- Reference implementation of WSMO
User Service versus Platform Service in SWS Systems

External SOA

User Service

Platform Service: Data Mediation

SEE

Platform Service: Discovery

Execution Management (Execution Semantics)

Platform Service: [...]
Vertical and Horizontal Services

- Vertical services remain invisible to horizontal services, and during its execution, the horizontal services remain unaware that vertical services are executed together with them.
- Vertical services invoke horizontal services, coordinating overall workflow, rather than horizontal service invoking the vertical.
Overview

- Introduction to SWS
  - WSMO
- Introduction to SOA
  - WSMX
- Means of Interoperability
- WSMT
- Conclusions
WSMX Introduction

- Software framework for runtime binding of service requesters and service providers
- WSMX interprets service requester’s goal to
  - discover matching services
  - select (if desired) the service that best fits
  - provide mediation (if required)
  - make the service invocation
- Is based on the conceptual model provided by WSMO
- Has a formal execution semantics
- Service Oriented and event-based architecture
  - based on microkernel design using technologies as J2EE, Hibernate, Spring, JMX, etc.
WSMX Motivation

- Provide middleware ‘glue’ for Semantic Web Services
  - Allow service providers focus on their business
- Provide a reference implementation for WSMO
  - Eat our own cake
- Provide an environment for goal based service discovery and invocation
  - Run-time binding of service requester and provider
- Provide a flexible Service Oriented Architecture
  - Add, update, remove components at run-time as needed
- Keep open-source to encourage participation
  - Developers are free to use in their own code
- Define formal execution semantics
  - Unambiguous model of system behaviour
WSMX Usage Scenario

[Diagram showing a three-tier architecture for B2C and a two-tier architecture for B2B, with User (Service Requester), Web Server / Application Server, WSMX Server, and WSMX Client connected.]
WSMX Usage Scenario - P2P

- A P2P network of WSMX ‘nodes’
- Each WSMX node described as a SWS
- Communication via WSML over SOAP
- Distributed discovery – first aim
- Longer term aim - distributed execution environment
WSMX Usage Scenario - P2P

[Diagram showing Peer-to-Peer communication with messages passing through the Internet and a central WSMX SWS ARCHITECTURE]
WSMX Usage Scenario - P2P
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
Benefits of SOA

- **Better reuse**
  - Build new functionality (new execution semantics) on top of existing Business Services
- **Well defined interfaces**
  - Manage changes without affecting the Core System
- **Easier Maintainability**
  - Changes/Versions are not all-or-nothing
- **Better Flexibility**
WSMX Architecture

![WSMX Architecture Diagram]

The WSMX Architecture diagram illustrates the components and interfaces involved in Web Service Management. It includes:

- **WSMX Management**
- **WSMX Monitor**
- **WSML Editor**
- **Ontology Visualizer**
- **Mapping Tools**

The diagram shows the integration of various components such as:

- **Interface**
- **Communication Manager**
- **Resource Manager**
- **Parser**
- **Discovery**
- **Selector**
- **Data Mediator**
- **Process Mediator**
- **Choreography**

This architecture supports service-oriented architectures (SOA) by managing and monitoring web services, ensuring efficient service requesters and providers can interact effectively.
Selected Components

- Adapters
- Parser
- Invoker
- Choreography
- Process Mediator
- Discovery
- Data Mediator
- Resource Manager
- Reasoning
Adapters

- To overcome data representation mismatches on the communication layer
- Transforms the format of a received message into WSML compliant format
- Based on mapping rules
Parser

- WSML compliant parser
  - Code handed over to wsmo4j initiative
    http://wsmo4j.sourceforge.net/
- Validates WSML description files
- Compiles WSML description into internal memory model
- Stores WSML description persistently (using Resource Manager)
Communication Manager – Invoker

- WSMX uses
  - The SOAP implementation from Apache AXIS
  - The Apache Web Service Invocation Framework (WSIF)
- WSMO service descriptions are grounded to WSDL
- Both RPC and Document style invocations possible
- Input parameters for the Web Services are translated from WSML to XML using an additional XML Converter component.
Choreography

- Requester and provider have their own observable communication patterns
  - Choreography part of WSMO
- Choreography instances are loaded for the requester and provider
  - Both requester and provider have their own WSMO descriptions
- Choreography Engine
  - Evaluation of transition rules
    - Prepares the available data
  - Sends data to the Process Mediator
    - The Process Mediator filters, changed or even replaced data
  - Receive data from PM and forwards it to the Communication manager
    - Data to be finally sent to the communication partner
Process Mediator

- Requester and provider have their own communication patterns
- Only if the two match precisely, a direct communication may take place
- The Process Mediator provides the means for runtime analyses of two choreography instances and uses mediators to compensate possible mismatches
Discovery

- Responsible for finding appropriate Web Services to achieve a goal (discovery)
- Current discovery component is based on simple matching
  - Keywords identified in the NFP of the goal
  - Matched against NFPs of the published WSs
  - Variable set of NFPs to be considered for this process
  - To be extended
    - Values in NFPs might be concepts from ontologies
    - More elaborate string matching algorithms

- Advanced semantic discovery in prototypical stage
Data Mediator

- Ontology-to-ontology mediation
- A set of mapping rules are defined and then executed
- Initially rules are defined semi-automatic
- Create for each source instance the target instance(s)
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
Reasoner

**Mins**
- Datalog + Negation + Function Symbols Reasoner Engine
- Features
  - Built-in predicates
  - Function symbols
  - Stratified negation

**WSMO4J**
- validation, serialization and parsing

**WSML2Reasoner**
- Reasoning API
  - mapping from WSML to a vendor-neutral rule representation
- Contains:
  - Common API for WSML Reasoners
  - Transformations of WSML to tool-specific input data (query answering or instance retrieval)

**WSML-DL-Reasoner**
- Features:
  - T-Box reasoning (provided by FaCT++)
  - Querying for all concepts
  - Querying for the equivalents, for the children, for the descendants, for the parents and for all ancestors of a given concept
  - Testing the satisfiability of a given concept with respect to the knowledge base
  - Subsumption test of two concepts with respect to the knowledge base
  - Wrapper of WSML-DL to the XML syntax of DL used in the DIG interface
System Entry Points

- `achieveGoal` (WSMLDocument): Context
- `getWebServices` (WSMLDocument): Context
- `invokeWebService` (WSMLDocument, Context): Context
Define “Business” Process
Generate Wrappers for Components

- Discover Web Services → Start
  - Create Choreography
  - Created
  - Discover Services
  - Mediate Data
    - Mediate Data
    - Return Mediated Data
      - Return Mediated Data
      - Return Web Services
        - Check Choreography
          - Confirmed
          - Call Invoker
            - Confirmed
            - End
  - Choreography Wrapper
  - Discovery Wrapper
  - Registry of known components

OASIS Symposium 2006
Context Data
Event-based Implementation

Core - Manager

"Business" Process - Internal Workflow

Event and Notification Distribution/Delivery Mechanism

Choreography Wrapper

Discovery Wrapper

Data Mediator Wrapper

Communication Manager Wrapper

Choreography

Discovery

Mediator

Communication Manager
WSMX Conclusions

- Conceptual model is WSMO
- End to end functionality for executing SWS
- Has a formal execution semantics
- Real implementation
- Open source code base at SourceForge
- Event-driven component architecture
Overview

- Introduction to SWS
  - WSMO
- Introduction to SOA
  - WSMX
- **Means of Interoperability**
- WSMT
- Conclusions
Means of Interoperability

- Format and Language heterogeneity
  - Adaptors to/from WSML
- Interface/communications formalism
  - Choreography and Orchestration
- Ontology heterogeneity
  - Data Mediation
- Interface/communication patterns heterogeneity
  - Process Mediation
Adapter Framework

• Overview
  – Overcomes mismatches at the communication layer
  – Is based on Java Connector Architecture (JCA)
  – Is based on SOA design principles
  – Adapters function independently
  – Adapters are built based on mapping rules
  – Is developed in Java

• Motivation
  – WSMX does not recognize message formats other than WSML
  – Backend applications that do not use WSML cannot communicate with WSMX without the help of adapters that transforms the format of a received message to WSML format
  – Provide a unified framework for developing and using adapters
Features

- Adapters can be added and removed at run time
- Secure pluggability
- Supports both synchronous and asynchronous communication
- Handles communication protocol heterogeneity, i.e., allow to communicate using HTTP(S), TCP/IP, UDP
- Provides simple operations:
  - *Deploy*: adds adapter to the adapter pool
  - *Undeploy*: removes adapter from the adapter pool, subject to security constraints
  - *Send*: send legacy message to WSMX
  - *Receive*: receive legacy message from WSMX
Adapter Framework – Deploy adapter

Request sent to deploy an adapter

WSMX

deploy (adapterName, someAdapter.adapter)

Validator (Message, Protocol)

Adapter Manager/Selector

Adapter Pool

<table>
<thead>
<tr>
<th>Inward</th>
<th>Outward</th>
</tr>
</thead>
<tbody>
<tr>
<td>UBL2WSML</td>
<td>WSML2UBL</td>
</tr>
<tr>
<td>EDI2WSML</td>
<td>WSML2EDI</td>
</tr>
<tr>
<td>RosettaNet2WSML</td>
<td>WSML2RosettaNet</td>
</tr>
<tr>
<td>...2WSML</td>
<td>WSML2 ...</td>
</tr>
</tbody>
</table>

Metadata Repository

OASIS Symposium 2006
Adapter Framework – Deploy adapter

```c
deploy (adapterName, someAdapter.adapter)
```

Communication type scanned
Adapter Framework – Deploy adapter

Fingerprint for this adapter created
Adapter Framework – Deploy adapter

Metadata updated

Adapter Pool

<table>
<thead>
<tr>
<th>Inward</th>
<th>Outward</th>
</tr>
</thead>
<tbody>
<tr>
<td>UBL2WSML</td>
<td>WSML2UBL</td>
</tr>
<tr>
<td>EDI2WSML</td>
<td>WSML2EDI</td>
</tr>
<tr>
<td>RosettaNet2WSML</td>
<td>WSML2RosettaNet</td>
</tr>
<tr>
<td>...</td>
<td>WSML2...</td>
</tr>
<tr>
<td>...</td>
<td>WSML2...</td>
</tr>
<tr>
<td>...</td>
<td>WSML2...</td>
</tr>
</tbody>
</table>

Validator (Message, Protocol)

Adapter Manager/Selector

WSMX
Adapter Framework – Deploy adapter

Adapter stored

Validator (Message, Protocol)
Adapter Manager/Selector

Adapter Pool
- Inward
- UBL2WSML
- EDI2WSML
- RosettaNet2WSML
- ... 2WSML
- ... 2WSML
- ... 2WSML
- ...2WSML
- adapterName2WSML

Outward
- WSML2UBL
- WSML2EDI
- WSML2RosettaNet
- WSML2...
- WSML2...
- WSML2...
- WSML2...
- WSML2adapterName

Metadata Repository

WSMX

Listener
Protocol Handler (Sync, Async)
Security Manager
Communication Manager
Adapter Framework – Deploy adapter

Validator (Message, Protocol)

Adapter Manager/Selector

Adapter Pool

<table>
<thead>
<tr>
<th>Inward</th>
<th>Outward</th>
</tr>
</thead>
<tbody>
<tr>
<td>UBL2WSML</td>
<td>WSML2UBL</td>
</tr>
<tr>
<td>EDI2WSML</td>
<td>WSML2EDI</td>
</tr>
<tr>
<td>RosettaNet2WSML</td>
<td>WSML2RosettaNet</td>
</tr>
<tr>
<td>...2WSML</td>
<td>WSML2...</td>
</tr>
<tr>
<td>...2WSML</td>
<td>WSML2...</td>
</tr>
<tr>
<td>...2WSML</td>
<td>WSML2...</td>
</tr>
<tr>
<td>adapterName2WSML</td>
<td>WSML2adapterName</td>
</tr>
</tbody>
</table>

Fingerprint returned in a requested communication mode
Adapter Framework – Deploy adapter

Fingerprint returned in to backend application
Adapter Framework – Send

Message send to WSMX via Adapter Framework

send (adapterName, message, fingerprint)
Adapter Framework – Send

Communication type scanned

send (adapterName, message, fingerprint)
Adapter Framework – Send

Fingerprint checked, valid fingerprint

send (adapterName, message, fingerprint)
Adapter Framework – Send

Adapter Pool

<table>
<thead>
<tr>
<th>Adapter Name</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inward</td>
<td>Outward</td>
</tr>
<tr>
<td>UBL2WSML</td>
<td>WSML2UBL</td>
</tr>
<tr>
<td>EDI2WSML</td>
<td>WSML2EDI</td>
</tr>
<tr>
<td>RosettaNet2WSML</td>
<td>WSML2RosettaNet</td>
</tr>
<tr>
<td>...2WSML</td>
<td>WSML2...</td>
</tr>
<tr>
<td>...2WSML</td>
<td>WSML2...</td>
</tr>
<tr>
<td>...2WSML</td>
<td>WSML2...</td>
</tr>
<tr>
<td>adapterName2WSML</td>
<td>WSML2adapterName</td>
</tr>
</tbody>
</table>

Message format checked, valid fingerprint
Adapter Framework – Send

Internal request sent to select adapterName2WSML

send (adapterName, message, fingerprint)
Adapter Framework – Send

adapterName2WSML selected looking into metadata repository

Validator (Message, Protocol)

Adapter Manager/Selector

Adapter Pool

<table>
<thead>
<tr>
<th>Inward</th>
<th>Outward</th>
</tr>
</thead>
<tbody>
<tr>
<td>UBL2WSML</td>
<td>WSML2UBL</td>
</tr>
<tr>
<td>EDI2WSML</td>
<td>WSML2EDI</td>
</tr>
<tr>
<td>RosettaNet2WSML</td>
<td>WSML2RosettaNet</td>
</tr>
<tr>
<td>...2WSML</td>
<td>WSML2...</td>
</tr>
<tr>
<td>...2WSML</td>
<td>WSML2...</td>
</tr>
<tr>
<td>...2WSML</td>
<td>WSML2...</td>
</tr>
<tr>
<td>adapterName2WSML</td>
<td>WSML2adapterName</td>
</tr>
</tbody>
</table>

send (adapterName, message, fingerprint)
Adapter Framework – Send

Message translated and sent to WSMX
Adapter Framework – Undeploy adapter

Request sent to undeploy an adapter together with its fingerprint

`undeploy (adapterName, D749 9163 9E5E BDFC 8018 E6B8 49DD 3252 ACF6 7294)`
Adapter Framework – Undeploy adapter

Fingerprint checked, valid fingerprint

undeploy (adapterName, D749 9163 9E5E BDFC 8018 E6B8 49DD 3252 ACF6 7294)
Adapter Framework – Undeploy adapter

Metadata updated
Adapter Framework – Undeploy adapter

WSMX

Adapter removed

Listener

Protocol Handler (Sync, Async)

Security Manager

Communication Manager

Validator (Message, Protocol)

Adapter Manager/Selector

Adapter Pool

<table>
<thead>
<tr>
<th>Inward</th>
<th>Outward</th>
</tr>
</thead>
<tbody>
<tr>
<td>UBL2WSML</td>
<td>WSML2UBL</td>
</tr>
<tr>
<td>EDI2WSML</td>
<td>WSML2EDI</td>
</tr>
<tr>
<td>RosettaNet2WSML</td>
<td>WSML2RosettaNet</td>
</tr>
<tr>
<td>...2WSML</td>
<td>WSML2...</td>
</tr>
<tr>
<td>...2WSML</td>
<td>WSML2...</td>
</tr>
<tr>
<td>...2WSML</td>
<td>WSML2...</td>
</tr>
</tbody>
</table>

Metadata Repository
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
Abstract State Machine

- **Formality**
  - a rigid framework to express dynamics.

- **Maximality**
  - expressive enough to model any aspect around computation

- **Minimality**
  - minimal set of modeling primitives – minimal ontological commitment
Choreography outline

Class choreography
  hasNonFunctionalProperties type nonFunctionalProperties
  hasStateSignature type stateSignature
  hasTransitionRules type transitionRules

• NFPs
  – The same as in WSML

• State Signature
  – Defines the state ontology used by the service together with the
    definition of the types of modes the concepts and relations may
    have

• Transition Rules
  – Express changes of states
States Signatures

Class stateSignature
  hasNonFunctionalProperties type nonFunctionalProperties
  importsOntology type ontology
  usesMediator type ooMediator
  hasStatic type mode
  hasIn type mode
  hasOut type mode
  hasShared type mode
  hasControlled type mode

Class mode subClass {concept, relation}
  hasGrounding type grounding
Transition Rules

• *if* $\varphi$ *then* $T$ *endif*
• *forall* $V$ *with* $\psi$ *do* $T'$ *endForall*
• *choose* $V$ *with* $\psi$ *do* $T'$ *endChoose*

• $\varphi$ is a first order formula with no free variables
• $V$ is a set of variables
• $\psi$ is a first order formula where the free variables are interpreted as parameters and all free variables in $\psi$ occur in $V$
• $T$ is a set of transition rules
• $T'$ is a set of transition rules and/or non-ground update rules, where each variable which occurs in any non-ground update rule in $T'$, occurs also in $V$
Update rules

• add(a)
• delete(a)
where \( a \) is a WSML atomic formula, which possibly includes
  – parameter variables, or
  – non-primitive update rules of the form:
• update(a_{\text{new}})
• update(a_{\text{old}} \rightarrow a_{\text{new}})

• \( S^U = S \setminus \{a | \text{delete}(a) \in U\} \cup \{a | \text{add}(a) \in U\} \)
where \( O \) is an ontology \( O \), \( S \) a state and \( U \) an update set
Machine behaviour

Given $C = (O, T, S)$

- $S_0 = S$
- for $0 \leq i \leq n-1$,
  - $S_i \neq S_{i+1}$
  - $U = \{\text{add}(a) \mid a \in S_{i+1} \setminus S_i\} \cup \{\text{delete}(a) \mid a \in S_i \setminus S_{i+1}\}$ is an update set associated with $S_i$, $O$ and $T$
  - $S_{i+1}$ is consistent with $O$, and $S_i$
- run terminated
Data Mediator

- Ontology-to-ontology mediation
- A set of mapping rules are defined and then executed
- Initially rules are defined semi-automatic
- Create for each source instance the target instance(s)
Design-time

- **Inputs**
  - Source Ontology and Target Ontology

- **Features**
  - Graphical interface
  - 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
Design-time Phase
Design-time Phase - Approach, Decomposition and Mapping Context

- Bottom-up -> training set
- Top-down -> decomposition, context
Design-time Phase - Suggestion Algorithms

- Eligibility Factor = f(Lexical Factor, Structural Factor)

- Lexical Factor:
  - WordNet
    - Synonyms, hyponyms, hipernyms
  - string analyzing algorithms
    - Tokenizer and string distance

- Structural Factor
  - Decomposition, EF for the composing concepts
    - Based on the already done mappings
Run-Time Data Mediator

- Main Mediation Scenario: Instance Transformation

- Inputs
  - Incoming data
    - Source ontology instances

- Features
  - Completely automatic process
  - Grounding of the abstract mappings to a concrete language
    - F-Logic, WSML
  - Uses a reasoner to evaluate the mapping rules
    - MINS

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

- Source
- Instance
- Reasoning Environment
- Target
- Instance
- Rules Generator
- Abstract Mappings Repr.
- Mapping Rules
- Ontologies

Abstract Mappings Repr.
Run Time Component – Features

- **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
- Caching mechanism can be used
Ontology Mapping Language

- **Language Neutral Mapping Language**
  - mapping definitions on meta-layer (i.e. on generic ontological constructs)
  - independent of ontology specification language
  - “Grounding” to specific languages for execution (WSML, OWL, F-Logic)

- **Main Features:**
  - Mapping Document (sources, mappings, mediation service)
  - direction of mapping (uni- / bidirectional)
  - conditions / logical expressions for data type mismatch handling, restriction of mapping validity, and complex mapping definitions
  - mapping constructs:
    - classMapping, attributeMapping, relationMapping (between similar constructs)
    - classAttributeMapping, classRelationMapping, classInstanceMapping
    - instanceMapping (explicit ontology instance transformation)
  - mapping operators:
    - =, <, <=, >, >=, and, or, not
    - inverse, symmetric, transitive, reflexive
    - join, split
Mapping Language Example

Ontology O1

Human
- name

Adult
Child

Ontology O2

Person
- name
- age

mick `memberOf` Person
- name = Mick Kerrigan
- age = 27

```
classMapping (unidirectional o2:Person o1.Adult
attributeValueCondition (o2.Person.age >= 18))
```

This allows to transform the instance ‘mick’ of concept person in ontology O2 into a valid instance of concept ‘adult’ in ontology O1.
Process Mediator

- Requester and provider have their own communication patterns
- Only if the two match precisely, a direct communication may take place
- The Process Mediator provides the means for runtime analyses of two choreography instances and uses mediators to compensate possible mismatches
Compatibility

- Two business partners are compatible if their public processes are matching.

![Diagram showing compatibility between Business Partner 1 and Business Partner 2 with processes A, B, C, D, E matching]
Two business partners are compatible if their public processes are matching.
Process Mediator – Addressed Mismatches
Process Mediator – Unsolvable Mismatches

[Diagram showing interactions between Business Partner 1 and Business Partner 2 through a process mediator (PM). Arrows indicate flow and interactions such as A, B, and Ack.]
Process Mediation Example

REQUEST

itinerary[origin, destination, date]

time

price

Processes Mediator

origin

destination

itinerary[origin, destination]

date

ticket[route, date, time, price]

SERVICE
Processes Mediation Example

REQUEST

itinerary[origin, destination, date]

Processes Mediator

SERVICE

origin

destination

itinerary[origin, destination]

date

ticket[route, date, time, price]
Process Mediation Example

REQUEST

itinerary[origin, destination, date]

time

price

Processes Mediator

SERVICE

origin

destination

itinerary[origin, destination]

date

ticket[route, date, time, price]
Process Mediation Example

REQUEST

- itinerary[origin, destination, date]
- time
- price

Processes Mediator

SERVICE

- origin
- destination
- itinerary[origin, destination]
- date
- ticket[route, date, time, price]

OASIS Symposium 2006
Process Mediation Example

REQUEST

- itinerary[origin, destination, date]
- time
- price

Processes Mediator

SERVICE

- origin
- destination
- itinerary[origin, destination]
- date
- ticket[route, date, time, price]
Overview

• Introduction to SWS
  – WSMO
• Introduction to SOA
  – WSMX
• Means of Interoperability
• WSMT
• Conclusions
Web Services Modeling Toolkit

- The aim of the Web Services Modeling Toolkit (WSMT) is to provide high-quality tools for designing, mediating and using Semantic Web Services, through the WSMO paradigm.

- The focus is currently on the following areas:
  - Creation of ontologies, web services, goals and mediators in WSMO
  - Creation of mappings between pairs of ontologies to allow runtime instance transformation
  - Management of Execution Environments for Semantic Web Services like WSMX and IRSIII
WSML Perspective

- Perspectives in the Eclipse framework allow for a number of Editors and views to be grouped and positions.
- The WSML perspective offers editors and views related to engineering of semantic descriptions in WSMO through the WSML language.
- Other General features include:
  - WSML file validation
  - Problems view (errors and warnings on files in the workspace)
  - Label highlighting (marking of errors and warnings in navigator view)
WSML Editors and Views in the WSML perspective

**Editors**
- WSML Text Editor
- WSML Conceptual Editor
- WSML Visualizer

**Views**
- Navigator view
- Problems view
- WSML Reasoner
Editors and Views in the WSML perspective

Editors
WSML Text Editor
WSML Conceptual Editor
WSML Visualizer

Views
Navigator view
Problems view
WSML Reasoner
Editors and Views in the WSML perspective

Editors
WSML Text Editor
WSML Conceptual Editor
WSML Visualizer

Views
Navigator view
Problems view
WSML Reasoner
Editors and Views in the WSML perspective

Editors
- WSML Text Editor
- WSML Conceptual Editor
- WSML Visualizer

Views
- Navigator view
- Problems view
- WSML Reasoner
Editors and Views in the WSML perspective

Editors
WSML Text Editor
WSML Conceptual Editor
WSML Visualizer

Views
Navigator view
Problems view
WSML Reasoner
Editors and Views in the WSML perspective

**Navigator view**

**Problems view**

WSML Reasoner
Editors and Views in the WSML perspective

Editors
WSML Text Editor
WSML Conceptual Editor
WSML Visualizer

Views
Navigator view
Problems view
WSML Reasoner
Editors

**AML Text Editor**
**AML Conceptual Editor**
**AML View Based Editor**

Views

- Concept 2 Concept View
- Attribute 2 Attribute View
- Concept 2 Attribute View
- Attribute 2 Concept View
- Status View
Editors
AML Text Editor
AML Conceptual Editor
AML View Based Editor

Views
Concept 2 Concept View
Attribute 2 Attribute View
Concept 2 Attribute View
Attribute 2 Concept View
Status View
Editors, Views for the Abstract Mapping Language

Editors
AML Text Editor
AML Conceptual Editor
AML View Based Editor

Views
Concept 2 Concept View
Attribute 2 Attribute View
Concept 2 Attribute View
Attribute 2 Concept View
Status View
Editors, Views for the Abstract Mapping Language

<table>
<thead>
<tr>
<th>Source Concepts</th>
<th>Target Concepts</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>integer</td>
<td>No conditions associated</td>
</tr>
<tr>
<td>string</td>
<td>string</td>
<td>No conditions associated</td>
</tr>
<tr>
<td>date</td>
<td>date</td>
<td>No conditions associated</td>
</tr>
</tbody>
</table>

Views
- Concept 2 Concept View
- Attribute 2 Attribute View
- Concept 2 Attribute View
- Attribute 2 Concept View
- Status View
Editors, Views for the Abstract Mapping Language

### Views

- **Concept 2 Concept View**
- **Attribute 2 Attribute View**
- Concept 2 Attribute View
- Attribute 2 Concept View
- Status View

<table>
<thead>
<tr>
<th>Source Attributes</th>
<th>Target Attributes</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>[(date) day =&gt; integer]</td>
<td>[(date) day =&gt; integer]</td>
<td>No conditions associated</td>
</tr>
<tr>
<td>[(date) month =&gt; integer]</td>
<td>[(date) month =&gt; integer]</td>
<td>No conditions associated</td>
</tr>
<tr>
<td>[(date) year =&gt; integer]</td>
<td>[(date) year =&gt; integer]</td>
<td>No conditions associated</td>
</tr>
</tbody>
</table>
Editors, Views for the Abstract Mapping Language

<table>
<thead>
<tr>
<th>Source Concepts</th>
<th>Target Attributes</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ticket</td>
<td>((TravelVoucher.bearer =&gt; name)</td>
<td>No conditions associated</td>
</tr>
</tbody>
</table>

Concept 2 Concept View
Attribute 2 Attribute View
**Concept 2 Attribute View**
Attribute 2 Concept View
Status View
Editors, Views for the Abstract Mapping Language

<table>
<thead>
<tr>
<th>Source Attributes</th>
<th>Target Concepts</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>[(tennis price =&gt; cost)]</td>
<td>payment</td>
<td>No conditions associated</td>
</tr>
</tbody>
</table>

- Concept 2 Concept View
- Attribute 2 Attribute View
- Concept 2 Attribute View
- **Attribute 2 Concept View**
- Status View
Editors, Views for the Abstract Mapping Language

Concept 2 Concept View
Attribute 2 Attribute View
Concept 2 Attribute View
Attribute 2 Concept View
Status View
Overview

• Introduction to SWS
  – WSMO
• Introduction to SOA
  – WSMX
• Means of Interoperability
• WSMT
• Conclusions
Conclusions

- **Semantic Enabled SOA** combines the benefits of semantics with best practices from industry

- **WSMO** - conceptual model for Semantic Web Services
  - Ontology of core elements for Semantic Web Services

- Clear separation between layers
  - Specification and realization
  - Interface and implementation

- **WSMX/SEE** – a Semantic Enabled SOA
  - Service Oriented Architecture
  - Reference implementation of WSMO

- Semantic Enabled SOA offers multiple means for interoperability
  - Mediation framework
  - Interface/communication disambiguation

- **WSMT** – emerging tool to handle semantics
  - High-quality tools for designing, mediating and using Semantic Web Services
References

- The central location where WSMO work and papers can be found is WSMO Working Group: [http://www.wsmo.org](http://www.wsmo.org)


- WSMO implementation
  - WSMX working group: [http://www.wsmx.org](http://www.wsmx.org)
  - WSMX open source can be found at: [https://sourceforge.net/projects/wsmx/](https://sourceforge.net/projects/wsmx/)
References

- [WSMO Use Case] Stollberg, M.; Lausen, H.; Polleres, A.; Lara, R. (ed.): WSMO Use Case Modeling and Testing, WSMO Working Drafts D3.2; D3.3; D3.4; D3.5, 05 November 2004.
References


- [Cimpian and Mocan, 2005] Emilia Cimpian, Adrian Mocan: WSMX Process Mediation Based on Choreographies, 1st International Workshop on Web Service Choreography and Orchestration for Business Process Management (BPM 2005), September 2005, Nancy, France


References


References


References


- [Stolberg et al., 2006] Michael Stollberg, Emilia Cimpian, Adrian Mocan, Dieter Fensel: A Semantic Web Mediation Architecture, Canadian Semantic Web Working Symposium (CSWWS 2006), June 2006, Québec city, Canada

Acknowledgements

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.

The WSMO working groups are funded by the European Commission under the projects ASG, DIP, Knowledge Web, SEKT, SemanticGov, SWWS, AKT and Esperonto; by Science Foundation Ireland under the DERI-Lion project; and by the Austrian government under the FIT-IT program.