Semantic Web Services
Tutorial

Michael Stollberg¹, Matt Moran², John Domingue³, and Liliana Cabral³

5th International Conference on Web Engineering (ICWE 2005)
Sydney, Australia, 2005 July 25

¹DERI, Innsbruck, Austria
²DERI, Galway, Ireland
³Knowledge Media Institute, The Open University, UK
Agenda

Part I: Introduction to Semantic Web Services
  – Vision of Next Generation Web Technology
  – Semantic Web Service Challenges

Part II: The Web Service Modeling Ontology WSMO
  – Aims & Design Principles
  – Top Level Element Definitions

BREAK

Part III: A Walkthru Example
  – Virtual Travel Agency Example
  – Roles, Elements, Semantic Web Service technology usage

LUNCH

Part IV: The Web Service Execution Environment WSMX
  – Aims & Design Principles
  – Architecture & Components

BREAK

Part V: Hands-On Session with IRS III
  – IRS III introduction
  – Hands-on Session (explanation, hands-on)
PART I:
Introduction to Semantic Web Services

• The vision of the Semantic Web
• Ontologies as the basic building block
• Current Web Service Technologies
• Vision and Challenges for Semantic Web Services
The Vision

- 500 million users
- more than 3 billion pages

Static

WWW

URI, HTML, HTTP
The Vision

Serious Problems in

- information finding,
- information extracting,
- information representing,
- information interpreting and
- and information maintaining.

Static

<table>
<thead>
<tr>
<th>WWW</th>
<th>Semantic Web</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI, HTML, HTTP</td>
<td>RDF, RDF(S), OWL</td>
</tr>
</tbody>
</table>
The Vision

Dynamic

Web Services
UDDI, WSDL, SOAP

Static

WWW
URI, HTML, HTTP

Syntactic

Semantic Web
RDF, RDF(S), OWL

Semantic

Bringing the computer back as a device for computation
The Vision

Bringing the web to its full potential

Dynamic

Web Services
UDDI, WSDL, SOAP

Static

WWW
URI, HTML, HTTP

Syntactic

Semantic Web
Services

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

• the next generation of the WWW

• information has machine-processable and machine-understandable semantics

• not a separate Web but an augmentation of the current one

• Ontologies as basic building block
Ontology Definition

formal, explicit specification of a shared conceptualization

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

Concept
conceptual entity of the domain

Property
attribute describing a concept

Relation
relationship between concepts or properties

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

holds(Professor, Lecture) => Lecture.topic = Professor.researchField
Ontology Technology

To make the Semantic Web working we need:

- **Ontology Languages:**
  - expressivity
  - reasoning support
  - web compliance

- **Ontology Reasoning:**
  - large scale knowledge handling
  - fault-tolerant
  - stable & scalable inference machines

- **Ontology Management Techniques:**
  - editing and browsing
  - storage and retrieval
  - versioning and evolution Support

- **Ontology Integration Techniques:**
  - ontology mapping, alignment, merging
  - semantic interoperability determination

- and ... **Applications**
Web Services

- 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
The Promise of Web Services

web-based SOA as new system design paradigm

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

- WSDL
  - Describes Service

- Service Consumer
  - Communicates with XML Messages
  - SOAP

- Web Service
WSDL

- Web Service Description Language
- W3C effort, WSDL 2 final construction phase

describes interface for consuming a Web Service:
- Interface: operations (in- & output)
- Access (protocol binding)
- Endpoint (location of service)
UDDI

- Universal Description, Discovery, and Integration Protocol
- OASIS driven standardization effort

Registry for Web Services:
- provider
- service information
- technical access
SOAP

- Simple Object Access Protocol
- W3C Recommendation

XML data transport:
- sender / receiver
- protocol binding
- communication aspects
- content
Deficiencies of WS Technology

- current technologies allow usage of Web Services
- but:
  - only syntactical information descriptions
  - syntactic support for discovery, composition and execution
  => **Web Service usability, usage, and integration needs to be inspected manually**
  - no semantically marked up content / services
  - no support for the Semantic Web

=> current Web Service Technology Stack failed to realize the promise of Web Services
Semantic Web Services

**Semantic Web Technology**
- allow machine supported data interpretation
- ontologies as data model

**Web Service Technology**
automated discovery, selection, composition, and web-based execution of services

=> Semantic Web Services as integrated solution for realizing the vision of the next generation of the Web
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

Usage Process:
• Publication: Make the description of a Web service available on the Web
• Discovery: Detect suitable services for a solving given task
• Selection: Choose the most appropriate services among the usable ones
• Composition: Combine services to achieve a goal
• Mediation: Solve mismatches (data, protocol, process) among the elements that shall interoperate
• Execution: Invoke services according to consumption interface and programmatic conventions
Semantic Web Services

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
PART II:
The Web Service Modeling Ontology WSMO

• Aims & Working Groups
• Design Principles
• Top Level Notions
  – Ontologies
  – Web Services
  – Goals
  – Mediators
• Comparison to OWL-S
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

• a 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
- Goal-driven
- Strict Decoupling
- Centrality of Mediation
- Description versus Implementation
- Execution Semantics
WSMO Top Level Notions

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

every WSMO element 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, etc.
## Non-Functional Properties List

<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>
</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

Connectors between components with mediation facilities for handling heterogeneities

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

- Advertising of Web Service
- Support for WS Discovery

Capability

DC + QoS + Version + financial

realization of functionality by aggregating other Web Services
- functional decomposition
- interaction with aggregated WS

Choreography --- Service Interfaces --- Orchestration

Web Service Implementation

(not of interest in Web Service Description)

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Flight, Hotel</th>
<th>Error</th>
<th>Confirmation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

Control Structure for aggregation of other Web Services

- decomposition of service functionality
- interaction with aggregated Web Services
- all service interaction via choreographies
WSMO Web Service Interfaces

- service interfaces are concerned with service consumption and interaction
- Choreography and Orchestration as sub-elements of Web 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.
Service Interface Description Approach

• Ontologies as data model:
  – all data elements interchanged are ontology instances
  – service interface = evolving ontology

• Abstract State Machines (ASM) as formal framework:
  – dynamics representation: high expressiveness & low ontological commitment
  – core principles: state-based, state definition by formal algebra, guarded transitions for state changes
  – overcome the “Frame Problem”

• further characteristics:
  – not restricted to any specific communication technology
  – ontology reasoning for service interoperability determination
  – basis for declarative mediation techniques on service interfaces
Service Interface Description Model

• Vocabulary $\Omega$:
  – ontology schema(s) used in service interface description
  – usage for information interchange: in, out, shared, controlled

• States $\omega(\Omega)$:
  – a stable status in the information space
  – defined by attribute values of ontology instances

• Guarded Transition $GT(\omega)$:
  – state transition
  – general structure: if (condition) then (action)
  – different for Choreography and Orchestration
  – additional constructs: add, delete, update
Service Interface Example

Communication Behavior of a Web Service

\[ \Omega_{\text{in}} \text{ hasValues } \{
\text{concept A [}
\text{att1 ofType X}
\text{att2 ofType Y]}
\ldots\}\]

\[ \Omega_{\text{out}} \text{ hasValues } \{
\text{concept B [}
\text{att1 ofType W}
\text{att2 ofType Z]}
\ldots\}\]

\begin{align*}
\text{State } \omega_1, \\
\text{a memberOf A [}
\text{att1 hasValue x}
\text{att2 hasValue y]} \\
\text{received ontology instance } a
\end{align*}

\begin{align*}
\text{Guarded Transition } GT(\omega_1) \\
\text{IF (a memberOf A [}
\text{att1 hasValue x ] }) \\
\text{THEN} \\
\text{(b memberOf B [}
\text{att2 hasValue m ]}) \rightarrow \\
\text{State } \omega_2 \\
\text{b memberOf B [}
\text{att2 hasValue m]} \\
\text{sent ontology instance } b
\end{align*}

Vocabulary:
- Concept A in \(\Omega_{\text{in}}\)
- Concept B in \(\Omega_{\text{out}}\)
Future Directions

**Choreography:**
- interaction of services / service and client
- "choreography interface" describes the behavior of a Web Service for client-service interaction for consuming its functionality

**Orchestration:**
- how Web Service interacts as a requester with other Web Services that it aggregates for achieving its functionality
- extends Choreography descriptions by extended action definitions in Guarded Transitions.

**Conceptual models**

**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

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 Architecture:**
  - requester formulates objective independently
  - ‘intelligent’ mechanisms detect suitable services for solving the Goal
  - allows re-use of Services for different purposes

• **Derived from different AI-approaches for intelligent systems**
  – Intelligent Agents (BDI Architectures)
  – Problem Solving Methods

• Requests may in principle not be satisfiable

• Ontological relationships & mediators used to link goals to web services

• Goal Resolution Process open to 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:
  – Mediators as components that resolve mismatches
  – Declarative Approach:
    • Semantic description of resources
    • ‘Intelligent’ mechanisms that resolve mismatches independent of content
  – Mediation cannot be fully automated (integration decision)

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

WSMO Mediator

uses a Mediation Service via

1 .. n

Source Component

1

Target Component

- as a Goal
- directly
- optionally incl. Mediation

Source Component

Mediation Services
OO Mediator - Example

Merging 2 ontologies

Train Connection Ontology (s1) → OO Mediator Mediation Service → Train Ticket Purchase Ontology

Purchase Ontology (s2) → Discovery

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

Mediation Services
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
Comparison to OWL-S

- Mapping to WSDL
  - communication protocol (RPC, HTTP, …)
  - marshalling/serialization
  - transformation to and from XSD to OWL

- Control flow of the service
  - Black/Grey/Glass Box view
  - Protocol Specification
  - Abstract Messages
Perspective

- OWL-S is an ontology and a language to describe Web services
  - Strong relation to Web Services standards
    - rather than proposing another WS standard, OWL-S aims at enriching existing standards
    - OWL-S is grounded in WSDL and it has been mapped into UDDI
  - Based on the Semantic Web
    - Ontologies provide conceptual framework to describe the domain of Web services and an inference engine to reason about the domain
    - Ontologies are essential elements of interoperation between Web services
- WSMO is a conceptual model for the core elements of Semantic Web Services
  - core elements: Ontologies, Web Services, Goals, Mediators
    - language for semantic element description (WSML)
    - reference implementation (WSMX)
  - Mediation as a key element
  - Ontologies as data model
    - every resource description is based on ontologies
    - every data element interchanged is an ontology instance
OWL-S and WSMO

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

- OWL-S uses Profiles to express existing capabilities (advertisements) and desired capabilities (requests)
- WSMO separates provider (capabilities) and requester points of view (goals)
OWL-S and WSMO

OWL-S Process Model $\sim$ WSMO Service Interfaces

- **Perspective:**
  - OWL-S Process Model describes operations performed by Web Service, including consumption as well as aggregation
  - WSMO separates Choreography and Orchestration

- **Formal Model:**
  - OWL-S formal semantics has been developed in very different frameworks such as Situation Calculus, Petri Nets, Pi-calculus
  - WSMO service interface description model with ASM-based formal semantics
  - OWL-S Process Model is extended by SWRL / FLOWS

*both approaches are not finalized yet*
OWL-S and WSMO

OWL-S Grounding $\approx$ current WSMO Grounding

• OWL-S provides default mapping to WSDL
  – clear separation between WS description and interface implementation
  – other mappings could be used

• WSMO also defines a mapping to WSDL, but aims at an ontology-based grounding
  – avoid loss of ontological descriptions throughout service usage process
  – ‘Triple-Spaced Computing’ as innovative communication technology
Mediation in OWL-S and WSMO

• OWL-S does not have an explicit notion of mediator
  – Mediation is a by-product of the orchestration process
    • E.g. protocol mismatches are resolved by constructing a plan that coordinates the activity of the Web services
    – …or it results from translation axioms that are available to the Web services
      • It is not the mission of OWL-S to generate these axioms
  
• WSMO regards mediators as key conceptual elements
  – Different kinds of mediators:
    • OO Mediators for ensuring semantic interoperability
    • GG, WG mediators to link Goals and Web Services
    • WW Mediators to establish service interoperability
  – Reusable mediators
  – Mediation techniques under development
Semantic Representation

- OWL-S and WSMO adopt a similar view on the need of ontologies and explicit semantics but they rely on different logics:
  - OWL-S is based on OWL / SWRL
    - OWL represent taxonomical knowledge
    - SWRL provides inference rules
    - FLOWS as formal model for process model
  - WSMO is based on WSML a family of languages with a common basis for compatibility and extensions in the direction of Description Logics and Logic Programming
OWL and WSML

- WSML aims at overcoming deficiencies of OWL
- Relation between WSML and OWL+SWRL to be defined
## Summary

<table>
<thead>
<tr>
<th></th>
<th>OWL-S</th>
<th>WSMO</th>
<th>current Web Service technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Discovery</strong></td>
<td>Profile</td>
<td>Goals and Web Services (capability)</td>
<td>UDDI API</td>
</tr>
<tr>
<td>detection of suitable WS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Consumption &amp; Interaction</strong></td>
<td>Process Model</td>
<td>Service Interfaces (Choreography + Orchestration)</td>
<td>BPEL4WS / WS-CDL</td>
</tr>
<tr>
<td>How to consume &amp; aggregate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Invocation</strong></td>
<td>Grounding+ WSDL/SOAP</td>
<td>Grounding (WSDL / SOAP, ontology-based)</td>
<td>WSDL / SOAP</td>
</tr>
<tr>
<td>How to invoke</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Mediation</strong></td>
<td>-</td>
<td>Mediators</td>
<td>-</td>
</tr>
<tr>
<td>Heterogeneity handling</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PART III: A Walkthrough Example

• Virtual Travel Agency Use Case Overview
• Semantic Web Services Modeling
• Discovery
• Conversation Validation
• Mediation
Virtual Travel Agency Use Case

- James is employed in DERI Austria and wants to book a flight and a hotel for the ICWE conference
- the start-up company VTA provides tourism and business travel services based on Semantic Web Service technology

=> how does the interplay of James, VTA, and other Web Services look like?
Domain Ontologies

• All terminology that we use in resource descriptions need to be based on ontologies; also, all information interchanged should be ontology instances

• Modular ontology design
  – an ontology should define terminology of a domain
  – an ontology should be shared & agreed in a community
  – modular ontologies are combined in specific

• Domain Ontologies needed for this Use Case:
  – Trip Reservation Ontology
  – Location Ontology
  – Date and Time Ontology
  – Purchase Ontology
  – … possibly more
Trip Reservation Ontology

• defines the terminology for trips (traveling, accommodation, holiday / business travel facilities) and reservations
• provided by community of interest (e.g. Austrian Tourism Association)

• main concepts:
  – TRIP
    • describes a trip (a journey between locations)
    • passenger, origin & destination, means of travel, etc.
  – RESERVATION
    • describes reservations for tickets, accommodation, or complete trips
    • customer, trip, price, payment
  – RESERVATION REQUEST
  – RESERVATION OFFER
  – RESERVATION CONFIRMATION

• uses other ontologies:
  – Location Ontology for origin & destination specification
  – Date and Time Ontology for departure, arrival, duration information
  – Purchase Ontology for payment related aspects
Trip Reservation Ontology

namespace {"http://www.wsmo.org/ontologies/tripReservationOntology",
dc _"http://purl.org/dc/elements/1.1#",
xsd _"http://www.w3.org/2001/XMLSchema#",
dt _"http://www.wsmo.org/ontologies/dateandtime.wsml#",
po _"http://www.wsmo.org/ontologies/purchase.wsml#"}

ontology _"http://www.wsmo.org/ontologies/tripReservationOntology#"

nonFunctionalProperties
dc#title hasValue "Trip Reservation Ontology"
dc#creator hasValue _"http://www.deri.org"
dc#description hasValue "domain ontology for travel and accommodation reservation"
dc#publisher hasValue "Austrian Tourism Association"
version hasValue "$Revision 1.17 $"
endNonFunctionalProperties

importsOntology  {"http://www.wsmo.org/ontologies/dateandtime.wsml",
                   _"http://www.wsmo.org/ontologies/purchase.wsml"}

usesMediator {"http://www.wsmo.org/mediators/owl2wsml.wsml"}
Trip Reservation Ontology

concept trip
passenger impliesType po#person
origin impliesType loc#location
destination impliesType loc#location
departureDate ofType dt#dateandtime
returnDate ofType dt#dateandtime
meansOfTransport impliesType meansOfTransport
accomodation impliesType accomodation

concept reservation
nonFunctionalProperties
dc#description hasValue "reservations for tickets, accomodation, or complete trips"
dc#relation hasValue reservationItemDef
endNonFunctionalProperties
customer impliesType po#customer
reservationItem impliesType wsml#true
price impliesType po#price
payment impliesType po#payment
axiom reservationItemDef
definedBy
forall {?x, ?y} (?xmemberOf reservation[reservationItem hasValue ?y] impliedBy
(?ymemberOf ticket) or (?ymemberOf accomodation) or (?ymemberOf trip) ).
Ontology Modelling Remarks

• Ontology Design for the Semantic Web
  – “real ontologies, no crappy data models” (Dieter Fensel)
  – (re-)use existing, widely accepted ontologies

• Ontology Design is a very difficult and challenging tasks
  – determine agreed conceptualization of domain
  – correct formalization (e.g. misuse of is_a / part_of relations)
  => requires expertise in knowledge engineering

• Ontology Engineering Methodologies & Technology
  Support essential
  – editing, browsing, maintenance
  – storage and retrieval
  – ontology evolution support
  – ontology integration techniques
Goal Description

- “book flight and hotel for the ICWE 2005 for James”
- goal capability postcondition: get a trip reservation for this

```owl
goal _"http://www.wsmo.org/examples/goals/icwe2005"
importsOntology {_"http://www.wsmo.org/ontologies/tripReservationOntology", …}
capability
  postcondition
    definedBy
      ?tripReservation memberOf tr#reservation[
        customer hasValue fof#james,
        reservationItem hasValue ?tripICWE] and
      ?tripICWE memberOf tr#trip[
        passenger hasValue fof#james,
        origin hasValue loc#innsbruck,
        destination hasValue loc#sydney,
        meansOfTransport hasValue ?flight,
        accomodation hasValue ?hotel] and
      ?flight[airline hasValue tr#staralliance] memberOf tr#flight and
      ?hotel[name hasValue “Capitol Square Hotel”] memberOf tr#hotel .
```
VTA Service Description

• book tickets, hotels, amenities, etc.
• capability description (pre-state)

\[
\text{capability} \text{ VTA}\text{capability}
\]
\[
\text{sharedVariables} \{ ?\text{creditCard}, ?\text{initialBalance}, ?\text{item}, ?\text{passenger} \}
\]
\[
\text{precondition}
\]
\[
\text{definedBy}
\]
\[
?\text{reservationRequest}[\ 
\text{reservationItem hasValue} \ ?\text{item}, \ 
\text{passenger hasValue} \ ?\text{passenger}, \ 
\text{payment hasValue} \ ?\text{creditcard}, \ 
] \text{memberOf} \ tr#\text{reservationRequest} \ \text{and} \ 
((?\text{item memberOf} \ tr#\text{trip}) \ \text{or} \ ((?\text{item memberOf} \ tr#\text{ticket})) \ \text{and} \ 
?\text{creditCard}[\text{balance hasValue} \ ?\text{initialBalance}] \text{memberOf} \ po#\text{creditCard} .
\]
\[
\text{assumption}
\]
\[
\text{definedBy}
\]
\[
po\#\text{validCreditCard}(?\text{creditCard}) \ \text{and} \ 
(?\text{creditCard[type hasValue} \ po\#\text{visa}] \ \text{or} \ ?\text{creditCard[type hasValue} \ po\#\text{mastercard}]).
\]
VTA Service Description

- capability description (post-state)

**postcondition**

*definedBy*

- ?reservation[
  - reservationItem *hasValue* ?item,
  - customer *hasValue* ?passenger,
  - payment *hasValue* ?creditcard
]
  *memberOf* tr#reservation .

**assumption**

*definedBy*

- reservationPrice(?reservation, "euro", ?tripPrice) and
- ?finalBalance= (?initialBalance - ?ticketPrice) and
Web Service Discovery

James has Objective: "book a flight and a hotel for me for the ICWE 2005."

Goal definition

Service Registry searches WS Discoverer result set includes VTA
Semantic Web Service Discovery

find appropriate Web Service for automatically resolving a goal as the objective of a requester

• Aims:
  – high precision discovery
  – maximal automation
  – effective discoverer architectures

• Requirements:
  – infrastructure that allows storage and retrieval of information about Web services
  – description of Web services functionality
  – description of requests or goals
  – algorithms for matching requesters for capabilities with the corresponding providers
Discovery Techniques

• different techniques available
  – trade-off: ease-of-provision <-> accuracy
  – resource descriptions & matchmaking algorithms

Key Word Matching
match natural language key words in resource descriptions

Controlled Vocabulary
ontology-based key word matching

Semantic Matchmaking
… what Semantic Web Services aim at
Semantic Web Services in UDDI

- Mapping semantic resource descriptions into UDDI
- OWL-S Service Profile mapping to UDDI
- WSMO elements to UDDI mapping (for all top level elements)

⇒ mapping semantic descriptions to syntactic repository

⇒ allows retrieval of structural information
Controlled Vocabulary

WSMO non-functional properties

• Ontology keywords in non-functional properties
  – dc#subject contains main ontology concepts related to Web Service
  – allows pre-filtering similar to OWL-S Profile Hierarchy, but on basis on ontologies (“controlled vocabulary”)

• Example
  – a Web Service for selling train tickets in Austria
    dc#subject hasValue _{tc#trainticket, po#purchase, loc#austria}
  – does not precisely describe Web Service functionality
    => accuracy of discovery result meager

Controlled Vocabulary

OWL-S Profile Hierarchies

- hierarchy of Web Services
  - functional similarities (domain, in- / outputs)
  - allows pre-filtering of services on basis of categorization

http://www.daml.org/services/owl-s/1.0/ProfileHierarchy.owl
Matchmaking Notions & Intentions

Exact Match:
\[ G, WS, O, M \models \forall x. (G(x) \iff WS(x)) \]

PlugIn Match:
\[ G, WS, O, M \models \forall x. (G(x) \Rightarrow WS(x)) \]

Subsumption Match:
\[ G, WS, O, M \models \forall x. (G(x) \subseteq WS(x)) \]

Intersection Match:
\[ G, WS, O, M \models \exists x. (G(x) \land WS(x)) \]

Non Match:
\[ G, WS, O, M \not\models \exists x. (G(x) \land WS(x)) \]

Discovery Approach

- Matchmaking Notion to be used defined for each goal capability element
- Basic Procedure:

```
 valid pre-state?  
 no    yes
 abort     

 valid post-state?  
 no    yes
 abort     
```

**Goal Capability**

- Precondition
- Assumption
- Postcondition
- Effect

**Web Service Capability**

- Precondition
- Assumption
- Postcondition
- Effect

- **Plug-In**: Exact
- **Intersection**: Exact
- **Match**: Match

- abort
- yes
- no

- abort
- yes
- no
Discoverer Architecture

- Discovery as central Semantic Web Services technology
- Integrated Discoverer Architectures (under construction):

  1. **Resource Repository** (UDDI or other)
  2. **Keyword-/ Classification-based Filtering**
  3. **Controlled Vocabulary Filtering**
  4. **Semantic Matchmaking**

  - Retrieve Service Descriptions
  - Efficient narrowing of search space (relevant services to be inspected)

- Usable Web Service
- Invoke Web Service
Service Interfaces

**Goal**
- **Requested Capability**: book flight & hotel
- **Requested Interface**
  1) send request
  2) select from offer
  3) receive confirmation

**VTA**
- **VTA WS**: Trip Booking
  - **Interface (Chor.)**
    1) get request
    2) provide offer
    3) receive selection
    4) send confirmation

**Flight WS**
- **Capability**
  - Interface (Chor.)
    1) get request
    2) provide offer
    3) receive selection
    4) send confirmation

**Hotel WS**
- **Capability**
  - Interface (Chor.)
    1) get request
    2) provide offer
    3) receive selection
    4) send confirmation

**Choreography Interface**: how entity can interact
**Choreography**: interaction between entities
**Orchestration**: service aggregation for realizing functionality
VTA Service Description

- Choreography Interface
- transition “get request” to “provide offer”

```plaintext
choreography VTABehaviorInterface
importsOntology {"http://www.wsmo.org/ontologies/tripReservationOntology"} vocabularyIn {reservationRequest, …}
vocabularyOut {reservationOffer, …}
guardedTransitions VTABehaviorInterfaceTransitionRules
if (reservationRequest memberOf tr#reservationRequest[
    customer hasValue ?Customer,
    reservationItem hasValue ?Trip] and
    ?Trip memberOf tr#trip[
    passenger hasValue ?Passenger,
    origin hasValue ?LocationInAustria] and
    ?LocationInAustria memberOf loc#location[
    inCountry hasValue loc#austria]
then reservationOffer memberOf tr#reservationOffer[
    customer hasValue ?Customer,
    reservationItem hasValue ?Trip ] .
```
Choreography Discovery

- **Goal**
- **Requested Capability**
  - book flight & hotel
- **Requested Interface**
  - 1) send request
  - 2) select from offer
  - 3) receive confirmation

- **VTA**
  - **Capability**
  - Interface (Chor.)
    - 1) get request
    - 2) provide offer
    - 3) receive selection
    - 4) send confirmation
  - Interface (Orch.)
    - 1) flight request
    - 2) hotel request
    - 3) book flight
    - 4) book hotel

- **VTA WS**
  - ‘Trip Booking’
  - **Capability**
    - Interface (Chor.)
      - 1) get request
      - 2) provide offer
      - 3) receive selection
      - 4) send confirmation
    - Interface (Orch.)
      - 1) flight request
      - 2) hotel request
      - 3) book flight
      - 4) book hotel

- **Flight WS**
  - **Capability**
    - Interface (Chor.)
      - 1) get request
      - 2) provide offer
      - 3) receive selection
      - 4) send confirmation
    - Interface (Orch.)
      - 1) flight request
      - 2) hotel request
      - 3) book flight
      - 4) book hotel

- **Hotel WS**
  - **Capability**
    - Interface (Chor.)
      - 1) get request
      - 2) provide offer
      - 3) receive selection
      - 4) send confirmation
    - Interface (Orch.)
      - 1) flight request
      - 2) hotel request
      - 3) book flight
      - 4) book hotel

- Both choreography interfaces given ("static")
- Correct & complete consumption of VTA
  => existence of a valid choreography?

- VTA Orchestration & Chor. Interfaces of aggregated WS given
  => existence of a valid choreography between VTA and each aggregated WS?

---

**Choreography Discovery** as a central reasoning task in Service Interfaces
- ‘choreographies’ do not have to be described, only existence determination
WSMO Service Interface Description Model

- common formal model for Service Interface description
  - ontologies as data model
  - based on ASMs
  - not restricted to any executable communication technology

- general structure:
  - Vocabulary \( \Omega \):
    - ontology schema(s) used in service interface description
    - usage for information interchange: in, out, shared, controlled
  - States \( \omega(\Omega) \):
    - a stable status in the information space
    - defined by attribute values of ontology instances
  - Guarded Transition \( \text{GT}(\omega) \):
    - state transition
    - general structure: \textbf{if} (condition) \textbf{then} (action)
    - different for Choreography and Orchestration
    - additional constructs: \textbf{add}, \textbf{delete}, \textbf{update}
Service Interface Example

Choreography Interface of a Web Service

Vocabulary:
- Concept A in $\Omega_{\text{in}}$
- Concept B in $\Omega_{\text{out}}$

State $\omega_1$

received ontology instance $a$

$\Omega_{\text{in}}$ hasValues
concept A [att1 ofType X, att2 ofType Y]
...

$\Omega_{\text{out}}$ hasValues
concept B [att1 ofType W, att2 ofType Z]
...

Guarded Transition GT($\omega_1$)

IF (a memberOf A [att1 hasValue x])
THEN
(b memberOf B [att2 hasValue m])

State $\omega_2$

sent ontology instance $b$

a memberOf A [att1 hasValue x, att2 hasValue y]

b memberOf B [att2 hasValue m]

defined evolving ontology instance store
Choreography Discovery

• a valid choreography exists if:
  1) **Information Compatibility**
     • compatible vocabulary
     • homogeneous ontologies
  2) **Communication Compatibility**
     • start state for interaction
     • a termination state can be reached without any additional input
Information Compatibility

If choreography participants have compatible vocabulary definitions:

– $\Omega_{in}(S1)$ and $\Omega_{shared}(S1) = \Omega_{out}(S2)$ and $\Omega_{shared}(S2)$
– determinable by Intersection Match from Discovery

$S_{IS1}, S_{IS2}, O, M \models \exists x. (\Omega_{S1(in \ U \ shared)}(x) \land \Omega_{S2(out \ U \ shared)}(x))$
– more complex for multi-party choreographies

Prerequisite: choreography participants use homogeneous ontologies:

– semanticInteroperability($S1, S2, \ldots, Sn$)
– usage of same ontologies in Service Interfaces or respective OO Mediators
Communication Compatibility

- Definitions (for “binary choreography” (only 2 services), more complex for multi-party choreographies)

Valid Choreography State:
\[ \omega_x(C(S1, S2)) \text{ if } \text{informationCompatibility}(\Omega S1(\omega_x), \Omega S2(\omega_x)) \]
- means: action in GT of S1 for reaching state \( \omega_x(S1) \) satisfies condition in GT of S2 for reaching state \( \omega_x(S2) \), or vice versa

Start State:
\[ \omega_{\emptyset}(C(S1, S2)) \text{ if } \Omega S1(\omega_{\emptyset})=\emptyset \text{ and } \Omega S2(\omega_{\emptyset})=\emptyset \text{ and } \exists \omega_1(C(S1, S2)) \]
- means: if initial states for choreography participants given (empty ontology, i.e. no information interchange has happened), and there is a valid choreography state for commencing the interaction

Termination State:
\[ \omega_T(C(S1, S2)) \text{ if } \Omega S1(\omega_T)=\text{noAction} \text{ and } \Omega S2(\omega_T)=\text{noAction} \text{ and } \exists \omega_T(C(S1, S2)) \]
- means: there exist termination states for choreography participants (no action for transition to next state), and this is reachable by a sequence of valid choreography states

- Communication Compatibility given if there exists a start state and a termination state is reachable without additional input by a sequence of valid choreography states
Communication Compatibility Example

James’ Goal Behavior Interface

\[ \Omega_{S1}(\omega_\emptyset) = \{\emptyset\} \]
\[ \text{if } \emptyset \text{ then request} \]
\[ \Omega_{S1}(\omega_1) = \{\text{request(out)}\} \]
\[ \text{if cnd1(offer) then changeReq} \]
\[ \Omega_{S1}(\omega_2a) = \{\text{offer(in)}, \text{changeReq(out)}\} \]
\[ \text{if cnd2(offer) then order} \]
\[ \Omega_{S1}(\omega_2b) = \{\text{offer(in)}, \text{order(out)}\} \]
\[ \text{if conf then } \emptyset \]
\[ \Omega_{S1}(\omega_3) = \{\text{offer(in)}, \text{conf(in)}\} \]

VTA Behavior Interface

\[ \Omega_{S2}(\omega_\emptyset) = \{\emptyset\} \]
\[ \text{if request then offer} \]
\[ \Omega_{S2}(\omega_1) = \{\text{request(in)}, \text{offer(out)}\} \]
\[ \text{if changeReq then offer} \]
\[ \Omega_{S2}(\omega_2a) = \{\text{changeReq(in)}, \text{offer(out)}\} \]
\[ \text{if order then conf} \]
\[ \Omega_{S2}(\omega_2b) = \{\text{order(in)}, \text{conf(out)}\} \]

existence of a valid Choreography
Orchestration

Interaction with aggregated Web Services + Control Structure

- formally described service functionality decomposition
- only those aspects of WS realization wherefore other WS are aggregated
- aggregated WS used via their behavior interface
Orchestration Description & Validation

• Orchestration Description:
  – interaction behavior of “Orchestrator” with “orchestrated Web Services”
  – formal description as Choreography, extended Guarded Transitions
  – Orchestration Guarded Transitions general structure:
    \[
    \text{if condition then operation} \quad \text{Operation} = (\text{Orchestrator, Web Service, Action})
    \]
  – Orchestrator serves as client for aggregated Web Services

• Orchestration Validation:
  – need to ensure that interactions with aggregated Web Service can be executed successfully
  => Choreography Discovery for all interaction of Orchestrator with each aggregated Web Service
Orchestration Validation Example

**VTA Web Service Orchestration**

- **if Ø then** (FWS, flightRequest)
- **if flightOffer**
  - **then** (HWS, hotelRequest)
- **if selection**
  - **then** (FWS, flightBookingOrder)
- **if selection, flightBookingConf**
  - **then** (HWS, hotelBookingOrder)

**Flight WS Behavior Interface**

- **Start** (VTA, FWS)
- **if request then offer**
- **if order then confirmation**

**Termination** (VTA, FWS)

**Hotel WS Behavior Interface**

- **Start** (VTA, HWS)
- **if request then offer**
- **if order then confirmation**

**Termination** (VTA, HWS)

Orchestration is valid if valid choreography exists for interactions between Orchestrator and each aggregated Web Service, done by choreography discovery.
Composition and Orchestration

Composition Synthesis

Complex Goal

client

composite service invocation

Composition

Orchestration

specification of the process of the composite service

additional requirements for orchestration

service descriptions

service 1

service n

non-functional features

service descriptions

functional features

non-functional features

service descriptions

functional features

non-functional features

5th Internation Conference on Web Engineering (ICWE 2005), Sydney, Australia, July 2005
Service Composition and Orchestration

- Web Service Composition:
  - the realization of a Web Service by dynamically composing the functionalities of other Web Services
    - The new service is the **composite service**
    - The invoked services are the **component services**
  - a composite service can provide the skeleton for a Web Service (e.g. the VTA Web Service)

- Current Composition techniques only partially cover aspects for valid orchestrations:
  - functional Web Service composition (on capability descriptions)
  - dynamic control and data flow construction for composite Web Service
  - delegation of client / goal behavior to component services

=> Orchestration Validation needed to ensure executability of Web Service aggregations
Mediation

• Heterogeneity as inherent characteristic of (Semantic) Web:
  – heterogeneous terminology
  – heterogeneous languages / formalisms
  – heterogeneous communication protocols and business processes

• WSMO identifies Mediators as top level element, i.e. central aspect of Semantic Web Services
  – levels of mediation: data, protocol, processes
  – WSMO Mediator types

• Approach: declarative, generic mismatch resolution
  – classification of possible & resolvable mismatches
  – mediation definition language & mediation patterns
  – execution environment for mappings
Data Level (OO) Mediation

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

• Data Level Mismatch Classification
  – Conceptualization Mismatches
    • same domain concepts, but different conceptualization
    • different levels of abstraction
    • different ontological structure
    => resolution only incl. human intervention
  – Explication Mismatches
    • mismatches between:
      T (Term used) D (definition of concepts), C (real world concept)
    => automated resolution partially possible
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)
  – mapping between Ontology Constructs:
    • classMapping, attributeMapping, relationMapping (between similar constructs)
    • classAttributeMapping, classRelationMapping, classInstanceMapping
    • instanceMapping (explicit ontology instance transformation)
  – Conditions / logical expressions for data type mismatch handling, restriction of mapping validity, and complex mapping definitions
  – 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

1234 `memberOf` Person
  - name = James
  - age = 22

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

this allows to transform the instance 1234 of ontology O2 into a valid instance of ‘adult’ in ontology O1
Protocol & Process Level Mediation

- if a choreography does not exist, then find an appropriate WW Mediator that
  - resolves possible mismatches to establish Information Compatibility (OO Mediator usage)
  - resolves process / protocol level mismatches in to establish Communication Compatibility
Process Mediation – Addressed Mismatches

a) 

b) 

c) 

d) 

e)
Unsolvable Mismatches

Business Partner1  PM  A  Business Partner2

Business Partner1  B  A  PM  A  B  Business Partner2

Business Partner1  A  Ack  PM  ?  Business Partner2
Process Mediation Example

REQUEST

itinerary[origin, destination, date]

<table>
<thead>
<tr>
<th>origin</th>
</tr>
</thead>
</table>

| destination |

| date |

| time |

| price |

itinerary[origin, destination]

SERVICES

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

REQUEST

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

Processes Mediator

SERVICE

| origin |
| destination |
| itinerary[origin, destination] |
| date |
| itinerary [route, date, time, price] |
Process Mediation Example

REQUEST

itinerary[origin, destination, date] → time

Processes Mediator

origin

destination

S
E
R
V
I
C
E

itinerary[origin, destination]

date

itinerary [route, date, time, price] → price
Process Mediation Example

REQUEST

origin

destination

itinerary [origin, destination, date]

Processes Mediator

date

itinerary [route, date, time, price]

SERVICE

time

price

itinerary [origin, destination]
Process Mediation Example

REQUEST

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

Processes Mediator

| origin |
| destination |
| itinerary[origin, destination] |
| date |
| itinerary [route, date, time, price] |

SERVICE
Conclusions

• Semantic Web Service descriptions require
  – expertise in ontology & logical modeling

  => tool support for users & developers under development
  – understanding of Semantic Web Service technologies
    • what it does, and how it works
    • which are the related descriptive information

• Semantic Web Service technologies aim at automation of the Web Service usage process
  – users only define goal with tool support
  – ‘intelligent’ SWS middleware for automated Web Service usage

• state of the art in technology & tool development
  – theoretical approaches are evolving & converging
  – prototypical SWS technologies existent
  – industrial strength SWS technology suites aspired in upcoming efforts
PART IV:
The Web Service Execution Environment (WSMX)

• Introduction, background and motivation
• Structural architecture
• Dynamic behaviour
• Future plans
• Demos
Introduction, Background and Motivation
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
• SO 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

Three Tier Architecture

User (Service Requester)
Client / Browser
WSMX Client

Retailer
Web Server / Application Server
WSMX Server

Manufacturer
WSMX
Back-End Application
Adapter / WSDL Interface
Back-End Application

Two Tier Architecture

B2C

B2B
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
WSMX Usage Scenario - P2P
WSMX Usage Scenario - P2P
Development Process & 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 0.1.6)
  - July 2005 (WSMX 0.2.0)
  - November 2005 (WSMX 0.3.0)
  - November 2004 (WSMX 0.1.5)
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 Architecture

WSMX Manager

WSMX Manager Core

Resource Manager Interface

WSMO Objects datastore

Non-WSMO Object datastore

Reasoner Interface

Reasoner

Reasoner

Reasoner

Service Oriented Architectures

WSMT – Web Services Modelling Toolkit

WSML Editor

Choreography Editor

Administration Framework

WSMX Monitoring

WSMX Management

Data and Communication Protocols Adaptors

Service Requesters

Service Providers

Web Service 1

Web Service 2

Web Service p

Agent 1 acting on behalf of user a

Agent 2 acting on behalf of user b

Agent 3 acting on behalf of user m

CM Wrapper

RM Wrapper

Parser Wrapper

Discovery Wrapper

Selector Wrapper

DM Wrapper

PM Wrapper

Choreography Wrapper

Orchestration Wrapper

Resource Manager Interface

Invoker

Receiver

Grounding

WSMO Objects Datastore

Non-WSMO Object Datastore

Reasoner

Flora/KSB

WSMO Reasoner
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
Service Oriented State

• The interface to the service is implementation-independent
• The service can be dynamically invoked
  – Runtime binding
• The service is self-contained
  – Maintains its own state
Messaging

• Messaging is peer-to-peer facility
• Distributed communication
  – Loosely coupled
• Sender does not need to know receiver (and vice versa)
• Asynchronous mechanism to communicate between software applications
Components & System Architecture

WSMX – Web Services Modelling Toolkit

WSMX Management  WSMX Monitor  WSML Editor  Choreography Editor  Mediator Editor

WSMX Manager

WSMX Manager Core

CM Wrapper  RM Wrapper  Parser Wrapper  Discovery Wrapper  Selector Wrapper  DM Wrapper  PM Wrapper  Choreography Wrapper

Interface  Interface  Interface  Interface  Interface  Interface  Interface  Interface

Communication Manager

Invoker  Receiver

Grounding

Resource Manager Interface

WSMO Objects  Non WSMO

Reasoner Interface

Reasoner

Component Wrapper

Interface

New Component

Service Requesters

Back-End Application

Agent acting on behalf of service requester

Data and Communication Protocols Adapters

Service Providers

Web Service 1

Web Service 2

Web Service p

5th Internation Conference on Web Engineering (ICWE 2005), Sydney, Australia, July 2005
Selected Components

- Adapters
- Parser
- Invoker
- Choreography
- Process Mediator
- Discovery
- Data Mediator
- Resource Manager
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 Mgr – 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

• A choreography instance is loaded for each
  – Both requester and provider have their own WSMO descriptions

• The Choreography component examines a service’s choreography to determine next step in communication

• The Choreography component raises events for the Invoker to make actual service invocations
Process Mediator

- Requester and provider have their own communication patterns
- Only if the two match precisely, a direct communication may take place
- At design time equivalences between the choreographies’ conceptual descriptions is determined and stored as set of rules
- The Process Mediator provides the means for runtime analyses of two choreography instances and uses mediators to compensate possible mismatches
Process Mediator

![Diagram of process mediation]
Discovery

• Responsible for finding appropriate Web Services to achieve a goal (discovery)
• Current discovery component is based on simple matching
• Advanced semantic discovery in prototypical stage
Discovery

Keyword-based with Natural Language Processing (NLP)

Coarse grained Service and Goal descriptions

Fine grained Service and Goal descriptions

{Keyword}

WS

Level of Abstraction

Syntactic

Semantic ("Light")

Semantic ("Heavy")
Discovery

Keyword-based with Natural Language Processing (NLP)

{Keyword}

W1 ... WL

Syntactic

Coarse grained Service and Goal descriptions

Fine grained Service and Goal descriptions

Level of Abstraction

Semantic ("Light")

Semantic ("Heavy")
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
Dynamic Behaviour
System entry points

- `storeEntity(WSMOEntity):Confirmation`
  - provides an administration interface for storing any WSMO-related entities (Web Services, Goals, Ontologies)

- `realizeGoal(Goal, OntologyInstance):Confirmation`
  - service requester expects WSMX to discover and invoke Web Service without exchanging additional messages

- `receiveGoal(Goal, OntologyInstance, Preferences):WebService[]`
  - list of Web Services is created for given Goal
  - requester can specify the number of Web Services to be returned

- `receiveMessage(OntologyInstance,WebServiceID, ChoreographyID):ChoreographyID`
  - back-and-forth conversation to provide all necessary data for invocation
  - involves execution of choreographies and process mediation between service interfaces
Define “Business” Process

1. Discover Web Services
   - Start
   - Create Choreography
     - Created

2. Discover Services
   - Mediate Data
     - Mediate Data
     - Return Mediated Data
       - Return Mediated Data
         - Return Web Services
           - Check Choreography
             - Confirmed
               - Confirmed
                 - End

               - Call Invoker
                 - Confirmed
Generate Wrappers for Components

1. Discover Web Services → Start
   - Create Choreography
   - Created

2. Discover Services
   - Mediate Data
   - Return Mediated Data

3. Return Mediated Data
   - Mediate Data
   - Data Mediator Wrapper
   - Check Choreography
   - Confirmed

4. Call Invoker
   - Confirmed
   - End

Registry of known components

Discovery Wrapper

Communication Manager Wrapper

Choreography Wrapper
Context Data
Event-based Implementation

Core – Manager

“Business” Process – Internal Workflow

Event and Notification Distribution/Delivery Mechanism

Choreography Wrapper

Discovery Wrapper implements Mediator Interface

Data Mediator Wrapper

Communication Manager Wrapper

Choreography

Discovery

Mediator

Communication Manager
Walk Through
Execution Semantics

Request to discover Web services.
Goal expressed in WSML is sent to WSMX System Interface

WSMX Manager

WSMX Manager Core

CM Wrapper
RM Wrapper
Parser Wrapper
Discovery Wrapper
Selector Wrapper
DM Wrapper
PM Wrapper
Choreography Wrapper

Interface
Resource Manager
Parser
Discovery
Selector
Data Mediator
Process Mediator
Choreography

WSMO Objects
Non WSMO
Reasoner

Component Wrapper
Interface
New Component
Execution Semantics

Com. M. implements the interface to receive WSML goals
Execution Semantics

WSMT – Web Services Modelling Toolkit

WSMX Managment  WSMX Monitor  WSML Editor  Choreography Editor  Mediator Editor

WSMX

WSMX Manager

WSMX Manager Core

CM Wrapper
RM Wrapper
Parser Wrapper
Discovery Wrapper
Selector Wrapper
DM Wrapper
PM Wrapper
Choreography Wrapper

WSMX Management

Service Requesters

Back-End Application

Agent acting on behalf of service requester

Data and Communication Protocols Adapters

Adapter 1
Adapter 2
Adapter n

Service Providers

Web Service 1
Web Service 2
...
Web Service n

Execution Semantics

Com. M. informs Core that Goal has been received

Resource Manager Interface

WSMO Objects
Non WSMO

Reasoner Interface

Reasoner

Component Wrapper

Interface

New Component

Reasoner Interface

WSMT – Web Services Modelling Toolkit

WSMX Managment  WSMX Monitor  WSML Editor  Choreography Editor  Mediator Editor

WSMX

WSMX Manager

WSMX Manager Core

CM Wrapper
RM Wrapper
Parser Wrapper
Discovery Wrapper
Selector Wrapper
DM Wrapper
PM Wrapper
Choreography Wrapper

WSMX Management

Service Requesters

Back-End Application

Agent acting on behalf of service requester

Data and Communication Protocols Adapters

Adapter 1
Adapter 2
Adapter n

Service Providers

Web Service 1
Web Service 2
...
Web Service n

Execution Semantics

Com. M. informs Core that Goal has been received

Resource Manager Interface

WSMO Objects
Non WSMO

Reasoner Interface

Reasoner

Component Wrapper

Interface

New Component

Reasoner Interface

WSMT – Web Services Modelling Toolkit

WSMX Managment  WSMX Monitor  WSML Editor  Choreography Editor  Mediator Editor

WSMX

WSMX Manager

WSMX Manager Core

CM Wrapper
RM Wrapper
Parser Wrapper
Discovery Wrapper
Selector Wrapper
DM Wrapper
PM Wrapper
Choreography Wrapper

WSMX Management

Service Requesters

Back-End Application

Agent acting on behalf of service requester

Data and Communication Protocols Adapters

Adapter 1
Adapter 2
Adapter n

Service Providers

Web Service 1
Web Service 2
...
Web Service n

Execution Semantics

Com. M. informs Core that Goal has been received

Resource Manager Interface

WSMO Objects
Non WSMO

Reasoner Interface

Reasoner

Component Wrapper

Interface

New Component

Reasoner Interface

WSMT – Web Services Modelling Toolkit

WSMX Managment  WSMX Monitor  WSML Editor  Choreography Editor  Mediator Editor

WSMX

WSMX Manager

WSMX Manager Core

CM Wrapper
RM Wrapper
Parser Wrapper
Discovery Wrapper
Selector Wrapper
DM Wrapper
PM Wrapper
Choreography Wrapper

WSMX Management

Service Requesters

Back-End Application

Agent acting on behalf of service requester

Data and Communication Protocols Adapters

Adapter 1
Adapter 2
Adapter n

Service Providers

Web Service 1
Web Service 2
...
Web Service n

Execution Semantics

Com. M. informs Core that Goal has been received

Resource Manager Interface

WSMO Objects
Non WSMO

Reasoner Interface

Reasoner

Component Wrapper

Interface

New Component

Reasoner Interface

WSMT – Web Services Modelling Toolkit

WSMX Managment  WSMX Monitor  WSML Editor  Choreography Editor  Mediator Editor

WSMX

WSMX Manager

WSMX Manager Core

CM Wrapper
RM Wrapper
Parser Wrapper
Discovery Wrapper
Selector Wrapper
DM Wrapper
PM Wrapper
Choreography Wrapper

WSMX Management

Service Requesters

Back-End Application

Agent acting on behalf of service requester

Data and Communication Protocols Adapters

Adapter 1
Adapter 2
Adapter n

Service Providers

Web Service 1
Web Service 2
...
Web Service n

Execution Semantics

Com. M. informs Core that Goal has been received

Resource Manager Interface

WSMO Objects
Non WSMO

Reasoner Interface

Reasoner

Component Wrapper

Interface

New Component

Reasoner Interface

WSMT – Web Services Modelling Toolkit

WSMX Managment  WSMX Monitor  WSML Editor  Choreography Editor  Mediator Editor

WSMX

WSMX Manager

WSMX Manager Core

CM Wrapper
RM Wrapper
Parser Wrapper
Discovery Wrapper
Selector Wrapper
DM Wrapper
PM Wrapper
Choreography Wrapper

WSMX Management

Service Requesters

Back-End Application

Agent acting on behalf of service requester

Data and Communication Protocols Adapters

Adapter 1
Adapter 2
Adapter n

Service Providers

Web Service 1
Web Service 2
...
Web Service n

Execution Semantics

Com. M. informs Core that Goal has been received

Resource Manager Interface

WSMO Objects
Non WSMO

Reasoner Interface

Reasoner

Component Wrapper

Interface

New Component

Reasoner Interface

WSMT – Web Services Modelling Toolkit

WSMX Managment  WSMX Monitor  WSML Editor  Choreography Editor  Mediator Editor

WSMX

WSMX Manager

WSMX Manager Core

CM Wrapper
RM Wrapper
Parser Wrapper
Discovery Wrapper
Selector Wrapper
DM Wrapper
PM Wrapper
Choreography Wrapper

WSMX Management

Service Requesters

Back-End Application

Agent acting on behalf of service requester

Data and Communication Protocols Adapters

Adapter 1
Adapter 2
Adapter n

Service Providers

Web Service 1
Web Service 2
...
Web Service n

Execution Semantics

Com. M. informs Core that Goal has been received

Resource Manager Interface

WSMO Objects
Non WSMO

Reasoner Interface

Reasoner

Component Wrapper

Interface

New Component

Reasoner Interface

WSMT – Web Services Modelling Toolkit

WSMX Managment  WSMX Monitor  WSML Editor  Choreography Editor  Mediator Editor

WSMX

WSMX Manager

WSMX Manager Core

CM Wrapper
RM Wrapper
Parser Wrapper
Discovery Wrapper
Selector Wrapper
DM Wrapper
PM Wrapper
Choreography Wrapper

WSMX Management

Service Requesters

Back-End Application

Agent acting on behalf of service requester

Data and Communication Protocols Adapters

Adapter 1
Adapter 2
Adapter n

Service Providers

Web Service 1
Web Service 2
...
Web Service n

Execution Semantics

Com. M. informs Core that Goal has been received

Resource Manager Interface

WSMO Objects
Non WSMO

Reasoner Interface

Reasoner

Component Wrapper

Interface

New Component

Reasoner Interface

WSMT – Web Services Modelling Toolkit

WSMX Managment  WSMX Monitor  WSML Editor  Choreography Editor  Mediator Editor

WSMX

WSMX Manager

WSMX Manager Core

CM Wrapper
RM Wrapper
Parser Wrapper
Discovery Wrapper
Selector Wrapper
DM Wrapper
PM Wrapper
Choreography Wrapper

WSMX Management

Service Requesters

Back-End Application

Agent acting on behalf of service requester

Data and Communication Protocols Adapters

Adapter 1
Adapter 2
Adapter n

Service Providers

Web Service 1
Web Service 2
...
Web Service n

Execution Semantics

Com. M. informs Core that Goal has been received

Resource Manager Interface

WSMO Objects
Non WSMO

Reasoner Interface

Reasoner

Component Wrapper

Interface

New Component

Reasoner Interface

WSMT – Web Services Modelling Toolkit

WSMX Managment  WSMX Monitor  WSML Editor  Choreography Editor  Mediator Editor

WSMX

WSMX Manager

WSMX Manager Core

CM Wrapper
RM Wrapper
Parser Wrapper
Discovery Wrapper
Selector Wrapper
DM Wrapper
PM Wrapper
Choreography Wrapper

WSMX Management

Service Requesters

Back-End Application

Agent acting on behalf of service requester

Data and Communication Protocols Adapters

Adapter 1
Adapter 2
Adapter n

Service Providers

Web Service 1
Web Service 2
...
Web Service n

Execution Semantics

Com. M. informs Core that Goal has been received

Resource Manager Interface

WSMO Objects
Non WSMO

Reasoner Interface

Reasoner

Component Wrapper

Interface

New Component

Reasoner Interface
Execution Semantics
Execution Semantics
Execution Semantics

WSMX – Web Services Modelling Toolkit

WSMX Managment | WSMX Monitor | WSML Editor | Choreography Editor | Mediator Editor

WSMX Manager

WSMX Manager Core

Core is notified that choreography instance has been created.

Resource Manager Interface

WSMO Objects | Non WSMO

Reasoner Interface

Reasoner

Component Wrapper

Interface

New Component

CM Wrapper

RM Wrapper

Parser Wrapper

Discovery Wrapper

Selector Wrapper

DM

PM

Choreography Wrapper

CM Wrapper

RM Wrapper

Parser Wrapper

Discovery Wrapper

Selector Wrapper

Invoker

Receiver

Grounding

Resource Manager

Parser

Discovery

Selector

Mediator

Choreography

Component

Interface

New Component

WSML Editor

WSMX Monitor

Choreography Editor

Mediator Editor

Service Requesters

Service Providers

Back-End Application

Agent acting on behalf of service requester

Data and Communication Protocols Adapters

Adapter 1

Adapter 2

Adapter n...

Web Service 1

Web Service 2

Web Service p

5th Internation Conference on Web Engineering (ICWE 2005), Sydney, Australia, July 2005
Execution Semantics

WSMX – Web Services Modelling Toolkit

WSMX Management | WSMX Monitor | WSML Editor | Choreography Editor | Mediator Editor

WSMX Manager

WSMX Manager Core

WSM Manager Core

Resource Manager Interface

WSMO Objects
Non WSMO

Reasoner Interface

Reasoner

Component Wrapper

New Component

WSML goal is parsed to internal format.
Execution Semantics

WSMT – Web Services Modelling Toolkit

WSMX Management
WSMX Monitor
WSML Editor
Choreography Editor
Mediation Editor

WSMX

WSMX Manager

WSMX Manager Core

Administration Framework Interface

CM Wrapper
RM Wrapper
Parser Wrapper
Discovery Wrapper
Selector Wrapper
DM Wrapper
PM Wrapper
Choreography Wrapper

CM Wrapper
RM Wrapper
Parser Wrapper
Discovery Wrapper
Selector Wrapper
DM Wrapper
PM Wrapper
Choreography Wrapper

Discovery is invoked for parsed goal.

Resource Manager Interface

WSMO Objects
Non WSMO

Reasoner Interface

Reasoner

Component Wrapper

Interface

New Component

Service Requesters

Data and Communication Protocols Adapters

Service Providers

Back-End Application
Agent acting on behalf of service requester

Agent acting on behalf of service requester

Adapters

Service Requesters

Web Service 1
Web Service 2
Web Service p

Execution Semantics

5th Internation Conference on Web Engineering (ICWE 2005), Sydney, Australia, July 2005
Execution Semantics

WSMT – Web Services Modelling Toolkit
- WSMX Managment
- WSMX Monitor
- WSML Editor
- Choreography Editor
- Mediator Editor

WSMX Manager
WSMX Manager Core
- CM Wrapper
- RM Wrapper
- Parser Wrapper
- Discovery Wrapper
- Selector Wrapper
- DM Wrapper
- PM Wrapper
- Choreography Wrapper

Administration Framework Interface

Data and Communication Protocols Adapters
- Adapter 1
- Adapter 2
- Adapter n...

Interface
- Invoker
- Receiver
- Grounding

Communication Manager
- Resource Manager

Interface
- Interface

Parser

Discovery

Resource Manager Interface
- WSMO Objects
- Non WSMO

Reasoner Interface
- Reasoner

Component Wrapper
- Interface
- New Component

Discovery may requires ontology mediation.

WSMO – Web Services Modelling Toolkit

Service Requesters
- Service Requesters
- Back-End Application
- Agent acting on behalf of service requester

WSML Editor
- WSML Monitor
- Choreography Editor

WSMT – Web Services Modelling Toolkit
- Web Service 1
- Web Service 2
- ...
- Web Service p

Service Providers

Web Service 1

Web Service 2

Web Service p

5th Internation Conference on Web Engineering (ICWE 2005), Sydney, Australia, July 2005
Execution Semantics

WSMX – Web Services Modelling Toolkit

WSMX Management | WSMX Monitor | WSML Editor | Choreography Editor | Mediator Editor

WSMX Manager

WSMX Manager Core

CM Wrapper | RM Wrapper | Parser Wrapper | Discovery Wrapper | Selector Wrapper | DM Wrapper | PM Wrapper | Choreography Wrapper

CM Wrapper

RM Wrapper

Parser Wrapper

Discovery Wrapper

Selector Wrapper

DM Wrapper

PM Wrapper

Choreography Wrapper

WSMX Management Interface

WSMX Monitor

WSML Editor

Choreography Editor

Mediator Editor

Service Requesters

Back-End Application

Agent acting on behalf of service requester

Data and Communication Protocols Adapters

Adapter 1

Adapter 2

... Adapter n

Service Providers

Web Service 1

Web Service 2

... Web Service p

After data mediation, Discovery iterates, if needed through last steps until result set is finished.

Resource Manager Interface

WSMO Objects

Non WSMO

Reasoner Interface

Reasoner

New Component

Component Wrapper

Interface

Selector Wrapper

Discovery Wrapper

Process Mediator

Choreography Interface

Resource Manager Interface

Invoker Receiver

Grounding

Communication Manager

Resource Manager

Parser

Discovery

Process Mediator

Choreography

CM Wrapper

RM Wrapper

Parser Wrapper

Discovery Wrapper

Selector Wrapper

DM Wrapper

PM Wrapper

Choreography Wrapper

WSMT – Web Services Modelling Toolkit

After data mediation, Discovery iterates, if needed through last steps until result set is finished.
Execution Semantics
Execution Semantics

WSMT – Web Services Modelling Toolkit

WSMX Management  WSMX Monitor  WSML Editor  Choreography Editor  Mediator Editor

WSMX

WSMX Manager

WSMX Manager Core

CM Wrapper  RM Wrapper  Parser Wrapper  Discovery Wrapper  Selector Wrapper  DM Wrapper  PM Wrapper  Choreography Wrapper

Interface  Interface  Interface  Interface  Interface  Interface  Interface  Interface

Communication Manager  Resource Manager  Parser  Discovery  Selector  Data Mediator  Pre Med

Invoker  Receiver  Grounding

Resource Manager Interface

WSMO Objects  Non WSMO  Reasoner

Reasoner Interface

Wrapper

Interface  New Component

Service Requesters

Back-End Application

Agent acting on behalf of service requester

Data and Communication Protocols Adapters

Adapter 1

Adapter 2

... Adapter n

Service Providers

Web Service 1

Web Service 2

... Web Service p

Choreography instance for goal requester is checked for next steps.
Execution Semantics

WSMT – Web Services Modelling Toolkit

WSMX Managment  WSMX Monitor  WSML Editor  Choreography Editor  Mediator Editor

WSMX Manager

WSMX Manager Core

Resource Manager Interface

WSMO Objects

Reasoner Interface

Reasoner

Non WSMO

Component Wrapper

Interface

New Component

WSML Editor

Choreography Editor

Mediator Editor
Execution Semantics
Execution Semantics

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

WSMX - Web Services Modelling Toolkit

WSM Managment  WSMX Monitor  WSML Editor  Choreography Editor  Mediator Editor

WSMX Manager

WSMX Manager Core

Communication Manager

Resource Manager

Parser

Discovery Wrapper

Selector Wrapper

Data Mediator

Process Mediator

Choreography Wrapper

Component Wrapper

Interface

New Component

Resource Manager Interface

WSMO Objects

Non WSMO

Reasoner Interface

Reasoner

Back-End Application

Agent acting on behalf of service requester

Data and Communication Protocols Adapters

WSMT - Web Services Modelling Toolkit

Service Providers

Web Service 1

Web Service 2

Web Service p

WSML Editor

Choreography Editor

WSMX Monitor

WSMX Management

WSMX Monitor

Choreography Editor

Mediator Editor

Component Wrapper

Interface

New Component

Requesters

Service Providers

Web

Service 1

Web

Service 2

...
Future Plans & Conclusions
WSMX Usage Scenario - P2P

- Complete the functionality for all the boxes
Conclusions

• Conceptual model is WSMO (with some add-ons)
• End to end functionality for executing SWS
• Has a formal execution semantics
• Real implementation
• Open source code base at SourceForge
• Event-driven component architecture
• Growing functionality - developers welcome 😊
Demos – WSMX and WSMT
Web Service Modelling Toolkit

• WSMT is a lightweight framework for rapid creation and deployment of homogenous tools for Semantic Web Services.

• Aims
  – Reduce application overhead
  – Provide common reusable functionality
  – Allow new tools to be added dynamically
  – Encourage the creation of tools

• Open source at:
WSMT Architecture

Launcher \[\leftrightarrow\] JAR Files \[\leftrightarrow\] Plug-in Descriptions

Dynamic Class Loader

Core

Communication Component

Invoker \[\leftrightarrow\] Plug-in Loader

Receiver

Logger

Plug-in 1 \[\leftrightarrow\] Plug-in n
Current Tools

- WSMX Invoker (v0.1)
- WSML Editor (v0.4)
- WSMX Data Mediation Mapping Tool (v0.2.2)
PART V: Hands-On Session

• Internet Reasoning Service (IRS III):
  – system overview
  – demonstration

• Hands-on Session:
  – Introduction to Use Case Scenario
  – Hands-on Session tasks
  – Exercises
IRS-III: A framework and platform for building Semantic Web Services

John Domingue and Liliana Cabral
The Internet Reasoning Service is an infrastructure for publishing, locating, executing and composing Semantic Web Services
Design Principles

• Ontological separation of User and Web Service Contexts
• Capability Based Invocation
• Ease of Use
• One Click Publishing
• Agnostic to Service Implementation Platform
• Connected to External Environment
• Open
• Complete Descriptions
• Inspectable
• Interoperable with SWS Frameworks and Platforms
Features of IRS-III (1/2)

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

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

• Enables publication of ‘standard code’
  – Provides clever wrappers
  – One-click publishing of web services

• Integrated with standard Web Services world
  – Semantic web service to IRS
  – ‘Ordinary’ web service
IRS-III Architecture

WSMX
Browser
Publishing Clients
Invocation Client

J a v a

A P I

S O A P

Publishing Platforms

Browser Handler
Publisher Handler
Invocation Handler

WS Publisher Registry
OCML
WSMO Library

OWL(-S) Handler

OWL(-S)

LispWeb Server

Web Service
Java Code
Web Application

Web Application
Java Code
Web Service
Publishing Platform Architecture

Publishing Clients

- SOAP
- IRS-III Server

Invocation Client

- SOAP

HTTP Server

- SOAP Handler
- Service Registrar
- Service Invoker

IRS-III Publishing Platform

WS Service Registry

Web Service 1
- Web Service 2
- Web Service 3
IRS-III/WSMO differences

- Underlying language OCML
- Goals have inputs and outputs
- IRS-III broker finds applicable web services via mediators
  - Used mediator within WS capability
  - Mediator source = goal
- Web services have inputs and outputs ‘inherited’ from goal descriptions
- Web service selected via assumption (in capability)
OWL-S 1.0 Translation

OWL-S Process \rightarrow \textit{OWL-S Translator} \rightarrow \textit{Web Service (Mediator and Goal)}

\textit{OWL Translator}
OWL Process to Web Service

• IOPEs are translated to:
  has-input, has-output, has-precondition and
  has-postcondition
  in the capability of a Web service.

• The type and condition definitions at the range of
  the above roles are translated by the OWL to
  OCML translator.

• Simple goal and mediators can be generated
  (optional) as template for later development.
IRS-III Demo
(including OWL-S Import)

John Domingue and Liliana Cabral
SWS Creation & Usage Steps

• Create a goal description
  – (e.g. exchange-rate-goal)
  – Add input and output roles
  – Include role type and soap binding

• Create a wg-mediator description
  – Source = goal
  – Possibly add a mediation service

• Create a web service description
  – Used-mediator of WS capability = wg-mediator above

• Specify Operation <-- Lisp function mapping in Choreography Grounding

• Publish against web service description

• Invoke web service by ‘achieve goal’
Multiple WS for goal

• Each WS has a mediator for used-mediator slot of capability
  – Some WS may share a mediator

• Define a kappa expression for assumption slot of WS capability

• Kappa expression format
  – (kappa (?goal) <ocml relations>)

• Getting the value of an input role
  – (wsmo-role-value ?goal <role-name>)
Defining a Mediation Service

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

• Classes are unary relations
  – e.g. (country ?x)

• Slots are binary relations
  – e.g. (is-capital-of ?x ?y)

• Standard relations in base (OCML toplevel) ontology
  =, ==, <, >, member
European Currency Assumption

(kappa (?goal)
  (member
    (wsmo-role-value
      ?goal
      'has_source_currency)
    '(euro pound))))
Goal Based Invocation

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

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

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

WS -> Capability -> Assumption
Web service selection
European-exchange-rate

Mediation
Mediate input values
‘$’ -> us-dollar

Invocation
Invoke selected web service
European-exchange-rate
Hands-On Session

John Domingue and Liliana Cabral
European Travel Scenario
European Travel Demo
IRS-III Hands On Task

• Develop an application for the European Travel scenario based on SWS. The application should support a person booking a train ticket between 2 European cities at a specific time and date

• Create Goal, Web service and Mediator WSMO descriptions in IRS-III (european-travel-service-descriptions) for available services. Your descriptions should choose a specific service depending on the start and end locations and the type of traveller. Use the assumption slot to do this

• Publish available lisp functions against your descriptions

• Invoke the web services

• Solution to be shown at the end of this session
Tutorial Setup

IRS Server (3000)

- Domain Models
- Web Service WSMO Descriptions
  + Registry of Implementors
- Goal WSMO Descriptions
  + SOAP Binding
- Mediator WSMO Descriptions

Travel Services (3001)

WSMX

IRS Lisp Publisher

IRS-III Knowledge Model Browser & Editor
Travel Related Knowledge Models
Key Classes, Relations, Instances

Is-in-country <city> <country> e.g.
(is-in-country berlin germany) -> true

(student <person>) -> true, for john matt michal
(business-person <person>) -> true, for liliana michael
Goals

1- Get train timetable
   – Inputs: origin and destination cities (city), date (date-and-time, e.g. (18 4 2004))
   – Output: timetable (string)

2- Book train
   – Inputs: passenger name (person), origin and destination cities, departure time-date (list-date-and-time, e.g. (20 33 16 15 9 2004))
   – Output: booking information (string)
Services

• 1 service available for goal 1
  – No constraints

• 6 services available for goal 2
  – As a provider write the constraints applicable to the services to satisfy the goal (assumption logical expressions)

• 1 wg-mediator mediation-service
  – Used to convert time in list format to time in universal format
Service constraints

• Services 2-5
  – Services for (origin and destination) cities in determined countries

• Service 4-5
  – Need a mediation service to map goal time-date to service time-date

• Services 6-7
  – Services for students or business people in Europe
Available Functions (1/3)

1- get-train-times

paris london (18 4 2004)
"Timetable of trains from PARIS to LONDON on 18, 4, 2004
  5:18
  ...23:36"

2- book-english-train-journey

christoph milton-keynes london (20 33 16 15 9 2004)
"British Rail: CHRISTOPH is booked on the 66 going from MILTON-KEYNES to
  LONDON at 16:49, 15, SEPTEMBER 2004. The price is 169 Euros."

3- book-french-train-journey

sinuhe paris lyon (3 4 6 18 8 2004)
"SNCF: SINUHE is booked on the 511 going from PARIS to LYON at 6:12, 18,
  AUGUST 2004. The price is 27 Euros."
Available Functions (2/3)

4- book-german-train-journey
christoph berlin frankfurt 3304251200
"First Class Booking German Rail (Die Bahn): CHRISTOPH is booked on the 323 going from BERLIN to FRANKFURT at 17:11, 15, SEPTEMBER 2004. The price is 35 Euros."

5- book-austrian-train-journey
sinuhe vienna innsbruck 3304251200
"Austrian Rail (OBB): SINUHE is booked on the 367 going from VIENNA to INNSBRUCK at 16:47, 15, SEPTEMBER 2004. The price is 36 Euros."
Available Functions (3/3)

6- book-student-european-train-journey
john london nice (3 4 6 18 8 2004)
"European Student Rail Travel: JOHN is booked on the 916 going from LONDON to NICE at 6:44, 18, AUGUST 2004. The price is 94 Euros."

7- book-business-european-train-journey
liliana paris innsbruck (3 4 6 18 8 2004)
"Business Europe: LILIANA is booked on the 461 going from PARIS to INNSBRUCK at 6:12, 18, AUGUST 2004. The price is 325 Euros."

8- mediate-time (lisp function) or JavaMediateTime/mediate (java)
(9 30 17 20 9 2004)
3304686609
Example: Multiply Goal

[Image of a software interface showing a goal with inputs and outputs, including fields for name, type, and SOAP type for each input and output.]
Example: Multiply Web Service
Example: Publishing
Tips

- Order matters for input roles
  - Input roles in goal must match order of arguments to function
- Need to specify both input roles and output role
- Be careful with soap binding
  - sexpr as default
  - String for one line output
  - Use xml for multiple line output
- Input roles for web services inherited from goal
- Slot names can not be the same as class names
- Goal <--> web service linking mediator in the capability used mediators
Closing, Outlook, References, Acknowledgements
Tutorial Wrap-up

• The targets of the presented tutorial were to:
  – understand aims & challenges within Semantic Web Services
  – understand Semantic Web Service Frameworks:
    • aims, design principles, and paradigms
    • ontology elements & description

• an overview of Semantic Web Service techniques:
  – element description
  – discovery
  – choreography and service interoperability determination
  – orchestration and composition

• present WSMX a future Web Service based IT middleware
  – design and architecture
  – components design

=> you should now be able to correctly assess emerging technologies & products for Semantic Web Services and utilize these for your future work
Beyond WSMO

• Although WSMO (and OWL-S) are the main initiatives on Semantic Web services, they are not the only ones:
  • Semantic Web Services Interest Group
    – Interest group founded at W3C to discuss issues related to Semantic Web Services (http://www.w3.org/2002/ws/swsig/)
    – Standardization Working Group in starting phase
  • SWSI: International initiative to push toward a standardization of SWS (http://www.swsi.org)
  • Semantic Web services are entering the main stream
    – UDDI is adopting OWL for semantic search
    – WSDL 2 will contain a mapping to RDF
    – The use of semantics is also discussed in the context of standards for WS Policies
SWSI (www.swsi.org)

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

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

• The central location where WSMO work and papers can be found is WSMO Working Group: http://www.wsmo.org

• WSMO languages
  – WSML Working Group http://www.wsml.org

• Web Service Execution Environment WSMX
  – WSMX working group : http://www.wsmx.org
  – WSMX open source can be found at: https://sourceforge.net/projects/wsmx/
References WSMO


References OWL-S

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

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

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

Overview


References
Ontology & Languages

• [Stencil Group] - www.stencilgroup.com/ideas_scope_200106wsdefined.html
References

Discovery


References

Discovery


References

Choreography, Orchestration, Mediation


The central location where WSMX work, papers, and software can be found is the WSMX working group homepage: http://www.wsmx.org.

The main documents are:

- Conceptual Model (http://www.wsmo.org/2004/d13/d13.1/v0.3/)
- Architecture (http://www.wsmo.org/TR/d13/d13.4/v0.2/)
- Implementation: open source at http://sourceforge.net/projects/wsmx
- Documentation (http://www.wsmo.org/TR/d22/v0.2/)
- Execution Semantics (http://www.wsmo.org/TR/d13/d13.2/)
- WSMX Toolkit (http://www.wsmo.org/TR/d9/d9.1/v0.2/)

Further Readings:


References IRS III tutorial


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

• 10-12 May 2004, Heraklion, Crete, Greece.

Acknowledgements

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

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. We dedicate special thanks to Chris Bussler, Michal Zaremba, François Scharffe, Adrian Mocan, and Emilia Cimpian for participation and input during preparation of the tutorial.