Semantic Web Services Systems and Tools

4th International Semantic Web Conference (ISWC 2005)

Galway, Ireland, 6 November 2005

Christoph Bussler
Mick Kerrigan
Matthew Moran
Brahmananda Sapkota
Michael Stollberg
Michal Zaremba

Liliana Cabral
John Domingue
Stefania Galizia
Barry Norton
Agenda

Part I: Introduction to Semantic Web Services and WSMO

Part II: The Web Service Execution Environment, WSMX
  – WSMX motivation, scope, approach
  – System Architecture

Coffee Break

Part III: The Internet Reasoning Service, IRS III
  – IRS Architecture
  – IRS Editing, Browsing, and Publication Environment

Part IV: Hands-On Session with IRS III
  – Create and publish WSMO descriptions
  – Use WSMX and IRS frameworks for goal-driven service invocation
PART I:
Introduction to Semantic Web Services and WSMO

- Semantic Web Services
- WSMO
  - Design principles
  - Top level concepts
Semantic Web Services

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

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

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

Usage Process:

- **Publication**: Make available the description of the capability of a service
- **Discovery**: Locate different services suitable for a given task
- **Selection**: Choose the most appropriate services among the available ones
- **Composition**: Combine services to achieve a goal
- **Mediation**: Solve mismatches (data, process) among the combined
- **Execution**: Invoke services following programmatic conventions
Semantic Web Services (3)

Usage Process – execution support

- **Monitoring**: Control the execution process
- **Compensation**: Provide transactional support and undo or mitigate unwanted effects
- **Replacement**: Facilitate the substitution of services by equivalent ones
- **Auditing**: Verify that service execution occurred in the expected way
Web Service Modeling Ontology (WSMO)

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

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

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

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

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

- Provide the formally specified terminology of the information used by all other components
- Connectors between components with mediation facilities for handling heterogeneities

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

**WSMO D2, version 1.2, 13 April 2005 (W3C submission)**
PART II: The Web Service Execution Environment (WSMX)

• Introduction, background and motivation
• Structural architecture
• Dynamic behaviour
• Future plans
WSMX Introduction

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

• Provide **middleware ‘glue’** for Semantic Web Services  
  – Allow service providers focus on their business
• Provide a **reference implementation** for WSMO  
  – Eat our own cake
• Provide an environment for **goal based** service  
  discovery and invocation  
  – Run-time binding of service requester and provider
• Provide a flexible **Service Oriented Architecture**  
  – Add, update, remove components at run-time as needed
• Keep **open-source** to encourage participation  
  – Developers are free to use in their own code
• Define **formal execution semantics**  
  – Unambiguous model of system behaviour
WSMX Usage Scenario
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
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:
  - November 2004 (WSMX 0.1.5)
  - January 2005 (WSMX 0.1.6)
  - July 2005 (WSMX 0.2.0)
  - November 2005 (WSMX 0.3.0)
Design Principles

Strong Decoupling & Strong Mediation

autonomous components with mediators for interoperability

Interface vs. Implementation

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

Peer to Peer

interaction between equal partners (in terms of control)

WSMO Design Principles == WSMX Design Principles

== SOA Design Principles
Benefits of SOA

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

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

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

• Better Flexibility
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
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
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

W1 … WL

Level of Abstraction

Syntactic

Semantic ("Light")

Semantic ("Heavy")
Discovery

Keyword-based with Natural Language Processing (NLP)

{Keyword}

W1 ... WL

Coarse grained Service and Goal descriptions

Fine grained Service and Goal descriptions

WS

Syntactic

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
System Entry Points

**storeEntity**
- Service Provider
  - Editor
  - Communication Manager (Requester Side)
  - Parser
  - Service Repository
- WSML

**receiveGoal**
- Service Requester
  - Adapter
  - Communication Manager (Requester Side)
  - Parser
  - Matchmaker
  - Selector
- WSML

**receiveMessage**
- Service Requester
  - Adapter
  - Communication Manager (Provider Side)
  - Parser
  - Communication Manager (Requester Side)
  - Choreography Engine
  - Data Mediator
  - Selecter
  - Data Mediator
  - SOAP
  - Service Provider

Legend:
- WSMX components
- External entities
- Execution Flow
- Usage
Define “Business” Process

Start

Create Choreography

Discover Web Services

Created

Discover Services

Mediate Data

Mediate Data

Return Mediated Data

Return Mediated Data

Return Web Services

Check Choreography

Confirmed

Call Invoker

Confirmed

End
Generate Wrappers for Components

- Discover Web Services
- Create Choreography
- Start
- Created
- Data Mediator Wrapper
- Check Choreography
- Confirmed
- Choreography Wrapper
- Communication Manager Wrapper
- Call Invoker
- Confirmed
- End

Discovery Wrapper

Mediate Data

Return Mediated Data

Return Web Services

Mediate Data

Mediate Data

Registry of known components

Start
Context Data
Event-based Implementation

“Business” Process – Internal Workflow

Event and Notification Distribution/Delivery Mechanism

Core – Manager

Choreography Wrapper

Discovery Wrapper

Data Mediator Wrapper

Communication Manager Wrapper

Choreography

Discovery

Mediator

Communication Manager
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

Interface  Interface  Interface  Interface  Interface  Interface  Interface  Interface

Communication Manager  Resource Manager  Parser  Discovery  Selector  Data Mediator  Process Mediator  Choreography

Invoker  Receiver

Grounding

Resource Manager Interface

WSMO Objects  Non WSMO

Reasoner Interface

Reasoner

Component Wrapper

Interface

New Component

Request to discover Web services.
Execution Semantics

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
Interface
Interface
Interface
Interface
Interface
Interface

Communication Manager
Resource Manager
Parser
Discovery
Selector
Data Mediator
Process Mediator
Choreography

Invoker
Receiver
Grounding

WSMO Objects
Non WSMO
Reasoner

Component Wrapper
Interface
New Component

WSMT – Web Services Modelling Toolkit

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

Goal expressed in WSML is sent to WSMX System Interface
Execution Semantics

WSMT – Web Services Modelling Toolkit

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

WSMX Manager Core

Administration Framework Interface

CM Wrapper
CM Interface
Communication Manager
Invoker
Receiver
Grounding

RM Wrapper
RM Interface
Resource Manager
Parser
Selector
Data Mediator
DM Wrapper
Process Mediator
PM Wrapper
Choreography

Adapter 1
Adapter 2
... Adapter n

Data and Communication Protocols Adapters

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

Service Providers
Web Service 1
Web Service 2
... Web Service p

WSM – Web Services Modelling Toolkit

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

Service Providers
Web Service 1
Web Service 2
... Web Service p

WSML Editor
WSMX Monitor
Choreography Editor
Mediator Editor

Resource Manager Interface
WSMO Objects
Non WSMO
Reasoner Interface
Reasoner

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

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

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

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

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

WSMX Service Providers
- Web Service 1
- Web Service 2
- Web Service p

WSML Editor
- WSMT - Web Services Modelling Toolkit
- WSMO Objects
- Resource Manager Interface

WSMX Monitor
- WSMX Management

Choreography Editor
- Choreography Interface

Component Wrapper
- New Component

Reasoner Interface
- Reasoner

Parser Interface
- Parser

Invoker Receiver
- Communication Manager

Grounding
- Discovery Wrapper
- Selector
- Data Mediator
- Process Mediator

CM M. informs Core that Goal has been received

Execution Semantics
Execution Semantics
Execution Semantics

WSMX – Web Services Modelling Toolkit

WSMX Management
WSMX Monitor
WSML Editor
Choreography Editor
Mediator Editor

WSMX Manager
WSMX Manager Core

WSMX

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

Resource Manager Interface
WSMO Objects
Non WSMO

Reasoner Interface
Reasoner

Component Wrapper
Interface
New Component

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

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

WSML Editor
WSMX Monitor
Choreography Editor
WSMX Management

Execution Semantics
WSMX – Web Services Modelling Toolkit

WSMX Manager

WSMX Manager Core

Administration Framework Interface

WSMX

WSMX Managment
WSMX Monitor
WSML Editor
Choreography Editor
Mediator Editor

Service Requesters

Service Providers

Back-End Application

Agent acting on behalf of service requester

Adapter 1

Adapter 2

Adapter n...

Data and Communication Protocols Adapters

WSML goal is parsed to internal format.

WSMO Objects
Non WSMO
Reasoner

Component Wrapper
Interface
New Component

Resource Manager Interface

Reasoner Interface

Execution Semantics
Execution Semantics

WSMT – 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

Administration Framework Interface

Invocation Interface

Resource Manager Interface

WSMO Objects
Non WSMO
Reasoner Interface

Reasoner

Discovery is invoked for parsed goal.

Component 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
Execution Semantics

WSMT – 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

Discovery may requires ontology mediation.

WSMO Objects
Non WSMO
Reasoner

Component Wrapper
Interface
New Component

Service Requesters
Service Providers

Data and Communication Protocols Adapters

WSML Editor
WSMX Monitor
Choreography Editor
Mediator Editor

WSMT – Web Services Modelling Toolkit

Web Service 1
Web Service 2
Web Service p

Agent acting on behalf of service requester
Execution Semantics

WSMT – Web Services Modelling Toolkit

WSMX Management
WSMX Monitor
WSML Editor
Choreography Editor
Mediator Editor

WSMX

Service Requesters

Data and Communication Protocols Adapters

Adapter 1
Adapter 2
... Adapter n

Back-End Application
Agent acting on behalf of service requester

CM Wrapper
RM Wrapper
DM Wrapper

Parser Wrapper
Discovery Wrapper
Selector Wrapper

Resource Manager
Invoker
Receiver
Grounding

Resource Manager Interface
WSMO Objects
Non WSMO

Reasoner Interface
Reasoner

Choreography

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

Component Wrapper
Interface
New Component

Service Providers

Web Service 1
Web Service 2
... Web Service p

WSML Editor
WSMX Monitor
Choreography Editor
Execution Semantics

WSMX – Web Services Modelling Toolkit

WSMX Manager

WSMX Manager Core

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

Interface

Communication Manager
Resource Manager
Parser
Discovery
Selector
Process
Choreography

Resource Manager Interface
WSMO Objects
Non WSMO

Reasoner Interface
Reasoner

Component Wrapper
Interface
New Component

Selection is invoked to relax result set to finally one service.

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

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

Reasoner Interface

Service Providers

Web Service 1
Web Service 2
... Web Service p
Execution Semantics

WSMT – Web Services Modelling Toolkit

WSMX Manager

WSMX Manager Core

CM Wrapper
RM Wrapper
Selector Wrapper
Discovery Wrapper
Data Wrapper
Selector Wrapper
PM Wrapper
Choreography Wrapper

Interface
Interface
Interface
Interface
Interface
Interface
Interface
Interface
Interface

Communication Manager
Resource Manager
Parser
Discovery
Selector
Data Mediator
Pre Mediator

Resource Manager Interface

WSMO Objects
Non WSMO
Reasoner

Reasoner Interface

Wrapper
Interface
New Component

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

Service Providers

Web Service 1
Web Service 2
...
Web Service p

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

WSMX – 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

Grounding

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

Resource Manager Interface

WSMO Objects
Non WSMO

Reasoner Interface

Reasoner

Component Wrapper
Interface
New Component

Result is returned to Com. Man. to be forwarded to the service requester.

Service Requesters

Back-End Application
Agent acting on behalf of service requester

Service Providers

Web Service 1
Web Service 2
...
Web Service p

Execution Semantics
Set of Web Service descriptions expressed in WSML sent to adapter.
Execution Semantics

Set of Web Service descriptions expressed in requester's own format returned to goal requester.
WSMX Usage Scenario - P2P

- Complete the functionality for all the boxes
WSMX Conclusions

- Conceptual model is WSMO
- End to end functionality for executing SWS
- Has a formal execution semantics
- Real implementation
- Open source code base at SourceForge
- Event-driven component architecture
- Growing functionality - developers welcome 😊
PART III: Internet Reasoning Service - IRS

- Design principles
- Architecture and features
- Demonstration
IRS-III

- IRS-III: The Internet Reasoning Service is an infrastructure for publishing, locating, executing and composing Semantic Web Services
- Internet Reasoning Service (IRS-III):
  - System overview
  - Demonstration
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 Framework

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

IRS Publisher
- Lisp
- Java
- Java WS

IRS Client

SOAP
IRS-III Architecture
Publishing Platform Architecture
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)
IRS-III Demo
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 expression**

- Web service selection
- European-exchange-rate

**Mediation**

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

**Invocation**

- Invoke selected web service
- European-exchange-rate
PART III:
Joint WSMX and IRS
Hands-on Session

• IRS III hands-on part
• WSMX hands-on part
IRS-III Hands-On Session
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

Diagram showing relationships between various ontologies and knowledge bases related to travel.
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
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
WSMX Hands-on session
Aims of the Session

• To create WSMO descriptions of Goals and Web Services using different Ontologies
• To register a Web Service description with WSMX
• To create mappings between the different Ontologies used for the Goal and Web Service to allow Data Mediation
• To register Ontology Mappings with WSMX
• To use the Goal created to discover the Web Service description and invoke it.
Creating Semantic Descriptions

• You will be provided with two sets of Ontologies, one for the Goal and one for the Web Services.
• The two Ontologies simulate the case where the user and service provider use different Ontologies.
• You play the role of both the user and the service provider.
• Use the WSML Editor in the WSMT to view the provided Ontologies and create the Goal and Web Service based on the different sets provided.
Register the Web Service

• Use the WSMX Invoker in the WSMT to register the Web Service you created with WSMX.
Create Ontology Mappings

• When there are multiple Ontologies that describe the same thing are used in different descriptions, it is necessary to create mappings between them to allow instances of one be converted to instances of the other.

• Use the WSMX Data Mediator Mapping Tool in the WSMT to create the mappings between the supplied Ontologies and store these mappings in WSMX.

• The tool will provide you with recommendations of which mappings to create. Note: the recommendations become more accurate as you create more mappings.
Storing the Mappings

• Use the WSMX Data Mediation Mapping Tool to store the mappings between the source and target Ontologies.
Invoking the Web Service

- Use the WSMX Invoker in the WSMT to execute the achieveGoal entry-point of WSMX with the Goal you created.
- Communication with WSMX is asynchronous and the messages you sent and the responses you received can be monitored through the WSMX Invoker.
Wrap-up

• Summary
• References
• Acknowledgements
Summary

• WSMX and IRS are two WSMO compliant execution engines
• Both focus on goal-driven service invocation
• Both focus on solving heterogeneity through mediators
• Both share a common public interface
  – Integration at the system level
References WSMX

- The central location where WSMX work, papers, and software can be found is the WSMX working group homepage: [http://www.wsmx.org](http://www.wsmx.org).
- The main documents are:
  - Architecture ([http://www.wsmo.org/TR/d13/d13.4/v0.2/](http://www.wsmo.org/TR/d13/d13.4/v0.2/))
  - Implementation: open source at [http://sourceforge.net/projects/wsmx](http://sourceforge.net/projects/wsmx)
  - Documentation ([http://www.wsmo.org/TR/d22/v0.2/](http://www.wsmo.org/TR/d22/v0.2/))
  - WSMX Toolkit ([http://www.wsmo.org/TR/d9/d9.1/v0.2/](http://www.wsmo.org/TR/d9/d9.1/v0.2/))
- Further Readings:
References IRS III


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

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

Acknowledgements

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