Semantic Web Services
SS 2016

Semantic Web Service Stack – 
WSMO, WSML, WSMX
Anna Fensel
02.05.2016

Where are we?

<table>
<thead>
<tr>
<th>#</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Web Science + TourPack project (separate slideset)</td>
</tr>
<tr>
<td>3</td>
<td>Service Science</td>
</tr>
<tr>
<td>4</td>
<td>Web services</td>
</tr>
<tr>
<td>5</td>
<td>Web2.0 services</td>
</tr>
<tr>
<td>6</td>
<td>Semantic Web + ONLIM APIs (separate slideset)</td>
</tr>
<tr>
<td>7</td>
<td>Semantic Web Service Stack (WSMO, WSML, WSMX)</td>
</tr>
<tr>
<td>8</td>
<td>OWL-S and the others</td>
</tr>
<tr>
<td>9</td>
<td>Semantic Services as a Part of the Future Internet and Big Data Technology</td>
</tr>
<tr>
<td>10</td>
<td>Lightweight Annotations</td>
</tr>
<tr>
<td>11</td>
<td>Linked Services</td>
</tr>
<tr>
<td>12</td>
<td>Applications</td>
</tr>
<tr>
<td>13</td>
<td>Mobile Services</td>
</tr>
</tbody>
</table>
Outline

- Motivation
- WSMO
  - Technical Solution
  - Illustration by a Larger Example
- WSML
  - Technical Solution
  - Illustration by a Larger Example
- WSMX
  - Technical Solution
  - Illustration by a Larger Example
- Summary
- References

MOTIVATION
Motivation for SWS

- 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

- Mechanized support is needed for
  - Annotating/designing services and the data they use
  - Finding and comparing service providers
  - Negotiating and contracting services
  - Composing, enacting, and monitoring services
  - Dealing with numerous and heterogeneous data formats, protocols and processes, i.e. mediation

So what is needed?

- Existing approaches to SWS (OWL-S, SWSF, WSDL-S) do not provide a unifying solution for SWS
  ⇒ WSMO Approach

- A language for the WSMO approach: WSML
  - RDFS and OWL have no support for web services, goals or mediators
  - OWL-S is not expressive enough to cover all aspects of Web Services

- An execution environment to enable the WSMO approach: WSMX
  - Reference implementation
  - Service offerings, required capabilities and exchanged data are semantically annotated
  - An environment to bridge service providers and requesters
  - Automation of tasks with reasoning
Web Service Modeling Ontology (WSMO) - Approach Overview

Conceptual Model for SWS

Ontology & Rule Language for the Semantic Web with built-in support for WSMO

Semantic Execution Environments and independent broker services

http://www.sti2.org/content/conceptual-models-services-working-group-cms-wg
http://www.wsmo.org/wsmx/

WSMO – Design Principles

Web Compliance  Ontology-Based
Strict Decoupling  Ontological Role Separation
Centrality of Mediation  Execution Semantics
Description versus Implementation
WSMO – Design Principles (1)

• Web Compliance
  – WSMO inherits the concept of URI (Universal Resource Identifier) for unique identification of resources as the essential design principle of the Word Wide Web
  – WSMO adopts the concept of Namespaces for denoting consistent information spaces, supports XML and other W3C Web technology recommendations, as well as the decentralization of resources

WSMO – Design Principles (2)

• Ontology-Based
  – Ontologies are used as the data model throughout WSMO
  – All resource descriptions as well as all data interchanged during service usage are based on ontologies
  – WSMO supports the ontology languages defined for the Semantic Web
WSMO – Design Principles (3)

• Strict Decoupling
  – WSMO resources are defined in isolation
  – Each resource is specified *independently* without regard to possible usage or interactions with other resources

WSMO – Design Principles (4)

• Centrality of Mediation
  • Complementary design principle to strict decoupling
  • Mediation addresses the handling of *heterogeneities* that naturally arise in open environments
  • Heterogeneity can occur in terms of data, underlying ontology, protocol or process.
  • Mediation a first class component of the WSMO framework
• Ontological Role Separation
  – Users, or more generally clients, exist in specific contexts which will not be the same as for available Web services
  – The underlying epistemology of WSMO differentiates between the desires of users or clients and available services

• Description versus Implementation
  – WSMO differentiates between the descriptions of Semantic Web services elements (description) and executable technologies (implementation)
  – WSMO aims at providing an appropriate ontological description model, and to be complaint with existing and emerging technologies
WSMO – Design Principles (7)

• Execution Semantics
  – In order to verify the WSMO specification, the formal execution semantics of reference implementations like WSMX as well as other WSMO-enabled systems provide the technical realization of WSMO.

WSMO – Design Principles (8)

• Service versus Web service
  – A Web service is a computational entity which is able (by invocation) to achieve a users goal.
  – A service in contrast is the actual value provided by this invocation.
  – WSMO provides means to describe Web services that provide access (searching, buying, etc.) to services; WSMO is designed as a means to describe the former and not to replace the functionality of the latter.
Web Service Modeling Framework (WSMF)

- The **Web Service Modeling Ontology** (WSMO) is derived from and based on the **Web Service Modeling Framework** (WSMF)

- WSMF provides the appropriate conceptual model for developing and describing web services and their composition

- WSMF is based on the following principle:
  - Strong de-coupling of the various components that realize an e-commerce application,
  - Strong mediation service enabling anybody to speak with everybody in a scalable manner.

Web Service Modeling Framework (WSMF)

- WSMF consists of four main different elements:
  - **ontologies** that provide the terminology used by other elements
  - **goal repositories** that define the problems that should be solved by web services
  - **web services** descriptions that define various aspects of a web service
  - **mediators** which bypass interoperability problems.
Top-level elements defined by WSMO

Objectives that a client may have when consulting a Web Service

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

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

Connectors between components with mediation facilities for handling heterogeneities

Dublin Core and Annotations

- The Dublin Core metadata element set is a standard for cross-domain information resource description.
- Annotations are used in the definition of WSMO elements (reuse of Dublin Core metadata elements)

Examples:
- The creator of the institute identified by http://www.sti-innsbruck.at/ is Dieter Fensel
- The date on which the website http://www.sti-innsbruck.at/ was created is 01.01.2006

Each WSMO element has an attached set of annotations

WSMO – Ontologies

- In WSMO, Ontologies are the key to linking conceptual real-world semantics defined and agreed upon by communities of users.

Examples:
- The Location Ontology (http://www.wsmo.org/ontologies/location) contains the concepts “Country” and “Address”
- The Location Ontology (http://www.wsmo.org/ontologies/location) contains the “Austria” and “Germany” instances

Ontology Specification

- Non functional properties: author, date, ID, etc.
- Imported Ontologies: importing existing ontologies where no heterogeneities arise
- Used mediators: OO Mediators (ontology import with terminology mismatch handling)

Ontology Elements:
- Concepts: set of entities that exists in the world / domain
- 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)
**Concepts** constitute the basic elements of the agreed terminology for some problem domain

- From a high-level perspective, a concept – described by a concept definition – provides attributes with names and types
- A concept can be a subconcept of several (possibly none) direct superconcepts as specified by the isA-relation

```
Class concept sub-Class wsmoElement
  hasSuperConcept type concept
  hasAttribute type attribute
  hasDefinition type logicalExpression multiplicity = single-valued

Class attribute sub-Class wsmoElement
  hasRange type concept multiplicity = single-valued
```

Example:
- The concept “Border” defines the border between two countries. It is a subclass of a more general concept “GeographicLocation”. It has two attributes countryA and countryB whose ranges are instances of concept “Country”

---

**Logical Expressions for the Definition of Concepts**

- The definition of a concept is a logical expression which can be used to define formally the semantics of the concept
  - The logical expression defines (or restricts, respectively) the extension (i.e. the set of instances) of the concept. If C is the identifier denoting the concept then the logical expression takes one of the following forms

  ```
  forAll ?x ( ?x memberOf C implies l-expr(?x) )
  forAll ?x ( ?x memberOf C impliedBy l-expr(?x) )
  forAll ?x ( ?x memberOf C equivalent l-expr(?x) )
  ```

  where l-expr(?x) is a logical expression with precisely one free variable ?x

Example:
- The concept “Human” is defined as the intersection of the concepts “Primate” and “LegalAgent”
**WSMO Ontologies – Relations**

- **Relations** are used in order to model interdependencies between several concepts (respectively instances of these concepts)

  ```
  Class relation sub-Class wsmoElement
  hasSuperRelation type relation
  hasParameter type parameter
  hasDefinition type logicalExpression multiplicity = single-valued
  
  Class parameter sub-Class wsmoElement
  hasDomain type concept multiplicity = single-valued
  ```

**Example:**
- The relation “distanceInKm” has three parameters: two concepts and an integer. The relation represents the distance between two cities. It is a sub-relation of the measurement relation.

**Logical Expressions for the Definition of Relations**

- The definition of a relation is a logical expression defining the set of instances (n-ary tuples, if n is the arity of the relation) of the relation
  - If the parameters are specified, the relation is represented by an n-ary predicate symbol with named arguments. If R is the identifier denoting the relation, then the logical expression takes one of the following forms:
    ```
    forAll ?v1,...,?vn ( R[p1 hasValue ?v1,...,pn hasValue ?vn] implies l-exp(?v1,...,?vn) )
    forAll ?v1,...,?vn ( R[p1 hasValue ?v1,...,pn hasValue ?vn] impliedBy l-exp(?v1,...,?vn) )
    forAll ?v1,...,?vn ( R[p1 hasValue ?v1,...,pn hasValue ?vn] equivalent l-exp(?v1,...,?vn) )
    ```
  - If the parameters are not specified, then the relation is represented by a predicate symbol where the identifier of the relation is used as the name of the predicate symbol. If R is the identifier denoting the relation, then the logical expression takes one of the following forms:
    ```
    forAll ?v1,...,?vn ( R(?v1,...,?vn) implies l-exp(?v1,...,?vn) )
    forAll ?v1,...,?vn ( R(?v1,...,?vn) impliedBy l-exp(?v1,...,?vn) )
    forAll ?v1,...,?vn ( R(?v1,...,?vn) equivalent l-exp(?v1,...,?vn) )
    ```

  where l-exp(?v1,...,?vn) is a logical expression with precisely ?v1,...,?vn as its free variables and p1,...,pn are the names of the parameters of the relation.
WSMO Ontologies – Instances

- Instances are either defined explicitly or by a link to an instance store, i.e., an external storage of instances and their values
- An explicit definition of instances of concepts is as follows:

```
Class instance sub-Class wsmoElement
  hasType type concept
  hasAttributeValues type attributeValue

Class attributeValue sub-Class wsmoElement
  hasAttribute type attribute multiplicity = single-valued
  hasValue type {instance, literal, anonymousId}
```

Example:
- Mary is parent of the twins Paul and Susan

- Instances of relations (with arity n) can be seen as n-tuples of instances of the concepts which are specified as the parameters of the relation

```
Class relationInstance sub-Class wsmoElement
  hasType type relation
  hasParameterValue type parameterValue

Class parameterValue sub-Class wsmoElement
  hasParameter type parameter multiplicity = single-valued
  hasValue type {instance, literal, anonymousId} multiplicity = single-valued
```

Example:
- The distance between Innsbruck and Munich is 234 kilometers

WSMO – the Web Service Element

- WSMO Web service descriptions consist of non-functional, functional, and the behavioral aspects of a Web service
  - A Web service is a computational entity which is able (by invocation) to achieve a users goal.
  - A service in contrast is the actual value provided by this invocation

![Diagram of WSMO Web Service Element]

- Advertising of Web Service
- Support for WS Discovery
- quality aspects
- Web Service Management

Capability
- functional description

Non-functional Properties
- DC + QoS + Version + financial

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

Choreography — Service Interfaces — Orchestration
WSMO – Web Service Non-Functional Properties

• Non-functional properties:
  – Accuracy - the error rate generated by the service
  – Financial - the cost-related and charging-related properties of a service
  – Network-related QoS - QoS mechanisms operating in the transport network which are independent of the service
  – Performance - how fast a service request can be completed
  – Reliability - the ability of a service to perform its functions (to maintain its service quality)
  – Robustness - the ability of the service to function correctly in the presence of incomplete or invalid inputs.
  – Scalability - the ability of the service to process more requests in a certain time interval
  – Security - the ability of a service to provide authentication, authorization, confidentiality, traceability/auditability, data encryption, and non-repudiation
  – Transactional - transactional properties of the service
  – Trust - the trust worthiness of the service

Example:
• If the client is older than 60 or younger than 10 years old the invocation price is lower than 10 euro

WSMO – Web Service Capability

• A capability defines the Web service by means of its functionality

Class capability sub-Class wsmoElement
  importsOntology type ontology
  usesMediator type {ooMediator, wgMediator}
  hasNonFunctionalProperties type nonFunctionalProperty
  hasSharedVariables type sharedVariables
  hasPrecondition type axiom
  hasAssumption type axiom
  hasPostcondition type axiom
  hasEffect type axiom

Example:
• The input for a birth registration service in Germany has to be boy or a girl with birthdate in the past and be born in Germany. The effect of the execution of the service is that after the registration the child is a German citizen.

• Precondition - the information space of the Web service before its execution
• Assumption - the state of the world before the execution of the Web service
• Postcondition - the information space of the Web service after the execution of the Web service
• Effect - the state of the world after the execution of the Web service
• Shared Variables - variables that are shared between preconditions, postconditions, assumptions and effects
WSMO – Web Service Interface

• An interface describes how the functionality of the Web service can be achieved (i.e. how the capability of a Web service can be fulfilled) by providing a twofold view on the operational competence of the Web service:
  – Choreography decomposes a capability in terms of interaction with the Web service
  – Orchestration decomposes a capability in terms of functionality required from other Web services

Class interface sub-Class wsmoElement
  importsOntology type ontology
  usesMediator type ooMediator
  hasNonFunctionalProperties type nonFunctionalProperty
  hasChoreography type choreography
  hasOrchestration type orchestration

WSMO Choreography: An Abstract State Machine Model (1)

• Why ASMs-based model?
  – Minimality: ASMs are based on a small assortment of modeling primitives
  – Expressivity: ASMs can model arbitrary computations
  – Formality: ASMs provide a formal framework to express dynamics

• Basic mechanism in ASMs:
  – A signature defines predicates and functions to be used in the description.
  – Ground facts specify the underlying database states.
  – State changes are described using transition rules, which specify how the states change by falsifying (deleting) some previously true facts and inserting (making true) some other facts.
Abstract State Machines

• Basic ASM are finite sets of conditional state transition rules of the form:

  \[
  \text{if Condition then Updates}
  \]

• A state is represented by a first order structure; a set with relations and functions

• Every algorithm can be rewritten as a finite number of transition rules

Abstract State Machines

• Signature is a finite collection of function names
  – each name comes with an indication of its arity

• Updates is a finite set of assignments of the form

  \[
  f(t_1,\ldots,t_n) := t
  \]

• Execution can be understood as changing (or defining, if there was none) in parallel the value of the occurring functions \( f \) at the indicated arguments to the indicated value
Abstract State Machines

• A guarded rule is a transition
  \[ \text{if } \text{Condition} \text{ then } \text{Updates} \]
  where Condition is the guard under which a rule is applied

• A set of guarded updates are written usually as a list

• They are executed in parallel, so order is immaterial

• All guarded updates on the list are performed simultaneously

Abstract State Machines

• Execution of an ASM
  1. Check which rules apply
  2. Randomly select a/all rule(s)
  3. Perform update
WSMO Choreography: An Abstract State Machine Model

- In WSMO:
  - Signatures are defined using ontologies
  - The ground facts that populate database states are instances of concepts and relations defined by the ontologies
  - State changes are described in terms of creation of new instances or changes to attribute values of objects.

- Transition rules used in WSMO:
  - if Condition then Rules
  - forall Variables with Condition do Rules
  - choose Variables with Condition do Rules

Examples:
- The state signature of the Amazon E-Commerce Service includes the concepts ItemSearchRequest and ItemLookupRequest with mode "in" and BrowseNodeLookupResponse, ItemContainer with mode "out".
- The ItemSearch transition rule checks for the presence of a request ItemSearchRequest and adds an instance of the corresponding ItemSearchResponse to the state (i.e. the state of the execution is changed).

WSMO Goals

- Goals are representations of an objective for which fulfillment is sought through the execution of a Web service. Goals can be descriptions of Web services that would potentially satisfy the user desires.

Examples:
- A person named Paul has to go to the German registration board to register his son.
Example: Web Service Discovery

- Distinguish between abstract service and a specific one
  - Abstract service: a computational entity able to provide many services
  - Service: a concrete invocation of a Web service
- The task
  - Client is interested in getting a specific service
  - Identify possible service providers, which may be able to provide the requested service $S$ for its clients
- Discovery
  - Given a goal and some Service repository determine the set of relevant service providers

Example: Web Service Discovery (cont')

Goal: buy a travel ticket from Vienna to Berlin

Web service: sells train tickets for trips within Europe

Reasoning

Match!

Europe

Vienna & Berlin

Train Ticket

Travel Ticket
WSMO Mediators

• Mediation
  – Data Level - mediate heterogeneous Data Sources
  – Protocol Level - mediate heterogeneous Communication Patterns
  – Process Level - mediate heterogeneous Business Processes

• Four different types of mediators in WSMO
  – ggMediators: mediators that link two goals. This link represents the refinement of the source goal into the target goal or state equivalence if both goals are substitutable
  – ooMediators: mediators that import ontologies and resolve possible representation mismatches between ontologies
  – wgMediators: mediators that link Web services to goals, meaning that the Web service (totally or partially) fulfills the goal to which it is linked. wgMediators may explicitly state the difference between the two entities and map different vocabularies (through the use of ooMediators)
  – wwMediators: mediators linking two Web services

Examples:
• The ooMediator identified by http://example.org/ooMediator translates the owl description of the iso ontology to wsml and adds the necessary statements to make them memberOf> loc:country concept of the wsmo location ontology
• The ggMediator identified by http://example.org/ggMediator links the general goal of getting a citizenship with the concrete goal of registering George

Class mediator sub-Class wsmoElement
  importsOntology type ontology
  hasNonFunctionalProperties type nonFunctionalProperty
  hasSource type (ontology, goal, webService, mediator)
  hasTarget type (ontology, goal, webService, mediator)
  hasMediationService type (goal, webService, wwMediator)

Class ooMediator sub-Class mediator
  hasSource type (ontology, ooMediator)

Class ggMediator sub-Class mediator
  usesMediator type ooMediator
  hasSource type (goal, ggMediator)
  hasTarget type (goal, ggMediator)

Class wgMediator sub-Class mediator
  usesMediator type ooMediator
  hasSource type (webService, goal, wgMediator, ggMediator)
  hasTarget type (webService, goal, ggMediator, wgMediator)

Class wwMediator sub-Class mediator
  usesMediator type ooMediator
  hasSource type (webService, wwMediator)
  hasTarget type (webService, wwMediator)
Example: Process Mediation

- Heterogeneity may exist between exposed communication interfaces of service providers and those expected by service requesters
  - Messages in the wrong order
  - Messages sent separately that are expected together
  - Messages sent together that are expected separately
  - Messages sent that are never expected
  - Messages expected but never sent

- Process Mediation required to addresses these heterogeneity issues and enable dynamic communication between requester and provider

Example: Process Mediation (cont')

- Run-time Process Mediation
  - Input
    - 2 or more processes
  - Output
  - Advantage:
    - Automatic
  - Disadvantage:
    - Un-solvable mismatches

- Design-time Process Mediation
  - Input
    - 2 or more processes
  - Output
    - 1 mediator process
  - Advantage:
    - No un-solvable mismatches
  - Disadvantage:
    - Manual -> Semi-automatic
Example: Process Mediation (cont')
ILLUSTRATION BY A LARGER EXAMPLE

Aim: automatically find shipment services

- The scenario is about how to identify possibly relevant services.
- With an invocation of one of the Web Services you can order a shipment by specifying, sender’s address, receiver’s address, package information and a collection interval during which the shipper will come to your premises to collect the package.
- The request contains the interval in which the shipper shall come to the requesters premises to pick up the package.
- A shipper either responses with the estimated pickup (respecting the given time constraints) or with a fault message indicating that a pick up is not possible in the requested time interval.
- If no constraints on the business hours (earliest and latest pick up time) are given one can assume 8am to 8pm. If a shipper specifies a constraint on how long in advance a shipment can be ordered, this means that the requested collection interval must end before this date. If no constraints on the length of the interval is given one can assume that a shipper requires at least an interval of 60 Minutes.
SWS Scenario – Shipment Discovery (2)

• All dates and times in the advertised services are assumed to be local to the shippers' office.
• For simplicity we only regard Sundays as non-business days.
• All prices are assumed to be in US dollars unless otherwise stated.
• If your package has a large size-to-weight ratio, you may need to consider your package’s dimensional weight. The weight that is used to determine the price of a package, respectively that is considered with respect to the maximum weight restriction of a shipper is the maximum value of its actual and its dimensional weight. The dimensional weight is calculated as follows: Dimensional Weight = (L*W*H)/166 [where L = length, W= width, and H=height] L*W*H yields an amount in cubic inches and is rounded up to the nearest pound.
• We use the definition of continents and countries given by the United Nations
• Each shipper has a guaranteed delivery time. The delivery time is specified in days. The first day of delivery is the day after the package has been picked up. The times are always local.

Examples of ontology elements for the shipment discovery scenario

• Ontology ShipmentOntology has annotations, dc#title (has value "Shipment Domain Ontology"), dc#contributor (has value "Maciej Zaremba, Matt Moran"), dc#date (has value 2006.10.23),
• Ontology ShipmentOntology has concepts:
  – OrderRequest has annotation dc#description whose value is "Information provided for a pickup request" and has a set of attributes: from (of type ContactInfo), to (of type ContactInfo), type (of type ShipmentType), package (of type Package)
  – Package has annotation dc#description whose value is "concept of a package" and a set of attributes: quantity (of type integer), length (of type decimal), width (of type decimal), height (of type decimal), weight (of type decimal)
  – Country has annotation dc#description whose value is "concept of a country" and attributes name (of type string), continent (of type Continent)
  – …
Examples of ontology elements for the shipment discovery scenario (cont')

- Ontology *ShipmentOntology* has relations:
  - *cityIsOnContinent* (relation that holds between a city and the continent it belongs to) with two parameters whose types are City and Continent
  - *cityIsInCountry* (relation that holds between a city and the country it belongs to) with two parameters whose types are City and Country
- Ontology *ShipmentOntology* has instances:
  - *Europe* (member of Continent) whose *name* has value "Europe"
  - *NY* (member of City) whose *name* has value "New York" and *country* has value USA
  - *Luxembourg* (member of City) whose *name* has value "Luxembourg" and *country* has value LuxembourgCountry
- Ontology *ShipmentOntology* has the axiom:
  - *cityIsOnContinentDef* defined by a logical formula that states that if a given city is in a certain country and that country is in a certain continent, then the given city is part of that continent

Examples of Shipping Services

**Muller**
Rates on Request
- Only packages weighing 50 lbs or less are shipped
- Ships to Africa, North America, Europe, Asia (all countries)

Constraints on Collection:
- There must be at least an interval of 90 minutes for collection.
- Collection is possible between 7am and 8pm.
- Collection can be ordered max 2 working days in advance.

Delivery Time:
- Ships in 2/3 (domestic/international) business days if collected by 5pm;

In WSMO:
The WSMuller Web Service has a set of annotations: dc:title (has value "Muller Web Service"), dc:description (has value "We ship to Africa, North America, Europe, Asia (all countries)").
- dc:contributor (has value "Maciej Zaremba, Matt Moran").
The WSMuller Web Service imports the ShipmentOntology and the ShipmentOntologyProcess ontologies.
The WSMuller Web Service Capability WSMullerCapability has a precondition stating that before the execution of the service there must be an order request containing a request for a package weighing 50 lbs or less, and that the destination mentioned in the request should be a location in Africa, North America, Europe, or Asia. In case the request contains a collection, additional conditions are imposed on the request (e.g., the collection can be ordered max 2 working days in advance).
The WSMuller Web Service Capability WSMullerCapability has a postcondition stating that after the execution of the service there will be an order response which will contain the shipping price for the package described in the precondition.
The WSMuller Web Service Choreography will describe a state signature containing the order request and response concepts with the modes “in”, resp “out”. There will be one transition rule stating that for all requests a response will be added to the knowledge base.
Examples of Shipping Services

**Racer**
Rates (flat fee/each lb): Europe (4/1.75), Asia (47/5.75), North America (26/25.4), Rates for South America like North America, Rates for Oceania like Asia.
Furthermore for each collection order 12.50 are added!
List of Countries Racer ships to is included in the WSDL file
Only packages weighing 70lbs or less are shipped

Constraints on Collection:
- There should be at least an interval of 120 minutes for collection.
- Latest Collection time is 8pm.

Delivery Time:
- Ships in 2/3 (domestic/international) business days if collected by 6pm.

**Runner**
Rates (flat fee/each lb): Europe (50/5.75), Asia (60/8.5), North America (15/0.5), South America (56.75/13.5), Africa (96.75/13.5), Oceania has the same rates then Asia
Exact list of countries included in WSDL file
When ordering a shipment using the Web Services, per invocation the shipment of one package can be ordered.
If package weight exceeds 70 lbs, weight, length and height are required (the order has to be done via phone or fax)

Constraints on Collection:
- Collection can be ordered max 5 working days in advance.
- Minimum Advance notice for collection is 1 hour
- Collection is possible between 1am - 12pm
- There must be at least an interval of 30 minutes for collection.

Delivery Time:
- Ships in 2 business days if collected by 10am;
- Ships in 3 business days otherwise.

Constraints on Collection:
- There should be at least an interval of 120 minutes for collection.
- Latest Collection time is 8pm.

Delivery Time:
- Ships in 2/3 (domestic/international) business days if collected by 6pm.

**Walker**
Rates (flat fee/each lb): Europe (41/5.5), Asia (65/10), North America (34/5.3), South America (59/12.3), Africa (85.03/13), Rates for Oceania like Asia.

Only packages weighing 50 lbs or less are shipped
Exact list of countries included in WSDL file
Constraints on Collection:
- Shipment can be ordered maximum 2 business days in advance (the end of the pickup interval must be at most two business days in advance at the time of ordering).
- Pickup time must be between 6 am and 11.00pm.
- There must be at least an interval of 30 minutes for collection.

Delivery Time:
- Ships in 2 business days if collected by 5pm

**Weasel**
Rates (flat fee/each lb): United States (10/1.5)
Delivery only in United States
Constraints on Collection:
- The pickup interval must be at least 5 hours
- The max. pickup interval is 4 days
- Collection can be ordered until 8pm

Delivery Time:
- 1 day if collected before 2pm
## Examples of goals

<table>
<thead>
<tr>
<th>Goal C3</th>
<th>Goal D1</th>
<th>Goal E1</th>
</tr>
</thead>
<tbody>
<tr>
<td>to Smithers (Bristol)</td>
<td>to Szyslak (Tunis)</td>
<td>to Gumble (New York)</td>
</tr>
<tr>
<td>no of packages: 1</td>
<td>no of packages: 2</td>
<td></td>
</tr>
<tr>
<td>package dimensions: (l/w/h) 10/2/3 (inch)</td>
<td>package dimensions: (l/w/h) 5/3/2 (inch)</td>
<td>package dimensions: (l/w/h) 10/2/3 (inch)</td>
</tr>
<tr>
<td>package weight: 20 lbs for less than 120$</td>
<td>package weight: 60 lbs (each)</td>
<td>package weight: 5 lbs for less than 20$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Current Time is 7:30 am</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Next day delivery</td>
</tr>
</tbody>
</table>

**Goal C3 in WSMO:**
- The Goal C3 has a set of annotations: dc:title (has value "Goal C3"), dc:description (has value "Goal of shipping a package to Smithers (Bristol), no of packages: 1, package dimensions: (l/w/h) 10/2/3 (inch), package weight: 20 lbs, for less than 120$"), dc:contributor (has value "Maciej Zaremba, Matt Moran, Tomas Vitvar, Thomas Haselwanter").
- The Goal C3 imports the ShipmentOntology and the ShipmentOntologyProcess ontologies.
- The Goal C3 requested capability has the postcondition stating that the user wants to ship one package with dimensions 10/2/3 (l/w/h) and weight 20 lbs to a specific destination in Bristol. Additionally, in the postcondition is stated that price of the shipment must be less than 120$.

## Result of Discovery Process

<table>
<thead>
<tr>
<th>Goal C3</th>
<th>Goal D1</th>
<th>Goal E1</th>
</tr>
</thead>
<tbody>
<tr>
<td>to Smithers (Bristol)</td>
<td>to Szyslak (Tunis)</td>
<td>to Gumble (New York)</td>
</tr>
<tr>
<td>no of packages: 1</td>
<td>no of packages: 2</td>
<td></td>
</tr>
<tr>
<td>package dimensions: (l/w/h) 10/2/3 (inch)</td>
<td>package dimensions: (l/w/h) 5/3/2 (inch)</td>
<td>package dimensions: (l/w/h) 10/2/3 (inch)</td>
</tr>
<tr>
<td>package weight: 20 lbs for less than 120$</td>
<td>package weight: 60 lbs (each)</td>
<td>package weight: 5 lbs for less than 20$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Current Time is 7:30 am</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Next day delivery</td>
</tr>
</tbody>
</table>

- **Goal C3**
  - Muller (includes a request For quote)
  - NOT: Racer (price is 176$)
  - NOT: Runner (price is 176$)
  - NOT: Walker (price is 151$)
  - NOT: Weasel (ships not to UK)

- **Goal D1**
  - Runner (2 invocations, since schema does not allow to order multiple packages in one invocation)
  - NOT: Racer (does not ship to Tunesia)
  - NOT: Muller (does only ship 50lbs)
  - NOT: Walker (does only ship 50lbs)
  - NOT: Weasel (does not ship to Tunesia)

- **Goal E1**
  - Weasel NOT: Muller (2 days)
  - NOT: Racer (2 days)
  - NOT: Runner (3 days)
  - NOT: Walker (2 days)
EXTENSIONS

WSMO-Lite

WSMO-Lite Ontology

annotations point to

SAWSDL
extends

layer of semantic annotations

MicroWSMO
extends

WSDL
service description layer

hRESTS
A set of concrete languages for the various tasks:

- Ontology / Rule Languages (static view)
  - WSML Core
    - efficiency and compatibility
  - WSML DL
    - decidability, open world semantics
  - WSML Rule
    - efficient existing rule engines
  - WSML Full
    - unifying language, theorem proving
- Languages for dynamics
  - Transaction Logic over ASMs
- Mapping languages
  - for dynamics (process mediation)
  - or data (data mediation)
Key Features of WSML

- **Three** pillars of WSML, namely
  
  (1) a *language independent conceptual model* for Ontologies, Web Services, Goals and Mediators, based on WSMO
  
  (2) *Reuse* of several well-known logical language paradigms in *one* syntactical framework
  
  (3) *Web Compliance*

Key Features of WSML (cont')

- **One syntactic framework** for a set of layered languages
  
  - Different Semantic Web and Semantic Web Service applications need languages of different expressiveness
  
  - No single language paradigm will be sufficient for all use cases
  
  - WSML investigates the use of *Description Logics* and *Logic Programming* for Semantic Web Services
Key Features of WSML (cont’)

• Separation of conceptual and logical modeling
  – The conceptual syntax of WSML has been designed in such a way that it is independent of the underlying logical language
  – No or only limited knowledge of logical languages is required for the basic modeling of Ontologies, Web Services, Goals, and Mediators
  – The logical expression syntax allows expert users to refine definitions on the conceptual syntax using the full expressive power of the underlying logic, which depends on the particular language variant chosen by the user

Key Features of WSML (cont’)

• Semantics based on well known formalisms
  – WSML captures well known logical formalisms such as Datalog and Description Logics in a unifying syntactical framework
    • WSML maintains the established computational properties of the original formalisms through proper syntactic layering
  – The variants allow the reuse of tools already developed for these formalisms
    • Efficient querying engines developed for Datalog
    • Efficient subsumption reasoners developed in the area of Description Logics
    • Inter-operation between the above two paradigms is achieved through a common subset based on Description Logic Programs
Key Features of WSML (cont’)

• **WWW Language**
  – WSML adopts the IRI standard, the successor of URI, for the identification of resources, following the Web architecture
  – WSML adopts the namespace mechanism of XML and datatypes in WSML are compatible with datatypes in XML Schema and datatype functions and operators are based on the functions and operators of XQuery
  – WSML defines an XML syntax and an RDF syntax for exchange over the Web

WSML Variants

• WSML Variants - allow users to make the trade-off between the provided expressivity and the implied complexity on a per-application basis
WSML Variants (cont’)

• **WSML-Core** - defined by the intersection of Description Logic and Horn Logic, based on Description Logic Programs
  – It has the least expressive power of all the languages of the WSML family and therefore has the most preferable computational characteristics
• **WSML-DL** - an extension of WSML-Core which fully captures the Description Logic SHIQ(D), which captures a major part of the (DL species of the) Web Ontology Language OWL
• **WSML-Flight** - an extension of WSML-Core with meta-modeling, constraints and nonmonotonic negation features
• **WSML-Rule** - an extension of WSML-Flight in the direction of Logic Programming
• **WSML-Full** - unifies WSML-DL and WSML-Rule under a First-Order syntactic umbrella with extensions to support the nonmonotonic negation of WSML-Rule

Syntax Basics – Main Parts

• A WSML specification is separated into **two parts:**
  – The first part provides a strictly ordered *meta-information* block about the specification:
    • WSML variant identification
    • Namespace references
    • Annotations
    • Import of ontologies
    • References to mediators used
    • The type of the specification
  – The second part of the specification, consisting of (unordered) elements such as concepts, attributes, relations (in the case of an ontology specification), capability, interfaces (in the case of a goal or web service specification), etc.
Namespaces and Identifiers

- WSML adopts the namespace mechanism of RDF; a namespace can be seen as part of an IRI
  - Namespaces can be used to syntactically distinguish elements of multiple WSML specifications and, more generally, resources on the Web
  - A namespace denotes a syntactical domain for naming resources
- An identifier in WSML is either a data value, an IRI, an anonymous ID, or a variable
  - The sets of identifiers of the following items are disjoint: ontology, goal, Web service, ooMediator, ggMediator, wgMediator, wwMediator, capability, interface, choreography, orchestration, state signature, grounding identifier, variant identifier, datatype wrapper identifier, built-in predicate identifier
Data Values

- WSML has direct support for different types of concrete data, namely, strings, integers and decimals, which correspond to the XML Schema primitive datatypes string, integer and decimal.
- Concrete values can then be used to construct more complex datatypes, corresponding to other XML Schema primitive and derived datatypes, using datatype constructor functions.
- Examples:

  ```xml
  xsd#date(2005,3,12)
  xsd#boolean("true")
  age ofType xsd#integer
  hasChildren ofType xsd#boolean
  ```

Internationalized Resource Identifiers

- The IRI (Internationalized Resource Identifier) mechanism provides a way to identify resources.
  - IRIs may point to resources on the Web (in which case the IRI can start with 'http://'), but this is not necessary (e.g. books can be identified through IRIs starting with 'urn:isbn:')
    ```xml
    _"http://example.org/PersonOntology#Human"
    _"http://www.uibk.ac.at/"
    ```
- An IRI can be abbreviated to a Compact URI - it consists of two parts, namely, the namespace prefix and the local part.
  - A Compact URI is written using a namespace prefix and a localname, separated by a hash ('#'): namespace_prefix#localname
    ```xml
    dc#title (http://purl.org/dc/elements/1.1#title)
    foaf#name (http://xmlns.com/foaf/0.1/name)
    xsd#string (http://www.w3.org/2001/XMLSchema#string)
    Person (http://example.org/#Person)
    hasChild (http://example.org/#hasChild)
    ```
Anonymous Identifiers

• An anonymous identifier represents an IRI which is meant to be globally unique
  – Unnumbered anonymous IDs are denoted with '_#'. Each occurrence of '_#' denotes a new anonymous ID and different occurrences of '_#' are unrelated
  – Numbered anonymous IDs are denoted with '_#n' where n stands for an integer denoting the number of the anonymous ID

• Anonymous identifiers are disallowed for the following elements:
  – the top-level elements ontology, goal, webService, ooMediator, ggMediator, wgMediator and wwMediator
  – capability, interface, choreography and orchestration

Variables

• Variable names start with an initial question mark "?"
• May only occur in place of concepts, attributes, instances, relation arguments or attribute values
  – A variable may not, for example, replace a WSML keyword however
• Variables may only be used:
  – Inside of logical expressions, and
  – As values in a non-functional property definition or in the sharedVariables block within capability definitions
• Examples: ?x, ?y1, ?myVariable
WSML Prologue

- WSML Prologue contains all those elements that are in common between all types of WSML specifications and all WSML variants:
  - WSML Variant e.g. the WSML variant reference for a WSML-Flight specification is:
    ```
    wsmlVariant http://www.wsmo.org/wsml/wsml-syntax/wsml-flight
    ```
  - Namespace References, e.g.
    ```
    namespace _http://www.example.org/ontologies/example#,
    dc _http://purl.org/dc/elements/1.1#,
    foaf _http://xmlns.com/foaf/0.1#,
    wsml _http://www.wsmo.org/wsml-syntax#,
    loc _http://www.wsmo.org/ontologies/location#,
    oo _http://example.org/ooMediator#
    
    namespace _http://www.example.org/ontologies/example#
    ```
  - WSML header - consists of items that any WSML specification may have: annotations, import ontologies and use mediators

WSML header

- Annotations
  ```
  annotations
  dcterms:title hasValue "WSML example ontology"
  dcterms:subject hasValue "family"
  dcterms:description hasValue "fragments of a family ontology to provide WSML examples"
  dcterms:contributor hasValue ( _http://homepage.uiib.ac.at/~c703240/foaf.rdf,
    _http://homepage.uiib.ac.at/~csaa5569/,
    _http://homepage.uiib.ac.at/~c703239/foaf.rdf,
    _http://homepage.uiib.ac.at/homepage/~c703319/foaf.rdf )
  dcterms:date hasValue xsd:dateTime("2004-11-22")
  dcterms:format hasValue "text/html"
  dcterms:language hasValue "en-US"
  dcterms:rights hasValue _http://www.deri.org/privacy.html"
  wsml:version hasValue "$Revision: 1.238 $"
  endAnnotations
  ```
- Importing Ontologies
  ```
  importsOntology (_http://www.wsmo.org/ontologies/location", _http://xmlns.com/foaf/0.1")
  ```
- Using Mediators
  ```
  usesMediator _http://example.org/ooMediator
  ```
Ontologies in WSML – Concepts

- WSML allows inheritance of attribute definitions, which means that a concept inherits all attribute definitions of its superconcepts

```xml
<concept>Human</concept>
subConceptOf {Primate, LegalAgent}

annotations

dc#description hasValue "concept of a human being"
dc#relation hasValue humanDefinition

endAnnotations

hasName ofType foaf#name
hasRelative impliesType Human
hasAncestor impliesType Human
hasParent impliesType Human
hasChild impliesType Human
hasWeightInKG ofType xsd#float
hasBirthdate ofType xsd#date
dateOfDeath ofType xsd#date
hasBirthplace ofType loc#location
isMarriedTo impliesType Human
hasCitizenship ofType oo#country
```

- Axioms can be used to define concepts, e.g.

```xml
<axiom>humanDefinition

definedBy

?x memberOf Human equivalent ?x memberOf Primate and ?x memberOf LegalAgent.
```

Concepts with Attribute Definitions

- WSML allows two kinds of attribute definitions
  - Constraining definitions: An attribute definition of the form $A ofType D$, where $A$ is an attribute identifier and $D$ is a concept identifier, is a constraint on the values for attribute $A$; If the value for the attribute $A$ is not known to be of type $D$, the constraint is violated and the attribute value is inconsistent with respect to the ontology
  - Inferring definitions: An attribute definition of the form $A impliesType D$, where $A$ is an attribute identifier and $D$ is a concept identifier, implies membership of the concept $D$ for all values of the attribute $A$

- Attributes which do not have a datatype range can be specified as being reflexive, transitive, symmetric, or being the inverse of another attribute
- WSML allows the specification of cardinality and range constraints (defined like integrity constraints in databases)
Concepts with Attribute Definitions – Example

```xml
<concept name="Human">
  <dc:description>concept of a human being</dc:description>
  <hasName type="foaf:name">Maria Smith</hasName>
  <hasBirthdate type="xsd:date">1949-09-12</hasBirthdate>
  <hasChild type="(Paul, Susan)"/>
</concept>
```

Ontologies in WSML – Instances

- The `memberOf` keyword identifies the concept to which the instance belongs, e.g.

  ```xml
  <instance name="Mary">
    <memberOf type="(Parent, Woman)"/>
    <dc:description>"Mary is parent of the twins Paul and Susan"</dc:description>
    <hasBirthdate type="xsd:date">1949-09-12</hasBirthdate>
    <hasChild type="(Paul, Susan)"/>
  </instance>
  ```

- WSML allows instances which are not members of a particular concept, e.g.

  ```xml
  <instance name="Mary">
    <hasName type="">"Maria Smith"</hasName>
  </instance>
  ```
Ontologies in WSML – Relations

- Relations in WSML can be used in order to model interdependencies between several concepts (respectively instances of these concepts)
- The parameters of a relation are strictly ordered and their domain can be optionally specified using the keyword `impliesType` or `ofType`
- Examples of relations and relation instances:

```xml
relation distanceInKm (ofType City, ofType City, impliesType _decimal) subRelationOf measurement
relation distanceInKm(3)
relationInstance distance(Innsbruck, Munich, 234)
```

Ontologies in WSML – Axioms

- The usage of axioms in WSML allows for example to refine the definition already given in the conceptual syntax, e.g.

```xml
axiom humanDefinition
  definedBy
    ?x memberOf Human equivalent ?x memberOf Animal and ?x memberOf LegalAgent.
```

- WSML allows the specification of database-style constraints, e.g.

```xml
axiom humanBMIConstraint
  definedBy
    `!- naf bodyMassIndex[bmi hasValue ?b, length hasValue ?l, weight hasValue ?w] and ?x memberOf Human and
    ?x[length hasValue ?l, weight hasValue ?w, bmi hasValue ?b].`
```
Web Service Capabilities in WSML

- The desired and provided functionality of services are described in WSML in the form of capabilities
  - The desired capability is part of a goal and the provided capability is part of a Web service
- Core elements of capabilities:
  - Shared variables: the variables which are shared between the preconditions, postconditions, assumptions and effects
  - Preconditions: conditions on the inputs of the service
  - Postconditions: the relation between the input and the output of the service
  - Assumptions: what must hold (but cannot be checked beforehand) of the state of the world for the Web service to be able to execute successfully
  - Effects: the real-world effects of the execution of the Web service which are not reflected in the output
- A WSML goal or Web service may only have one capability

Web Service Capabilities in WSML – Example

```
capability
    sharedVariables ?child

precondition
    annotations
      dc#description hasValue "The input has to be boy or a girl with birthdate in the past and be born in Germany."
    endAnnotations
    definedBy
      ?child memberOf Child
      and ?child[hasBirthdate hasValue ?birthdate] with birthdate in the past
      and ?child[hasBirthplace hasValue ?location]
      and ?location[locatedIn hasValue oo#de] or
      ( ?child[hasParent hasValue ?parent] and ?parent[hasCitizenship hasValue oo#de] )

effect
    annotations
      dc#description hasValue "After the registration the child is a German citizen"
    endAnnotations
    definedBy
      ?child memberOf Child
      and ?child[hasCitizenship hasValue oo#de].
```
Web Service Interfaces in WSML

• An interface describes how the functionality of the Web service can be achieved by providing a twofold view on the operational competence of the Web service: Choreography decomposes a capability in terms of interaction with the Web service; and Orchestration decomposes a capability in terms of functionality required from other Web services

• Basic mechanism for representing choreographies:
  – A signature defines predicates and functions to be used in the description. Ground facts specify the underlying database states.
    • Signatures are defined using ontologies
    • The ground facts that populate database states are instances of concepts and relations defined by the ontologies
    • State changes are described in terms of creation of new instances or changes to attribute values of objects.
  – State changes are described using transition rules, which specify how the states change by falsifying (deleting) some previously true facts and inserting (making true) some other facts
    • if Condition then Rules
    • forall Variables with Condition do Rules
    • choose Variables with Condition do Rules

Interfaces in WSML – Examples

interface choreography _"http://example.org/mychoreography"
orchestration _"http://example.org/myorchestration"

interface (_"http://example.org/mychoreography", _"http://example.org/mychoreography")

interface buyInterface
  choreography buyChoreography
  annotations
dc:title hasValue "Multimedia Shopping Service Choreography"
dc:description hasValue "Describes the steps required for shopping multimedia items over this web service"
endAnnotations

stateSignature
importsOntology {
  _"http://example.org/ontologies/products/Products",
  _"http://example.org/ontologies/tasks/ShoppingTasks",
  _"http://example.org/ontologies/shopping/Shopping",
  _"http://example.org/ontologies/products/MediaProducts",
  _"http://example.org/ontologies/Media"
}

in
  shoptasks#SearchCatalog withGrounding
  _"http://example.org/webservices/shopping/mediashoppingservice/wsdl.interfaceMessageReference(MediaShoppingServicePortType/SearchCatalog/In)"

out
  mediaproduct#MediaProduct withGrounding
  _"http://example.org/webservices/shopping/mediashoppingservice/wsdl.interfaceMessageReference(MediaShoppingServicePortType/SearchCatalog/Out)"
Interfaces in WSML – Example (cont')

transitionrules
forall ?search
with
  {?search}
  byTitle hasValue ?title,
  byArtist hasValue ?artist,
  byMinPrice hasValue ?minPrice,
  byMaxPrice hasValue ?maxPrice,
  byMinRating hasValue ?minRating,
  byMaxRating hasValue ?maxRating
} memberOf shoptasks#SearchCatalog
and
  ?artist memberOf media#Artist
and
exists ?item{ 
  ?item memberOf mediaproduct#MediaProduct and
  ?item[hasContributor hasValue ?artist] or
  ?item[hasTitle hasValue ?title] or
  (?item[hasPrice hasValue ?price] and
  ?price >= ?minPrice and
  ?price <= ?maxPrice) or
  (?item[hasRating hasValue ?rating] and
  ?rating >= ?minRating and
  ?rating <= ?maxRating) )
}
do
  add {?item{ 
    hasContributor hasValue ?artist,
    hasTitle hasValue ?title,
    hasPrice hasValue ?price,
    hasRating hasValue ?rating } memberOf mediaproduct#MediaProduct }
  add {?artist memberOf media#Artist}
delete {?search memberOf shoptasks#SearchCatalog}
endForall

Non-functional Properties in WSML

- Properties which strictly belong to a Web service, goal, capability, interface or mediator and which are not functional and behavioral
- A WSML Web service, goal, capability, interface or mediator may specify multiple non-functional properties
- Example:

```xml
nonFunctionalProperty
  poiPrice hasValue ?price
  annotations
dc:description hasValue "If the client is older than 60 or younger than 10 years old the invocation price is lower than 10 Euro"
endAnnotations
definedBy
  {client[age hasValue ?age] memberOf hu#human and
  ?age[amount hasValue ?years, units hasValue hu#YearsDuration] 
  memberOf hu#age and
  (greaterEqual(?years, 60) or lessEqual(?years, 10))}
implies
  {poiPrice hasAmount hasValue ?amount, hasCurrency hasValue cur#Euro} memberOf
  poiAbsolutePrice and
  lessEqual(?amount, 10).
```
Goals, Web services, and Mediators in WSML

- A WSML goal specification is identified by the `goal` keyword optionally followed by an IRI which serves as the identifier of the goal, e.g.
  
  ```
  goal _"http://example.org/Germany/GetCitizenShip"
  ```

- A WSML Web service specification is identified by the `webService` keyword optionally followed by an IRI which serves as the identifier of the Web service, e.g.
  
  ```
  webService _"http://example.org/Germany/BirthRegistration"
  ```

- WSML allows for the specification of four kinds of mediators, namely ontology mediators, mediators between Web services, mediators between goals and mediators between Web services and goals, e.g.
  
  ```
  ooMediator _"http://example.org/ooMediator"
  ```

Logical Expressions in WSML

- Logical expressions occur within axioms and the capabilities which are specified in the descriptions of goals and Semantic Web services

- We give a syntax specification for general logical expressions in WSML; the general logical expression syntax encompasses all WSML variants and is thus equivalent to the WSML-Full logical expression syntax
  - Vocabulary
  - Terms
  - Set of Formulae

- We specify for each of the WSML variants the restrictions the variant imposes on the logical expression syntax
### Examples of WSML Logical Expressions

- No human can be both male and female:
  \[
  \neg \exists x [\text{gender hasValue } \{?y, ?z\}] \text{memberOf Human and } ?y = \text{Male and } ?z = \text{Female}.
  \]

- A human who is not a man is a woman:
  \[
  \exists x [\text{gender hasValue Woman}] \text{IMPLIEDBy neg } \exists x [\text{gender hasValue Man}].
  \]

- The brother of a parent is an uncle:
  \[
  \exists x [\text{uncle hasValue } ?z] \text{IMPLIEDBy } \exists x [\text{parent hasValue } ?y] \text{ and } ?y[\text{brother hasValue } ?z].
  \]

- Do not trust strangers:
  \[
  \exists x [\text{distrust hasValue } ?y] \text{ :- naf } \exists x [\text{knows hasValue } ?y].
  \]

### WSML Variants and Feature Matrix – Summary

<table>
<thead>
<tr>
<th>Feature</th>
<th>Core</th>
<th>DL</th>
<th>Flight</th>
<th>Rule</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical Negation (neg)</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Existential Quantification</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>(Head) Disjunction</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>n-ary relations</td>
<td>-</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Meta Modeling</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Default Negation (naf)</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>LP implication</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Integrity Constraints</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Function Symbols</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Unsafe Rules</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Vehicle Ontology

wsmlVariant "http://www.wsmonline.org/wsml/wsml-syntax/wsml-flight"
namespace {'http://www.sti-innsbruck.at/ontologies/vehicle#'}

ontology VehicleOntology
  concept Vehicle
    hasTires ofType _integer
    hasMotor ofType _boolean
  concept Automobile subConceptOf Vehicle
    hasLimitSpeed ofType _integer
  concept Car subConceptOf Automobile
    hasDoors ofType _integer
  concept TwoWheeler subConceptOf Vehicle
  concept Motorcycle subConceptOf {Automobile, TwoWheeler}
  concept Bicycle subConceptOf TwoWheeler

axiom definedBy
// a vehicle with a motor is an automobile
?x memberOf Vehicle and ?x hasMotor hasValue _boolean("true") implies ?x memberOf Automobile.
// an automobile with 4 tires is a car
?x memberOf Automobile and ?x hasTires hasValue 4] implies ?x memberOf Car.
// a vehicle with 2 tires is a TwoWheeler
?x memberOf Vehicle and ?x hasTires hasValue 2] implies ?x memberOf TwoWheeler.
// a two-wheeler with motor is a motorcycle
?x memberOf TwoWheeler and ?x hasMotor hasValue _boolean("true") implies ?x memberOf Motorcycle.
// a two-wheeler without motor is a bicycle.
?x memberOf TwoWheeler and ?x hasMotor hasValue _boolean("false") implies ?x memberOf Bicycle.
Find An Automobile Goal

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace { _"http://www.sti-innsbruck.at/goals#",
  vehicle _"http://www.sti-innsbruck.at/ontologies/vehicle#",
  discovery _"http://wiki.wsmx.org/index.php?title=DiscoveryOntology#" }
goal FindAnAutomobile
importsOntology {vehicle#VehicleOntology}
capability FindAnAutomobileCapability
nonFunctionalProperties
discovery#discoveryStrategy hasValue discovery#HeavyweightDiscovery
discovery#discoveryStrategy hasValue discovery#NoPreFilter
endNonFunctionalProperties
sharedVariables ?x
precondition findAnAutomobilePre
definedBy
  ?x memberOf vehicle#Vehicle and ?x[vehicle#hasMotor hasValue _boolean("true")].
postcondition findAnAutomobilePost
definedBy
  ?x memberOf vehicle#Automobile.
```

Automobile Vendor Web Service

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace { _"http://www.sti-innsbruck.at/services#",
  vehicle _"http://www.sti-innsbruck.at/ontologies/vehicle#",
  discovery _"http://wiki.wsmx.org/index.php?title=DiscoveryOntology#" }
webService AutomobileVendor
importsOntology {vehicle#VehicleOntology}
capability AutomobileCapability
nonFunctionalProperties
discovery#discoveryStrategy hasValue discovery#HeavyweightDiscovery
discovery#discoveryStrategy hasValue discovery#NoPreFilter
endNonFunctionalProperties
sharedVariables ?x
precondition automobilePre
definedBy
  ?x memberOf vehicle#Vehicle and ?x[vehicle#hasMotor hasValue _boolean("true")].
postcondition automobilePost
definedBy
  ?x memberOf vehicle#Automobile.
```
Car Vendor Web Service

wsmlVariant ._"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"

webService CarVendor

importsOntology {vehicle#VehicleOntology}
capability CarCapability

definitions

definitions

definitions

definitions

nonFunctionalProperties
discovery#discoveryStrategy hasValue discovery#HeavyweightDiscovery
discovery#discoveryStrategy hasValue discovery#NoPreFilter
definitions endNonFunctionalProperties

sharedVariables ?x

precondition

carPre

definedBy

?x memberOf vehicle#Automobile.

postcondition

carPost

definedBy

?x memberOf vehicle#Automobile and ?x[vehicle#hasTires hasValue 4].

WSML – Recent development

• 1 new variant and 4 updated versions of WSML developed in SOA4All project:
  – WSML-Quark
  – WSML-Core 2.0
  – WSML-DL 2.0
  – WSML-Flight 2.0
  – WSML-Rule 2.0
Overview
Lifecycle

WSMX aims to support the complete Semantic Web Service lifecycle:

1) Discovery - determines usable services for a request,
2) Composition - combine services to achieve a goal,
3) Selection - chooses most appropriate service among the available ones,
4) Mediation - solves mismatches (data, protocol, process) hampering interoperation,
5) Choreography – interactions and processes between the service providers and clients,
6) Grounding – lifting and lowering between the semantic and syntactic data representations, and
7) Invocation - invokes Web service following programmatic conventions.

WEB SERVICE EXECUTION ENVIRONMENT
Conceptual Model
Architecture

Core Management is a kernel of WSMX with the following objectives
- It realizes the overall operational semantics in order to achieve the functional semantics of its client-side interface
- It orchestrates the functionality of the middleware components into a coherent process in an orderly and consistent fashion called Execution Semantics (see later)
- It ensures the proper inter-component communication through publishing and subscribing to the data as sets of triples over triple space or directly in the synchronous communication manner
WSMX Components
Communication Manager and Invoker

- Responsible for interaction with services and entities that are external to WSMX.
- Should be open to support as many transport and messaging protocols as possible (transparently to WSMX).
- WSMX uses
  - The SOAP implementation from Apache AXIS, and
  - The Apache Web Service Invocation Framework (WSIF).
- Both RPC and Document style invocations possible

![Diagram of WSMX Components](image)

WSMX Components
Grounding

- WSMO service descriptions are grounded to WSDL by the means of XSLT lifting and lowering

![Diagram of WSMX Grounding](image)

Conceptual Model – WSMX Components

Discovery

- Responsible for finding appropriate Web Services capable of fulfilling a goal
- Different techniques available
  - trade-off: ease-of-provision vs. accuracy
  - resource descriptions & matchmaking algorithms

Key Word Matching
  - match natural language key words in resource descriptions,

Controlled Vocabulary
  - ontology-based key word matching, and

Semantic Matchmaking
  - what Semantic Web Services aim at.

Conceptual Model – WSMX Components

Ranking and Selection

- One service which best satisfies the user preferences is selected from the candidate services returned by the service discovery.
- Selection
  - determines best candidate out of discovered WS,
- Ranking
  - determines a priority list of discovered WS.
- The process is run after “functional” discovery
- Criteria:
  - Quality of Service (security, robustness, availability),
  - Context (regional, business / social communities),
  - Preferences and policies,
  - Financial criteria,
  - …
Ontology-to-ontology mediation

A set of mapping rules are defined and then executed

- Ontology Mapping Language

Initially rules are defined semi-automatic

Create for each source instance the target instance(s)

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

* Not implemented yet!
Technical Solution
Conceptual Model – WSMX Components
Choreography
• Requester and provider have their own observable communication patterns
  – Choreography part of WSMO
• Choreography instances are loaded for the requester and provider
  – Both requester and provider have their own WSMO descriptions
• Abstract State Machines (ASM)-based Choreography Engine
  – Evaluation of transition rules
    • prepares the available data
  – Sends data to the Process Mediator
    • filters, changes or replaces data
  – Receives data from PM and forwards it to the Communication manager
    • data to be finally sent to the communication partner

Conceptual Model – WSMX Components
Choreography

• The model used to express typical characteristics of a choreography must be compliant to the following features:
  – Abstract – hides away any details regarding the underlying message exchange protocols and message formats,
  – State-based – the interactions are described in the form of updates on a state of the choreography,
  – Expressive – allows describing features such as sequences of message interactions and branching, and
  – Ontology Reliance – ontologies are used as the underlying data model for message exchanges.

• The abstract state model chosen to address the requirements is inspired by the Abstract State Machines (ASM) methodology.
The choreography model consists primarily of:
- *state signature* – defines ontology which is used by the choreography,
- set of *transition rules* – express the interaction steps in the choreography and also the updates over the state.

State signature allows to define:
- Set of imported ontologies
- Set of *modes* which are associated with the concepts and/or relations.

Modes can be of the following types:
- static – extension of the concept/relation cannot be changed,
- in - extension of the concept/relation can only be changed by the client and read by the choreography instance; grounding mechanism must be provided.
- out - extension of the concept/relation can only be changed by the the choreography instance and read by the client; grounding mechanism must be provided.
- shared - extension of the concept/relation can be changed and read by both choreography instance and client; grounding mechanism must be provided, and
- controlled - extension of the concept/relation is changed and read only by the choreography instance.

Transition rules define actual behavior of the choreography.

The rules can take the form of:
- if Condition then Rules endif – if the condition holds executes the updates.
- forall Variables with Condition do Rules endForall - simultaneous execution of updates for each binding of a variable satisfying a given condition.
- choose Variables with Condition do Rules endChoose - executes an update with an arbitrary binding of a variable chosen among those satisfying the selection condition.

where
- condition (guard) - is an arbitrary logical expression as defined by WSML,
- rules - may take the form of Updates, whose execution is to be understood as changing (or defining, if there was none) instances in an ontology
  - add – adds new instances in the state ontology,
  - delete – deletes instances from the state ontology, and
  - update – updates instances in the state ontology.
Different components within WSMX necessitate efficient and different reasoning functionality:

- **Discovery**
  - Simple ontological reasoning and query answering as well as logical entailment between preconditions and postconditions of SWS and Goals
  - Both description logic-based and logic programming–based reasoning is required.
- **Selection**
  - Evaluation of the logical rules that are used to model the non-functional properties of services
  - Logic programming–based reasoning is required.
- **Data mediation**
  - Ontology mapping rules, source instances and source and target schema information are loaded into the reasoning space where rules are evaluated in order to produce target instances.
  - Logic programming–based reasoning is required.
- **Process Mediation**
  - Reasoning is used to check whether messages are expected at the certain phase of the communication.
  - Evaluation of transition rules is required.

Different reasoning functionality is provided to WSMX through WSML2Reasoning framework.

---

**Conceptual Model – Entry Points**

- Represent input ports to which messages can be sent for initiating specific execution semantics.
- Published as SOAP Endpoints

  - **getWebServices(WSMLDocument)**: Web Services
    - A service requester wishes to discover a list of SWS fulfilling its requirements provided by as a goal description using WSML.
    - A set of WSML Web Service descriptions whose capability matches the goal is returned.

  - **invokeWebService(WSMLDocument, Context)**: Context
    - Used to invoke already known Semantic Web Service by relying on data provided in the form of WSML ontology and conversation context.

  - **achieveGoal(WSMLDocument)**: Context
    - A service requester wishes to use WSMX for all aspects of goal-based service invocation (discovery, mediation, invocation) by providing both goal and data in the single WSML document.
    - Processing of the message is identified by the conversation context.
Behavioral Model
Execution Semantics

• Execution Semantics is a formal description of the operational behavior of the system in terms of computational steps
• The benefits of having behavioral models are in
  – Greater flexibility in SESA implementations,
  – Foundations for model testing,
  – Executable representation, and
  – Improved model understanding among humans.
• Mandatory execution semantics
  – Goal-Based Web Service Discovery
  – Web Service Invocation
  – Goal-Based Service Execution

Illustration by a Larger Example
The goal is to discover a suitable solution for the transportation of a package with defined size and weight.

Candidate Web Services have different constraints regarding the transportation destinations, package size and weight acceptance, as well as pricing schemas.

Has been part of Semantic Web Service Challenge contests.

I want to have my package shipped from CA, USA to Tunis, Africa size (7/6/4), weight 1 lbs, the cheaper the better.
Illustration by larger example
AchieveGoal execution semantics

Goal expressed in WSML is sent to the WSMX Entry Point
AchieveGoal execution semantics

Discovery is employed in order to find suitable Web Service. Discovery consults appropriate ontologies and Web Service descriptions. Web Service may be invoked in order to discover service availability.

Ships only to US ($10/1.5 lb). Cannot be used for Africa.
Illustration by larger example
AchieveGoal execution semantics

List of candidate Web Services is ranked and best solution is selected

Requester and provider choreographies are instantiated and processed

Invocation of Web Service occurs
Illustration by larger example
AchieveGoal execution semantics – choreography exec

choreography WSMullerShipmentOrderChoreography
stateSignature WSMullerShipmentOrderStateSignature

in sop#ShipmentOrderReq
  withGrounding (_"http://sws-challenge.org/shipper/v2/muller.wsdl#wsdl.interfaceMessageReference(muller/ShipmentOrder/in0)"
  in so#ContactInfo
  in so#ShipmentDate
  in so#Package
  in so#Address
  out sop#ShipmentOrderResp

transitionRules WSMullerShipmentOrderTransitionRules
forall {?request} with
  (?request memberOf sop#ShipmentOrderReq)
do
  add(_#1 memberOf sop#ShipmentOrderResp)
delete(?request memberOf sop#ShipmentOrderReq)
endForall

<shipmentOrderReq(soi#MoonContactInfo, soi#shipmentDate1, package, soi#SzyslakContactInfo),
  package(1, 7.0, 6.0, 4.0, 1.0),
  shipmentDate1("2009-01-21T13:00:00.046Z", "2009-01-22T13:00:00.046Z")>

<shipmentOrderResp("2009-01-21T15:00:00.046Z", 65.03),
  package(1, 7.0, 6.0, 4.0, 1.0),
  shipmentDate1("2009-01-21T13:00:00.046Z", "2009-01-22T13:00:00.046Z")>

Illustration by larger example
AchieveGoal execution semantics

Result is returned to the client in the form of WSML message
SUMMARY

Summary

• Semantic Web Services
  – Have the potential of improving the usability of services
  – Lots of progress in the last years
• The WSMO Approach is an active initiative in the area of SWS
• WSML provides a formal syntax and semantics for representing WSMO
  – Designed based on principles of the Semantic Web
• WSMX is a reference implementation of WSMO
REFERENCES

References

• Mandatory reading:

• Further reading:
  – http://www.wsmo.org/TR/d3/d3.4/v0.1/
  – http://www.wsmo.org/
  – http://www.wsmo.org/TR
  – http://www.wsmo.org/wsml
  – http://www.wsmx.org/
  – http://sourceforge.net/projects/wsmx
  – http://www.oasis-open.org/committees/semantic-ex/

• Wikipedia link:
Next Lecture

<table>
<thead>
<tr>
<th>#</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Web Science + TourPack project (separate slideset)</td>
</tr>
<tr>
<td>3</td>
<td>Service Science</td>
</tr>
<tr>
<td>4</td>
<td>Web services</td>
</tr>
<tr>
<td>5</td>
<td>Web2.0 services</td>
</tr>
<tr>
<td>6</td>
<td>Semantic Web + ONLIM APIs (separate slideset)</td>
</tr>
<tr>
<td>7</td>
<td>Semantic Web Service Stack (WSMO, WSML, WSMX)</td>
</tr>
<tr>
<td>8</td>
<td>OWL-S and the others</td>
</tr>
<tr>
<td>9</td>
<td>Semantic Services as a Part of the Future Internet and Big Data Technology</td>
</tr>
<tr>
<td>10</td>
<td>Lightweight Annotations</td>
</tr>
<tr>
<td>11</td>
<td>Linked Services</td>
</tr>
<tr>
<td>12</td>
<td>Applications</td>
</tr>
<tr>
<td>13</td>
<td>Mobile Services</td>
</tr>
</tbody>
</table>

Questions?