STI · INNSBRUCK

Semantic Web Services

The Web Service Modeling Ontology

Lecture VI – 23rd April 2009

Dieter Fensel

Where are we?

#	Date	Title
1	5th March	Introduction
2	12th March	Web Science
3	19th March	Service Science
4	26th March	Web Services (WSDL. SOAP, UDDI, XML)
5	2nd April	Web 2.0 and RESTful services
6	**23rd April**	**WSMO**
7	30th April	WSML
8	7th May	WSMX
9	14th May	OWL-S and others
10	28th May	WSMO-Lite, MicroWSMO
11	4th June	SWS Use Cases
12	18th June	seekda: the business point of view
13	25th June	Mobile services
14	2nd July	Exam Preparation

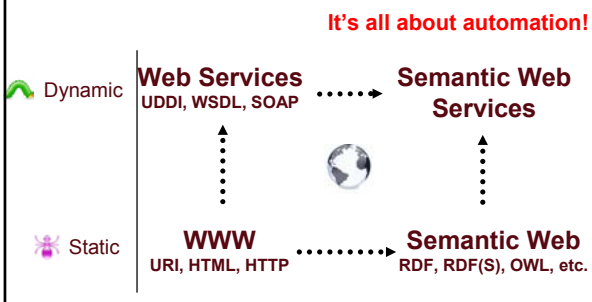
Outline

- Motivation
- Technical solution
  - Ontologies
  - Web Services
  - Goals
  - Mediators
- Illustration by a larger example
- Summary and Conclusions

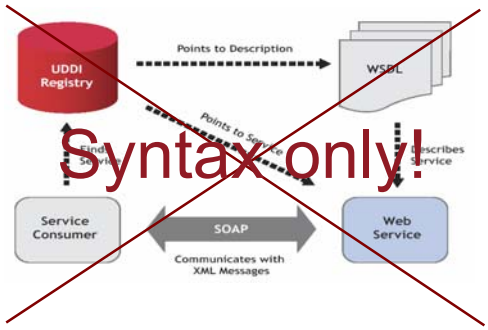
Outline

- Motivation
- Technical solution
  - Ontologies
  - Web Services
  - Goals
  - Mediators
- Illustration by a larger example
- Summary and Conclusions

Semantic Web and Web Services - SWS



Motivation for SWS (1)



Motivation for SWS (2)

- 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

=> Initial Web Service Technology Stack failed to realize the SOA Vision

Problem: Lack of technologies to cope with the scale envisioned for WS
Solution: Techniques for automated support for service related tasks

So what is needed?

- 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

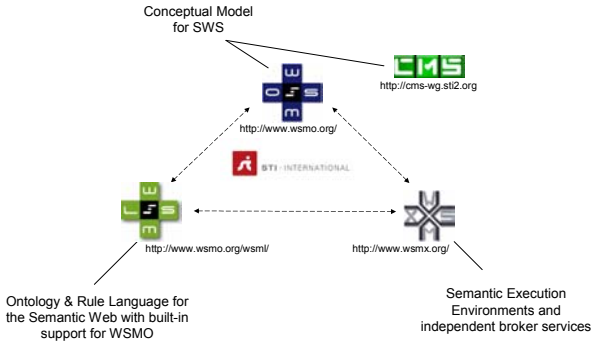
=> Conceptual Models, Formal Languages, Execution Environments

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

Outline

- Motivation
- Technical solution
  - Ontologies
  - Web Services
  - Goals
  - Mediators
- Illustration by a larger example
- Summary and Conclusions

The WSMO Approach



WSMO – Design Principles

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
- 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 (2)

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

WSMO – Design Principles (2)

- 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 (3)

- 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
- 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

Top-level elements defined by WSMO

(http://www.wsmo.org)

Objectives that a client may have when consulting a Web Service

Goals

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

Ontologies

Web Services

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

Mediators

Connectors between components with mediation facilities for handling heterogeneities

Dublin Core

- The Dublin Core metadata element set is a standard for cross-domain information resource description.
- An architecture and abstract model that can be used to develop application profiles to describe resources in a machine processable manner.
- 15 elements or attribute-value pairs – “simple DC”
- 55 elements or attribute-value pairs – “qualified DC”
- Elements may be displayed in any order
- Extensible
- International in scope

Annotations

- Annotations are used in the definition of WSMO elements (reuse of Dublin Core metadata elements)

Class annotation
- hasContributor **type** dc:contributor
- hasCoverage **type** dc:coverage
- hasCreator **type** dc:creator
- hasDate **type** dc:date
- hasDescription **type** dc:description
- hasFormat **type** dc:format
- hasIdentifier **type** dc:identifier
- hasLanguage **type** dc:language
- hasOwner **type** owner
- hasPublisher **type** dc:publisher
- hasRelation **type** dc:relation
- hasRights **type** dc:rights
- hasSource **type** dc:source
- hasSubject **type** dc:subject
- hasTitle **type** dc:title
- hasType **type** dc:type
- hasVersion **type** version

Dublin Core 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

Class wsmoElement
hasAnnotation **type** annotation

WSMO – Ontologies

WSMO – Ontologies

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

Class ontology **sub-Class** wsmoElement
importsOntology **type** ontology
usesMediator **type** ooMediator
hasConcept **type** concept
hasRelation **type** relation
hasFunction **type** function
hasInstance **type** instance
hasRelationInstance **type** relationInstance
hasAxiom **type** axiom

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)

WSMO Ontologies – Concepts

- **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-expr(?v1,...,?vn))
    forAll ?v1,...,?vn (R[p1 **hasValue** ?v1,...,pn **hasValue** ?vn] **impliedBy** l-expr(?v1,...,?vn))
    forAll ?v1,...,?vn (R[p1 **hasValue** ?v1,...,pn **hasValue** ?vn] **equivalent** l-expr(?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-expr(?v1,...,?vn))
    forAll ?v1,...,?vn (R(?v1,...,?vn) **impliedBy** l-expr(?v1,...,?vn))
    forAll ?v1,...,?vn (R(?v1,...,?vn) **equivalent** l-expr(?v1,...,?vn))

  where l-expr(?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

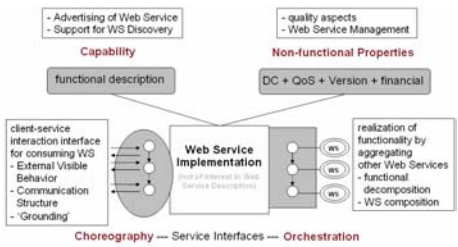
Class relationInstance **sub-Class** wsmoElement
  hasType **type** relation
  hasParameterValue **type** parameterValue
Class parameterValue **sub-Class** wsmoElement
  hasParameter **type** parameter *multiplicity = single-valued*
  hasValue **type** {instance, literal, anonymousId} *multiplicity = single-valued*

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

The Web Service Element

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



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 the 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, postconditons, 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.

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

- 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

- A guarded rule is a transition
  if *Condition* **then** *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

- 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 (2)

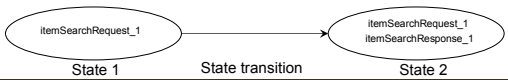
Class choreography
hasNonFunctionalProperties **type** nonFunctionalProperties
hasStateSignature **type** stateSignature
hasState **type** state
hasTransitionRules **type** transitionRules

- 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*

A logical expression, as defined by WSML.

A set of ASM rules: primitive state changes, like *add*, *delete*, or *update* (modify) a fact.

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

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

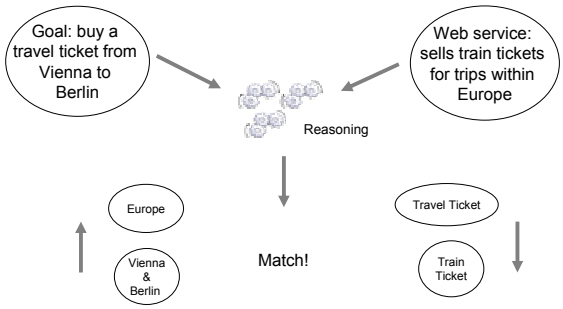
Class goal **sub-Class** wsmoElement
importsOntology **type** ontology
usesMediator **type** {ooMediator, ggMediator}
hasNonFunctionalProperties **type** nonFunctionalProperty
requestsCapability **type** capability *multiplicity = single-valued*
requestsInterface **type** interface

Example:
• A person named Paul has to goal to register his son with the German birth registration board

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')



WSMO Mediators

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

WSMO Mediators (cont')

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}

Examples:

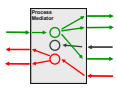
- The ooMediator identified by http://example.org/ooMediator translates the owl description of the iso ontology to wsml and adds the necessary statements to make them memberOf> loc:country concept of the wsmo location ontology
- The ggMediator identified by http://example.org/ggMediator links the general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

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

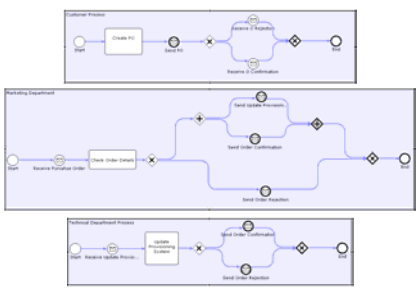
Example: Process Mediation (cont')

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

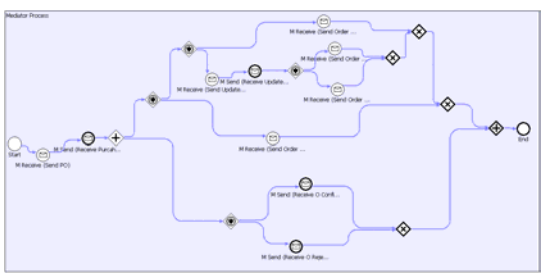


- **Design-time Process Mediation**
  - Input
    - 2 or more processes
  - Output
    - *1 mediator process*
  - Advantage:
    - No un-solvable mismatches
  - Disadvantage:
    - Manual -> Semi-automatic

Example: Process Mediation (cont')

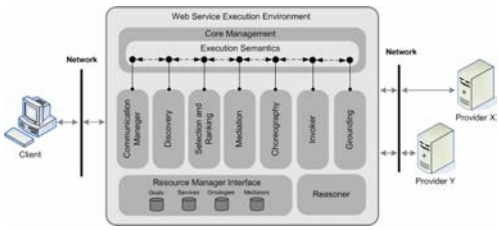


Example: Process Mediation (cont')



WSMX – Web Service Execution Environment

- WSMX – reference implementation for WSMO/L
- Architecture and execution environment



Outline

- Motivation
- Technical solution
  - Ontologies
  - Web Services
  - Goals
  - Mediators
- Illustration by a larger example
- Summary and Conclusions

Illustration by larger example
Scenario description

- 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
- For more information visit:
  - http://sws-challenge.org/wiki/index.php/Scenario:_Shipment_Discovery

Illustration by larger example
Goal description

I want to have my package shipped from CA, USA to Tunis, Africa size (7/6/4), weight 1 lbs, the cheaper the better.

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

goal GoalA1

capability GoalA1Capability
  postcondition
    definedBy
      ?x[sop#price hasValue ?price] memberOf
sop#PriceQuoteResp
        and sop#isShipped(shipmentOrderReq) ).

interface GoalA1Interface
  choreography GoalA1Choreography
    stateSignature GoalA1StateSignature

    in sop#ShipmentOrderReq
    out sop#ShipmentOrderResp

    transitionRules GoalA1TransitionRules

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

```
ontology GoalRequest

instance shipmentOrderReq memberOf sop#ShipmentOrderReq
  sop#from hasValue sol#MoonContactInfo
  sop#shipmentDate hasValue sol#shipmentDate1
  sop#package hasValue package
  sop#to hasValue sol#SzyslakContactInfo

instance package memberOf sol#Package
  sol#quantity hasValue 1
  sol#length hasValue 7.0
  sol#width hasValue 6.0
  sol#height hasValue 4.0
  sol#weight hasValue 1.0

instance shipmentDate1 memberOf sol#ShipmentDate
  sol#earliest hasValue "2009-01-21T13:00:00.046Z"
  sol#latest hasValue "2009-01-22T13:00:00.046Z"
```

Illustration by larger example
AchieveGoal execution semantics

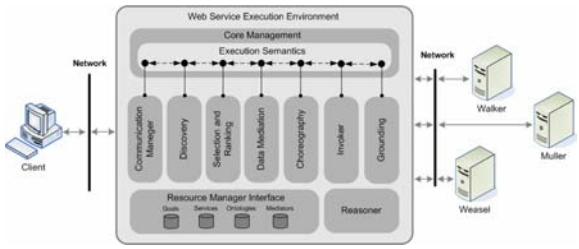


Illustration by larger example
AchieveGoal execution semantics

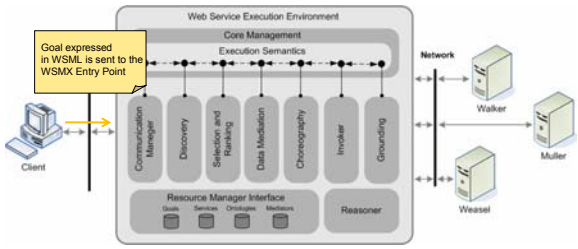


Illustration by larger example
AchieveGoal execution semantics

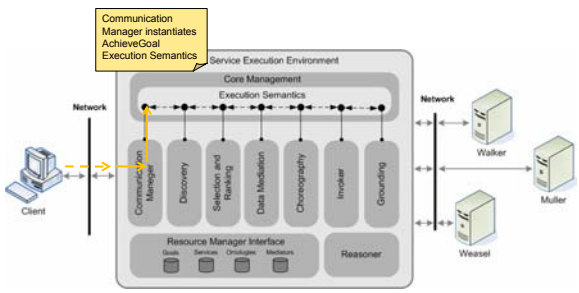


Illustration by larger example
AchieveGoal execution semantics

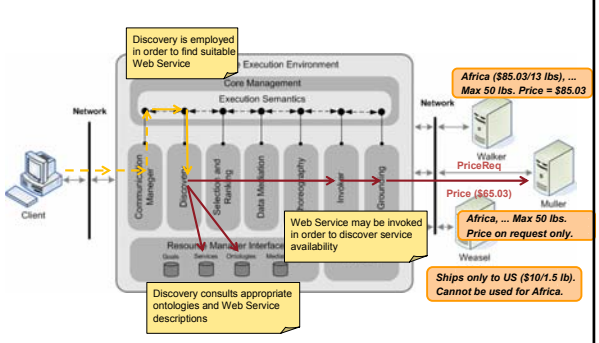


Illustration by larger example
AchieveGoal execution semantics

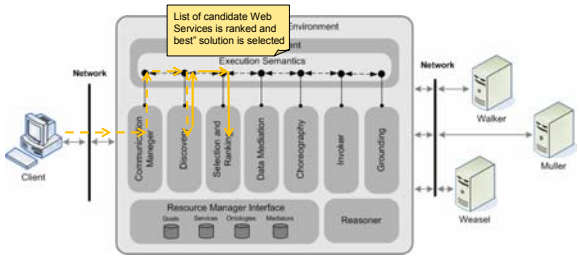


Illustration by larger example
AchieveGoal execution semantics

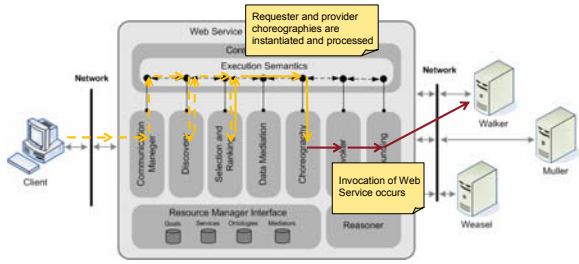


Illustration by larger example
AchieveGoal execution semantics – choreography exec

```
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")>
```

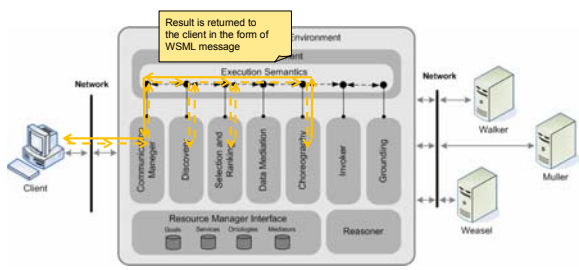
S1

⇩

```
<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")>
```

S2

Illustration by larger example
AchieveGoal execution semantics



SWS Scenario – Shipment Discovery (1)

SWS-Challenge Workshop: http://sws-challenge.org/

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, senders address, receivers 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)

SWS-Challenge Workshop: http://sws-challenge.org/

- 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* hasvalue 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(41/6.75), Asia(47.5/7.15), North America(26.25,4.15), 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(65.75/12), 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.

Examples of Shipping Services (cont')

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 pick up interval must be at least 5 hours
- the max. pick up interval is 4 days
- collection can be ordered until 8pm

Delivery Time
- 1 day if collected before 2pm

Examples of goals

Goal C3
to Smithers (Bristol)
no of packages: 1
package dimensions: (l/w/h) 10/2/3 (inch)
package weight: 20 lbs
for less than 120$

Goal D1
to Szyslak (Tunis)
no of packages: 2
package dimensions: (l/w/h) 5/3/2 (inch)
package weight: 60 lbs (each)

Goal E1
to Gumble (New York)
package dimensions: (l/w/h) 10/2/3 (inch)
package weight: 5 lbs
for less then 20$
Current Time is 7:30 am
Next day delivery

Goal G3 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

Goal C3
to Smithers (Bristol)
no of packages: 1
package dimensions: (l/w/h)
10/2/3 (inch)
package weight: 20 lbs
for less than 120$

⇒ 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
to Szyslak (Tunis)
no of packages: 2
package dimensions: (l/w/h) 5/3/2 (inch)
package weight: 60 lbs (each)

⇒ 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
to Gumble (New York)
package dimensions: (l/w/h) 10/2/3 (inch)
package weight: 5 lbs
for less then 20$
Current Time is 7:30 am
Next day delivery

⇒ Weasel NOT: Muller (2 days)
NOT: Racer (2 days)
NOT: Runner (3 days)
NOT: Walker (2 days)

Outline

- Motivation (for SWS and WSMO)
- Technical solution (the WSMO Approach)
  - Ontologies
  - Web Services
  - Goals
  - Mediators
- Illustration by a larger example
- Summary and Conclusions

Summary and Conclusions

- 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
  - The WSMO conceptual model consists of four core elements: Ontologies, Web Services, Goals, and Mediators
- Standardization based on the WSMO Approach is emerging
  - OASIS SEE TC

Additional References

- WSMO
  - http://www.wsmo.org/
  - http://www.wsmo.org/TR
  - http://www.wsmo.org/TR/d2/v1.3/
  - http://www.wsmo.org/TR/d3/d3.4/v0.1/
- CMS WG
  - http://cms-wg.sti2.org/
- WSML
  - http://www.wsmo.org/wsml
- WSMX
  - http://www.wsmx.org/
  - http://sourceforge.net/projects/wsmx
- WSMT
  - http://wsmt.sourceforge.net
- WSMO Studio
  - http://www.wsmostudio.org
- OASIS SEE TC
  - http://www.oasis-open.org/committees/semantic-ex/
- SWS-Challenge
  - http://sws-challenge.org/

Next Lecture

#	Date	Title
1	5th March	Introduction
2	12th March	Web Science
3	19th March	Service Science
4	26th March	Web Services (WSDL. SOAP, UDDI, XML)
5	2nd April	Web 2.0 and RESTful services
6	23rd April	WSMO
7	**30th April**	**WSML**
8	7th May	WSMX
9	14th May	OWL-S and others
10	28th May	WSMO-Lite, MicroWSMO
11	4th June	SWS Use Cases
12	18th June	seekda: the business point of view
13	25th June	Mobile services
14	2nd July	Exam Preparation

Questions?