Agenda

• Part I: Introduction to Semantic Web Services

• Part II: Semantic Web Service Ontologies
  – OWL-S
  – WSMO
  – differences & commonalities

  Coffee Break: 10.30 – 11.00

• Part III: Addressing Semantic Web Service Challenges
  – Discovery
  – Composition

  Lunch: 12.30 – 14.00

• Part IV: Semantic Web Service Tools & Systems
  – CMU OWL-S Browser
  – Service Composer
  – WSMX

  Coffee Break: 15.30 – 16.00

• Part V: Hands-On Session
PART I:
Introduction to Semantic Web Services

Michael Stollberg
Contents

• The vision of the Semantic Web

• Ontologies as the basic building block

• Current Web Service Technologies

• Vision and Challenges for Semantic Web Services
The Vision

- 500 million users
- more than 3 billion pages

Static

WWW

URI, HTML, HTTP
The Vision

Serious Problems in

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

Static

WWW
URI, HTML, HTTP

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

Dynamic

Web Services
UDDI, WSDL, SOAP

Static

WWW
URI, HTML, HTTP

Semantic Web
RDF, RDF(S), OWL

Bringing the computer back as a device for computation
The Vision

Bringing the web to its full potential

Dynamic

Web Services
UDDI, WSDL, SOAP

Semantic Web Services

Static

WWW
URI, HTML, HTTP

Semantic Web
RDF, RDF(S), OWL

2nd European Semantic Web Conference, Heraklion, Greece, May 2005
The Semantic Web

- the next generation of the WWW
- information has machine-processable and machine-understandable semantics
- not a separate Web but an augmentation of the current one
- Ontologies as basic building block
Ontology Definition

formal, explicit specification of a shared conceptualization

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

To make the Semantic Web working we need:

• **Ontology Languages:**
  – expressivity
  – reasoning support
  – web compliance

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

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

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

• and … **Applications**
Web Services

- loosely coupled, reusable components
- encapsulate discrete functionality
- distributed
- programmatically accessible over standard internet protocols
- add new level of functionality on top of the current web
The Promise of Web Services

web-based SOA as new system design paradigm

UDDI Registry

Points to Description

WSDL

Points to Service

Describes Service

Service Consumer

Communicates with XML Messages

SOAP

Web Service

Finds Service
WSDL

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

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

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

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

- Simple Object Access Protocol
- W3C Recommendation

XML data transport:
- sender / receiver
- protocol binding
- communication aspects
- content
Lacks of Web Service Technology

- current technologies allow usage of Web Services
- but:
  - only syntactical information descriptions
  - syntactic support for discovery, composition and execution

=> **Web Service usability, usage, and integration needs to be inspected manually**
  - no semantically marked up content / services
  - no support for the Semantic Web

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

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

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

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

- define exhaustive description frameworks for describing Web Services and related aspects *(Web Service Description Ontologies)*

- support ontologies as underlying data model to allow machine supported data interpretation *(Semantic Web aspect)*

- define semantically driven technologies for automation of the Web Service usage process *(Web Service aspect)*
Semantic Web Services

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

Execution support:

- **Monitoring**: Control the execution process
- **Compensation**: Provide transactional support and undo or mitigate unwanted effects
- **Replacement**: Facilitate the substitution of services by equivalent ones
- **Auditing**: Verify that service execution occurred in the expected way
PART II:
Semantic Web Service
Ontologies
Katia Sycara
Michael Stollberg
Dumitru Roman
Contents

- **OWL-S**
  - Upper Ontology
  - Service Profile
  - Process Model
  - Service Grounding

- **WSMO**
  - WSMO top level notions
  - Choreography and Mediation
  - Mediation

- **Differences and Commonalities**
OWL-S

Katia Sycara
OWL-S Ontology

- OWL-S is an OWL ontology to describe Web services
- OWL-S leverages on OWL to
  - Support capability based discovery of Web services
  - Support automatic composition of Web Services
  - Support automatic invocation of Web services

*Complete do not compete*
- OWL-S does not aim to replace the Web services standards
  rather OWL-S attempts to provide a semantic layer
  - OWL-S relies on WSDL for Web service invocation (*see Grounding*)
  - OWL-s Expands UDDI for Web service discovery (*OWL-S/UDDI mapping*)
OWL-S Upper Ontology

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

- Capability specification
- General features of the Service
  - Quality of Service
  - Classification in Service taxonomies

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

Service Profile
- Presented by a service.
- Represents what the service provides
- Two main uses:
  1. Advertisements of Web Services capabilities
  2. Request of Web services with a given set of capabilities
OWL-S Profile in a Nutshell

- Describes Web service
  - What capabilities it provides:
    - What transformation the service computes
    - Type of service and products
  - General features such as
    - Agent providing the service
    - Security requirements
    - Quality guarantees of service
- Primary role: to assist discovery
  - Allows capability based search
  - Allows selection based on requirements of the requester
- Profile does not specify use/invocation
OWL-S Service Profile

Capability Description

- **Preconditions**
  - Set of conditions that should hold prior to service invocation

- **Inputs**
  - Set of necessary inputs that the requester should provide to invoke the service

- **Outputs**
  - Results that the requester should expect after interaction with the service provider is completed

- **Effects**
  - Set of statements that should hold true if the service is invoked successfully.

- **Service type**
  - What kind of service is provided (eg selling vs distribution)

- **Product**
  - Product associated with the service (eg travel vs books vs auto parts)
OWL-S Service Profile

Additional Properties

- **Security Parameters**
  - Specify the security capabilities of a Web service (e.g., support for X509 Encryption)
  - Specify the security requirements of a Web service (e.g., a client should be able to provide X509 Encryption)

- **Quality rating**
  - What level of service quality does the Web service provide?

- **Description with standard business taxonomies**
  - How would the service be classified in standard taxonomies such as UNSPSC or NAICS?

This is not a closed set, new properties can be added using existing ontologies.
Process Model

- Process Model
  - Describes how a service works: internal processes of the service
  - Specifies service interaction protocol
  - Specifies abstract messages: ontological type of information transmitted

Facilitates
- Web service invocation
- Composition of Web services
- Monitoring of interaction
Viewpoints of Process Model

• Three viewpoints of a Web service
  – **Glass Box:**
    • The Web service reveals all its internal structure
    • Which parts of the service it performs in-house, which one it subcontracts, etc
  – **Black Box:**
    • The Web service model does not reveal anything about the internal working of the service
    • It just specifies what data it gathers and what data it sends back
  – **Grey Box:**
    • The Web service selectively hides some parts of its Process Model, while it publicizes others
Definition of Process

• A Process represents a transformation (function). It is characterized by four parameters
  – **Inputs**: the inputs that the process requires
  – **Preconditions**: the conditions that are required for the process to run correctly
  – **Outputs**: the information that results from (and is returned from) the execution of the process
  – **Results**: a process may have different outcomes depending on some condition
    • **Condition**: under what condition the result occurs
    • **Constraints on Outputs**
    • **Effects**: real world changes resulting from the execution of the process
Motivation for Results

• Processes may terminate in exceptional states:
  – The credit company may fail to charge the credit card
  – The book may be out of stock
  – The deliver of the goods may fail

• Results support modeling of non-deterministic outcomes of Web services
  – The condition specifies when an outcome is generated
  – Each outcome is characterized by
    • a set of constraints on outputs
    • a set of effects
Example of Process

<process:AtomicProcess rdf:ID="LogIn">
  <process:hasInput rdf:resource="#AcctName"/>
  <process:hasInput rdf:resource="#Password"/>
  <process:hasOutput rdf:resource="#Ack"/>
  <process:hasPrecondition isMember(AccName)/>
  <process:hasResult>
    <process:Result>
      <process:inCondition>
        <expr:SWRL-Condition>
          correctLoginInfo(AccName,Password)
        </expr:SWRL-Condition>
      </process:inCondition>
      <process:withOutput rdf:resource="#Ack">
        <valueType rdr:resource="#LoginAcceptMsg"/>
      </process:withOutput>
      <process:hasEffect>
        <expr:SWRL-Condition>
          loggedIn(AccName,Password)
        </expr:SWRL-Condition>
      </process:hasEffect>
    </process:Result>
  </process:hasResult>
</process:AtomicProcess>
Ontology of Processes

- **Process**
  - **Atomic**: Invokable bound to grounding
  - **Simple**: Provides abstraction, encapsulation etc.
  - **Composite**: Defines a workflow composed of process performs
Process Model Organization

• **Process Model is described as a tree structure**
  – Composite processes are internal nodes
  – Simple and Atomic Processes are the leaves

• **Simple processes represent an abstraction**
  – Placeholders of processes that aren’t specified
  – Or that may be expressed in many different ways

• **Atomic Processes correspond to the basic actions that the Web service performs**
  – Hide the details of how the process is implemented
  – Correspond to WSDL operations
Composite Processes

- Composite Processes specify how processes work together to compute a complex function
- Composite processes define
  1. **Control Flow**
     Specify the temporal relations between the executions of the different sub-processes
  2. **Data Flow**
     Specify how the data produced by one process is transferred to another process
Example of Composite Process

- **Perform**
  - Airline
  - Depart
  - Arrive

- **Get Flights**
  - Flights

- **Select Flight**
  - Flight

**Control Flow Links**
Specify order of execution

**Data-Flow Links**
Specify transfer of data

**Perform statements**
Specify the execution of a process
**Perform** Construct

- **Perform** provides invocation mechanism
  - Specify context of process execution
    - input data flow
    - hooks for output data flow
- Distinction between definition and invocation of a process
  - Definition specifies the process’ I/P/R
  - Perform specify when the process is invoked and with what parameters
Control Flow

- Processes can be chained to form a workflow
- OWL-S supports the following control flow constructs
  - **Sequence/Any-Order**: represents a list of processes that are executed in sequence or arbitrary order
  - **Conditionals**: if-then-else statements
  - **Loops**: while and repeat-until statements
  - **Multithreading and synchronization**: split process in multiple threads, and rendezvous (joint) points
  - **Non-deterministic choices**: (arbitrarily) select one process of a set
Data Flow

Dataflow allows information that is transferred from process to process.

**Output → Input:**

The information produced by one process is transferred to another in the same control construct

**Input → Input:**

The information received by a composite process is transferred to the sub-processes

**Output → Output:**

The information produced by a subprocess is transferred to a super-process
Process Model: take home lesson

• Service Model describes
  – Set of processes that define the operations performed by the Web service
  – Control flow describing the temporal flow of processes
  – Data flow describing the transfer of information between sub-processes
Service Grounding

- Service Grounding
  - Provides a specification of service access information.
  - Service Model + Grounding give everything needed for using the service
  - Builds upon **WSDL** to define message structure and physical binding layer

- Specifies:
  - communication protocols, transport mechanisms, communication languages, etc.
Rationale of Service Grounding

• Provides a specification of service access information.

• Service Model + Grounding give everything needed for using the service
  – Service description is for reasoning about the service
    • Decide what information to send and what to expect
  – Service Grounding is for message passing
    • Generate outgoing messages, and get incoming messages
    • Mapping XML Schemata to OWL concepts

• Builds upon **WSDL** to define message structure and physical binding layer
Mapping OWL-S / WSDL 1.1

- **Operations** correspond to Atomic Processes
- **Input/Output messages** correspond to Inputs/Outputs of processes
Example of Grounding

Sequence
BookFlight

Perform
Get Flights
Perform
Select Flight

Get Flights Op
Select Flight op

WSDL

Airline
Depart
Arrive

Flights

Flight
Result of using the Grounding

- **Invocation mechanism for OWL-S**
  - Invocation based on WSDL
  - Different types of invocation supported by WSDL can be used with OWL-S

- **Clear separation between service description and invocation/implementation**
  - Service description is needed to reason about the service
    - Decide how to use it
    - Decide how what information to send and what to expect
  - Service implementation may be based on SOAP an XSD types
  - The crucial point is that the information that travels on the wires and the information used in the ontologies is the same

- **Allows any web service to be represented using OWL-S**
  - For example: Amazon.com
Handling stateful vs stateless Web services

1. Stateless Web services
   - The server does not maintain the state of the computation
   - Dataflow links specify how the client communicate the state to the service

2. Stateful Web services
   - The service does maintain the state
   - No need of dataflow links since transfer of information is opaque to the client
Representing Stateful Web services

**Stateless**: no information is transferred between the two operations.
Representing Stateless Web services

**Stateful**: information is recorded by the server, no need of transfer between the two operations.
Conclusion OWL-S section

• OWL-S provides a language for the description of Web services
  – Service Profile provides description of capabilities of Web Service
    • Allows capability-based discovery
  – Process Model provides the description of how to use a Web service
    • Allows automatic invocation of Web service
  – Service Grounding maps Atomic Processes into WSDL operations
    • Allows separation between description and implementation
    • Supports description of arbitrary Web services
Web Service Modeling Ontology
WSMO

Michael Stollberg
Outline

• WSMO
  – aims & objectives
  – working structure
• Design Principles
• Top Level Notions
  – Ontologies
  – Web Services
  – Goals
  – Mediators
WSMO is ..

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

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

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

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

- Web Compliance
- Ontology-Based
- Strict Decoupling
- Centrality of Mediation
- Ontological Role Separation
- Description versus Implementation
- Execution Semantics
WSMO Top Level Notions

Objectives that a client wants to achieve by using Web Services

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

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

Connectors between components with mediation facilities for handling heterogeneities

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

every WSMO element is described by properties that contain relevant, non-functional aspects

• Dublin Core Metadata Set:
  – complete item description
  – used for resource management

• Versioning Information
  – evolution support

• Quality of Service Information
  – availability, stability

• Other
  – Owner, financial
## Non-Functional Properties List

### Dublin Core Metadata
- Contributor
- Coverage
- Creator
- Description
- Format
- Identifier
- Language
- Publisher
- Relation
- Rights
- Source
- Subject
- Title
- Type

### Quality of Service
- Accuracy
- NetworkRelatedQoS
- Performance
- Reliability
- Robustness
- Scalability
- Security
- Transactional
- Trust

### Other
- Financial
- Owner
- TypeOfMatch
- Version
WSMO Ontologies

Objectives that a client wants to achieve by using Web Services

- **Capability** (functional)
- **Interfaces** (usage)

Connectors between components with mediation facilities for handling heterogeneities

Provide the formally specified terminology of the information used by all other components
Ontology Usage & Principles

• **Ontologies are used as the ‘data model’ throughout WSMO**
  - all WSMO element descriptions rely on ontologies
  - all data interchanged in Web Service usage are ontologies
  - Semantic information processing & ontology reasoning

• **WSMO Ontology Language WSML**
  - conceptual syntax for describing WSMO elements
  - logical language for axiomatic expressions (WSML Layering)

• **WSMO Ontology Design**
  - **Modularization:** import / re-using ontologies, modular approach for ontology design
  - **De-Coupling:** heterogeneity handled by **OO Mediators**
Ontology Specification

- Non functional properties (see before)
- Imported Ontologies importing existing ontologies where no heterogeneities arise
- Used mediators OO Mediators (ontology import with terminology mismatch handling)

Ontology Elements:
- **Concepts** set of concepts that belong to the ontology, incl.
- **Attributes** set of attributes that belong to a concept
- **Relations** define interrelations between several concepts
- **Functions** special type of relation (unary range = return value)
- **Instances** set of instances that belong to the represented ontology
- **Axioms** axiomatic expressions in ontology (logical statement)
WSMO Web Services

Objectives that a client wants to achieve by using Web Services

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

Connectors between components with mediation facilities for handling heterogeneities

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

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

Capability
- Advertising of Web Service
- Support for WS Discovery

WS - Advertising of Web Service
- Support for WS Discovery

Web Service Implementation
- functional description

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

Choreography --- Service Interfaces --- Orchestration

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

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

- Non functional properties
- Imported Ontologies
- Used mediators
  - OO Mediator: importing ontologies with mismatch resolution
  - WG Mediator: link to a Goal wherefore service is not usable a priori
- Pre-conditions
  What a web service expects in order to be able to provide its service. They define conditions over the input.
- Assumptions
  Conditions on the state of the world that has to hold before the Web Service can be executed
- Post-conditions
  describes the result of the Web Service in relation to the input, and conditions on it
- Effects
  Conditions on the state of the world that hold after execution of the Web Service (i.e. changes in the state of the world)
Choreography & Orchestration

• **VTA example:**

  - **Choreography** = how to interact with the service to consume its functionality
  - **Orchestration** = how service functionality is achieved by aggregating other Web Services
Choreography Aspects

Interface for consuming Web Service

- **External Visible Behavior**
  - those aspects of the workflow of a Web Service where Interaction is required
  - described by workflow constructs: sequence, split, loop, parallel
- **Communication Structure**
  - messages sent and received
  - their order (communicative behavior for service consumption)
- **Grounding**
  - executable communication technology for interaction
  - choreography related errors (e.g. input wrong, message timeout, etc.)
- **Formal Model**
  - reasoning on Web Service interfaces (service interoperability)
  - allow mediation support on Web Service interfaces
Orchestration Aspects

Control Structure for aggregation of other Web Services

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

• service interfaces are concerned with service consumption and interaction
• Choreography and Orchestration as sub-concepts of Service Interface
• common requirements for service interface description:
  1. represent the dynamics of information interchange during service consumption and interaction
  2. support ontologies as the underlying data model
  3. appropriate communication technology for information interchange
  4. sound formal model / semantics of service interface specifications in order to allow operations on them.
Service Interface Description

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

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

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

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

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

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

Communication Behavior of a Web Service

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

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

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

State $\omega_1$

a memberOf A [att1 hasValue x att2 hasValue y]

received ontology instance $a$

Guarded Transition $GT(\omega_1)$

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

State $\omega_2$

b memberOf B [att2 hasValue m]

sent ontology instance $b$
Future Directions

Choreography:
- interaction of services / service and client
- a „choreography interface“ describes the behavior of a Web Service for client-service interaction for consuming the service

Orchestration:
- how the functionality of a Web Service is achieved by aggregating other Web Services
- extends Choreography descriptions by control & data flow constructs between orchestrating WS and orchestrated WSs.

Conceptual models

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

workflow constructs as basis for describing service interfaces:
- workflow based process models for describing behavior
- on basis of generic workflow constructs (e.g. van der Aalst)

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

Grounding:
- making service interfaces executable
- currently grounding to WSDL

Ontologies as data model:
- every resource description based on ontologies
- every data element interchanged is ontology instance
WSMO Goals

Objectives that a client wants to achieve by using Web Services

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

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

Connectors between components with mediation facilities for handling heterogeneities
Goals

• **Ontological De-coupling of Requester and Provider**

• **Goal-driven Approach**, derived from AI rational agent approach
  - Requester formulates objective independently
  - ‘Intelligent’ mechanisms detect suitable services for solving the Goal
  - allows re-use of Services for different purposes

• **Usage of Goals within Semantic Web Services**
  – A Requester (human or machine) defines a Goal to be resolved
  – Web Service Discovery detects suitable Web Services for solving the Goal automatically
  – Goal Resolution Management is realized in implementations
Goal Specification

- **Non functional properties**
- **Imported Ontologies**
- **Used mediators**
  - *OO Mediators:* importing ontologies with heterogeneity resolution
  - *GG Mediator:*
    - Goal definition by reusing an already existing goal
    - allows definition of **Goal Ontologies**
- **Requested Capability**
  - describes service functionality expected to resolve the objective
  - defined as capability description from the requester perspective
- **Requested Interface**
  - describes communication behaviour supported by the requester for consuming a Web Service *(Choreography)*
  - Restrictions / preferences on orchestrations of acceptable Web Services
WSMO Mediators

Objectives that a client wants to achieve by using Web Services

- Capability (functional)
- Interfaces (usage)

Connectors between components with mediation facilities for handling heterogeneities

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

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

• Heterogeneity …
  – Mismatches on structural / semantic / conceptual / level
  – Occur between different components that shall interoperate
  – Especially in distributed & open environments like the Internet

• Concept of Mediation (Wiederhold, 94):
  – Mediators as components that resolve mismatches
  – Declarative Approach:
    • Semantic description of resources
    • ‘Intelligent’ mechanisms that resolve mismatches independent of content
  – Mediation cannot be fully automated (integration decision)

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

WSMO Mediator uses a Mediation Service via

- as a Goal
- directly
- optionally incl. Mediation

Source Component

Source Component

Target Component

Mediation Services
**OO Mediator - Example**

**Merging 2 ontologies**

- **Train Connection Ontology (s1)**
- **Purchase Ontology (s2)**
- **OO Mediator Mediation Service**
- **Train Ticket Purchase Ontology**
- **Goal:**
  - “merge s1, s2 and s1.ticket subclassof s2.product”
  - **Discovery**
  - **Mediation Services**
GG Mediators

• **Aim:**
  – Support specification of Goals by re-using existing Goals
  – Allow definition of **Goal Ontologies** (collection of pre-defined Goals)
  – Terminology mismatches handled by OO Mediators

• **Example: Goal Refinement**

![Diagram showing goal refinement process]

**Source Goal**
"Buy a ticket"

**GG Mediator**
Mediation Service

**Target Goal**
"Buy a Train Ticket"

**postcondition:**
"aTicket memberof trainticket"
WG & WW Mediators

• **WG Mediators:**
  – link a Web Service to a Goal and resolve occurring mismatches
  – match Web Service and Goals that do not match a priori
  – handle terminology mismatches between Web Services and Goals
  ⇒ broader range of Goals solvable by a Web Service

• **WW Mediators:**
  – enable interoperability of heterogeneous Web Services
  ⇒ support automated collaboration between Web Services

  – **OO Mediators** for terminology import with data level mediation
  – Protocol Mediation for establishing valid multi-party collaborations
  – Process Mediation for making Business Processes interoperable
OWL-S and WSMO

Commonalities and Differences
Outline

• Perspectives
• Relation of Ontology Elements
• Interoperability and Mediation
• Semantic Representation
OWL-S Perspective

- OWL-S is an ontology and a language to describe Web services
- Guiding lines for the development of OWL-S
  - Strong relation to Web Services standards
    - Rather than proposing another WS standard, OWL-S aims at enriching existing standards
    - OWL-S is grounded in WSDL and it has been mapped into UDDI
  - Based on the Semantic Web
    - Ontologies provide conceptual framework to describe the domain of Web services and an inference engine to reason about the domain
    - Ontologies are essential elements of interoperation between Web services
  - Build upon 30 years of AI research on Knowledge Representation and Planning
WSMO Perspective

- WSMO is a conceptual model for the core elements of Semantic Web Services
  - core elements: Ontologies, Web Services, Goals, Mediators
  - ontology for precise, unambiguous, element description
  - language for semantic element description (WSML)
  - reference implementation (WSMX)
- Focus on solving the integration problem
- Mediation as a key element
- Ontologies as a data model
  - every resource description is based on ontologies
  - every data element interchanged is an ontology instance
- Based on Knowledge Engineering and B2B Integration experience
OWL-S and WSMO

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

• Request
  – OWL-S uses Profiles to express existing capabilities (advertisements) and desired capabilities (requests)
  – WSMO separates provider (capabilities) and requester points of view (goals)
• Conceptually, OWL-S requested profile and WSMO goal are not exactly the same
  – Requested service profile vs requester objectives
OWL-S and WSMO

OWL-S Process Model ≈ WSMO Service Interfaces

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

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

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

OWL-S Grounding \( \approx \) current WSMO Grounding

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

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

- Interaction of Web services is bound to produce many forms of mismatch
  - **Data mismatch**: the interacting parties do not agree on the data format that they are using
  - **Ontology mismatch**: the interacting parties refer to different ontologies
  - **Protocols mismatch**: the interacting parties expect information at different times
  - **Goals Mismatch**: the interacting parties attempt to achieve very different goals
  - **Interpretations Mismatch**: The interacting parties interpret the same information in very different ways
- These mismatches need to be reconciled for the interoperation to succeed.
- **Mediators are the components that reconcile these mismatches**
Mediation in OWL-S and WSMO

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

- OWL-S and WSMO adopt a similar view on the need of ontologies and explicit semantics but they rely on different logics
  - OWL-S is based on OWL/SWRL
    - OWL represents taxonomical knowledge
    - SWRL provides inference rules
    - FLOWS as formal model for process model
  - WSMO is based on WSML, a family of languages with a common basis for compatibility and extensions in the direction of Description Logics and Logic Programming
• WSML aims at overcoming deficiencies of OWL
• Relation between WSML and OWL+SWRL to be completed
## Summary

<table>
<thead>
<tr>
<th></th>
<th>OWL-S</th>
<th>WSMO</th>
<th>current Web Service technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Discovery</strong>&lt;br&gt;<em>What it does</em></td>
<td>Profile</td>
<td>Goals and Web Services (capability)</td>
<td><strong>UDDI API</strong></td>
</tr>
<tr>
<td><strong>Consumption &amp; Interaction</strong>&lt;br&gt;<em>How to consume &amp; realize</em></td>
<td>Process Model</td>
<td>Service Interfaces (Choreography + Orchestration)</td>
<td><strong>BPEL4WS</strong></td>
</tr>
<tr>
<td><strong>Invocation</strong>&lt;br&gt;<em>How to invoke</em></td>
<td>Grounding+ WSDL/SOAP</td>
<td>Grounding (WSDL / SOAP, ontology-based)</td>
<td><strong>WSDL/SOAP</strong></td>
</tr>
</tbody>
</table>
PART III: Addressing Semantic Web Service Challenges

Michael Stollberg
Daniela Berardi
Contents

• Aspects of Semantic Web Services

• Discovery
  – Problem of Discovery
  – Existing approaches overview

• Composition
  – Problem of Composition
  – Existing approaches overview
Challenges

- Web services as loosely coupled components that shall interoperate dynamically and automatically
- Techniques required for:
  - **Discovery**
    - How are Web services found and selected?
  - **Composition**
    - How to aggregate Web Services into a complex functionality?
  - **Conversation**
    - How to ensure automated interaction of Web Services?
  - **Invocation**
    - How to access and invoke Semantic Web Services?
  - **Mediation and Interoperability**
    - How are data and protocol mismatches resolved?
Discovery
Problems & Approaches Overview

Michael Stollberg
Outline

• Aspects of Discovery
• Terminology
• Discovery Process
• Discovery Techniques
  – keyword-word based retrieval
  – controlled vocabulary / pre-filtering
• Semantic Discovery
  – Matchmaking Notions
  – Approaches & Prototypes
Aspects of Discovery

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

- **Aims:**
  - high precision discovery
  - maximal automation
  - effective discoverer architectures
- **Requirements:**
  - infrastructure that allows storage and retrieval of information about Web services
  - description of Web services functionality
  - description of requests or goals
  - algorithms for matching requesters for capabilities with the corresponding providers
Terminology

• Web Services:
  – abstract Web Services
    • provides access to concrete Web Services
    • has a description
  – concrete Web Services
    • a concrete execution of a Web Service with given input values
    • corresponds to an abstract Web Service

• Goals / Requests:
  – predefined goals (Goal Templates)
    • generic structure of user requests
    • ease goal / request creation by users
    • defined in Goal Ontologies
  – concrete goals (Goal Instances)
    • concrete objectives
    • serve as client for automated service usage

based on “Conceptual Architecture for Semantic Web Services”, C. Preist, ISWC 2004
Overall Discovery Process

Requester Desire → Goal Discovery → Selected Goal Template → Goal Refinement → Concrete Goal

Goal Repository
Goal Template

Service Repository
abstract WS

Abstract Service Discovery
usable abstract Web Services

Concrete Service Discovery & Selection

Concrete Web Service

Discovery Techniques

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

Key Word Matching

match natural language key words in resource descriptions

Controlled Vocabulary

ontology-based key word matching

Semantic Matchmaking

... what Semantic Web Services aim at
Keyword-based retrieval: UDDI

- Service Information by:
  - provider
  - services + binding templates
  - categories
- T-Models allow creating specific information models on resources
- standard API for finding & retrieving information on services and providers

=> allows finding all information on available Web Service

=> Web Service usage / integration to be done manually
Semantic Web Services in UDDI

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

⇒ mapping semantic descriptions to syntactic repository

⇒ allows retrieval of structural information


Controlled Vocabulary

OWL-S Profile Hierarchies

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

http://www.daml.org/services/owl-s/1.0/ProfileHierarchy.owl
Controlled Vocabulary

WSMO non-functional properties

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

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

Semantic Matchmaking

usability determination on basis of precise semantic descriptions

- **OWL-S Profile** provides capability description & request
  - Functional capabilities (what the Web services does)
  - Quality parameters (how the Web service does it)
  - Capability description & request are both Profile-based
  - OWL-S reliance on OWL provides basis for semantic matching

- **WSMO** separates requester and provider viewpoints
  - WSMO goals describe requester objectives
  - WSMO capabilities describe WS functionality
  - Non-functional properties used for security, trust, etc.
OWL-S Profile Matching

- Advertisements (service provider) and requests described as OWL-S Service Profiles
- Matching inputs and outputs of advertisement and request
- Five degrees of match:
  - Exact
  - PlugIn $R \subseteq A$
  - Subsumed $A \subseteq R$
  - Intersection $\neg (A \cap R \subseteq \bot)$
  - Fail when disjoint $A \cap R \subseteq \bot$

- This is ontology-subsumption matching


WSMO Capabilities: Set-based Modeling

- capability as state-relation (pre- & post state of service usage)
- each capability description element restricts the information space to the set of possible instances that satisfy the element
- used in both goals and service descriptions

Assumption

Effect

Precondition

Postcondition

shared variables

computational

non-computational

Information Space
all possible instances of used ontologies
Matchmaking Notions & Intentions

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

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

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

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

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

Discovery Approach

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

**Goal Capability**
- Precondition
- Assumption
- Postcondition
- Effect

**Web Service Capability**
- Precondition
- Assumption
- Postcondition
- Effect

**Assumption**
- Precondition
- Exact

**Effect**
- Postcondition
- Exact

**Plug-In**
- Valid pre-state?
  - No: abort
  - Yes: exact

**Intersection**
- Valid post-state?
  - No: abort
  - Yes: match
Prototype with TA/Flora2

- **Realization**
  - F-Logic Reasoner with Transaction Logic Support
  - Resource Modeling
    - Service Capability: set of rules for each element
    - Goal Capability: pre-state as facts, post-state as queries
  - State: model of a logical theory (facts & rules)
  - State-Transitions: Update of the logical theory
    - Insertion & Deletion of Facts and/or Rules
    - Matchmaking on basis of current state

- **Procedure:**
  1. Goal pre-state satisfies Service pre-state?
  2. Insert Goal pre-state into Knowledge Base (KB)
  3. Can KB satisfy Service post-state (hypoth. execution)?
  4. If yes, can Service post-state satisfy Goal post-state?

Prototype with VAMPIRE

- **Realization**
  - FOL Theorem Prover
  - Universe as Knowledge Definition:
    - ontology schemas (concepts, relations, axioms)
    - generic instances for all concepts and relations
  - Matchmaking: Proof Obligations
    - Goal and Service descriptions as logical theories
    - Matchmaking Notions as Proof Obligations
    - not bound to DL / LP reasoning support

- **Universe Definition:**
  - supports matchmaking without knowledge creation / insertion at runtime
  - handling of incomplete facts (modeling intention ≠ semantics in logics)


---

**Generic Instance**

\[ \forall v, \ldots, v \exists x \exists x. \\
\text{instanceOf}(x, \text{concept}) \\
\text{hasAttr}(x, \text{att}, v) \\
\ldots \\
\text{hasAttr}(x, \text{att}, v) \]

**Incomplete Facts**

\[ \text{concept} \text{ Person [} \\
\text{ age ofType integer} \\
\text{ sex ofType string} \text{]} \]

\[ ?x \text{ memberOf} \text{ Person[} \\
\text{ age hasValue } ?A \text{]} \\
\text{ and } ?A = 80 . \]
Conclusions

- Discovery as central Semantic Web Services technology
- Approaches from OWL-S and WSMO are converging
- Integrated Discoverer Architectures admired:
  - Resource Repository (UDDI or other)
  - Keyword-/ Classification-based Filtering
  - Controlled Vocabulary Filtering
  - Semantic Matchmaking
  - Usable Web Service

retrieve Service Descriptions
invoke Web Service

efficient narrowing of search space (relevant services to be inspected)
Automatic Service Composition: A Conceptual Perspective

Daniela Berardi

based on joint work with Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella
Composition

• Deals with the implementation of an application (in turn offered as a service) whose application logic involves the invocation of operations offered by other services
  – The new service is the *composite* service
  – The invoked services are the *component* services
Synthesis and Orchestration

• **(Composition) Synthesis**: building the specification of the composite service (i.e., the composition schema)
  – Manual
  – Automatic

• **Orchestration**: the run-time management of the composite service (invoking other services, scheduling the different steps, etc.)
  – Composition schema is the “program” to be executed
  – Similarities with WfMSs (Workflow Management Systems)
Composition Schema

- A **composition schema** specifies the "process" of the composite service
  - The "workflow" of the service
- Different clients, by interacting with the composite service, satisfy their specific needs (reach their goals)
  - A specific execution of the composition schema for a given client is an orchestration instance
Service Composition System

1. **Synthesis**
   - Service descriptions
     - Functional features
     - Non-functional features
     - Available service 1
   - Functional requirements of the target service
   - Non-functional requirements of the target service
   - Specification of the process of the composite service
   - Additional requirements for orchestration

2. **Orchestration**
   - Available service n
   - Additional requirements for orchestration

3. **Monitoring**
   - Available service invocation
   - Target service invocation
   - Client

---

**Terms and Concepts**
- Service Composition System
- Synthesis
- Orchestration
- Monitoring
- Client
- Target service invocation
- Available service invocation
1. Composition Synthesis:
   Input:
   – client request
   – set of available services
   Output:
   – specification of composite service

2. Orchestration:
   Input:
   – specification of composite service
   Output:
   – coordination of available services according to the composition schema
   – data flow and control flow monitoring
Service Description

- Services export a view of their behavior
  - I/O interface
    - Data Access
      - focus on data
      - for information gathering
    - Atomic Actions
      - focus on actions
      - world altering services
  - Complex Behavioral Description
    (typically represented using finite states, e.g., TSs)
The Whole Picture

Composition as (classical) planning

Knoblock’s group

Traverso’s group

McIlraith’s group

The Roman group

Hull’s group

Diagram inspired from Hull&Su 2004 SIGMOD tutorial

Statics of the system

Dynamics of target service

Dynamics of component services
Key Dimensions in Service Composition (1)

1. Statics of the composition system (i.e., static semantics):
   - e.g., ontologies of services (for sharing semantics of data/information), inputs and outputs, etc.

2. Dynamics of component services (i.e., dynamic semantics, process):
   - e.g., behavioral features, complex forms of dataflow, transactional attitudes, adaptability to varying circumstances
3. Dynamics of the target service (i.e., dynamic semantics, process)

The target service exposed as:

- single step
- (set of) sequential steps
- (set of) conditional steps
- while/loops, running batch
- while/loops, running under an external control
Key Dimensions in Service Composition: the 4\textsuperscript{th} dimension

4. Degree of (in)completeness in the specification of:
   \begin{itemize}
   \item Static Aspects (of the composition system)
   \item Dynamic Aspects (of component services)
   \item Target service specification
   \end{itemize}

• Note: Orthogonal to previous dimensions

For simplicity not shown in the following slides
What is Addressed from the Technical Point of View?

- Automatic composition techniques?
  - Which formal tools?
  - Sound and complete techniques?
  - Techniques/Problem investigated from computational point of view?
Analyzed Works

- Knoblock’s group (information oriented services)
- Composition as Planning (services as atomic actions)
- Traverso’s group
- McIlraith’s group
- Hull’s group
- The Roman group

as called by Rick Hull in his SIGMOD 2004 tutorial
Knoblock’s Group

• available service: data query
  – basic idea: informative services as views over data sources
  – each service described in terms of I/O parameters (of course, the latter being provided by the data source), binding patterns and additional constraints on the source

• client request:
  – data query, expressed in terms of inputs provided by the client and requested outputs
Knoblock’s Group

• **service composition problem:**
  
  – **Input:** *(i)* available services modeled as data-sources, and *(ii)* client request as user query
  
  – **Output:** (automatically obtained) composite service as integration plan for a *generalized* user query, so that all the user queries that differ only for intensional input values can be answered by the same (composite) service. Integration plan as a sequence of source queries, taking binding pattern into account
Composition as Planning

- available services: atomic actions
- client request: client (propositional) goal
- **service composition problem**: planning problem
  - **Input**: (i) client goal (also encodes initial condition)  
    (ii) available services as atomic actions
  - **Output**: composite service as a (possibly conditional) plan, i.e., sequence of actions that transform the initial state into a state satisfying the goal.
  - Sirin, Parsia, Wu, Hendler & Nau [Sirin et al ICWS03]
  - ICAPS 2003 Planning for Web Services workshop [P4WS03]
  - ICAPS 2004 Planning for Web and Grid Services workshop [P4WGS04]
- **NOTE**: the client has not influence over the control flow of the composite service
Example (1)

• Component Services
  – \( S_1: \text{True} \rightarrow \{S_1: \text{bookFlight}\} \) FlightBooked \( \land \) MayBookLimo
    MayBookLimo \( \rightarrow \{S_1: \text{bookLimo}\} \) LimoBooked
  – \( S_2: \text{True} \rightarrow \{S_2: \text{bookHotel}\} \) HotelBooked
    HotelBooked \( \rightarrow \{S_2: \text{bookShuttle}\} \) ShuttleBooked
  – \( S_3: \text{True} \rightarrow \{S_3: \text{bookEvent}\} \) EventBooked

• Ontology:
  – TravelSettledUp \( \land \) FlightBooked \( \land \) HotelBooked \( \land \) EventBooked
  – CommutingSettled \( \land \) ShuttleBooked \( \lor \) LimoBooked \( \lor \)
    TaxiAvailabilityChecked
  – ...

• Client Service Request:
  – Find a composition of the actions (i.e., a sequence, a program using
    such actions as basic instructions) such that a given property is
    fulfilled
Example (2)

• Component Services
    MayBookLimo → {S₁:bookLimo} LimoBooked
  – S₂: True → {S₂:bookHotel} HotelBooked
    HotelBooked → {S₂:bookShuttle} ShuttleBooked

• Ontology:
  – TravelSettledUp ∨ FlightBooked ∧ HotelBooked ∧ EventBooked
  – CommutingSettled ∨ ShuttleBooked ∨ LimoBooked ∨ TaxiAvailabilityChecked
  – ...

• Client Service Request:
  – Starting from: ¬FlightBooked ∧ ¬HotelBooked ∧ ¬EventBooked ∧ ¬CommutingSettled
  – Achieve: TravelSettledUp ∧ CommutingSettled
Example (3)

- Component Services
  - $S_1$: True $\rightarrow$ \{S$_1$:bookFlight\} FlightBooked $\land$ MayBookLimo
    MayBookLimo $\rightarrow$ \{S$_1$:bookLimo\} LimoBooked
  - $S_2$: True $\rightarrow$ \{S$_2$:bookHotel\} HotelBooked
    HotelBooked $\rightarrow$ \{S$_2$:bookShuttle\} ShuttleBooked
  - $S_3$: True $\rightarrow$ \{S$_3$:bookEvent\} EventBooked

- Ontology:
  - TravelSettledUp $\land$ FlightBooked $\land$ HotelBooked $\land$ EventBooked
  - CommutingSettled $\lor$ ShuttleBooked $\lor$ LimoBooked $\lor$ TaxiAvailabilityChecked
  - ...

- Client Service Request:
  Starting from:
  - $\neg$FlightBooked $\land$ $\neg$HotelBooked $\land$ $\neg$EventBooked $\land$ $\neg$CommutingSettled
  achieve:
  - TravelSettledUp $\land$ CommutingSettled

- Compositions:
Another Example (1)

- **Component Services:**
  - $S_1$: Registered $\rightarrow \{S_1:bookFlight\}$ FlightBooked
    $\neg$Registered $\rightarrow \{S_1:register\}$ Registered
  - $S_2$: True $\rightarrow \{S_2:bookHotel\}$ HotelBooked
    HotelBooked $\rightarrow \{S_2:bookShuttle\}$ ShuttleBooked
  - $S_3$: True $\rightarrow \{S_3:bookEvent\}$ EventBooked

- **Ontology:**
  - TravelSettedUp $\wedge$ FlightBooked $\wedge$ HotelBooked $\wedge$ EventBooked

- **Client Service Request:**
  **Starting from:**
  $\neg$FlightBooked $\wedge$ $\neg$HotelBooked $\wedge$ $\neg$EventBooked
  **Achieve:**
  TravelSettedUp
Another Example (2)

Client Service Request:
- Starting from: \( \neg \text{FlightBooked} \land \neg \text{HotelBooked} \land \neg \text{EventBooked} \)
- Achieve: \( \text{TravelSettedUp} \)

*What about Registered?*

The client does not know whether he/she/it is registered or not.

The composition must resolve this at runtime:

```java
if (\( \neg \text{Registered} \)) {
    S_1: \text{register};
}
S_1: \text{bookFlight};
S_2: \text{bookHotel};
S_3: \text{bookEvent}
```
Composition as Planning

Composition as (classical) planning

Statics of the system

Dynamics of target service

Dynamics of component services
Planning is a Rich Area!!!

- Sequential Planning (plans are sequences of actions)
- Conditional Planning (plans are programs with if’s and while’s)
- Conformant Planning (plans the work in spite of incomplete -non observable- information)
- Knowledge Producing Actions/Sensing (distinction between truth and knowledge)
- Plan Monitoring
- Interleaving Deliberation and Execution
- Form of the Goals:
  - Achieve something
  - Achieve something while keeping something else
  - Temporal goals
  - Main goal + exception handling
• available services:
  – non-deterministic transition systems characterized by a set of initial states and by a transition relation that defines how the execution of each action leads from one state to a set of states
  – among such services, one represents the client

• client request (called global goal):
  – it specifies a main execution to follow, plus some side paths that are typically used to resolve exceptional circumstances e.g., \( \text{Do } \Phi \text{ else } \text{Try } \Psi \)
• service composition problem: (extended)
planning problem
  – Input: (i) a set of services, including the one
    representing the client (behavior), and (ii) the global
    goal,
  – Output: a plan that specifies how to coordinate the
    execution of various services in order to realize the
    global goal.

• NOTE:
  – the composition is not tailored towards satisfying
    completely the client requested behavior, but concerns
    with the global behavior of the system in which some
    client desired executions may happen not to be
    fulfilled
Traverso’s Group

Statics of the system

Dynamics of component services

Dynamics of target service

Composition as (classical) planning

Knoblock’s group

Traverso’s group
McIlraith’s Group

• both available and composite service: behavioral description seen as procedures invokable by clients
  – Golog procedure, atomically executed, i.e., seen by its client as an atomic Situation Calculus action, presenting an I/O interface
  – each service stored in an OWL-S ontology
client request:
- skeleton of a Golog procedure expressing also client constraints and preferences

service composition problem:
- **Input:** (i) OWL-S ontology of services as atomic actions, and (ii) client request
- **Output:** Golog procedure obtained by automatically instantiating the client request with services contained in the ontology, by also taking client preferences and constraints into account

NOTE: the client has not influence over the control flow of the composite service
McIlraith’s Group

Knoblock’s group

Composition as (classical) planning

Dynamics of target service

Dynamics of component services

Traverso’s group

Statics of the system

McIlraith’s group
Hull’s Group

• both available and composite service (peer): behavioral description
  – Mealy machine, that exchanges messages with other peers according to a predefined communication topology (channels among peers)
  – peers equipped with (bounded) queue to store messages received but not yet processed
  – Conversation: sequence of messages exchanged by peers
  – At each step, a peer can either (i) send a message, or (ii) receive a message, or (iii) consume a message from the queue, or (iv) perform an empty move, by just changing state
Hull’s Group

- **Choreography mapping problem:**
  - **Input:** (i) a desired global behavior (i.e., set of desired conversations) as a Linear Temporal Logic formula, and (ii) an infrastructure (a set of channels, a set of peer names and a set of messages)
  - **Output:** Mealy machines *(automatically obtained)* for all the peers such that their conversations are compliant with the LTL specification
Hull’s Group

Composition as (classical) planning

Static of the system

Dynamics of target service

Dynamics of component services

Knoblock’s group

McIlraith’s group

Traverso’s group

Hull’s group
The Roman Group

- available service: **behavioral description**
  - service as an interactive program: at each step it presents the client with a set of actions among which to choose the next one to be executed
  - client choice depends on outcome of previously executed actions, but *the rationale behind this choice depends entirely on the client*
  - behavior modeled by a finite state transition system, each transition being labeled by a deterministic (atomic) action, seen as the abstraction of the effective input/output messages and operations offered by the service
The Roman Group

• client request (target service):
  – set of executions organized in a (finite state) *transition system* of the activities he is interested in doing

• service composition problem:
  – **Input:** *(i)* finite state transition system of available services, and *(ii)* finite state transition system of target service
  – **Output:** *(automatically obtained)* composite service that realizes the client request, such that each action of the target service is delegated to at least one available service, in accordance with the behavior of such service.

• NOTE: the client “strongly” influence the composite service control flow
The Roman Group

Knoblock's group
Composition as (classical) planning

Statics of the system

McIlraith's group

Dynamics of target service

Traverso's group

Dynamics of component services

Hull's group
The Whole Picture

Knoblock’s group

Composition as (classical) planning

McIlraith’s group

Traverso’s group

Hull’s group

The Roman group

Dynamics of component services

Dynamics of target service

Statics of the system
PART IV:
Semantic Web Service Tools & Systems

Katia Sycara
Massimo Mecella
Emilia Cimpian
Contents

- CMU OWL-S IDE
- Composer (Rome)
- WSMX
OWL-S IDE (CMU)

Integration of WS implementation, deployment, discovery, invocation and verification

Katia Sycara
WS Development and invocation

• **Web Service Development**
  – Implement Web service
  – Produce WSDL and OWL-S WS description
  – Deploy Web service
    • Advertise to available UDDI
    • Make service available for invocation

• **Web Service invocation on client side**
  – Find Web service in UDDI
  – Translate internal data representation to WS data representation
  – Invoke Web service consistently with specification of OWL-S Process Model

• **All descriptions should fit together** otherwise interaction with Web service fails
Integrated WS Development cycle

• OWL-S IDE aims at automating WS-Development and invocation cycle
  – Based on Eclipse to support WS programmers
  – (Semi) Automated generation of WSDL and OWL-S descriptions
  – Consistency checking
  – Automated publication with UDDI
  – Integrated Semantic discovery in UDDI
  – Automated generation of client code
Overview OWL-S IDE

- OWL-S Editor integrated with Eclipse
- Integrated editing of all OWL-S modules
- Automatic publication, inquiry, and capability-based discovery with Semantic UDDI

- OWL-S2UDDI Converter
- OWL-S API
- OWL-S VM
- OWL-S API provide easy processing in Java
- OWL-S VM provides an execution environment for OWL-S Web services

Embed guided generation of WSDL and schematic OWL-S directly from Java exploiting Java2WSDL and WSDL2OWL-S tools

- Profile
- Process
- Grounding
- Generated OWL-S
- Java Code
- WSDL Code
- WSDL2OWL-S Converter

Java Code

2nd European Semantic Web Conference, Heraklion, Greece, May 2005
OWL-S IDE a close look

Seamlessly transform Java into OWL-S

Coherent interface facilitate switching between views

Overview of OWL-S Service description

Detailed Process description

Profile/Process Model Overview
OWL-S IDE: a look inside

• The OWL-S IDE is based on the following components
  – **WSDL2OWL-S**
    • Map WSDL descriptions into OWL-S descriptions
  – **OWL-S API**
    • Transform OWL-S code in an equivalent set of Java classes for easy processing
  – **OWL-S Virtual Machine**
    • Control interaction with Web service consistently with Process Model and Grounding
  – **OWL-S/UDDI translator**
    • Translate OWL-S Profiles in UDDI statements
  – **Semantic UDDI**
    • Integrate UDDI Registry and OWL reasoning to facilitate discovery of Web services
Mapping WSDL to OWL-S

- Exploit relation between WSDL and OWL-S to generate (partial) OWL-S specification
  - Automatic generation of Grounding
  - Partial generation of Process Model and Profile
  - Up to 80% of work required to generate a OWL-S description is done automatically
  - Allow programmers to concentrate on the information that is really different between the two Web services descriptions

- Combined with Java2WSDL to provide Java2OWL-S
**OWL-S API**

- Parser for OWL-S Service descriptions
  - Transform OWL-S code in an equivalent set of Java classes for easy processing
  - Based on Jena OWL API
    - Jena does all the inferences
    - OWL-S descriptions are not parsed syntactically, but derived from Semantic Model
  - Complete version for OWL-S 1.1 available open source on semwebcentral.org
OWL-S/UDDI Translator

• UDDI and OWL-S Profile provide a description of Web services in two very different ways
  – UDDI stresses Business/Service/Binding representation
  – OWL-S stresses capability representation
• Mapping OWL-S representation into UDDI allows combination of
  – Use of widely accepted industry standard
  – Semantic Service Discovery supported by OWL-S
Expressing capabilities in OWL-S

• **OWL-S Profile** describes capabilities of Web services

• **Three types of representations:**
  1. **Functional representation**
     - Input/Output specify the information transformation produced by the Web service
     - Precondition/Effect specify the domain transformation produced by the Web service
  2. **Non-functional properties**
  3. **Type of service and product information**

• **Many capability matching algorithms have been proposed, here we discuss three.**
Provenance information in the OWL-S Profile is has a direct mapping to UDDI Business Entity.

Service Specification and features of the service can be added through the specification of a set of OWL-S specific TModels.
Semantic UDDI

Integrating OWL-S and UDDI

CMU UDDI is publicly available at
www.daml.ri.cmu.edu/matchmaker
or on SemWebCentral
www.semwebcentral.org

A variant of the CMU UDDI is in use at the NTT UDDI Business Registry (The main public UDDI in Japan) (see Kawamura et al 2003, 2004)

- CMU OWL-S Matching engine has been integrated within UDDI server
- CMU UDDI server provides
  - Normal UDDI Publish/Inquiry ports
  - Complete interoperability with any UDDI Client
- Capability Port provides OWL-S based capability requests
  (see Srinivasan et al 2004)
Matching Capabilities

- Matching of I/O of the request with I/O of the advertisement
- Five degrees of match
  - Exact
  - PlugIn $R \subseteq A$
  - Subsumed $A \subseteq R$
  - Intersection $\neg (A \cap R \subseteq \bot)$
  - Fail when disjoint $A \cap R \subseteq \bot$
- **Efficient implementation** given correct indexing of advertisements
  - Match within ms
  - Linear complexity on the size of the query
- **Current work** aims at matching preconditions/effects service and product types and service parameters
OWL-S Virtual Machine

- OWL-S VM a generic processor for the OWL-S Process Model
  - It can interact with any OWL-S Web service
  - Based on the Process Model formal semantics (Ankolekar et al. 2002)
  - Exploits Web services technology such as Axis and WSIF

- Provide generic client code for Web services described using OWL-S
Architecture OWL-S VM

Web Services

- SOAP
- WSDL

Web service Invocation

OWL-S Processor
- Grounding Processor
- Process Model Processor
- OWL Theorem Prover

OWL KB

Knowledge base of the client

Decision System

OWL-S Service description

make non-deterministic decisions, and provides data to OWL-S Processor
Beyond OWL-S IDE

• OWL-S IDE provides the bases for OWL-S development
  – Support WS implementation
  – Allow automatic client generation and WS invocation
  – Integrate WS discovery in the WS implementation

• OWL-S IDE has been used to experiment with
  – Security in OWL-S
  – WS Composition in OWL-S
  – Mediation in OWL-S
Security and Policies

- No standard OWL-S representation for Security and Policies has been published yet
  - But experimentation already underway
  - Adoption of a solution will depend on WS security standards

- Security Experiments with
  - representing security capability/requirements for discovery
  - Representing security information in Process Model.
    (See Denker et al 2003)

- Policies:
  - Experiments combining OWL-S and Rei
  - Rei statements included in Process Model to constrain the use of a Web service (see Kagal 2004)

- Recent work on Formal Verification of OWL-S Process Models provides a way to certify adherence to a policy
  (see Ankolekar et al “Spinning the OWL-S Process Model” In Semantic Web Services Workshop at ISWC ’04)
Composition with OWL-S

CMU Composition Architecture

- It integrates discovery and composition
  - OWL-S/UDDI Matchmaker for discovery
  - Retsina planner to control the agent
    - Interleaving of planning and execution to allow communication while planning
  - OWL Reasoner
  - OWL-S Virtual Machine to communicate with other Web Services
- Used in a number of applications: travel domain, supply chain management
- Connection with autonomous agent technology

in collaboration with TOSHIBA
Composition Example

• Task:
  – Organize trip to Conference
  – Use DAML-S VM for interaction

✓ Calendar Agent schedule trip
✓ PI Meeting Web Service provides information about location dates etc
✓ Checks on calendar stored in MS Outlook
✓ Airline and Rental car provide are information sources
✓ DAML-S Matchmaker uses UDDI
Mediation with OWL-S

- OWL-S is orthogonal to mediation
  - Mediators are architecture components
  - OWL-S is a language for the description of Web services
    - It works with any architecture that supports ontology specification
- To the extent that WSMO mediators are Web services, they can be described in OWL-S.

(See Paolucci et al. “Expressing WSMO Mediators in OWL-S” In Semantic Web Services Workshop at ISWC ’04)
Mediation with OWL-S (2)

- General schema to represent WSMO mediators:
  - any xy-mediator is represented by a Web service that takes input x and reports output y
- ...but the mediation is more complex than asserting the need for mappings
  - Discovery maps advertisements and requests
  - Planning systems to reconcile discrepancies between Web services
  - Data type Mapping rules are used in the OWL-S Groundings
- OWL-S assumes all these technologies for interoperation and mediation
Conclusion: How OWL-S Addresses WS problems

- **Discovery**
  - Provide formal representation of capabilities of WSs
  - Many different types of inferences possible to find Web services using OWL/OWL-S

- **Composition**
  - Support formal representation of WS Process Model of Web services
  - Process Model can be integrated into Planning systems for automatic composition

- **Invocation**
  - Support any type of WS invocation mechanism
  - Clear separation between WS description and implementation

- **Guaranteeing Security and Policies**
  - No explicit policy and security specification yet
  - Proposed solution will interoperate with WS security standards

- **Mediation and Interoperation**
  - Mediation services can be directly described
  - Interoperation allowed by ontology-based description of WS descriptions and data

- **The solutions are envisioned maintaining a strong relation with existing WS standards**
ESC

A Tool for Synthesizing Composite Web Services in the Roman Framework

Massimo Mecella

based on joint work with Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini
Basics on Services

- A Service is an interactive program that expose its behavior in terms of an abstract description.
- A client selects and interacts with it according to the description exposed.

Client

These are your options

This is my choice

Service
Behavior Exposed by a Service

- Behavior compactly modeled by a finite state transition system (TS): tree of all possible sequences of actions
- Each transition labeled by a deterministic (atomic) action
- Data produced by actions not explicitly modeled
  - data used by the client to interact with the service

Do you want to search by author or by title?

search by author, please

search by author

search by title

listen
Community of Services

- A community of services is
  - a finite set of services …
  - … that share implicitly a common understanding on a common (finite) set of actions …
  - … and expose their behavior using this common set of actions

- A client specifies his request as desired service behavior (Target Service) expressed as TS, using the common set of actions of the community
Example of composition

- Community of services
  *(expressed as TS)*

- Target service
  *(again expressed as TS)*

\[ \begin{align*}
  a & : \text{"search by author (and select)"} \\
  b & : \text{"search by title (and select)"} \\
  r & : \text{"listen (the selected song)"}
\end{align*} \]
Example of composition

Target Service

Service $S_0$

$\node{a} \rightarrow b \rightarrow \node{a}$

Composition schema

requests

Client A

request: $a$

delegation:

Service $S_1$

$\node{a} \rightarrow r$

Service $S_2$

$b \rightarrow \node{b} \rightarrow r$

$a$: "search by author (and select)"

$b$: "search by title (and select)"

$r$: "listen (the selected song)"
Example of composition

**Target Service**

- **Service S₀**
  - Request: a
  - Delegation: r

**Composition schema**

- **Composition**
  - Request: a
  - Delegation: r

**Client A**

- Request: a
- Delegation: r

**Service S₁**

- Request: a
- Delegation: r

**Service S₂**

- Request: b
- Delegation: r

- **a**: "search by author (and select)"
- **b**: "search by title (and select)"
- **r**: "listen (the selected song)"
Example of composition

Target Service

Composition schema

Service S₁

Service S₂

Client A

request: a r

delegation: S₁ S₁

a: "search by author (and select)"
b: "search by title (and select)"
r: "listen (the selected song)"
Example of composition

Target Service

Service $S_1$

Service $S_2$

Composition schema

Client A

Client B

request: $a$

delegation: $S_1$

request: $b$

delegation: $S_1$

request: $r$

$S_0$

requests

$a$: "search by author (and select)"

$b$: "search by title (and select)"

$r$: "listen (the selected song)"
Example of composition

Target Service

Service $S_0$

Composition schema

requests

Service $S_1$

Service $S_2$

Client A

request: $a$

delegation: $r$

Client B

request: $b$

delegation: $r$

$a$: "search by author (and select)"

$b$: "search by title (and select)"

$r$: "listen (the selected song)"
Example of composition

Target Service

Composition schema

Service S₁

Service S₂

requests

Client A

request:

delegation:

Client B

request:

delegation:

a: "search by author (and select)"
b: "search by title (and select)"
r: "listen (the selected song)"
Example of composition

**Goal**: to automatically synthesize Composition schema represented as

Note: we cannot label the Target Service directly
Algorithms for Composition: Key Ideas

- We exploit Propositional Dynamic Logic (PDL)

- Interesting properties of PDL:
  - EXPTIME decidability
  - Tree model property
  - Small model property

We can automatically build a composition schema
We can automatically build a finite state composition schema
\[ \Phi = \text{Init} \land ([u]\Phi_0 \land \bigwedge_{i=1,...,n} [u]\Phi_i \land [u]\Phi_{\text{aux}}) \]

- Initial states of all services
- PDL encoding of Target service
- PDL encoding of \( i \)-th component service
- PDL additional domain-independent conditions

PDL encoding is polynomial in the size of the service TSs
Results on Automatically Building Service Composition

PDL formula encoding the composition problem

Φ

Check satisfiability (and build a model)

composition schema

EXPTIME
Results

Thm 1: Composition schema exists iff PDL formula $\Phi$ is satisfiable

From composition labeling of the target Service one can build a tree model for the formula, and vice-versa

Cor 1: Composition schema existence of services, expressible as TS, is decidable in EXPTIME

$\Rightarrow$ A model of the PDL formula is a composition schema
Results

Thm 2: If composition schema exists then composition schema which is a TS exists.

From a small model of the PDL formula $\Phi$, one can build a composition schema which is a TS.

Cor 2: Finite state composition existence, when services are expressible as TSs, is decidable in EXPTIME.

$\Rightarrow$ A finite model of the PDL formula is a composition schema which is a TS.
Results

Thm 3: Our technique for Service composition is sound, complete and EXPTIME-terminating

⇒ We can automatically build a composition schema which is a finite state transition system in EXPTIME

From Thm 1 and Thm 2
Extensions to the Roman Framework

• Loose specification of target service:
  – target service “under-specified”
    • how to automatically compute composition?
      • how to extend ESC to this enhanced framework? ongoing work

Done: [Berardi-etal ICSOC04]

• Incomplete specification of Services of the community:
  – services export partial description of their behavior to the community
    • how to automatically compute composition?
      ongoing work (very hard!)

• Enriching the language for describing services:
  – Adding Data “in a smart way” (joint work with Rick Hull)
  – lower level of abstraction
  – new problems, e.g. how to deal with intrinsic nondeterminism?
    • Idea: exploiting database theory
      • how to automatically compute composition

First Results!
E-Service Composition (ESC) Tool

• The ESC Tool implements our automatic composition algorithm
• It is available as an open source project at: http://sourceforge.net/projects/paride/
• On this site we intend to release the various prototypes produced by our research.
• ESC developed in Java
Architecture of the ESC Tool

- Abstraction Module
  - service TSs
  - WS-DL + behavioral descriptions of services of the community

- Synthesis Engine
  - (PDL/DL SAT + FSM minimizer)

- Realization Module
  - TS of composition
  - WS-DL + behavioral descriptions of the target service
  - WS-BPEL specification of the composition schema to be enacted by the Orchestrator
• TS_2_ALC: Input: TS
  Output: ALC knowledge base

• ALC Tableau Algorithm: Input: ALC knowledge base
  Output: if ALC knowledge base satisfiable
   then return a model which is a composition schema (repr. as TS)
   else fail

• TS Minimizer: Input: TS
  Output: TS minimized
Our Algorithm for Service Composition

INPUT: $S_0$ /* TS of client specification */
$S_1..S_n$ /* TSs of Services in the Community C */

OUTPUT: if a composition of $S_0$ wrt $S_1..S_n$ exists
then return $T_S$ of composition schema
else return nil

begin

$\Phi = \text{TS}_2\text{-}\text{ALC}(S_0,S_1,..,S_n)$ /* encode client spec. and services of C into a PDL formula $\Phi$ */

$l_f = \text{ALC}\text{-}\text{Tableau}(\Phi)$ /* compute a finite model $l_f$ for $\Phi$ */

if ($l_f == \text{nil}$) /* if $l_f$ does not exist, i.e., no composition exists*/
then return nil
else /* else $l_f$ exist */

$S_c = \text{Extract}\_\text{TS}(l_f)$ /* extract a TS from $l_f$ */

$T_S = \text{Minimize}(S_c)$ /* minimize it */

return $T_S$ /* return it */

end
ESC Tool at work

target service

Service community

search_by_title

search_by_author

listen

search_by_title

search_by_author

listen

search_by_title

search_by_author

listen

ESC Synthesis Engine

Service community
Creating the initial context

Reading file "community.txt"...
...end reading file.
Reading target e-Service...
...End reading target e-Service.
Start reading e-Service belonging to the community...
...Reading e-Service: eservice1.
...Reading e-Service: eservice2.
...End reading community.
Start creating the System Constraints...
see file "kb.txt" for the System Constraints
see file "init.txt" for the initial state
see file "actions.xml" for the link between actions and input/output message
see file "moved.xml" for the link between moved and e-Service
...end creating System Constraints.

Starting the Tableau Algorithm

Reading the initial context...
...end reading initial context
Search for a solution...
...found solution<see file "FsmToMinimize.xml">

Starting the minimization phase

Reading fsm to minimize...
...end reading
init minimization...
...end minimization<see file "FsmMinimized.xml">
Press any key to continue...
ESC Tool at work

target service

search_by_title

S₀

S₀₀

search_by_author

listen

S₀₁

S₁

S₁₀

search_by_author

listen

S₁₁

S₂₁

search_by_title

listen

S₂

S₂₀

S₂₁

ESCESC Synthesis Engine

S₀

S₁

S₂

(look, S₁)

(look, S₂)

(search_by_author, S₁)

(search_by_title, S₂)

t₀

t₁

t₂
Abstracting over Technologies: Abstraction & Realization Modules
Modeling Services as Transition Systems
Intuition

- Abstraction Module:
  - WS-DL
  - OWL-S

- Realization Module:
  - TS --> WS-BPEL

see, e.g., [Pistore & Traverso ISWC04]
Transition Systems

A transition system (TS)
is a tuple
\( T = < A, S, S^0, \delta, F > \)
where:
- \( A \) is the set of actions
- \( S \) is the set of states
- \( S^0 \subseteq S \) is the set of initial states
- \( \delta \subseteq S \times A \times S \) is the transition relation
- \( F \subseteq S \) is the set of final states
Process Algebras and TSs

• Process theory:
  – a process is a term of an algebraic language
  – a transition $E \rightarrow_a F$ means that process $E$ may become $F$ by performing (participating in, or accepting) action $a$
  – structured rules guide the derivation

• A graph:
  – nodes are process terms
  – labelled directed arcs between nodes

Ven = $2p.2p\text{Inserted} + 1p.1p\text{Inserted}$

$2p\text{Inserted} = \text{big}.\text{Choice}_R$

$1p\text{Inserted} = \text{little}.\text{Choice}_L$

$\text{Choice}_B = \text{collect}_B.\text{Ven}$

$\text{Choice}_L = \text{collect}_L.\text{Ven}$
Web Service Definition Language (WS-DL)

- WS-DL (v2.0) provides a framework for defining
  - Interface: operations and input/output formal parameters
  - Access specification: protocol bindings (e.g., SOAP)
  - Endpoint: the location of service
Message Exchange Patterns (1)

- **in-only (no faults)**
  - Client
  - Service

- **robust in-only (message triggers fault)**
  - Client
  - Service

- **out-only (no faults)**
  - Client
  - Service

- **robust out-only (message triggers fault)**
  - Client
  - Service
Message Exchange Patterns (2)

**in-out (fault replaces message)**

1. Client → Service: input
2. Service → Client: output
3. Client → Service: output (2') fault

**out-in (fault replaces message)**

1. Service → Client: input
2. Client → Service: output
3. Client → Service: output (2') fault

**in-optional-out (message triggers fault)**

1. Client → Service: input
2. Service → Client: output
3. Client → Service: output (2') fault

**out-optional-in (message triggers fault)**

1. Service → Client: input
2. Client → Service: output
3. Client → Service: output (2') fault
WS-DL is the Set of Actions

• A message exchange pattern (and the related operation) represents an interaction with the service client
  – an action that the service can perform by interacting with its client
• Abstracting from formal parameters, we can associate a different symbol to each operation …
• … thus obtaining the alphabet of actions
An Example (1)

- The MP3ServiceInterface defines 3 actions:
  - search_by_title / st
  - search_by_author / sa
  - listen / l

- Formally A = \{st,sa,l\}
Definition of a message and its formal parameter

```xml
<definitions ...
  <types>
    <element name="ListOfSong_Type">
      <complexType><sequence>
        <element minOccurs="0" maxOccurs="unbound"
          name="SongTitle" type="xs:string"/>
      </sequence></complexType>
    </element>
    <element name="SearchByTitleRequest">
      <complexType><all>
        <element name="containedInTitle"
          type="xs:string"/>
      </all></complexType>
    </element>
    <element name="SearchByTitleResponse">
      <complexType><all>
        <element name="matchingSongs"
          xsi:type="ListOfSong_Type"/>
      </all></complexType>
    </element>
  </types>
</definitions>
```
An Example (3)

```xml
<element name="SearchByAuthorRequest">
  <complexType><all>
    <element name="authorName" type="xs:string"/>
  </all></complexType>
</element>

<element name="SearchByAuthorResponse">
  <complexType><all>
    <element name="matchingSongs" xsi:type="ListOfSong_Type"/>
  </all></complexType>
</element>

<element name="ListenRequest">
  <complexType><all>
    <element name="selectedSong" type="xs:string"/>
  </all></complexType>
</element>
```
<element name="ListenResponse">
  <complexType><all>
    <element name="MP3fileURL" type="xs:string"/>
  </all></complexType>
</element>

<element name="ErrorMessage">
  <complexType><all>
    <element name="cause" type="xs:string"/>
  </all></complexType>
</element>

</types>
An Example (5)

```xml
<interface name="MP3ServiceType">
  <operation name="search_by_title" pattern="in-out">
    <input message="SearchByTitleRequest"/>
    <output message="SearchByTitleResponse"/>
    <outfault message="ErrorMessage"/>
  </operation>

  <operation name="search_by_author" pattern="in-out">
    <input message="SearchByAuthorRequest"/>
    <output message="SearchByAuthorResponse"/>
    <outfault message="ErrorMessage"/>
  </operation>

  <operation name="listen" pattern="in-out">
    <input message="ListenRequest"/>
    <output message="ListenResponse"/>
    <outfault message="ErrorMessage"/>
  </operation>
</interface>
</definitions>
```
The ESC Abstraction Module

- **Input:** Services in the repository and the target Service
- **Output:** for each Service, its corresponding FMS

- Behavioral descriptions can be expressed in any language that allows to express a finite state machine
- Behavioral descriptions can be expressed in OWL-S (to be refined)
Intuition

- Abstraction Module:
  - WS-DL
  - OWL-S

- Realization Module:
  - TS --> WS-BPEL
Business Process Execution Language for Web Services (WS-BPEL)

- Allows specification of composition schemas of Web Services
  - Business processes as coordinated interactions of Web Services
  - Business processes as Web Services
- Allows abstract and executable processes
- Influenced from
  - Traditional flow models
  - Structured programming
  - Successor of WSFL and XLANG
- Component Web Services described in WS-DL (v1.1)
WS-BPEL Specification

An XML document specifying

• Roles exchanging messages with the composite service/process
• The (WSDL) interfaces supported by such roles

• The orchestration of the process
  – Variables and data transfer
  – Exception handling
  – Correlation information

Orchestration
- variables and data transfers,
- exception handling,
- correlation information (for instance routing)

Variables:
warehouse: URI
inStock, shippingAvail: bool
customer: String
...

1. receive orderGoods
2. invoke checkLocalStock
   - invoke checkShipAvailable
   - local service offered by the supplier
   - supplier
3. invoke cancelOrder
4. invoke confirmOrder

roles
interfaces

customer
warehouse
Process Model
(Activities)

• Primitive
  - `invoke`: to invoke a Web Service (in-out) operation
  - `receive`: to wait for a message from an external source
  - `reply`: to reply to an external source message
  - `wait`: to remain idle for a given time period
  - `assign`: to copy data from one variable to another
  - `throw`: to raise exception errors
  - `empty`: to do nothing

• Structured
  - `sequence`: sequential order
  - `switch`: conditional routing
  - `while`: loop iteration
  - `pick`: choices based on events
  - `flow`: concurrent execution (synchronized by `links`)
  - `scope`: to group activities to be treated “transactionally” (managed by the same fault handler, within the same transactional context)
Process Model
(Data Manipulation and Exception Handling)

• Blackboard approach
  – a blackboard of variables is associated to each orchestration instance (i.e., a shared memory within an orchestration instance)
  – variables are not initialized at the beginning; they are modified (read/write) by assignments and messages
  – manipulation through XPath

• Try-catch-throw approach
  – definition of fault handlers
  – … but also event handlers and compensation handlers (for managing transactionality as in the SAGA model)
From a TS to WS-BPEL (1)

Transition System

Mapping transitions

Transition Skeletons

Mapping states

State Skeletons

Connecting state skeletons on the basis of the graph

WS-BPEL Specification Skeleton

<process name = "...">

<partnerLinks>
...
</partnerLinks>

<variables>
...
</variables>

<flow>

<links>
...
</links>

<!-- state skel. -->
...
</flow>

</process>
From a TS to WS-BPEL (2)

Intuition [Baina etal CAISE04, Berardi etal VLDB-TES04]

1. Each transition corresponds to a WS-BPEL pattern consisting of (i) an 
   <onMessage> operation (in order to wait for the input from the client of 
   the composite service), (ii) followed by the effective logic of the 
   transition, and then (iii) a final operation for returning the result to the 
   client. Of course both before the effective logic and before returning the 
   result, messages should be copied forth and back in appropriate 
   variables

2. All the transitions originating from the same state are collected in a 
   <pick> operation, having as many <onMessage> clauses as 
   transitions originating from the state

3. The WS-BPEL file is built visiting the graph in depth, starting from the 
   initial state and applying the previous rules.

N.B.: (1) and (2) works for in-out interactions (the one shown in the following). Simple 
modifications are needed for in-only, robust-in-only and in-optional-out. The other 
kinds of interactions implies a proactive behaviour of the composite service, possibly 
guarded by <onAlarm> blocks. 
(3) works for acyclic TS. See later for cycle management.
Transition Skeletons

<onMessage ... >
  <sequence>
    <assign>
      <copy>
        <from variable="input" ... />
        <to variable="transitionData" ... />
      </copy>
    </assign>
    <!-- logic of the transition -->
    <assign>
      <copy>
        <from variable="transitionData" ... />
        <to variable="output" ... />
      </copy>
    </assign>
    <reply ... />
  </sequence>
</onMessage>
State Skeletons

- N transitions from state $S_i$ are mapped onto:

```xml
<pick name = "$S_i$">
  <!-- transition #1 -->
  <onMessage ...
  <!-- transition skeleton -->
  </onMessage>

  .... ....

  <!-- transition #N -->
  <onMessage ...
  <!-- transition skeleton -->
  </onMessage>
</pick>
```
Mapping the TS

- All the `<pick>` blocks are enclosed in a surrounding `<flow>`; the dependencies are modeled as `<link>`s
  - `<link>`s are controlled by specific variables $S_i$-to-$S_j$ that are set to TRUE iff the transition $S_i \rightarrow S_j$ is executed
  - Each state skeleton has many outgoing `<link>`s as states connected in output, each going to the appropriate `<pick>` block
Mapping Cyclic TSs
(Intuition)

- Identify all the cycles
- Enclose the involved state skeletons inside a `<while>` block controlled by a condition (`!exit`) (`exit` is a variable defined ad hoc)
  - `exit` is set to FALSE by any transition that “goes out” of the cycle
  - The overall `<while>` block is connected to other state skeletons by appropriate `<link>`s

- Special cases:
  - A state S with self-transitions can be represented as a `<pick>` block enclosed in a `<while>` block controlled by a condition (`vs`) (the variable `vs` is set to FALSE by other non self-transitions)
  - Cycles starting from the initial state should not be considered, as they can be represented as the start of a new instance
<partnerLinks>
    <!-- The "client" role represents the requester of this composite service -->
    <partnerLink name="client"
                partnerLinkType="tns:Transition"
                myRole="MP3ServiceTypeProvider"
                partnerRole="MP3ServiceTypeRequester"/>
    <partnerLink name="service"
                partnerLinkType="nws:MP3CompositeService"
                myRole="MP3ServiceTypeRequester"
                partnerRole="MP3ServiceTypeProvider"/>
</partnerLinks>
An Example (2)

```xml
<variables>
    <variable name="input" messageType="tns:listen_request"/>
    <variable name="output" messageType="tns:listen_response"/>
    <variable name="dataIn" messageType="nws:listen_request"/>
    <variable name="dataOut" messageType="nws:listen_response"/>
</variables>

<pick>
    <onMessage partnerLink="client" portType="tns:MP3ServiceType" operation="listen" variable="input">
        <sequence>
            <assign>
                <copy>
                    <from variable="input" part="selectedSong"/>
                    <to variable="dataIn" part="selectedSong"/>
                </copy>
            </assign>
            <assign>
                <copy>
                    <from variable="dataOut" part="MP3FileURL"/>
                    <to variable="output" part="MP3FileURL"/>
                </copy>
            </assign>
            <reply name="replyOutput" partnerLink="client" portType="tns:MP3ServiceType" operation="listen" variable="output"/>
        </sequence>
    </onMessage>
    ...
    <assign>
        <copy>
            ...
        </copy>
    </assign>
    ...
</pick>
```

2nd European Semantic Web Conference, Heraklion, Greece, May 2005
<process suppressJoinFailure = "no">
  <flow>
    <links>
      <link name="start-to-1"/>
      <link name="start-to-2"/>
    </links>
  </flow>
  <pick createInstance = "yes">
    <onMessage="sa">
      <sequence>
        <copy>...</copy>
        ... ...
        <copy>...</copy>
        <reply ... />
      </sequence>
    </onMessage>
    <onMessage="st">
      <sequence>
        <copy>...</copy>
        ... ...
        <copy>...</copy>
        <reply ... />
      </sequence>
    </onMessage>
    <source linkName="start-to-1" transitionCondition = "bpws:getVariableData('start-to-1') = 'TRUE'" />
    <source linkName="start-to-2" transitionCondition = "bpws:getVariableData('start-to-2') = 'TRUE'" />
  </pick>
</process>

A new instance is created in the initial state. This resolve also the presence of the cycles without the need of enclosing <while>

The <sa> transition skeleton should set variables:
start-to-1 = TRUE
start-to-2 = FALSE

The <st> transition skeleton should set variables:
start-to-1 = FALSE
start-to-2 = TRUE
<pick>
  <onMessage="I">
      <sequence>
          <copy>...</copy>
          ...
          <copy>...</copy>
          <reply ... />
      </sequence>
  </onMessage>
  <target linkName="start-to-1" />
</pick>

<pick>
  <onMessage="I">
      <sequence>
          <copy>...</copy>
          ...
          <copy>...</copy>
          <reply ... />
      </sequence>
  </onMessage>
  <target linkName="start-to-2" />
</pick>
</process>
Web Service Execution Environment (WSMX)

Emilia Cimpian

Digital Enterprise Research Institute (DERI)
Contents

• Scope of WSMX Development

• Architecture
  – Components
    • System Architecture
  – Inter-relationship of Components
    • Execution Semantics
  – Component Interfaces
    • Data Flow between Components
Scope of WSMX Development

• reference implementation for WSMO

• complete architecture for SWS discovery, mediation, selection and invocation

• initial functionality - achieving a user-specified goal by invoking WS described with the semantic markup
Architecture Overview

• Components
  – System Architecture
• Interrelationship of components
  – Execution semantics
• Component interfaces
  – Data flow between components
System Architecture
Execution Semantics

- Invocation order of components
- Process context
- Event-based implementation
- JMX implementation
Invocation Order

1. Discover Web Services
2. Create Choreography
3. Discover Services
4. Mediate Data
5. Create Mediated Data
6. Return Mediated Data
7. Return Web Services
8. Call Invoker
9. Check Choreography
10. Confirmed
11. End
Process Context

1. Registry of known components
   - Discover Web Services
     - Discover Services
       - Mediate Data
         - Return Mediated Data
           - Return Web Services
             - Call invoker
               - Confirmed
                 - Choreography object
                   - Mediated objects, Web Services entities
               - Errors
                 - Exceptions
   - Create Choreography
     - Created
       - Data Mediator Wrapper
         - Check Choreography
           - Confirmed
             - Choreography Wrapper
               - End
Event-based Implementation

Core – Manager

“Business” Process – Internal Workflow

Event and Notification Distribution/Delivery Mechanism

- Choreography Wrapper
- Discovery Wrapper
- Data Mediator Wrapper
- Communication Manager Wrapper

Choreography
Discovery
Mediator
Communication Manager
JMX Implementation
System Architecture
System Architecture

Request to discover Web services. May be sent to adapter or adapter may extract from backend app.
System Architecture

Goal expressed in WSML sent to WSMX System Interface
Comm Manager component implements the interface to receive WSML goals.
System Architecture

Comm Manager tells core
Goal has been received
System Architecture

Choreography wrapper picks up event for Choreography component.
A new choreography instance is created.
Core is notified that choreography instance has been created.
System Architecture

Parser wrapper picks up event for Parser component
WSML goal is parsed to internal format
System Architecture
System Architecture
System Architecture

Discovery is invoked for parsed goal
System Architecture
System Architecture
System Architecture

Discovery component requires data mediation.
System Architecture
System Architecture
System Architecture

After data mediation, discovery component completes its task.
System Architecture
System Architecture

WSMX

WSMX Manager

ResourceManager Interface

Datastore

NonWSMO Object datastore

FloraXSB

WSMO Reasoner

Reasoner Interface

Resource Manager Interface

WSMO Objects Datastore

New Component Interface

New Component Wrapper

Component Wrapper

Service Requesters

Back-end application 1

Back-end application 2

Back-end application n

Agent 1 acting on behalf of user a

Agent 2 acting on behalf of user b

Agent 3 acting on behalf of user m

Adapter

Adapter 1

Adapter 2

...
After discovery, the choreography instance for goal requester is checked for next step in interaction.
System Architecture
System Architecture

Next step in choreography is to return set of discovered Web services to goal requester
System Architecture

Set of Web Service descriptions expressed in WSML sent to appropriate adapter
System Architecture

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

• Methods
• Parameter
• Examples
  – Data Mediator
  – Invoker
Data Mediator

mediateData ( 
    URIRef sourceOntologyID, 
    URIRef targetOntologyID, 
    Identifiable ontologyFragment )

- Calls the mediator and returns back the mediated fragment of ontology

• Parameters
  - sourceOntologyID - the source ontology ID
  - targetOntologyID - the target ontology ID
  - ontologyFragment - the subject of the mediation (instances from the source ontology)

• Returns
  - the mediated payload (instances from the target ontology)
Invoker

sendWSMLMessage (  
   java.lang.String wsdl,  
   java.lang.String portType,  
   java.lang.String operation,  
   WSMLMessage message  
)
Conclusions

• Event based component architecture
• Conceptual model is WSMO (with some add-ons)
• End to end functionality for executing SWS
• Has a formal execution semantics
• Real implementation
• Open source code base at SourceForge
• Event driven component architecture
• Developers welcome
PART V:

Hands-On Session

John Domingue
Liliana Cabral
Matt Moran
Michal Zaremba
Contents

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

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

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

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

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

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

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

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

IRS-3 Server

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

IRS Publisher

IRS Publisher

IRS Publisher

IRS Publisher

IRS Publisher

IRS Client

Lisp

Java

Java WS

SOAP
IRS-III Architecture

Publishing Platforms

SOAP

Browser Handler
Publisher Handler
Invocation Handler

OWL(-S) Handler

WSMXML
Browser
Publishing Clients
Invocation Client

WSMXML
Java Code
Web Application

Browser
Publishing Platforms
WSMXML
SOAP
 Publishing Platforms

OWL(-S)

LispWeb Server

Java API

Invoking
Client

Invoking
Client

Web Service

Web Service

Web Service
Publishing Platform Architecture

Publishing Platform

Architectures

IRS-III Publishing Platform

HTTP Server

SOAP Handler

Service Registrar

Service Invoker

WS Service Registry

IRS-III Server

Invocation Client

Publishing Clients

SOAP

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

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

- OWL-S Process
- OWL-S Translator
- OWL Translator
- Web Service (Mediator and Goal)

2nd European Semantic Web Conference, Heraklion, Greece, May 2005
OWL Process to Web Service

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

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

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

John Domingue and Liliana Cabral
SWS Creation & Usage Steps

• Create a goal description
  – (e.g. exchange-rate-goal)
  – Add input and output roles
  – Include role type and soap binding
• Create a wg-mediator description
  – Source = goal
  – Possibly add a mediation service
• Create a web service description
  – Used-mediator of WS capability = wg-mediator above
• Specify Operation <-> Lisp function mapping in Choreography Grounding
• Publish against web service description
• Invoke web service by ‘achieve goal’
Multiple WS for goal

• Each WS has a mediator for used-mediator slot of capability
  – Some WS may share a mediator
• Define a kappa expression for assumption slot of WS capability
• Kappa expression format
  – (kappa (?goal) <ocml relations>)
• Getting the value of an input role
  – (wsmo-role-value ?goal <role-name>)}
Defining a Mediation Service

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

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

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

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

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

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

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

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

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

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

**Invocation**
Invoke selected web service
European-exchange-rate
Hands-On Session (with IRS III)

John Domingue and Liliana Cabral
European Travel Scenario
European Travel Demo

[Image of a travel booking interface with fields for Name, Type, Departure, Arrival, Departure date, and Departure time]
IRS-III Hands On Task

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

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

• Publish available lisp functions against your descriptions.

• Invoke the web services.

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

IRS Server (3000)
- Domain Models
- Web Service WSMO Descriptions + Registry of Implementors
- Goal WSMO Descriptions + SOAP Binding
- Mediator WSMO Descriptions

Travel Services (3001)
- IRS Lisp Publisher
- IRS-III Knowledge Model Browser & Editor

WSMX
Travel Related Knowledge Models
Key Classes, Relations, Instances

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

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

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

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

• 1 service available for goal 1
  – No constraints

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

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

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

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

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

1- get-train-times

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

2- book-english-train-journey

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

3- book-french-train-journey

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

4- book-german-train-journey

christoph berlin frankfurt 3304251200

"First Class Booking German Rail (Die Bahn): CHRISTOPH is booked on the 323 going from BERLIN to FRANKFURT at 17:11, 15, SEPTEMBER 2004. The price is 35 Euros."

5- book-austrian-train-journey

sinuhe vienna innsbruck 3304251200

"Austrian Rail (OBB): SINUHE is booked on the 367 going from VIENNA to INNSBRUCK at 16:47, 15, SEPTEMBER 2004. The price is 36 Euros."
Available Functions (3/3)

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

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

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

![Image of a software interface for editing and publishing a web service. The interface includes fields for name, ontology, and parent, as well as options for inputs and output, capability, interface, and web service mediators. There is also a section for publishing a Lisp function, with fields for web service name, web service ontology, and Lisp publisher URL.]
Tips

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

Michael Stollberg
Tutorial Wrap-up

The targets of the presented tutorial were to:
  – understand aims & challenges within Semantic Web Services
  – understand OWL-S and WSMO:
    • design principles & paradigms
    • ontology elements
  • an overview of ‘hot topics’ within the Semantic Web and Semantic Web Services
  • OWL-S and WSMO Tools and System Presentation
  • do-it-yourself Hands-On Session
=> you should now be able to correctly assess emerging technologies & products for Semantic Web Services and utilize these for your future work
OWL-S and WSMO

- North-American and European initiatives with converging aims
- Offer a SWS platforms to be used by B2C and B2B applications
- Provide a backbone for advanced integration and automation of industrial and business processes
- Are the most developed SWS technologies up to now available to be used in commercial and industrial applications
- Developments towards refining and interconnecting them
Future work – OWL-S

• OWL-S is close to conclusion, but a few issues still need to be addressed
  – An exception mechanism is still missing
  – There is a need of an exec instruction for loading and executing Process Models dynamically
  – A new Grounding for WSDL 2 should be developed
• Additional issues that OWL-S does not address
  – Security and Policies are not directly expressed in OWL-S yet
  – There are no facilities for Contracting and agreement
  – There are no facilities for Web service management
Future work – OWL-S (2)

• Standardization
  – The OWL-S coalition is planning to submit a W3C note to draw attention and create momentum for W3C standardization activities on Semantic Web services
  – Members of the OWL-S coalition are already active in standardization committee such as UDDI, WSDL 2 and WS Coordination

• The Future of OWL-S
  – OWL-S is nearing its completion and it will converge in the results of the SWSI working group or future standardization activities
  – The OWL-S coalition plans to remain in existence to maintain and further develop the language if needed
Future work - WSMO

- Further develop and consolidate concepts and implementation aspects of WSMO, WSML and WSMX
  - Choreography and orchestration
  - Business process execution
  - Web services composition
  - Process and protocol mediation
- Open to new ideas, contributions and suggestions
- Standardization …
Future Work WSMO (2)

- WSMO & WSMX – applied in several case studies within EU funded projects
- WSMO Studio development
- WSMX v2 to be release in November
- IRS III new release at the beginning of 2005
Beyond OWL-S and WSMO

• Although OWL-S and WSMO are the main initiatives on Semantic Web services, they are not the only activities

• Semantic Web Services Interest Group
  – Interest group founded at W3C to discuss issues related to Semantic Web Services (http://www.w3.org/2002/ws/swsig/)

• SWSI: International initiative to push toward a standardization of SWS (http://www.swsi.org)

• Semantic Web services are entering the main stream
  – UDDI is adopting OWL for semantic search
  – WSDL 2 will contain a mapping to RDF
  – The use of semantics is also discussed in the context of standards for WS Policies
**SWSI (www.swsi.org)**

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

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

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

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

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

• Fundamental


References OWL-S

• Discovery


Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, Katia Sycara; *Importing the Semantic Web in UDDI*. In *Proceedings of Web Services, E-business and Semantic Web Workshop, 2002*

References OWL-S

• Composition and Invocation


References OWL-S

• Formal Models and Verification

Anupriya Ankolekar, Massimo Paolucci, and Katia Sycara

**Spinning the OWL-S Process Model -- Toward the Verification of the OWL-S Process Models** In Proceedings of Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications (ISWC 2004)


References OWL-S

• Policies and Security


References OWL-S

• Applications


Aabhas V Paliwal, Nabil Adam, Christof Bornhövd, and Joachim Schaper Semantic Discovery and Composition of Web Services for RFID Applications in Border Control In Proceedings of Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications (ISWC 2004)


References WSMO

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

• WSMO languages – WSML Working Group: [http://www.wsml.org](http://www.wsml.org)

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


References WSMO

References WSMO

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

- OWL -- http://www.wsmo.org/2004/d20/d20.1/
- WSML-Core -- http://www.wsmo.org/2004/d16/d16.7/
References Discovery


References Discovery


Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, Katia Sycara; *Importing the Semantic Web in UDDI*. In *Proceedings of Web Services, E-business and Semantic Web Workshop, 2002*


Composition of Information Oriented Services: References

- **Read and exploit data integration literature!**

  **Survey on data integration**


  **Seminal papers**


- **See how other service-researchers have used it!**

Composition of Services as Atomic Actions: References

• Read and exploit planning and reasoning about actions literature!

Books
Chapters on Planning and on Reasoning about Actions in any Artificial Intelligence textbook.


Interesting papers


• See how other service-researchers have used it!
  – Proceedings of P4WGS – ICAPS Workshop 2004
  – Proceedings of P4WS – ICAPS Workshop 2003
Composition of Services exposing Complex Behavior: McIlraith & her group

Background


Papers


[Narayanan&McIlraith WWW02] S. Narayanan, S. A. McIlraith: Simulation, verification and automated composition of web services. WWW 2002:


Composition of Services exposing Complex Behavior: Hull & his group

Paper analyzed


Other papers


Composition of Services exposing Complex Behavior: Traverso & his group

Papers on Planning as Model Checking


Papers on Service Composition


Composition of Services exposing Complex Behavior: the Roman Group

Papers


SWS tools: ESC


References IRS III tutorial


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


Acknowledgements

We would like to acknowledge the contribution of the past and present members of the OWL-S coalition for their hard work in the development of the language. Furthermore, we would like to thank the community at large for contributing to tools and ideas.

Furthermore, we would like to thank to all the members of the WSMO, WSML, and WSMX working groups for their advice and input into this tutorial.

Special thanks to Sheila McIlraith, Craig Schlenoff, Daniel Elenius and Naveen Srinivasan for providing slides and suggestions on this tutorial.

Slide design by Harriett Cornish, Knowledge Media Insitute, The Open University
Acknowledgements

The development of OWL-S has been funded almost exclusively by the DAML DARPA program.

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