CM40212: Internet Technology

Julian Padget
Tel: x6971, E-mail: jap@cs.bath.ac.uk

Semantic Web Services and Service Oriented Architecture

Acknowledgements: Grigoris Antoniou and Frank van Harmelen (A Semantic Web Primer)
Katia Sycara and David Martin (OWL-S tutorial)

December 13, 2011
Outline

1. Introduction
2. OWL (Antoniou & van Harmelen)
3. OWL for Services: OWL-S
4. Service Discovery
5. Service Templates
6. Service Adaptors
7. Building Semantic Services
8. Service-Oriented Architecture
9. Summary
Outline

1 Introduction

2 OWL (Antoniou & van Harmelen)

3 OWL for Services: OWL-S

4 Service Discovery

5 Service Templates

6 Service Adaptors

7 Building Semantic Services

8 Service-Oriented Architecture

9 Summary
Objectives

- To outline the high-level web ontology language
- To make the case for its extension to service description
- To relate the semantic layer to the operational layer
- To make clear this is a rapidly evolving area...
- Question is not “if” but “when” will it deliver?
- Notably, this aspect of the web contains much science and challenging problems
Outline

1. Introduction
2. OWL (Antoniou & van Harmelen)
3. OWL for Services: OWL-S
4. Service Discovery
5. Service Templates
6. Service Adaptors
7. Building Semantic Services
8. Service-Oriented Architecture
9. Summary
Requirements for ontology languages

- Ontology languages allow users to write explicit, formal conceptualizations of domain models
- The main requirements are:
  - a well-defined syntax
  - efficient reasoning support
  - a formal semantics
  - sufficient expressive power
  - convenience of expression
- Tradeoff between expressive power and efficient reasoning support:
  - The richer the language, the more inefficient reasoning becomes
  - Sometimes it reaches the limits of computability
  - We need a compromise:
    - A language supported by reasonably efficient reasoners
    - A language that can express large classes of ontologies and knowledge
Reasoning in ontology languages

- Class membership
  - If $x$ is an instance of a class $C$, and
  - $C$ is a subclass of $D$,
  - then can infer that $x$ is an instance of $D$

- Equivalence of classes
  - If class $A$ is equivalent to class $B$, and
  - class $B$ is equivalent to class $C$,
  - then $A$ is also equivalent to $C$

- Consistency
  - $X$ instance of classes $A$ and $B$,
  - but $A$ and $B$ are disjoint
  - Indicates an error in the ontology

- Classification
  - Certain property-value pairs are a sufficient condition for membership in a class $A$; if an individual $x$ satisfies such conditions, we can conclude that $x$ must be an instance of $A
Uses for Reasoning

- Reasoning support is important for:
  - checking the consistency of the ontology and the knowledge
  - checking for unintended relationships between classes
  - automatically classifying instances in classes

- Such checks are valuable for:
  - designing large ontologies, where multiple authors are involved
  - integrating and sharing ontologies from various sources

Reasoning support is important for:

- checking the consistency of the ontology and the knowledge
- checking for unintended relationships between classes
- automatically classifying instances in classes

Such checks are valuable for:

- designing large ontologies, where multiple authors are involved
- integrating and sharing ontologies from various sources
Reasoning Support for OWL

- Semantics is a prerequisite for reasoning support
- Formal semantics and reasoning support are usually provided by
  - mapping an ontology language to a known logical formalism
  - using automated reasoners that already exist for those formalisms
- OWL is (partially) mapped on to a description logic
- Can use existing reasoners: FaCT, RACER, Pellet
- Description logics are a subset of predicate logic for which efficient reasoning support is possible
Description Logics

- A DL models concepts, roles and individuals, and their relationships
- Many varieties, depending on expressivity:
  - $\mathcal{F}$: Functional properties
  - $\mathcal{S}$: An abbreviation for $\mathcal{ALC}$ with transitive roles
  - $\mathcal{H}$: Role hierarchy (subproperties - rdfs:subPropertyOf)
  - $\mathcal{N}$: Nominals. (Enumerated classes of object value restrictions - owl:oneOf, owl:hasValue)
  - $\mathcal{I}$: Inverse properties
  - $\mathcal{R}$: Limited complex role inclusion axioms; reflexivity and irreflexivity; role disjointness
  - $\mathcal{N}$: Cardinality restrictions (owl:Cardinality, owl:MaxCardinality)
  - $\mathcal{Q}$: Qualified cardinality restrictions (available in OWL 2, cardinality restrictions that have fillers other than owl:thing) Use of datatype properties, data values or data types.

http://en.wikipedia.org/wiki/Description_logic, retrieved 20101205
Three dialects of OWL I

- W3C’s Web Ontology Working Group defined OWL as three different sublanguages:
  - OWL Full
  - OWL DL
  - OWL Lite

- Each sublanguage oriented at different aspects of requirements

- OWL Full
  - Offers all the OWL language primitives
  - Allows the (arbitrary) combination of these primitives with RDF and RDF Schema
  - Fully syntactically and semantically compatible with RDF
  - OWL Full is undecidable—no complete (or efficient) reasoning support
  - Corresponds to the description logic \(SROIQ\)
Three dialects of OWL II

■ OWL DL
  ■ A sublanguage of OWL Full: restricts application of the constructors from OWL and RDF
  ■ Constructor composition is disallowed
  ■ Corresponds to the description logic $SHIOn$
  ■ Efficient reasoning support
  ■ Not fully compatible with RDF:
    ■ Not every RDF document is a legal OWL DL document
    ■ Every legal OWL DL document is a legal RDF document

■ OWL Lite: subset of the language constructors
  ■ excludes enumerated classes
  ■ disjointness statements
  ■ arbitrary cardinality
  ■ simpler to learn, to implement
  ■ restricted expressivity
  ■ Corresponds to the description logic $SHIF$
Three dialects of OWL III

- Ontologically: OWL Lite ⊂ OWL DL ⊂ OWL Full
  - Meaning: if it’s a valid ontology in the first, it’s also valid in the second
- Reasoning: $X \vdash_{OWLLite} Y \Rightarrow X \vdash_{OWLDL} Y$
  - and
    $X \vdash_{OWLDL} Y \Rightarrow X \vdash_{OWLFull} Y$
  - Meaning: if it’s a valid conclusion in the first, it’s also valid in the second
- All varieties of OWL use RDF for their syntax
- Instances declared as in RDF, using RDF descriptions
- Type information in OWL constructors are specialization of corresponding RDF values
- Downward compatibility with RDF only achieved in OWL Full
Event Ontology

- Points to note:
  - Comment/label: identify events associated with service invocation
  - Classes: Actor ⊆ { Agent, Service }
  - More classes: Event ⊆ various event classes
  - Properties: caused and causedby, in particular

- Allows semantic annotations of events

- Decouples monitoring code from actual events
Outline

1. Introduction
2. OWL (Antoniou & van Harmelen)
3. OWL for Services: OWL-S
4. Service Discovery
5. Service Templates
6. Service Adaptors
7. Building Semantic Services
8. Service-Oriented Architecture
9. Summary
OWL-S Service Ontology

- First define an upper ontology:

  - **What does service provide?** see profile: this advertises the service
  - **How is it used?** see process model: captured in ServiceModel class. Instances Service have a property describedBy that refers to the ServiceModel
  - **How to invoke it?** see grounding: transport protocol details. Service supports ServiceGrounding

  ServiceProfile is for service discovery
  ServiceModel + ServiceGrounding are for service utilization
OWL-S Profile

- Defines properties of profile to refer to IOPEs:
  - **hasParameter**: instance of Parameter from Process ontology
  - **hasInput**: instances of Input
  - **hasOutput**: instances of Output
  - **hasPrecondition**: specifies a precondition of the service and ranges over a Precondition instance
  - **hasResult**: specifies a result of the service, as defined by the Result class in the Process ontology
OWL-S Process

- **Atomic processes**: directly invocable, no sub-process, single step. Need grounding to construct messages, deconstruct responses.
- **Simple processes**: not invocable, no grounding, but single-step... abstract processes realizedby atomic process or expandsTo composite process
- **Composite processes**: decomposable into other (non-composite or composite) processes
OWL-S Composite process constructors

- Names from programming language **BUT** describes how client can behave, not what service will do
- Thus a **Sequence** composite means client sends messages to invoke each step
- **Split**: bag of processes, completes when all scheduled
- **Split + Join**: as above + barrier synchronization
- **Choice**: one process from a bag
- **Any-Order**: but not concurrently, until all completed
- **If-Then-Else**: self explanatory
- **Iterate, Repeat-While and Repeat-Until**: likewise
Outline

1. Introduction
2. OWL (Antoniou & van Harmelen)
3. OWL for Services: OWL-S
4. Service Discovery
5. Service Templates
6. Service Adaptors
7. Building Semantic Services
8. Service-Oriented Architecture
9. Summary
Service Discovery

- Want to find service by function: IOPE
- Re-use well-defined notions of semantic matching (OWL-S MX):
  1. **Exact**: S is an exact match for Q
  2. **PlugIn**: S is a plug-in match for Q, if S could be always used for Q
  3. **Subsumes**: S subsumes Q. In this case S is more general than Q and could be used under some additional constraints making it more specific. In the case of several Ss, discrimination between them could be done based on those constraints.

- Want to take account of non-OWL metadata too
- Don’t want hard-coded ranking scheme
- Want to allow users to express preference relationships

Later:
- Adaptation is inevitable because exact match rare
- Want to connect with abstract workflows
Matchmaking

- Given:
  1. A query — description of the task
  2. A set of service descriptions

- Find one or more services which:
  - Are semantically compatible with the query (strict)
  - Are best suited to the query (lax) - according to some combination of some metrics.

- Rank matching services according to some policy.

- A number of possible metrics may be used:
  - Syntactic similarity (textual correspondence)
  - Ontological distance
  - Specific service properties (e.g. QoS)
  - Domain-specific metrics
Selection policies

- Separate query into task prototype and selection policy.
- Query: specifies service semantics
- Selection policy specifies:
  - Which metrics are used to assess service/query suitability, including the semantic degree of match.
  - How those metrics are composed to define a (partial order) ranking of service choices.

“I prefer a perfect semantic match unless it has less than 50% reliability, in which case give me a plugin match which has the highest reliability and syntactic similarity.”
Selection policies

- Define a preference algebra for expressing preferences (on services and preferences).

- Preference: \( p = (A, \succeq_p) \), s.t. \( x, y \in A, x \succeq_p y \), means “\( x \) is at least as preferred as \( y \)”

- Equally preferred: \( x \succeq_p y \land y \succeq x \equiv x \approx y \)

- Strictly preferred: \( x \succeq_p y \land \neg(y \succeq_p x) \equiv x \prec_p y \)

- Define preference algebra to express set of all preferences over some set of services \( A \).
Outline

1. Introduction
2. OWL (Antoniou & van Harmelen)
3. OWL for Services: OWL-S
4. Service Discovery
5. Service Templates
6. Service Adaptors
7. Building Semantic Services
8. Service-Oriented Architecture
9. Summary
Service Templates

- May want to capture requirements as workflow without identifying particular services
- Matchmaking against single services is often insufficient
- Sometimes service invocations require workflow-like compositions to achieve specific goals. (e.g. Login, select, checkout)
- Can be specified a priori (e.g. OWL-S Composites)
- Arbitrary composition? ≡ to planning over all services—too hard
- Consider compositions specified by given templates for operation
Template example

Input 1 → P 1 → P 2 → Output 1
Input 2 → P 1 → P 3 → P 4 → Output 2

Input 1' → S2 → P 2 → S4 → Output 1'
Input 2' → S1 → S3 → S4 → Output 2'
Service Template Description

- Template Description
  - \textit{slots} : abstract (partial) service queries
  - \textit{template variables} : cf. Generics (in java)
  - \textit{constraints} : logical restrictions on which services may fulfil slots.
  - \textit{abstract process model} : Outward facing template semantics
  - \textit{abstract workflow} : inward facing operational semantics

- Select candidate template(s) for a given query,
- Template is \textit{grounded} by matching services (or templates) to slots (given constraints)
- This grounds template variables to concrete types
- If abstract process model matches and constraints are met, then template is a \textit{match} for a given query.
Service templates (cont.)

- Once a template is matched, it is simply a composite service or deterministic workflow (OWL-S)
- Template matching integrated into service matchmaker
  - Simple and template matches
  - Adjust metrics for template properties (e.g. aggregate QoS, template depth, prefer simple matches to template ones, etc.)

- Templates matching still combinatoric, but less so than general matching (best effort in given time).
- Produce a library of “standard” template descriptions (and let users define new ones)
- Templates can assist in adaptation (retained partial matches)
Outline

1 Introduction
2 OWL (Antoniou & van Harmelen)
3 OWL for Services: OWL-S
4 Service Discovery
5 Service Templates
6 Service Adaptors
7 Building Semantic Services
8 Service-Oriented Architecture
9 Summary
Service Adaptors

- **Different message signatures:** names and/or types in service and query that are in fact compatible, e.g. most popular service ← top-10 popular services

- **Unit converters:** inputs or outputs need conversion, e.g. Fahrenheit ↔ Celsius

- **Split data flows:** A composite datum needs to be split up into several components, e.g. time → hh:mm:ss

- **Merge data flows:** And vice-versa, hh:mm:ss → time

- **Missing service data:** E.g. an object unrelated to the query, such as a credential or data that is redundant vis-à-vis the query

- **Ontology alignment:** different terms for same/similar concepts, e.g. tablet > iPad
Data mediation

- Candidate service may not match because parameter types are incompatible.
- However types may be adaptable for a given query (e.g. cm $\mapsto$ inch, $w \times h \mapsto area$). Making service compatible.
- An adapted service consists of:
  - **An underlying service**: the subject of the adaptation
  - **An outer service description**: This is the facade service presented to the caller
  - **Input adaptations**: Arbitrarily long chains of adaptors from outer service to underlying service
  - **Output adaptations**: Likewise, from underlying service to outer service
Adaptation

Introduction

OWL (Antoniou & van Harmelen)

OWL for Services: OWL-S

Service Discovery

Service Templates

Service Adaptors

Building Semantic Services

Service-Oriented Architecture

Summary
Example adaptation

The requested signature

\[ \text{IN1} \times \text{IN2} \times \text{IN3} \rightarrow \text{OUT1} \times \text{OUT2} \]

is matched by a service with the signature

\[ A \times \text{IN2} \times C \rightarrow D \times E \]

because of the depicted adaptations (services) applied to inputs and outputs as a result of the adaptations (above) applied.
Outline

1. Introduction
2. OWL (Antoniou & van Harmelen)
3. OWL for Services: OWL-S
4. Service Discovery
5. Service Templates
6. Service Adaptors
7. Building Semantic Services
8. Service-Oriented Architecture
9. Summary
OWL-S Builder

- Enables the generation of semantic service descriptions through annotation of existing Java web services
- Patterned after JAX-WS
- OWL-S builder permits semantic annotations in parallel with JAX-WS
- Designed for minimal information from programmer: defaults inferred from properties of Java classes
- Need to specify ontologies to use
- Write annotations for OWL-S input and output parameters
- Run OWL-S Builder:
  1. Build JAX-WS model
  2. Build OWL-S model
  3. Build web service grounding linking OWL-S to WSDL
- Can annotate existing WSDL services to add semantic descriptions
JAX-WS example

```java
package endpoint;

import javax.ws.rs.WebService;
import javax.ws.rs.WebMethod;

@WebService(
    name="Calculator",
    serviceName="CalculatorService",
    targetNamespace="http://techtip.com/jaxws/sample"
)
public class Calculator {

    public Calculator() {
    }

    @WebMethod(operationName="add", action="urn:Add")
    public int add(int i, int j) {
        int k = i + j;
        System.out.println(i + " + " + j + " = " + k);

        return k;
    }
}
```
OWL-S Builder process

1. Build domain ontology
2. Annotate classes
3. Parse annotations
4. Generate Java grounding
5. Generate core OWL-S model
6. Generate XSLT grounding

- OWL-S Semantic Annotations
- Bean Ontology Bindings
- Domain ontologies (OWL)
- OWL-S core descriptions (service, profile, process)
- OWL-S Java grounding
- OWL-S WSDL grounding

Existing java service interface. (e.g. JAX-WS SEI)

OWL-S Semantic Annotations

OWL-S for Services: OWL-S

Summary
OWL-S Builder example

```java
@WebService(targetNamespace = "http://exampleNamespace/")
@URINamespace(prefix = "num", value = "http://numbers.org/Numbers.owl")
@OwlsImport( "&num;"")
@OwlsClass(owlsLocation = "http://numsvc.org/NumberServices.owl", defaultOntology = "&num;#")
public class SimpleAddService {
    @OwlsService(name = "AddService", label = "Addition_Service")
    @OwlsOutParam(name = "#rv")
    @OwlsLocals( { @OwlsLocal(name = "valX", type = "&xsd;#int"),
                   @OwlsLocal(name = "valY", type = "&xsd;#int") })
    @OwlsPreconditions("swrlb:greaterThanOrEqual(#valX,0), swrlb:greaterThanOrEqual(#valY,0), "
                       + "num:hasValue(#x,#valX), num:hasValue(#y,#valY)"")
    @OwlsResult(effect = "swrlb:isa(#rv,&num;#EvenNumber)",
                 condition = "swrlb:mod(0,#sum,2), swrlb:add(#sum,#valX,#valY)",
                 vars = { @OwlsResultVar(name = "#sum", type = "&xsd:int")})
    @WebMethod
    public ANumber add(  
        @WebParam(name = "x")
        @OwlsInParam(name = "#x", owlType = "#Integer",
                      bindings = { @OwlsBinding(from = "value", to = "#hasValue") })
        ANumber x,
        @WebParam(name = "y")
        @OwlsInParam(name = "#y", owlType = "#Integer")
        ANumber y) {...
```
Outline

1. Introduction
2. OWL (Antoniou & van Harmelen)
3. OWL for Services: OWL-S
4. Service Discovery
5. Service Templates
6. Service Adaptors
7. Building Semantic Services
8. Service-Oriented Architecture
9. Summary
Service Oriented Architecture

- Resources separated and identified in a consistent manner
-universal model for automation, business, etc.

Principles of service orientation:

1. **Reusability**: divide logic into services for reuse
2. **Contract**: communication defined by service descriptions
3. **Loose coupling**: minimal dependency
4. **Abstraction**: service contract information only
5. **Composition**: ⇒ coordination
6. **Autonomy**: services encapsulate their logic
7. **Stateless**: minimization of information retention
8. **Discoverability**: descriptions ⇒ discovery
‘X’ ... as a Service

- Cloud computing forms:
  - Infrastructure as a Service (IaaS)
  - Platform as a Service (PaaS)
  - Software as a Service (SaaS)

- Delivery of function on “pay as you go” basis

- Technical challenges: security, trust, privacy, lack of standardization, (⇝ supplier lock-in)

- Examples:
  - Google AppEngine — PaaS
  - Amazon EC2 + Amazon Simple Storage Service (Amazon S3) — IaaS
  - Amazon SimpleDB + Amazon Simple Queue Service (Amazon SQS) — PaaS
  - Windows Azure — PaaS
  - Your web service running on AWS — SaaS (?)
Outline

1. Introduction
2. OWL (Antoniou & van Harmelen)
3. OWL for Services: OWL-S
4. Service Discovery
5. Service Templates
6. Service Adaptors
7. Building Semantic Services
8. Service-Oriented Architecture
9. Summary
Summary

- OWL provides a high level language both for defining ontologies and for efficient reasoning about semantic descriptions.
- OWL-S extends OWL to provide a semantic layer for service description (IOPE) to complement the operational layer (WSDL).
- These are the base tools: need higher level structures
  - service oriented architecture
  - virtual organizations
  - dynamic and adaptive workflows
Resources I

“Never believe one source”

- http://www.w3.org/Submission/OWL-S
- Google AppEngine: http://code.google.com/appengine/
- EU ALIVE project: http://www.ist-alive.eu
- JAX-WS: http://jax-ws.java.net/
A useful source of information is [Antoniou and van Harmelen, 2008].