Business Process Modelling & Semantic Web Services

Charlie Abela
Department of Artificial Intelligence
charlie.abela@um.edu.mt
Last Lecture

- Web services
- SOA
- Problems?
Lecture Outline

- Business Process Modelling
  - BPEL
- Semantic Web Services
  - OWL-S
Introduction

- Web services are increasingly utilized by organizations, to improve responsiveness and efficiency
  - Need to integrate them as part of business processes
- Current approaches for process modelling are based on workflow abstractions
  - UML provides graphical constructs that can be used to describe actions and activities, and temporal precedence and control flows.
  - BPEL4WS enables specification of processes
  - ebXML also considers the life cycle of processes
History of Business Process Standards

BPML (Intallio et al)  2000
BPSS (ebXML)  2001
WSCl (Sun et al)  2001
WS-Choreography (W3C)  2002
WSBPEL 1.1 (OASIS)  2002

XLang (Microsoft)  2000
WSFL (IBM)  2001
WSCL (HP)  2001
BPEL4WS 1.0 (IBM, Microsoft)  2002
BPEL4WS 1.1 (OASIS)  2003
Sample Business Process: Purchasing Order
From a Choreography Perspective

Choreography – The observable public exchange of messages
From an Orchestration Perspective

Orchestration – A private executable business process
Orchestration and Choreography

Two BPEL workflow templates reflecting a business agreement

Source: ORACLE, “BPELOverview”
**BPEL4WS or WS-BPEL**

- The new standard for orchestrating business process using web services
  - Joint IBM/Microsoft proposal, being standardized through OASIS
  - There are some competing languages, e.g. BPML

- Supported by more platform vendors than its predecessors that tried to achieve similar goals, such as ebXML
  - BPEL is supported by Microsoft, IBM, BEA, SAP, Hewlett-Packard, Oracle, Siebel, and others.
  - Choice of process engines
    - Standards lead to competitive offerings
Building Standards-Based Business Processes with Web Services

- Useful in defining both *concrete* and *abstract* processes
- Each activity is represented as a service with a WSDL interface
  - All BPEL interactions are via WSDL-defined interfaces
    - WSDL Message Exchange Patterns
      - WSDL 1.1 has several
      - Common practice / Basic Profile 1.0 is just to use
        - in-out = request-response offered
        - in = one-way received (invoke)
  - Supports compensation model of transactions for long-running processes and fault handling
BPEL Process

A composite Web service with a WSDL description
Semantic Web Services

- Semantic descriptions of Web services are necessary to enable
  - automatic discovery, composition and execution across heterogeneous users and domains.
- A Semantic Web Service is defined through a service ontology, which enables machine interpretability of its capabilities as well as integration with domain knowledge.
Publishing/advertising of SWS will allow agents or applications to discover services based on its goals and capabilities.

Discovery will not be solely based on keyword matching
- Semantic matching between request and published descriptions which involves also inputs, outputs, preconditions and effects
- Specify tasks or goals to be achieved in request and result will include services that can solve these tasks.
- Different degree of matching will be possible: exact, subsumed etc.

Service selection and ranking can be improved by considering non-functional properties such as costs and quality of service
Composition of simple or atomic services into more complex ones can be achieved either semi-automatically or automatically.

- Involves defining workflows of services by mapping outputs of one service to inputs of another.

Invocation involves validation of input types with domain ontology and monitoring of the execution process.
Semantic Web Services Architecture

Components

- **Reasoner**: provides the reasoning support for interpreting the semantic descriptions and queries.
- **Registry**: provides the mechanism for publishing and locating services in a semantic registry as well as functionalities for creating and editing service descriptions.
- **Matchmaker**: mediate between the requester and the registry during the discovery and selection of services.
- **Decomposer**: required for executing the composition model of composed services.
- **Invoker**: mediate between requester and provider or decomposer and provider when invoking services.
What is OWL-S?

- OWL-based Web service ontology
  - Supplies a core set of markup language constructs for describing Web services in an unambiguous, computer-interpretable form
    - Describe Web services capabilities
    - Describe Web services Process Model
    - Map Web services Process Model to WSDL for Web service invocation
  - OWL-S allows services to interact on the Semantic Web
    - Description of capabilities allows capability-based discovery of WS
    - Process Model allows construction of plans that compose the activities of different WS
    - Mapping to WSDL allows automatic invocation of WS
  - OWL-S objective
    - OWL-S does not aim to replace the Web services standards rather it attempts to provide a semantic layer
      - OWL-S relies on WSDL for Web service invocation
      - OWL-S expands UDDI for Web service discovery
Tasks OWL-S is expected to enable

- **Automatic Web service discovery**
  - Automated location of WSs that provide a particular service and adhere to requested constraints

- **Automatic Web service invocation**
  - Automated execution of an identified WS by a computer program or agent

- **Automatic Web service composition and interoperation**
  - Automatic selection, composition and interoperation of WSs to perform some task (e.g. arrangement for a conference)

- **Automatic Web service execution monitoring**
  - Individual services and composition services generally require some time to execute completely
  - It is useful to know the state of execution of services
Three essential types of knowledge about a service:

- **What does the service provide for prospective clients?**
  - The answer to this question is given in the "profile" which is used to advertise the service.
  - To capture this perspective, each instance of the class Service presents a ServiceProfile.

- **How is it used?**
  - The answer to this question is given in the "process model".
  - This perspective is captured by the ServiceModel class. Instances of the class Service use the property describedBy to refer to the service's ServiceModel.

- **How does one interact with it?**
  - The answer to this question is given in the "grounding".
  - A grounding provides the needed details about transport protocols.
  - Instances of the class Service have a supports property referring to a ServiceGrounding.
The class `Service` provides an organizational point of reference for a declared Web service

- One instance of Service will exist for each distinct published service.
- The properties `presents`, `describedBy`, and `supports` are properties of Service.
- The classes `ServiceProfile`, `ServiceModel`, and `ServiceGrounding` are the respective ranges of those properties.
- Each instance of Service will present a `ServiceProfile` description, be describedBy a `ServiceModel` description, and supports a `ServiceGrounding` description.

The `ServiceProfile` provides the information needed to automatically discover a service, while the `ServiceModel` and `ServiceGrounding`, taken together, provide enough information to make use of a service, once it is found.
Service Profile

- describes a service as a function of three basic types of information:
  - what organization provides the service,
  - what function the service computes, and
  - a host of features that specify characteristics of the service.

- It presents both
  - functional properties: inputs, outputs, preconditions and effects (IOPEs)
  - non-functional properties: name, contact, type, category, quality, etc

- profile class can be sub-classed and specialized, thus supporting the creation of profile taxonomies which subsequently describe different classes of services.
Service Process Model

- describes the composition or orchestration of one or more services in terms of constituent processes
- used both for reasoning about possible compositions and also for controlling the enactment/invocation of a service
- chief components:
  - **process**: enables planning, composition & agent/service interoperation
  - **process control model**: allows agents to monitor the execution of a service request (still to be defined)
- process classes:
  - **atomic**: a single, black box process description with exposed IOPEs
  - **composite**: hierarchically defined workflows, consisting of atomic or other composite processes.
  - **simple**: provides an abstract description of services or processes
Service Process Control structures

- **Sequence**: A list of control constructs to be done in order.
- **Split**: A bag of process components to be executed concurrently. Split completes as soon as all of its component processes have been scheduled for execution.
- **Split-Join**: The process consists of concurrent execution of a bunch of process components with barrier synchronization. Split+Join completes when all of its components processes have completed.
- **Any-Order**: Allows the process components (specified as a bag) to be executed in some unspecified order but not concurrently. *Execution and completion of all components is required.*
- **Choice**: Execution of a single control construct from a given bag of control constructs. Any of the given control constructs may be chosen for execution.
- **If-Then-Else**: Branching based on some condition
- **Iterate**: Is an "abstract" class, serves as the common superclass of Repeat-While, Repeat-Until, and potentially other specific iteration constructs.
  - **Repeat-While** and **Repeat-Until**: Iterate until a condition becomes false or true, following the familiar programming language conventions.
Service Grounding

- Providing details on how to interoperate/access the service
  - Protocol, message formats, serialization, …
  - A mapping from an abstract specification to a concrete realization
    - How the abstract inputs and outputs of an atomic process are to be realized concretely as messages (which carry these inputs and outputs)

- WSDL as a possible grounding approach
  - Exploiting the extensibility elements of WSDL
To construct an OWL-S/WSDL grounding one must first identify, in WSDL, the messages and operations by which an atomic process may be accessed, and then specify correspondences.
OWL-S & UDDI

- The main problem with UDDI is that it does not provide a **capability representation language** such as the OWL-S Service Profile.
  - UDDI supports the location of information about Web services, once it is known which Web service to use
  - **UDDI does not provide capability based search** (impossible to locate a Web service on the basis of what problems it solves)

- But …

  **OWL-S and UDDI complement each other**
  - Integrate OWL-S capability matching in the UDDI registry.
    - Mapping of OWL-S Service Profiles into UDDI Web service representations.
    - A set of specialized UDDI TModels to store OWL-S information that cannot be represented in the standard UDDI
  - **OWL-S/UDDI provides all the functionalities provided by UDDI using exactly the same API; any UDDI can interact with it to retrieve information about available Web services.**
    - OWL-S/UDDI supports capability matching by taking advantage of OWL-S capability representation.
Adding OWL-S to UDDI, implementation and throughput, Naveen Srinivasan, Massimo Paolucci, Katia Sycara
OWL-S Editing tools

- OWL-S Editor (Uom)

- OWL-S Editor (SRI)
  http://owlseditor.semwebcentral.org/
Suggested Reading

- W3C Web Services Architecture Group,
  - http://www.w3.org/TR/ws-arch/
- OWL-S
  - http://www.daml.org/services/owl-s/
- BPEL:
  http://www.eclipse.org/tptp/platform/documents/design/choreography_html/tutorials/wsbpel_tut.html#basicbpel
Good Luck for Exam 😊