Web Services and Planning or How to Render an Ontology of Random Buzzwords Useful?

Presented by Zvi Topol

May 12th, 2004

Agenda

- Web Services
- Semantic Web
- OWL-S
- Composition of Web Services using HTN Planning

Web Services

What are Web Services?

- Machine friendly software components designed to work interoperably when deployed over heterogenous computing environments.
- Interoperability is achieved using a set of standards based on XML:
  - WSDL (Web Services Description Language) describes in detail the interface which the WS exposes.
  - SOAP (Simple Object Access Protocol) defines how to interact with WS - through a message-based communication.
  - UDDI (Universal Discovery, Description and Integration) allows the discovery of WS.

Web Services Description Language

- A WSDL document defines Web Services as collections of concrete network endpoints. The following elements define a Web Service (WSDL 2.0, March 2004, latest W3C Working Draft):
  - **<message>**s are abstract, typed definitions of the data being communicated
    - **<types>** encloses data type definitions that are relevant for the exchanged messages (usually using XSD)
  - **< interface >** is a named set of abstract **<operation>**s and the abstract <input>, <output>, and <fault> messages involved
    - different transmission primitives: *one-way*, *request-response*, *solicit-response*, and *notification*.
  - **<binding>** defines concrete message format and protocol details for operations and messages defined by a particular <interface>
    - e.g., use SOAP (<soap:header>, <soap:body>, <soap:operation>).
  - **<service>** is a collection of related <endpoint>s
    - **< endpoint >** defines an individual endpoint by specifying a single address for a binding; e.g., a <soap:address>: a URL (SOAP over HTTP) or email address (SOAP over SMTP)

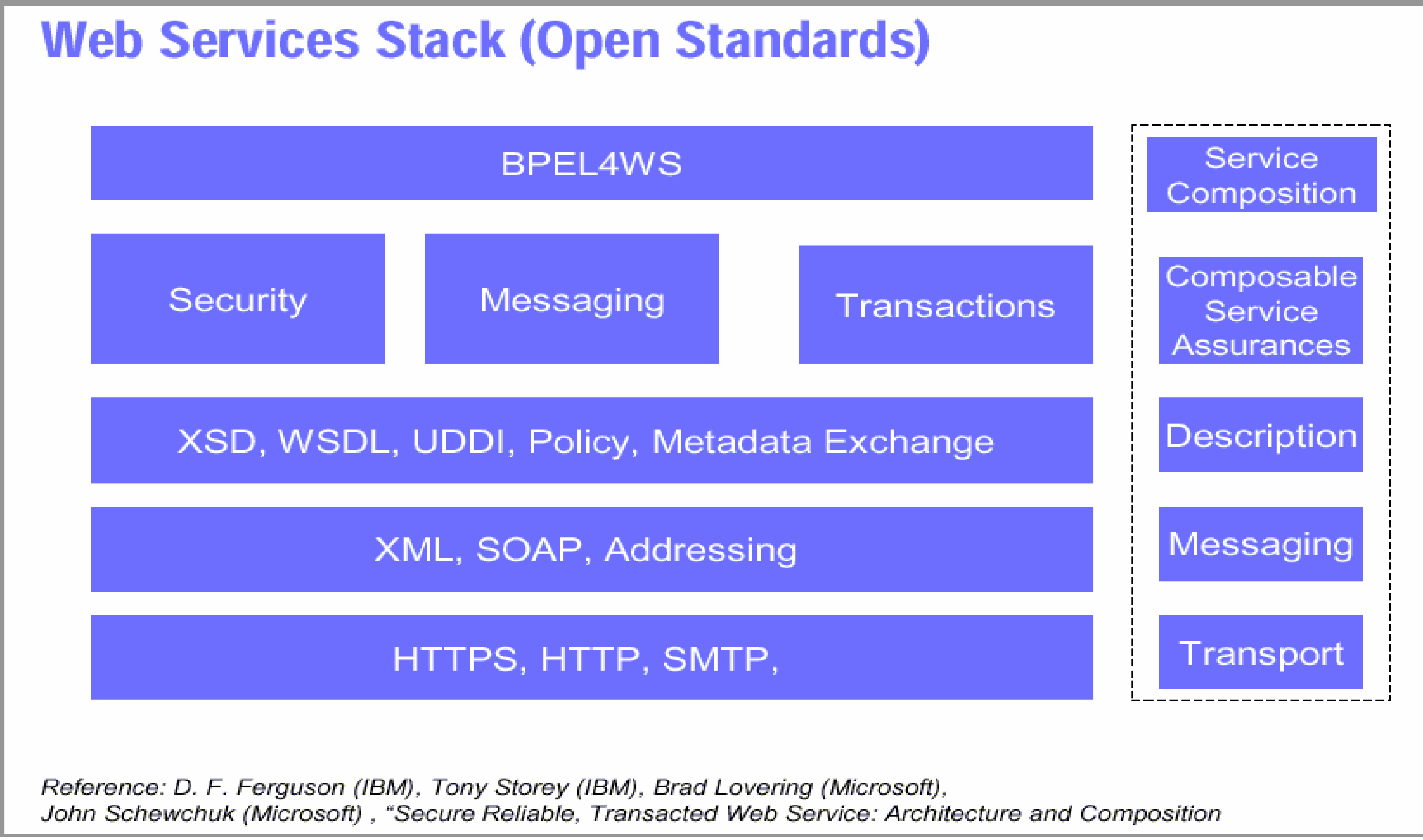
SOAP

- Latest Definition: SOAP 1.2, June 2003, W3C Recommendation
- SOAP = extensible XML messaging framework
  - SOAP messages flow from initial SOAP sender to ultimate SOAP receiver;
  - message path, possibly through multiple SOAP intermediaries;
  - different message exchange patterns are supported;
  - a SOAP node can act in one or more SOAP roles
  - messages can be exchanged over a variety of underlying protocols
- A SOAP message is an **<Envelope>** containing one or more **<Header>**s and a **<Body>**
  - Body and Headers can have specific attributes:
    - *encodingStyle* (body),
    - *mustUnderstand* (header).
    - *role* (indicates recipient of header if not final destination, but intermediary),
    - *relay* (header, if not processed)
  - Body can be any XML message, particularly a SOAP **<Fault>**
- SOAP binding defines how to carry a SOAP message within or on top of another (underlying) protocol; e.g., within an HTTP body, or over a TCP stream

Universal Description, Discovery and Integration

- Latest: UDDI Version 3.0.1, OASIS TC Spec., Oct. 2003
- Describes the Web services, data structures and behaviours of all instances of a UDDI registry
- A UDDI Registry is formed by one or more UDDI Nodes supporting one or more Node API Sets, and contains the following information:
  - *businessEntity* (descriptive info of business)
  - *businessService* (descriptive info of Web service)
  - *bindingTemplate* (tech. description of Web service)
  - *tModel* (‘technical fingerprint’ of the Web service)

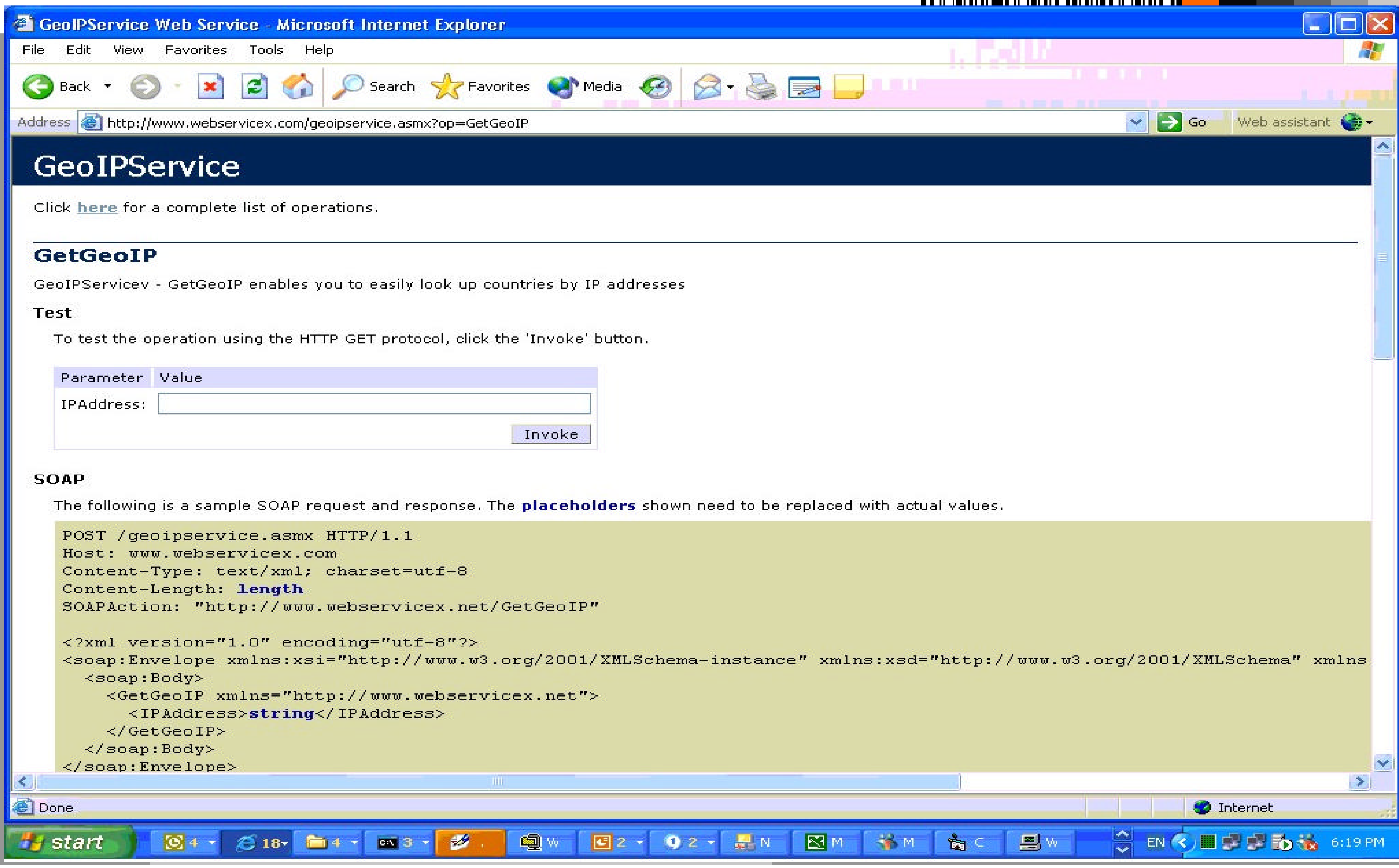
Web Services Stack



WS Example

- Example:
  - GEOIP Web Service: http://www.webservicex.com/geoipservice.asmx
  - Has two methods:
    - GetGeoIPContext
      enables to look up countries by Context
    - GetGeoIP
      enables to look up countries by IP addresses

WS Example (continued)



WS Example (continued)

- <?xml version="1.0" encoding="utf-8" ?>
- \- <GeoIP xmlns:xsd="**http://www.w3.org/2001/XMLSchema**" xmlns:xsi="**http://www.w3.org/2001/XMLSchema-instance**" xmlns="**http://www.webservicex.net**">
- <IP>**62.34.45.43**</IP>
- <CountryCode>**FR**</CountryCode>
- <CountryName>**France**</CountryName>
- <ReturnCode>**1**</ReturnCode>
- <ReturnCodeDetails>**Record Found**</ReturnCodeDetails>
- </GeoIP>

WS Orchestration and Choreography

- Different enterprises have implemented various Web Services.
- How to integrate them together into a complex and meaningful *business process*?
- Orchestration:
  - refers to an executable business process that may interact with both internal and external Web services.
  - describes how Web services can interact at the message level to result a long-lived, transactional process.
  - With orchestration, the process is always controlled from the perspective of one of the business parties.

WS Orchestration and Choreography (continued)

- Choreography:
  - each party involved in the process describes the part they play in the interaction.
  - Choreography tracks the sequence of messages that may involve multiple parties and multiple sources.
  - It is associated with the public message exchanges that occur between multiple Web services.
- Main difference: choreography is more collaborative in essence, dealing with message communication of multiple parties, whereas orchestration “tells the story” from the point of view of a single party.

BPEL4WS

- Business Process Execution Language For Web Services is a flow specification language used to compose Web Services.
- Based on XML and created as the merging of IBM’s WSFL and Microsoft’s XLANG.
- Latest: version 1.1, May 2003.
- Goal: allowing long-running transactions between Web Services.

BPEL4WS (continued)

- BPEL supports two usage scenarios:
  - Implementation of executable business processes
  - Description of non-executable abstract processes (=business protocols)
- A *business process* specifies
  - the potential execution order of operations from a collection of Web Services
  - the data shared between the Web Services
  - which partners are involved and how.
  - joint exception handling
- A *business protocol* specifies the public message exchanges between parties. Business protocols are not executable and do not describe the internal details of a process flow

BPEL4WS (continued)

- In its essence, BPEL is a flow-chart like description of an algorithm with
  - Primitive activities, such as:
    - <invoke> for service invocation
    - <receive> handles message reception
    - <reply> deals with response messages
    - <assign> copy data from one place to another
    - <catch> exception handling
    - <throw> exception handling
    - <terminate> termination of a process
  - Structured activities, such as:
    - <sequence>
    - <switch>
    - <while>
    - <pick>
    - <flow>

BPEL4WS (continued)

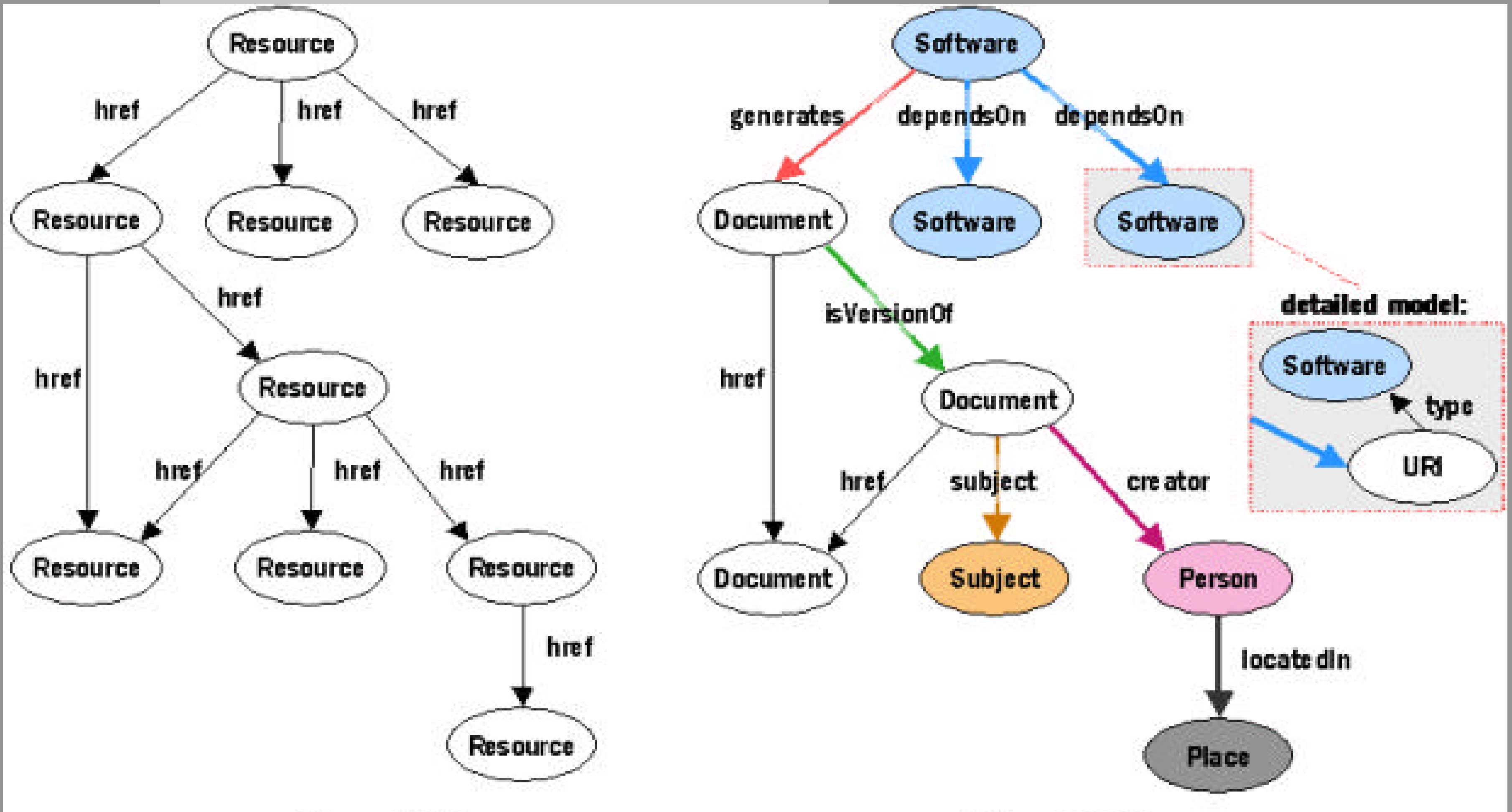
- Variables allow persistency: they identify the specific data exchanged in a message flow. When a business process receives a message, the appropriate variable is populated so that subsequent requests can access the data.
- Partners define and describe the different parties interacting with the process and their roles.

The Semantic Web

The Semantic Web

- A set of ontologies designed to give meaningful definitions to various resources.
- Faciltitates access and reasoning of agents w.r.t. information residing on the Web.
- Examples for ontologies: FOAF, RSS, OWL-S.

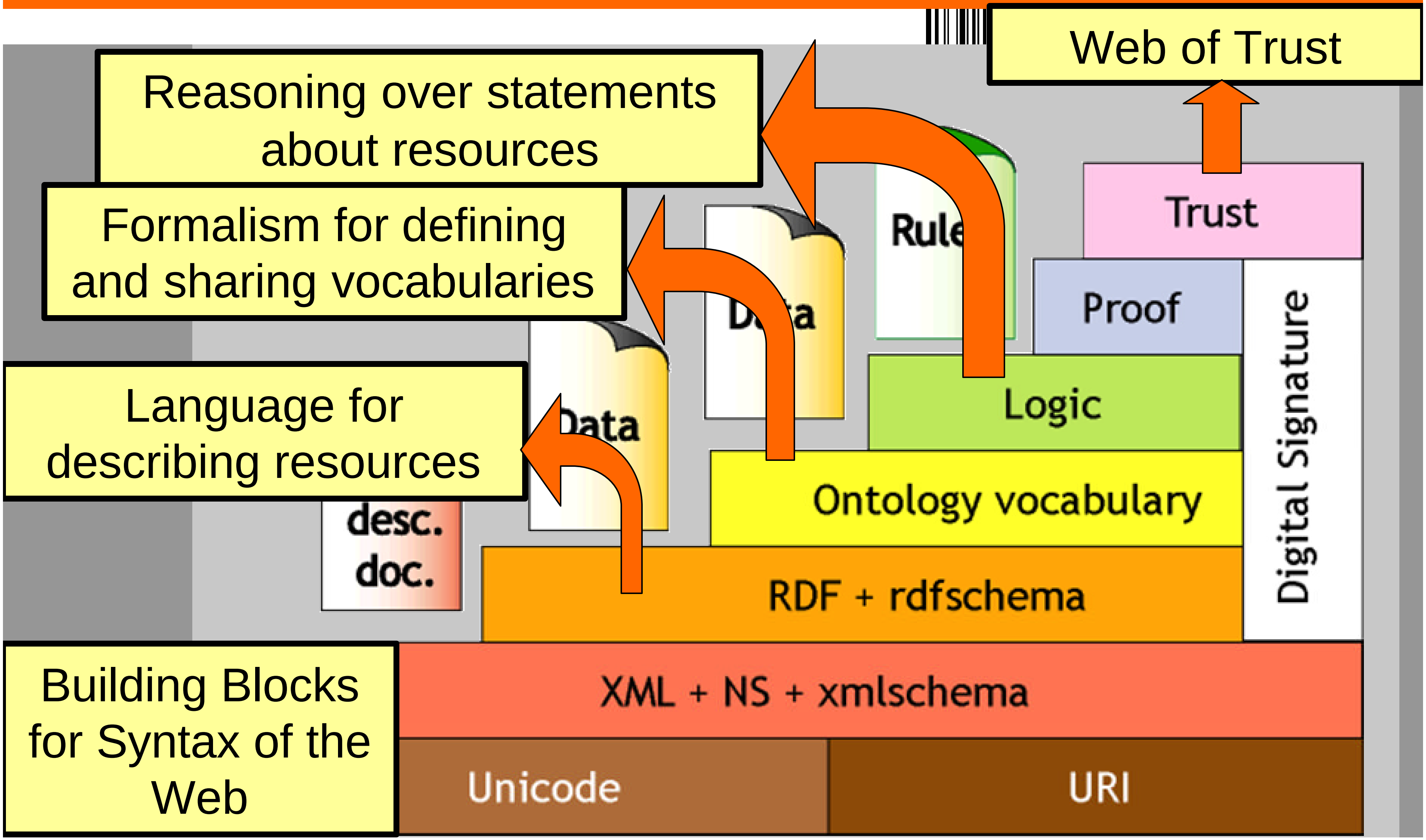
The Semantic Web (continued)



a) Current Web

b) Semantic Web

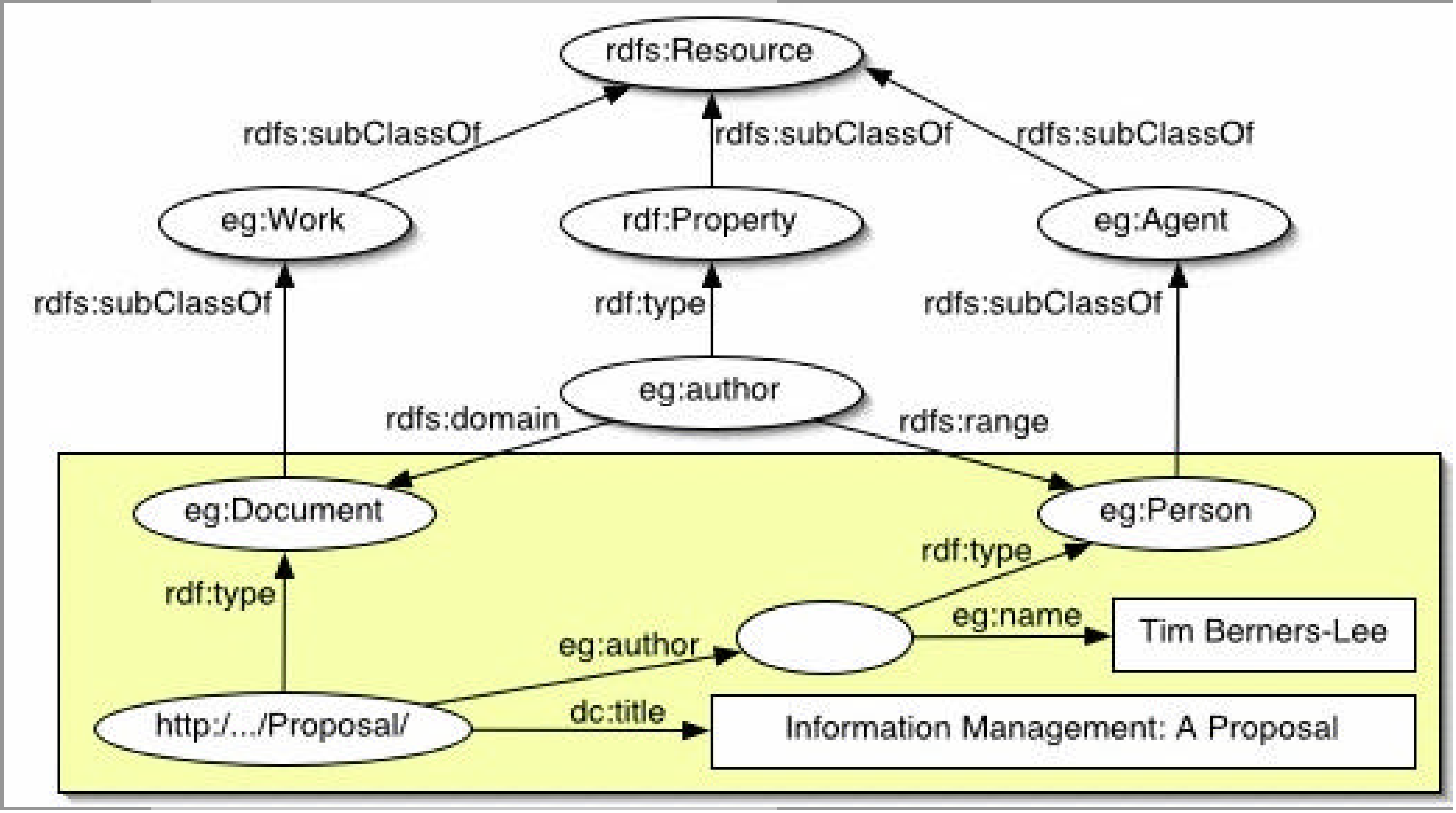
The Semantic Web (continued)



Resource Description Framework

- A framework for knowledge representation based on XML
- Statements are made w. r. t web resources
- Form of statements: triplets of <Subject, Predicate, Object>

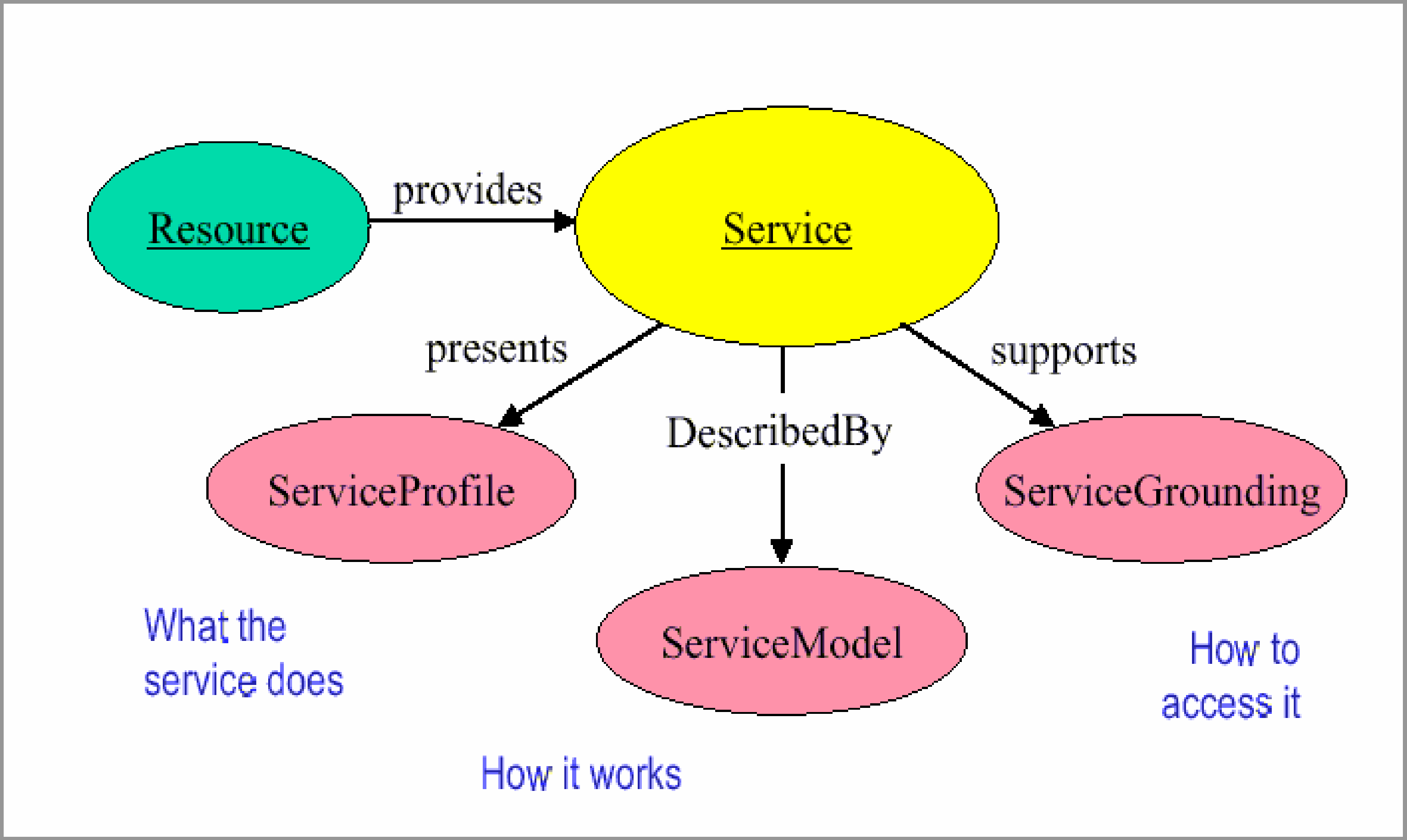
RDF - Example


rdfs:Resource
rdfs:subClassOf
rdfs:subClassOf
rdfs:subClassOf
eg:Work
rdf:Property
eg:Agent
rdfs:subClassOf
rdf:type
rdfs:subClassOf
eg:author
rdfs:domain
rdfs:range
eg:Document
eg:Person
rdf:type
rdf:type
eg:name
Tim Berners-Lee
eg:author
http:/.../Proposal/
dc:title
Information Management: A Proposal

OWL-S

- Web Ontology Language for Services
- Latest: version 1.0
- Three main parts
  - Service Profile: for advertisement and discovery of services
  - Process Model: for the provision of a detailed description of the service.
  - Grounding: How to interact with a service through messaging.

OWL-S



Resource
provides
Service
presents
supports
DescribedBy
ServiceProfile
ServiceGrounding
ServiceModel
What the service does
How to access it
How it works

Service Profile

- Allows discovery of service through any type of registry
- Describes which entity provides the service
- Provides functional description of the service:
  - Inputs
  - Outputs
  - Preconditions
  - Effects
- Allows description of properties of service:
  - Category of a given service
  - Quality rating of the service
  - Any other information (response time, etc)

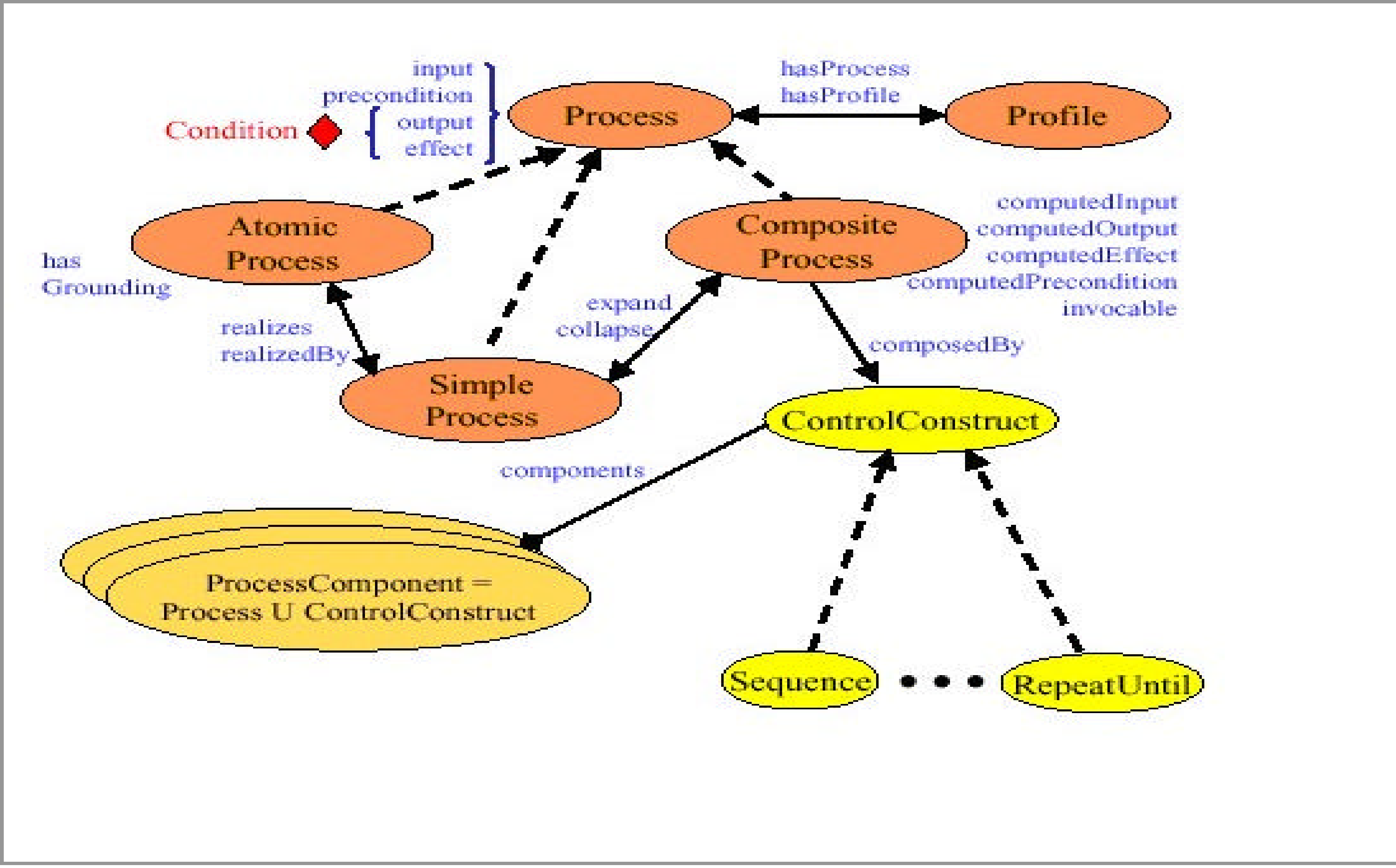
Service Model

- Services are modeled as *processes* that:
  - Receive inputs and produce outputs.
  - Produce a transition in the world from one state to another (preconditions and effects)
- Outputs and Effects could be conditioned.
- The class *Process* collects three types of processes:
  - Atomic Processes: directly invocable
  - Simple Processes: not invocable. Allow a specialized view of Atomic Processes or simplified representation of Composite Processes.
  - Composite Processes: decomposable into other non-composite or composite Processes.

Service Model (continued)

- Composite Process must have a *composedOf* property indicating the control structure using ControlConstruct.
- Possible ControlConstructs:
  - Sequence: defines a list of processes to be done in order
  - Split: process components to be executed concurrently
  - Split+Join: concurrent execution + barrier synchronization
  - Unordered: unordered execution, possibly concurrently.
  - Choice: possible choice of processes and execution control
  - If-Then-Else
  - Iterate
  - Repeat-Until

Service Model (continued)



Grounding

- Grounding is a mapping from an abstract to a concrete specification of the Service.
- Specifies how to access the service - a concrete specification of how inputs and outputs of an atomic process described in an abstract manner could be realized concretely as messages.
- WSDL is a good candidate for grounding

OWL-S/WSDL Grounding

- An OWL-S/WSDL grounding is based on 3 correspondences between OWL-S and WSDL:
  - An atomic process corresponds to a WSDL *operation*, e.g. an atomic process with both inputs and outputs corresponds to WSDL request-response operation.
  - The set of inputs and set of outputs each corresponds to the a WSDL *message*
  - The types (OWL-S classes) of the inputs and outputs correspond to WSDL *abstract type*.
- To construct such a grounding one should identify the messages and operations by which an atomic process could be accessed and then specify the correspondences.
- OWL-S WSDLGROUNDING (subclass of Grounding) allows to create references to the apporpriate WSDL specifications.

Resource Ontology

- Processes require resources, hence the need for an ontology for resources.
- A resource ontology for OWL-S is under development.
- Primary interest of this ontology is *resource tokens* - instances of resource types that could be consumed, replenished, locked and released.

Composition Through Planning

A Classical Example

- Say I want to plan a trip to Costa Rica.
- When I am in San Jose, I would like to stay only in hotels of 4 stars or better, at the center.
- I have other contraints regarding flights, restaurants, cocktails, etc...
- Wouldn‘t it be great to feed my constraints and get a clear, ordered trip-plan? (based on automatic composition of Web Services)

HTN Planning

- Hierarchical Task Network Planning is based on *hierarchical decoposition*
- Initial plan describing the problem, is viewed as a very high level description of the goal
- Plans are refined applying *task decompositions*, reducing a high-level *task* to a partially ordered set of *subtasks*
- The process continues until only *primitive tasks* remain in the plan

Motivation for using HTN Planning in WSC

- HTN encourages modularity which is a natural match for Web Service composition.
- HTN planning scales well to large numbers of methods and operators.
- Some HTN planners support sophisticated condition reasoning such as evaluation and integration of information-supplying Web Services

SHOP2 Planner

- A domain-independent HTN planning system
- Plans for tasks in the same order that they will be executed:
  - Current state of the world could be known at each step of planning.
  - Call for external information sources could be easily integrated.
- Knowledge about a domain consists of Operators, Methods and a KB.

SHOP2 Domain Knowledge

- Operators are similar to STRIPS: of the form of (h(v), Pre, Del, Add) where
  - h(v) is a primitive task, v input list
  - Pre - preconditions
  - Del - delete list
  - Add - add list
- Methods describe how to decompose a compound task into partially ordered subtasks. Of the form: ((h(v), Pre1, T1, Pre2, T2, ...) where
  - h(v) is a compound task, v input list
  - Pre_i is a precondition
  - T_i is a partially ordered set of subtasks.

Planning Problem

- A triple <S, T, D> where S is the initial state of the world, T is a task list and D is a domain knowledge is said to be a *planning problem* for SHOP2.

- SHOP2, given input <S, T, D> will return a plan P, which is a sequence of grounded operators, achieving T from S in D

SHOP2 Planning Procedure

```
procedure SHOP2(s, T, D)
    if T is empty then return empty plan
    Let t be the first task in T
    if t is a primitive task then
        Find an operator o = (h Pre Add Del) in D such that
            h unifies with t and s satisfies Pre
        if no such o exists then return failure
        Let s' be s after deleting Del and adding Add
        Let T' be T after removing t
        return [o, SHOP2(s', T', D)]
    else if t is a composite task
        Find a method m = (h Pre_1 T_1 Pre_2 T_2 ...) in D such that
            h unifies with t
        Find the task list T_i such that
            s satisfies Pre_i and does not satisfy Pre_k, k < i
        if no such T_i exists then return failure
        Let T' be T after removing t
            and adding all the elements in T_i at the beginning
        return SHOP2(s', T', D)
    end if
end SHOP2
```

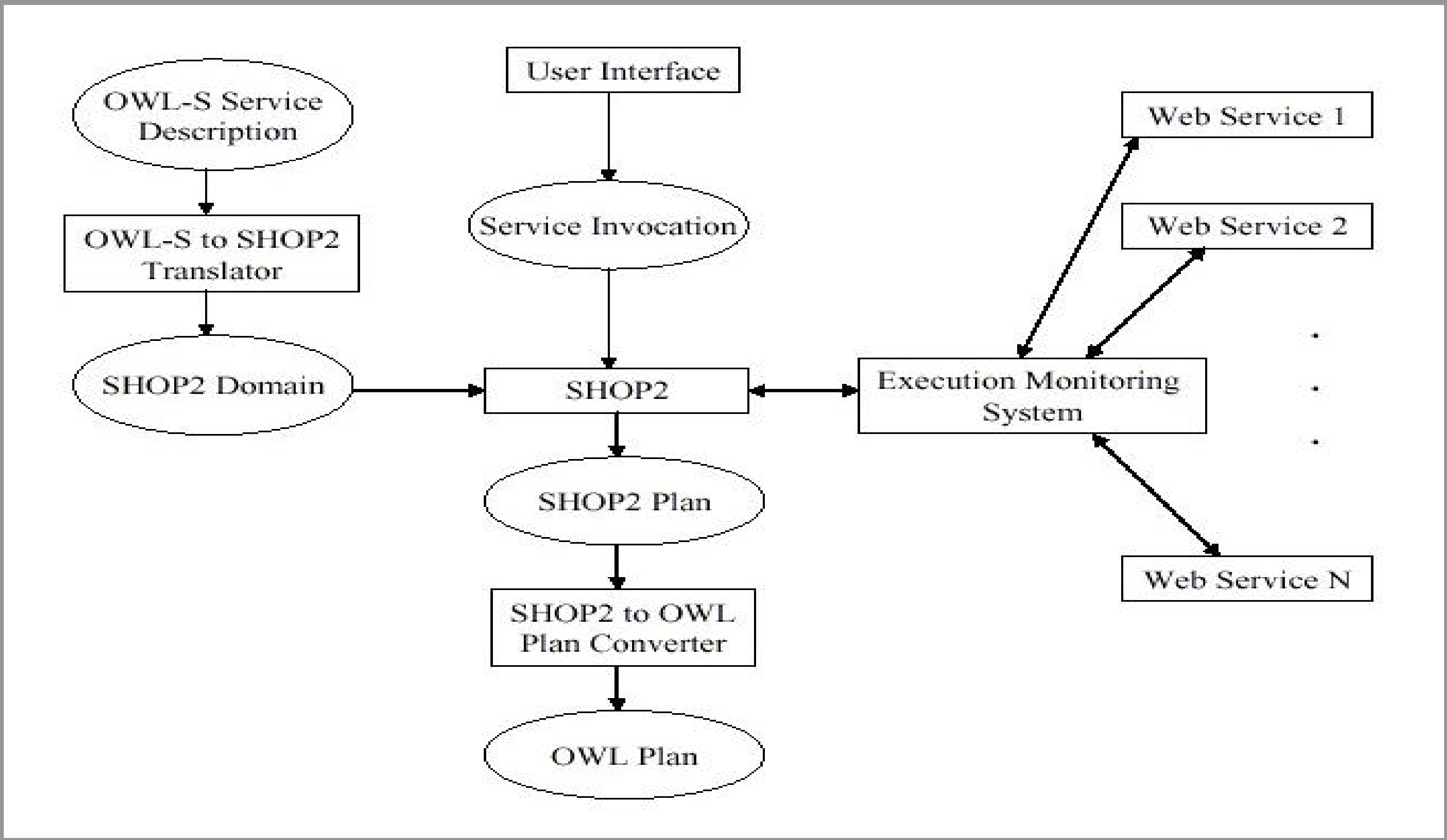
Translating OWL-S to SHOP2

- Two main assumptions: given a collection of OWL-S process models K={K_1, ..., K_n}
  - Atomic processes in K are either only information-providing (outputs without effects) or only world-altering (effects without outputs) - we would like to gather information from information-providing Web Services without changing the world.
  - There is no composite process in K with Split or Split + Join, as SHOP2 doesn’t support concurrency.

Translating OWL-S to SHOP2 (continued)

- Translation is quite straight-forward:
  - Each atomic process with effects is encoded as a SHOP2 operator simulating the effects of the world-altering WS.
  - Each atomic process with output is encoded as a SHOP2 operator whose precondition include a call to the information-providing WS and effects are the WS's output.
  - Each simple or composite process is encoded using one or more SHOP2 methods.
- Full details in the paper.

System Architecture



But….

- SHOP2 theorem prover makes closed-world assumption vs. open-world assumption of Semantic Web.
- Questions of expressiveness: OWL DL vs. SHOP2 axioms.
- Scalability questions w.r.t. the amount of data in the Semantic Web
- Current mapping of information-gathering processes to operators is not very elegant, this could be circumvented with some cost.

Bibliography

- http://www.w3.org/TR/wsdl20
- http://www.uddi.org
- http://www.w3.org/TR/soap
- http://www-106.ibm.com/developerworks/library/ws-bpel/
- Web Services Orchestration and Choreography, Chris Pelz. IEEE Computer, October 2003
- http://www.daml.org/services/owl-s/1.0/
- HTN planning for Web Service Composition Using SHOP2 , Evren Sirin, Bijan Parsia, Dan Wu, James Hendler, and Dana Nau. Submitted to Journal of Web Semantics.