

Intelligente Systeme im World Wide Web

Semantic Web Services

Folien zur Vorlesung im Sommersemester 2006

Anupriya Ankolekar

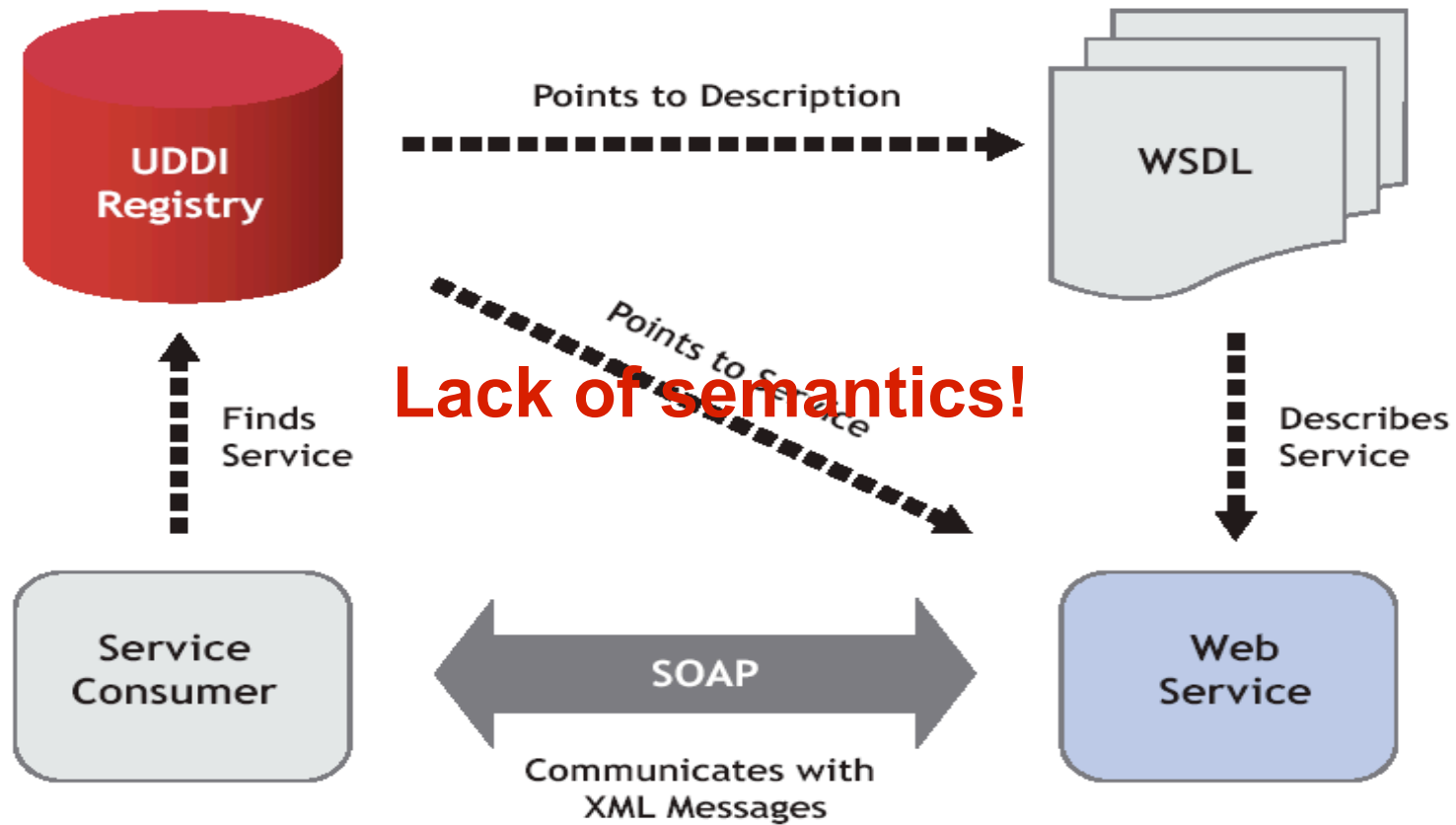
Institut für angewandte Informatik und Formale
Beschreibungsverfahren (AIFB)

Universität Karlsruhe (TH)

Outline

- Web Services
 - What is a Web Service and how can it be used?
 - SOAP
 - WSDL
 - UDDI
 - BPEL
 - Business case for Web services
- Semantic Web Services
 - Some criticisms of the standards
 - Semantics to reduce interoperability issues
 - OWL-S
 - WSMO

WS standards



Problem: Interoperability by using the same format, but still only syntax-based ... no way to describe functionality

Current State: Web Services Standards I

SOAP

- XML-based web services communication protocol
- *Limitations: no description of format and role of message in WS interaction*

WSDL

- Structured mechanism to describe a WS interface
 - Abstract operations that a Web Service can perform
 - Format of messages it can process
 - Protocols it can support
 - Physical bindings to URIs and protocols
- *Limitations: no semantics for message sequencing, correlation and content*

Current State: Web Services Standards II

UDDI

- Directory service for Web Services
- *Limitations: only keyword searches, limited functionality search*

BPEL

- Description of how Web Services are composed together
 - structure of the WS in terms of individual process steps, data flow links and control flow links (Flow Model)
 - interaction between provider and requester and mappings between internal operations and WSDL port types (Global Model)
- *Limitations: No description of inputs, outputs, preconditions and effects of WS, and only manually constructed compositions*

The Integration Challenge

Integrate multiple independent and heterogeneous

- Data repositories
- Processes
- Applications

Ensure

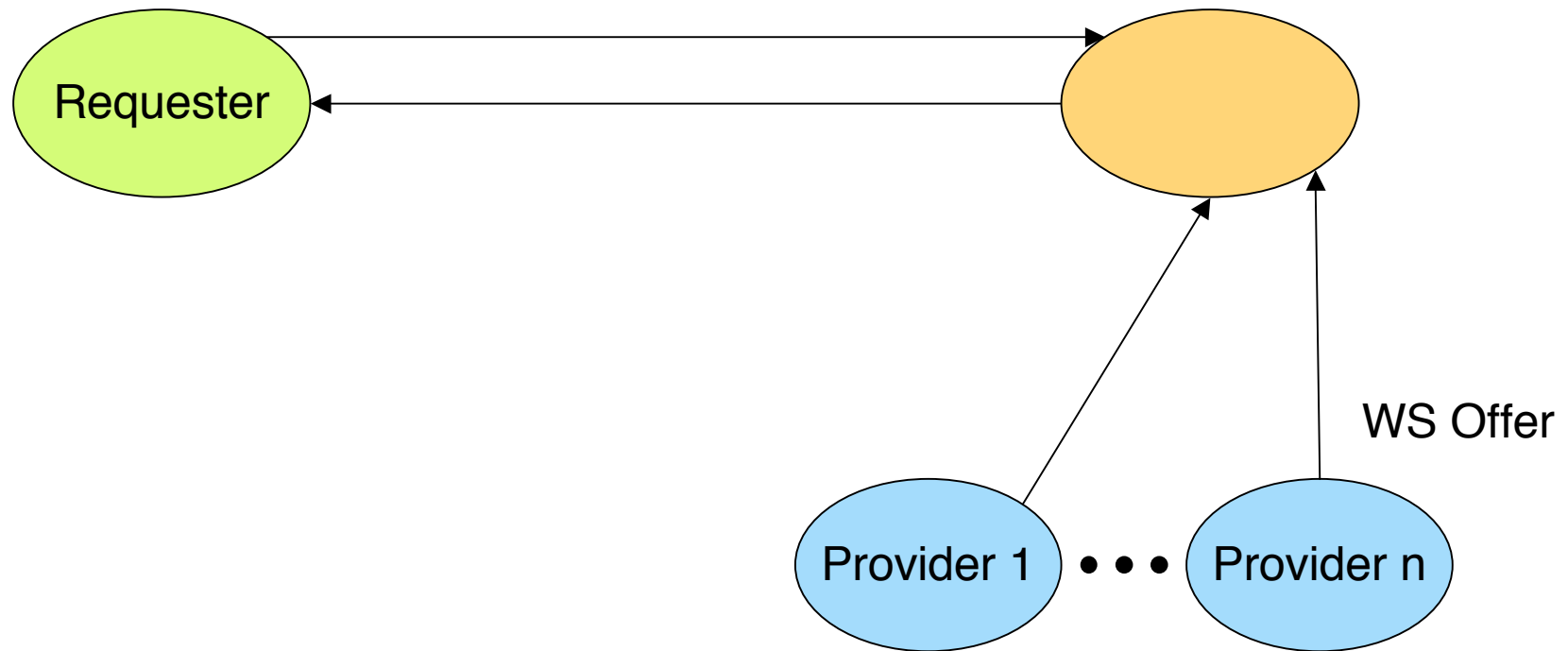
- Semantic equivalence of equivalent concepts
- Flexibility: systems enter and leave integration
- Performance

Lack of Semantic Interoperability ...

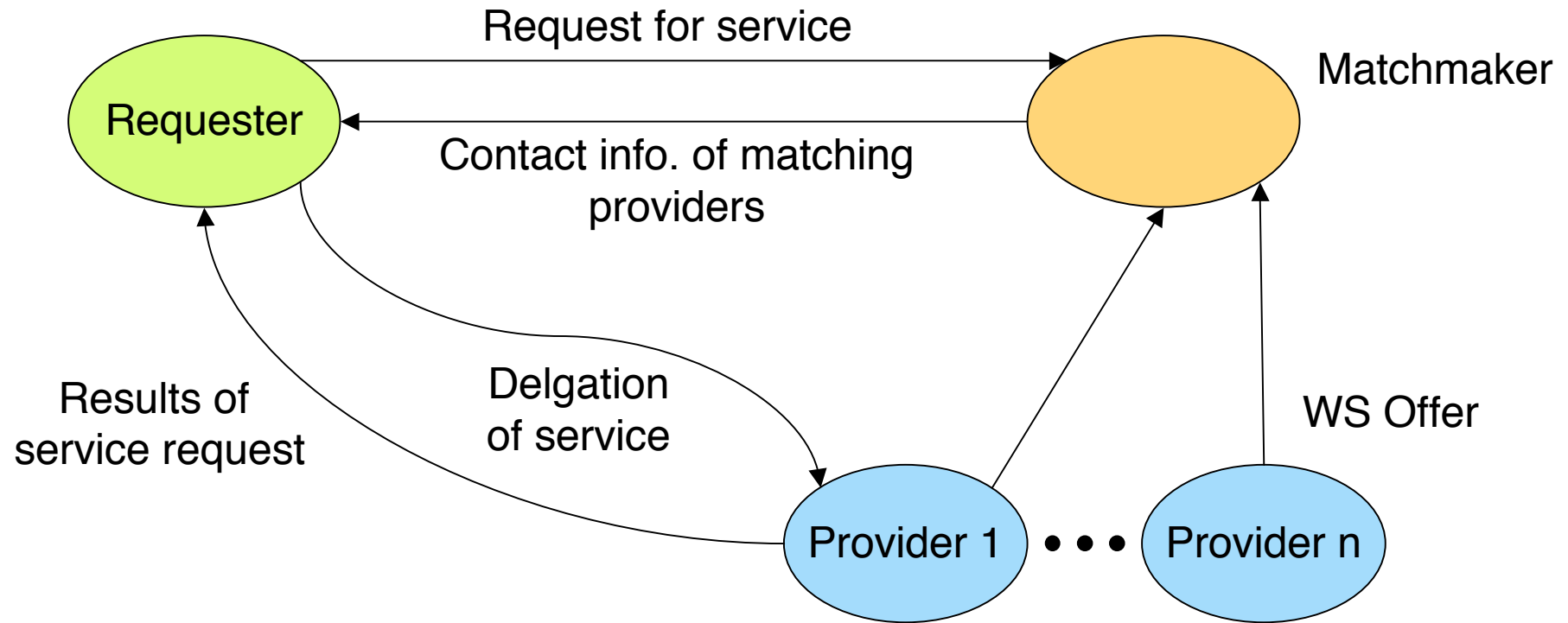
A problem for ...

- **Discovery**
 - Different terms used for offers and requests
- **Invocation**
 - Different specifications for messages, protocols and WS interface
- **Understanding**
 - Interpreting the results returned by the Web service
- **Composing Services**
 - Reconciling private goals with goals of the WS

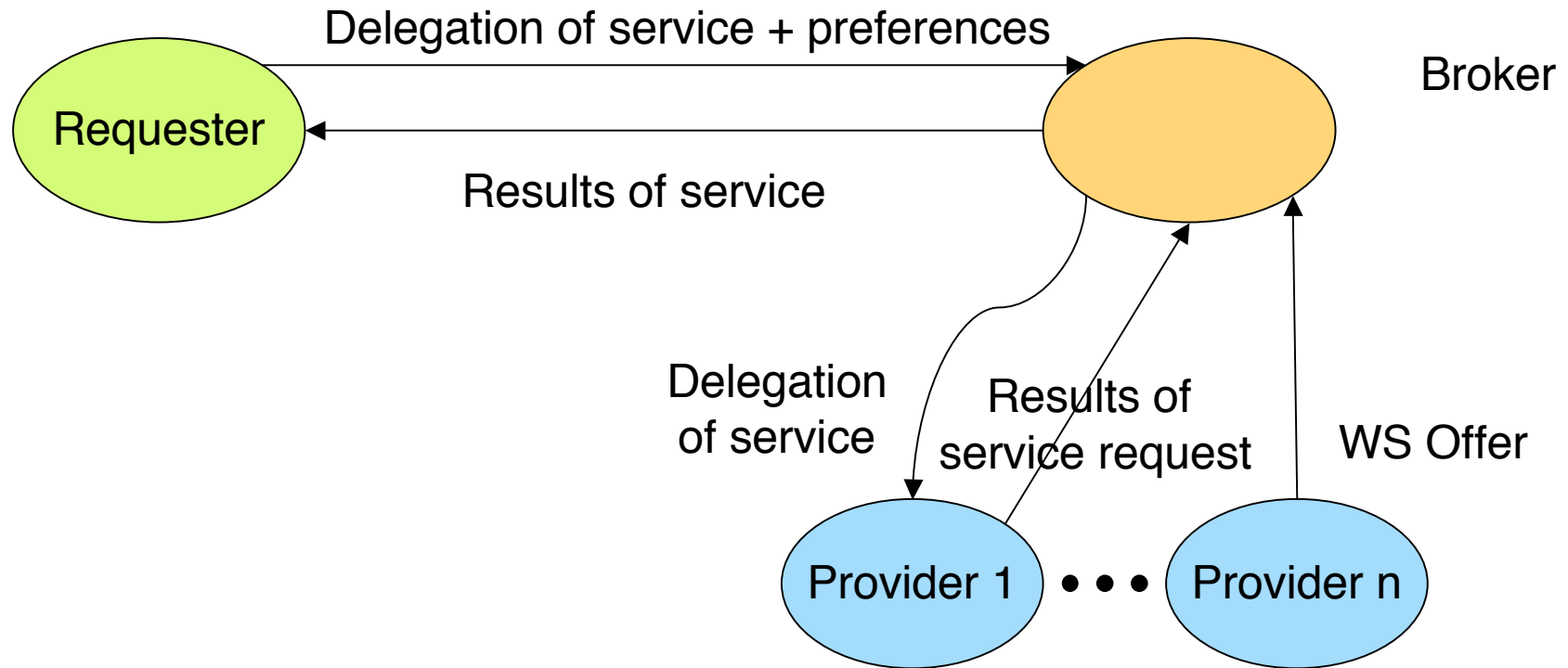
Models of WS interaction



Models of WS interaction



Models of WS interaction



Semantic Web Services

Objective: Semi-Automation of the **Usage Process**:

- **Publication**: Make available the description of the functionality 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
- **Monitoring**: Control the execution process
- **Compensation**: Provide transactional support and undo unwanted effects
- **Replacement**: Facilitate the substitution of services by equivalent ones

Problem of Discovery

- Requester and provider have different views
 - Provider sells Mutual Funds
 - Requester wants Mutual Funds of European Companies

There is no exact match

- Matchmaker or broker needs to exploit the semantics of advertisements and requests to recognize partial matches

Problem of Composition

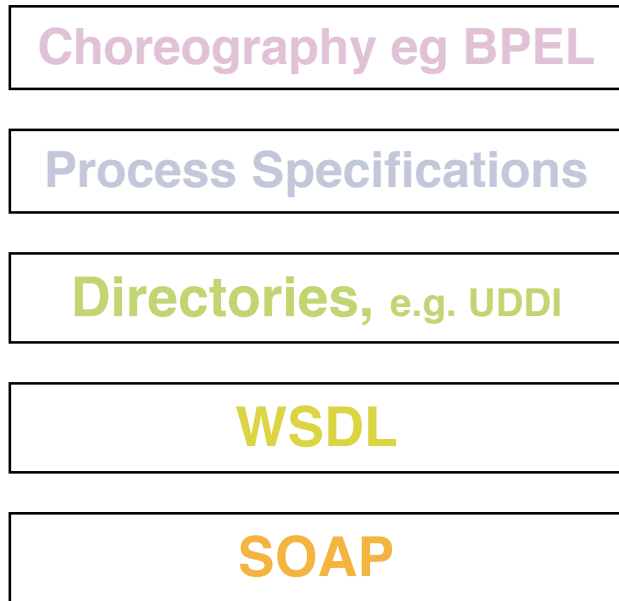
- No single Web service may achieve all goals of an agent
 - Composition is the process of chaining results from different Web services automatically
- Planning problem
 - How do the Web services fit together?
- Interoperation problem
 - How does the information returned fit together?

Semantics for WS usage process

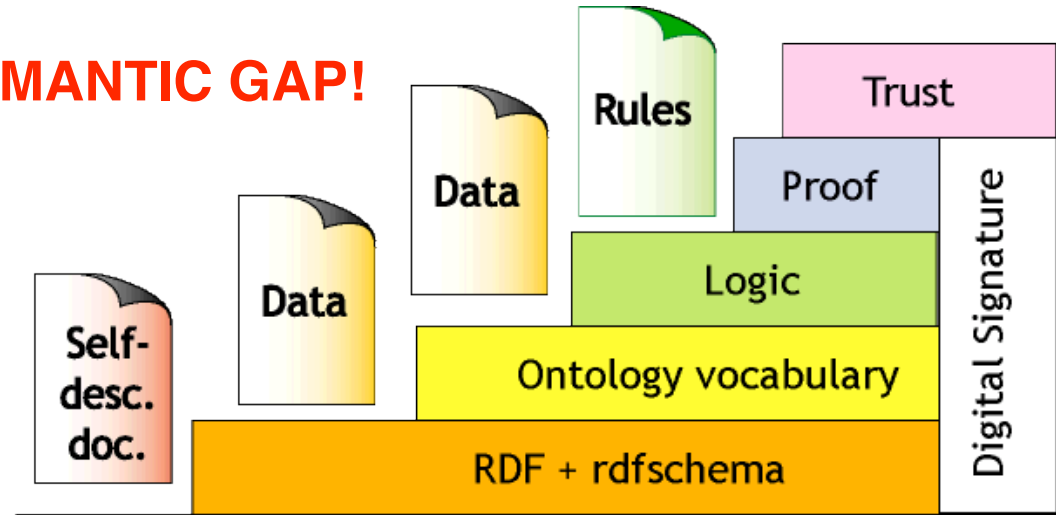
- Discovery
 - Provide formal representation of capabilities and goal
 - Conceptual model for service discovery
 - Support different approaches to web service discovery
- Composition
 - Provide formal representation of capabilities and choreographies
- Invocation
 - Support any type of WS invocation mechanism
 - Clear separation between WS description and implementation

Semantic Web Services Stack Diagram

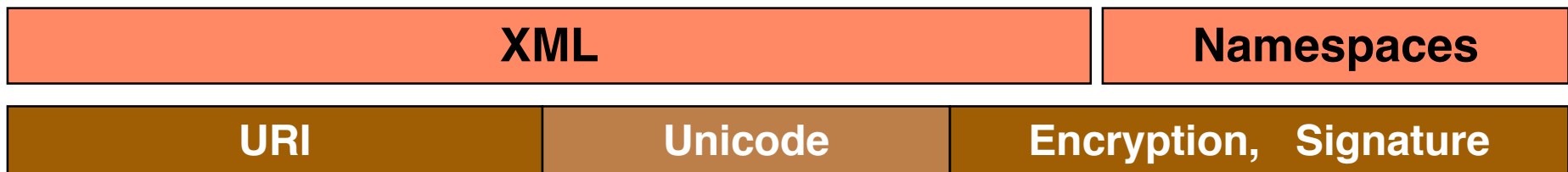
Web Services Stack



SEMANTIC GAP!



Semantic Web Stack



Service Description languages and ontologies to enable semantic markup

- Should describe all information necessary to enable
- automating discovery, composition, execution, etc.
 - semantically enhanced repositories

Two main efforts: **OWL-S** and **WSMO**

OWL-S Ontology

- OWL-S is an OWL ontology to describe Web services
- OWL-S has been accepted as a Note by W3C (12/1/2004)
- OWL-S uses OWL DL reasoning to
 - Support capability based discovery of Web services
 - Support automatic composition of Web services
 - Support automatic invocation of Web services
 - Support monitoring of the execution of Web services

Complete do not compete

- OWL-S does not aim to replace the Web services standards but attempts to provide a semantic layer
 - uses WSDL for Web service invocation
 - extends UDDI for Web service discovery

<http://www.daml.org/services/owl-s/>

Semantic Web Services

Relation to WS Standards

	OWL-S	Web Services Infrastructure
Discovery <i>What it does</i>	Profile	<i>UDDI API</i>
Orchestration <i>How is done</i>	Process Model	<i>BPEL4WS</i>
Invocation <i>How to invoke</i>	Grounding+ WSDL/SOAP	<i>WSDL/SOAP</i>

OWL-S Upper Ontology

WS functionality specification to assist **discovery**

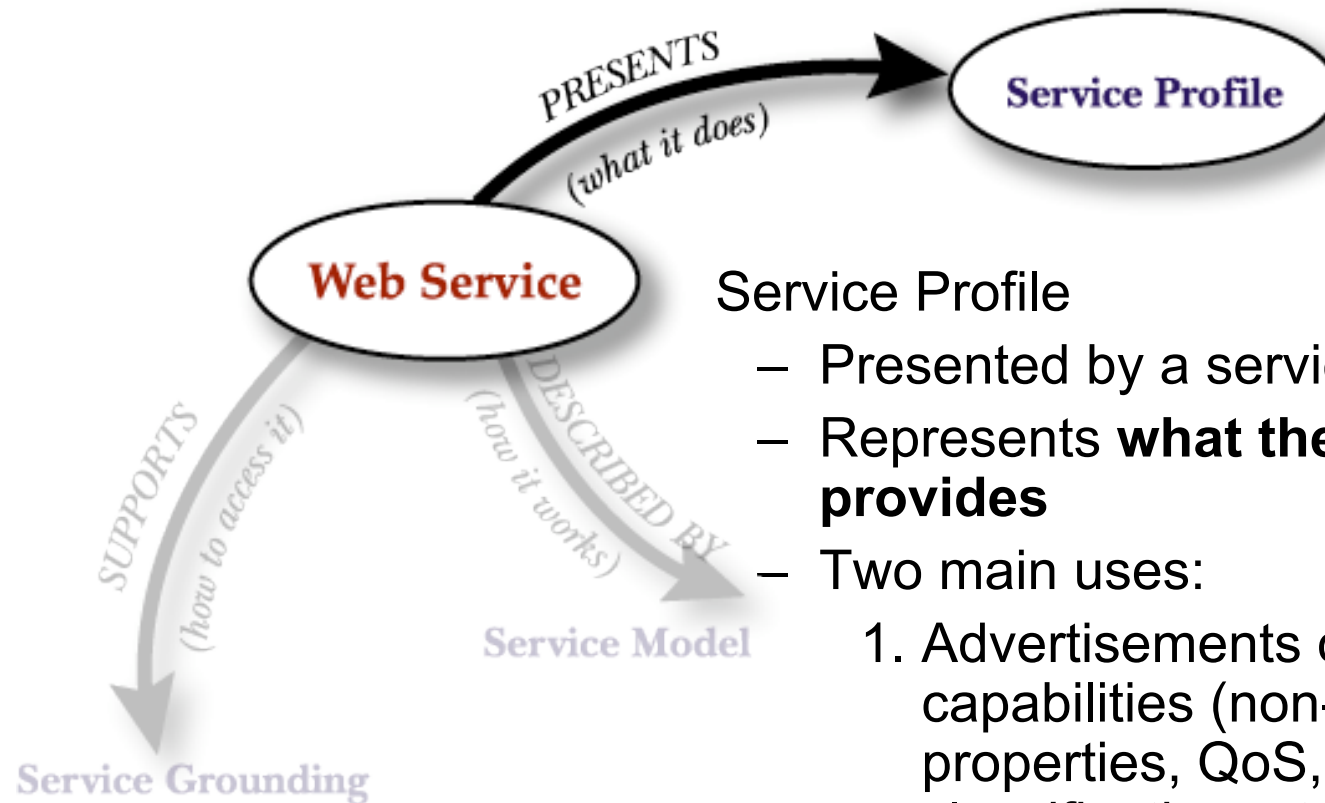
- Used for advertisement of service functionality
- Used for requesting WS with desired capabilities



Mapping to WSDL

- communication protocol (RPC, HTTP, ...)
- transformation to and from XSD to OWL
- 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 (non-functional properties, QoS, Description, classification, etc.)
 2. Request Web services with a given set of capabilities

Service Profile 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)
- **Service Parameters** (e.g. security, QoS etc)

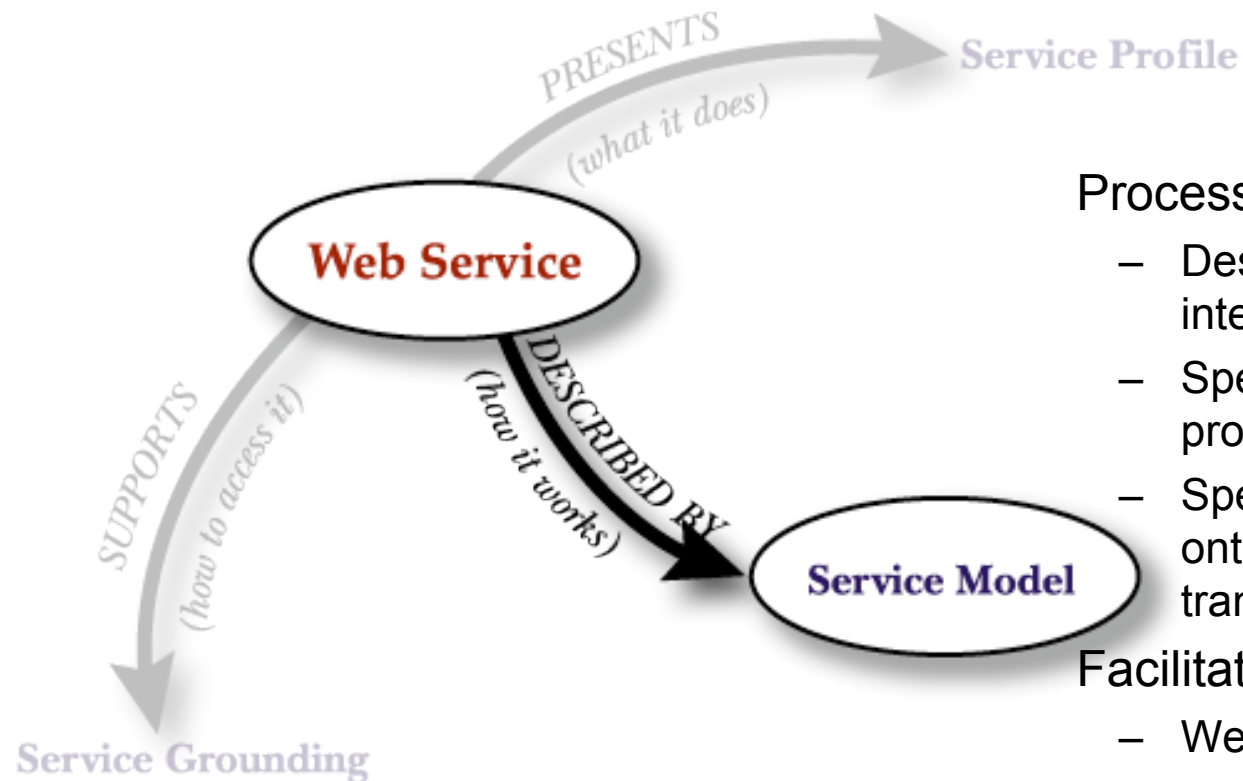
Example: Congo Profile

```

- <profileHierarchy:BookSelling rdf:ID="Profile_Congo_BookBuying_Service">
  <!-- reference to the service specification -->
  <service:presentedBy rdf:resource="http://www.daml.org/services/owl-s/1.1/CongoService.owl#ExpressCongoBuyService"/>
  <profile:has_process rdf:resource="http://www.daml.org/services/owl-s/1.1/CongoProcess.owl#ExpressCongoBuy"/>
  <profile:serviceName>Congo_BookBuying_Agent</profile:serviceName>
- <profile:textDescription>
  This agentified service provides the opportunity to browse a book selling site and buy books there
</profile:textDescription>
- <profile:contactInformation>
  - <actor:Actor rdf:ID="CongoBuy_contacts">
    <actor:name>ExpressCongoBuy</actor:name>
    <actor:title> Service Representative </actor:title>
    <actor:phone>412 268 8780 </actor:phone>
    <actor:fax>412 268 5569 </actor:fax>
    <actor:email>Bravo@Bravoair.com</actor:email>
  - <actor:physicalAddress>
    somewhere 2, OnWeb, Montana 52321, USA
    </actor:physicalAddress>
  - <actor:webURL>
    http://www.daml.org/services/owl-s/1.1/ExpressCongoBuy.html
    </actor:webURL>
  </actor:Actor>
</profile:contactInformation>
<profileHierarchy:deliveryRegion rdf:resource="http://www.daml.org/services/owl-s/1.1/Country.owl#UnitedStates"/>
<profile:hasInput rdf:resource="http://www.daml.org/services/owl-s/1.1/CongoProcess.owl#ExpressCongoBuyBookISBN"/>
<profile:hasInput rdf:resource="http://www.daml.org/services/owl-s/1.1/CongoProcess.owl#ExpressCongoBuySignInInfo"/>
<profile:hasInput rdf:resource="http://www.daml.org/services/owl-s/1.1/CongoProcess.owl#ExpressCongoBuyCreditCardNumber"/>
<profile:hasInput rdf:resource="http://www.daml.org/services/owl-s/1.1/CongoProcess.owl#ExpressCongoBuyCreditCardType"/>
<profile:hasInput rdf:resource="http://www.daml.org/services/owl-s/1.1/CongoProcess.owl#ExpressCongoBuyCreditCardExpirationDate"/>
<profile:hasPrecondition rdf:resource="http://www.daml.org/services/owl-s/1.1/CongoProcess.owl#ExpressCongoBuyAcctExists"/>
<profile:hasPrecondition rdf:resource="http://www.daml.org/services/owl-s/1.1/CongoProcess.owl#ExpressCongoBuyCreditExists"/>
<profile:hasResult rdf:resource="http://www.daml.org/services/owl-s/1.1/CongoProcess.owl#ExpressCongoBuyPositiveResult"/>
<profile:hasResult rdf:resource="http://www.daml.org/services/owl-s/1.1/CongoProcess.owl#ExpressCongoBuyNegativeResult"/>
<profile:hasOutput rdf:resource="http://www.daml.org/services/owl-s/1.1/CongoProcess.owl#ExpressCongoBuyOutput"/>
</profileHierarchy:BookSelling>

```

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

Definition of Process

A Process represents a transformation (function), characterized by four parameters

- **Inputs**: the inputs that the process requires
- **Preconditions**: the conditions that are required for the process to run correctly
- **Results**: a process may have different outcomes depending on some condition
 - **Condition**: under what condition the result occurs
 - **Outputs**: the data that results from the execution of the process
 - **Effects**: real world changes resulting from the execution of the process

Processes

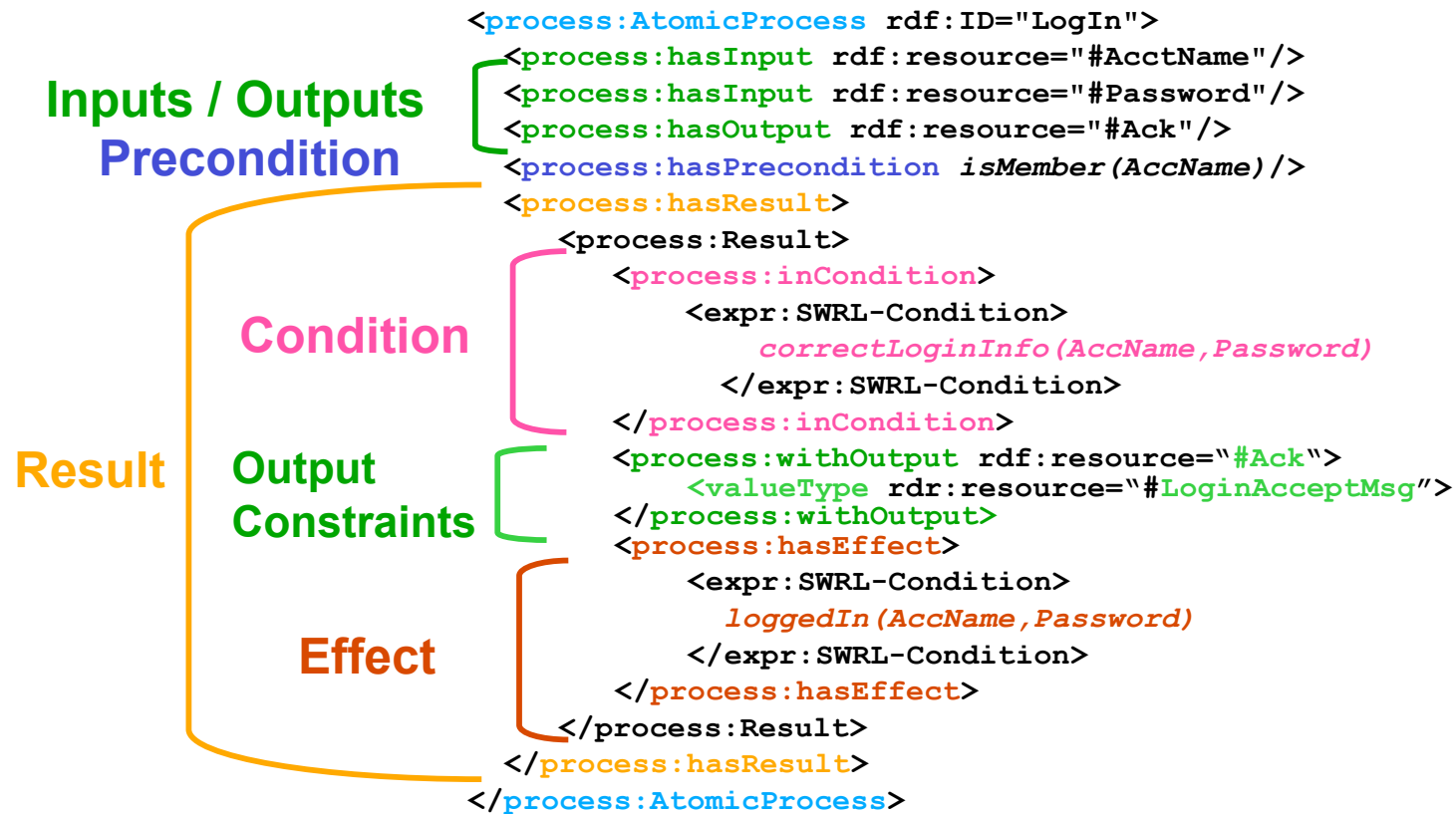
Atomic Processes

- Invocable, bound to WSDL operations

Composite Processes

- specify how processes work together to compute a complex function
- define **control flow**: specify the temporal relations between the executions of the different sub-processes
- define **data flow**: specify how the data produced by one process is transferred to another process

Example of an atomic Process



Congo Process Model

```

- <process:AtomicProcess rdf:ID="CreateAcct">
  - <process:hasInput>
    - <process:Input rdf:ID="CreateAcctInfo">
      - <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
        http://www.daml.org/services/owl-s/1.1/CongoProcess.owl#AcctInfo
      </process:parameterType>
    </process:Input>
  </process:hasInput>
  - <process:hasOutput>
    - <process:Output rdf:ID="CreateAcctOutput">
      - <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
        http://www.daml.org/services/owl-s/1.1/CongoProcess.owl#AcctID
      </process:parameterType>
    </process:Output>
  </process:hasOutput>
</process:AtomicProcess>

```

Control Flow

- Processes can be chained to form a workflow
- OWL-S supports the following control flow constructs
 - **Sequence/Unordered**: to represent a list of processes that are executed in sequence or random order
 - **Conditionals**: if-then-else statements
 - **Loops**: while and repeat-until statements
 - **Multithreading and synchronization**: split process in multiple threads, and join
 - **Non-deterministic choices**: (arbitrarily) select process from a set

Control flow Constructs

- `<process:CompositeProcess rdf:ID="CongoBuyBook">`
 - `<process:composedOf>`
 - `<process:Sequence>`
 - `<process:components rdf:parseType="Collection">`
 - `<process:CompositeProcess rdf:about="#BuySequence"/>`
 - `<process:AtomicProcess rdf:about="#SpecifyDeliveryDetails"/>`
 - `<process:AtomicProcess rdf:about="#FinalizeBuy"/>`
 - `</process:components>`
 - `</process:Sequence>`
 - `</process:composedOf>`
- `<process:CompositeProcess rdf:ID="SignInAlternatives">`
 - `<process:composedOf>`
 - `<process:Choice>`
 - `<process:components rdf:parseType="Collection">`
 - `<process:CompositeProcess rdf:about="#CreateAcctSequence"/>`
 - `<process:AtomicProcess rdf:about="#SignInSequence"/>`
 - `</process:components>`
 - `</process:Choice>`
 - `</process:composedOf>`

Data Flow

Dataflow allows information to be transferred between processes

Output→Input:

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

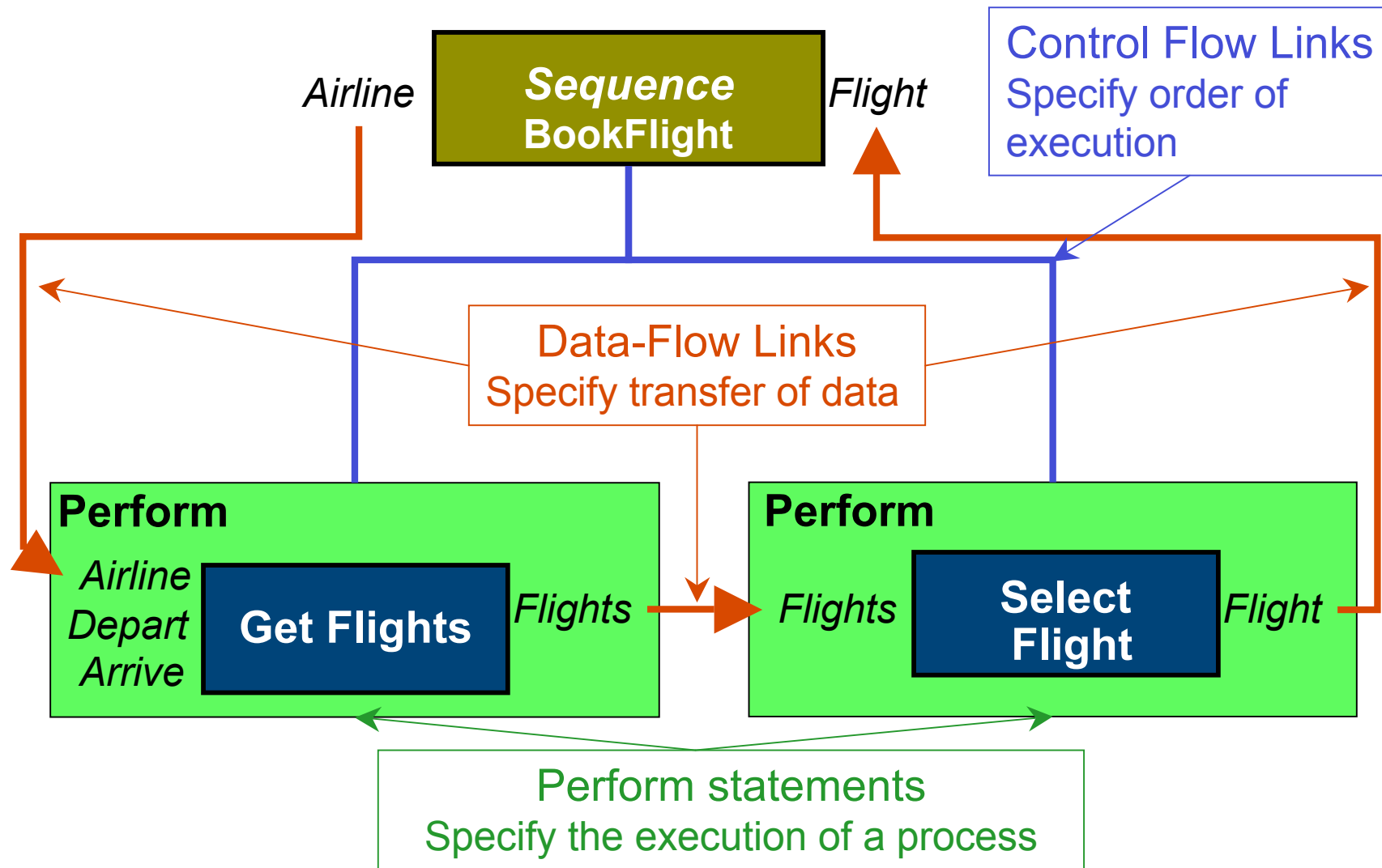
Input →Input:

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

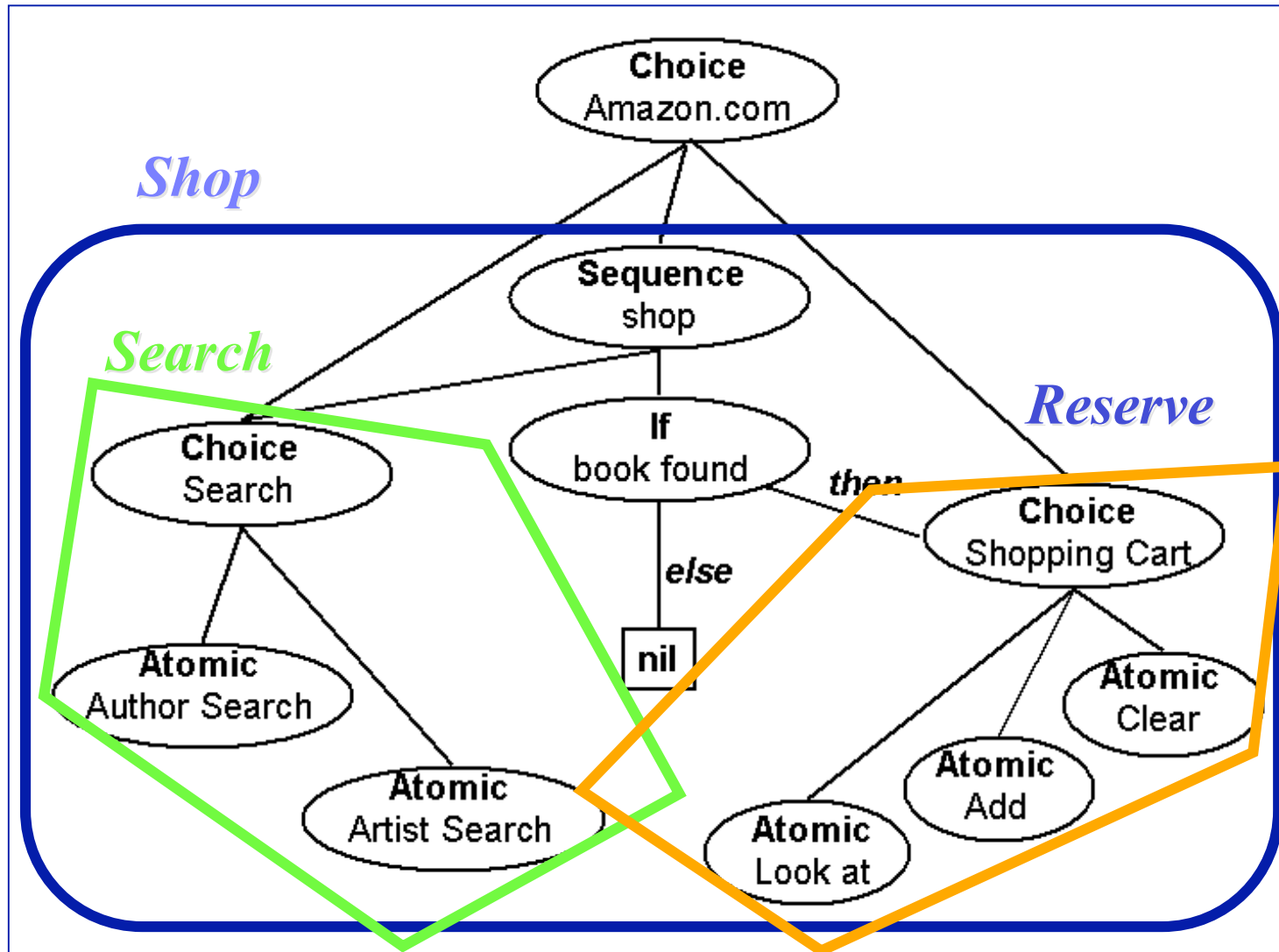
Output→Output:

The information produced by a subprocess is transferred to a super-process

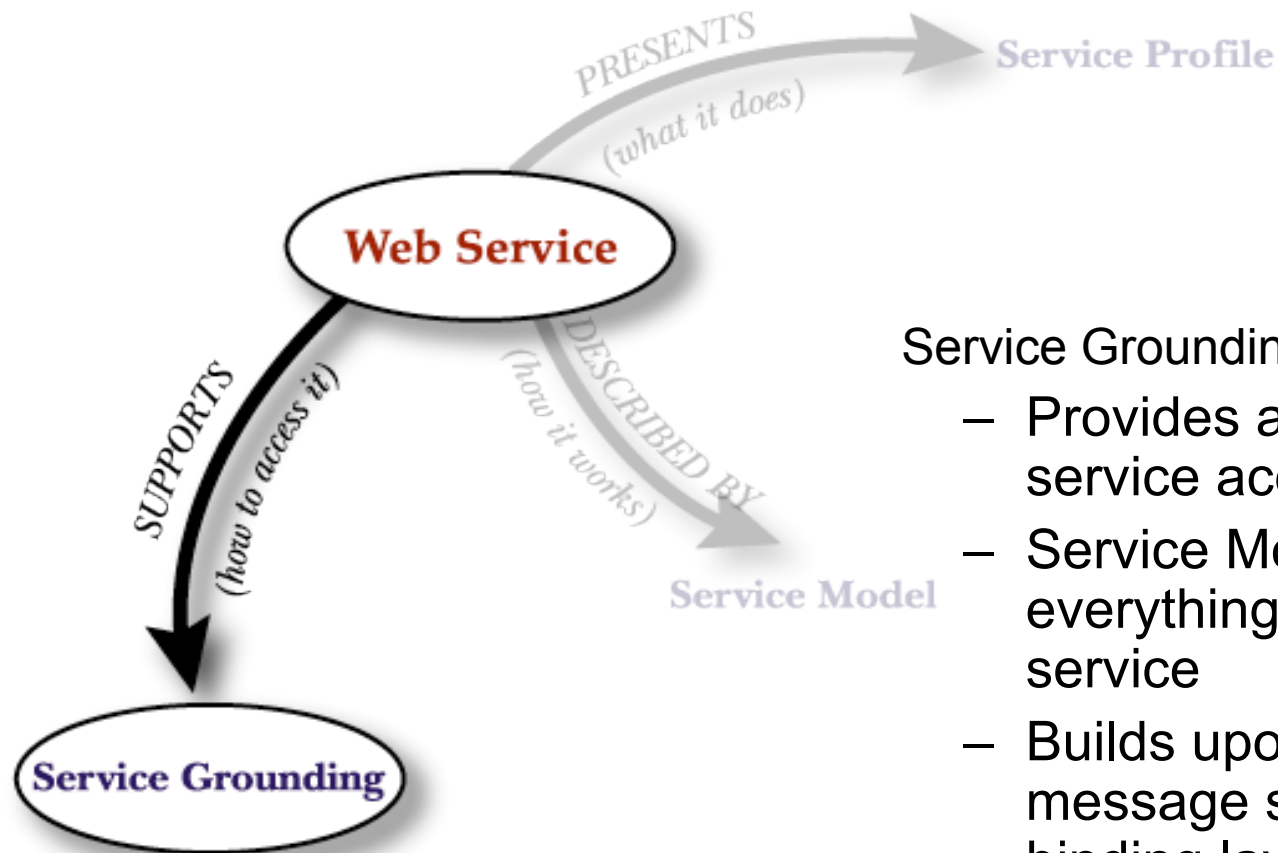
Example: Composite Process



Process Model of Amazon Web Service



Service Grounding

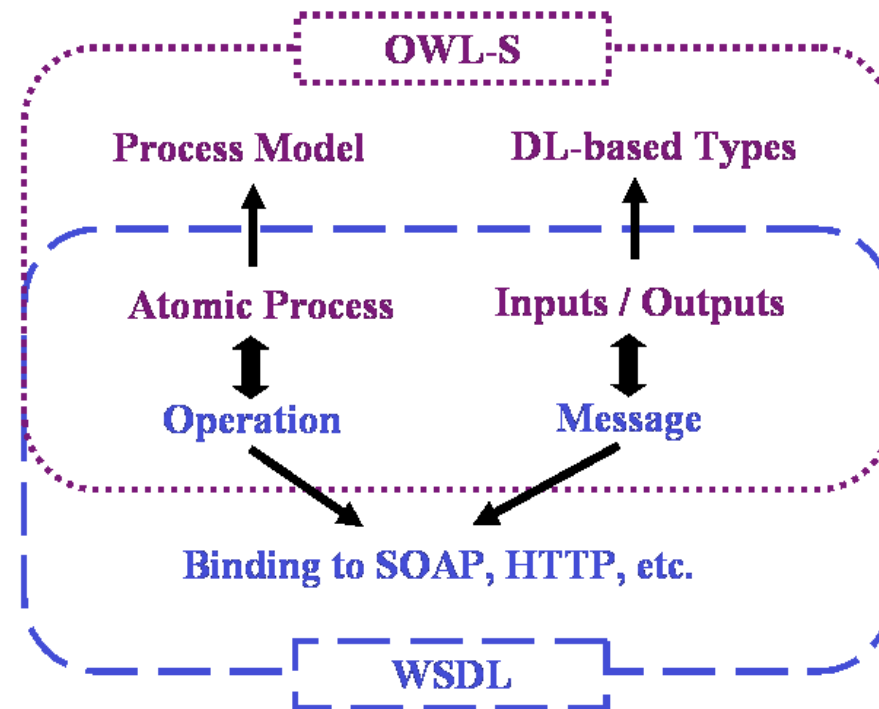


Service Grounding

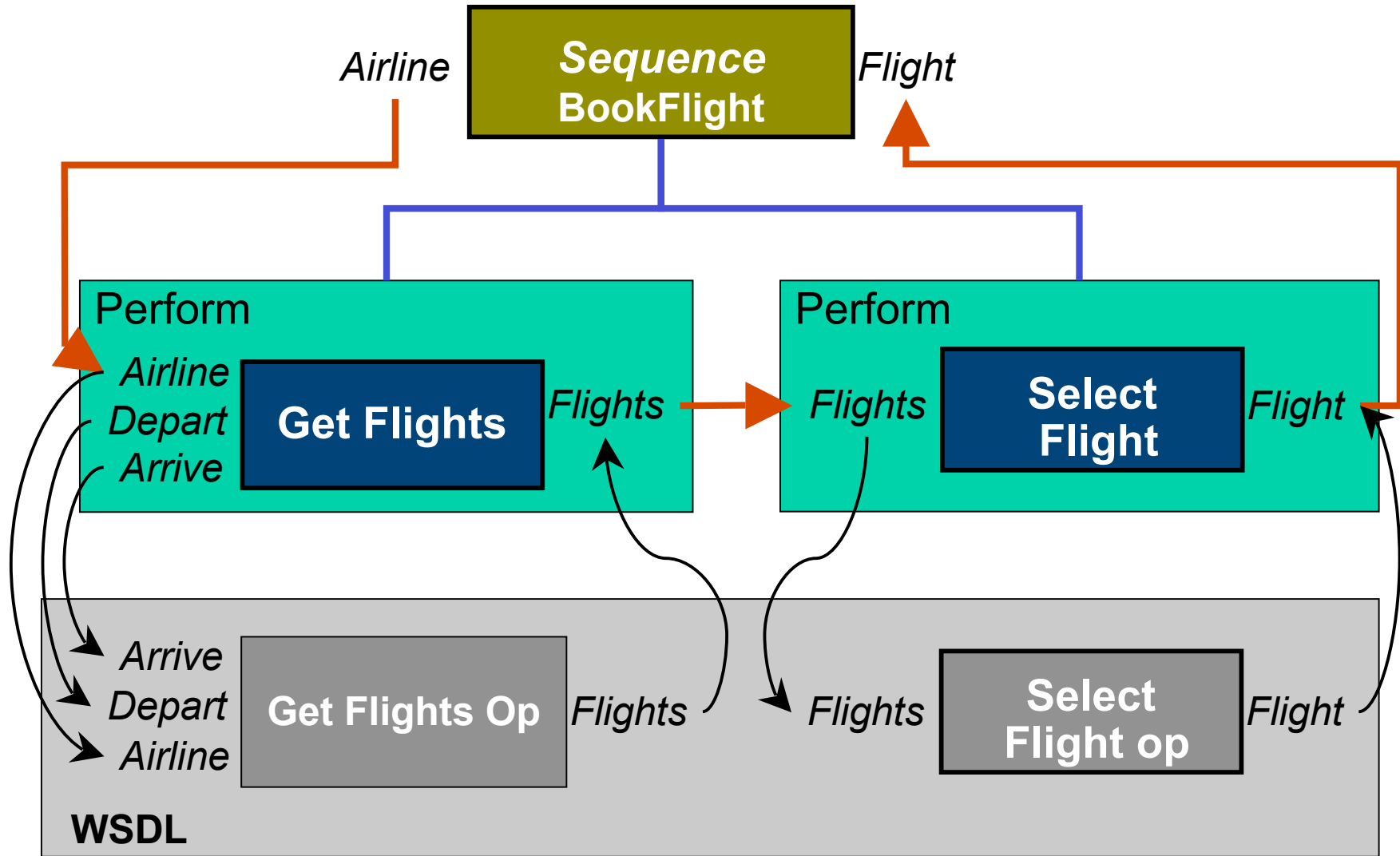
- Provides a specification of service access information.
- Service Model + Grounding give everything needed for using the service
- Builds upon **WSDL** to define message structure and physical binding layer

Mapping OWL-S / WSDL 1.1

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



Example of Grounding



Grounding

- **Invocation mechanism for OWL-S**
 - Invocation based on WSDL: map atomic processes to WSDL
 - Different types of WSDL invocations can be used with OWL-S
- **Clear separation between service description and invocation/implementation**
 - Invocation may be modified without changing semantic description
 - Service implementation may be based on SOAP and XSD types
 - Service description is needed to reason about the service
 - Decide how to use it
 - Decide how what information to send and what to expect
- **Allows any web service to be represented using OWL-S, e.g. Amazon.com**
 - without modifying the WSDL description or XSD types of the WS

Congo Grounding

```

- <grounding:WsdAtomicProcessGrounding rdf:ID="PutInCartGrounding">
  <grounding:owlsProcess rdf:resource="http://www.daml.org/services/owl-s/1.1/CongoProcess.owl#PutInCart"/>
  - <grounding:wslOperation>
    - <grounding:WslOperationRef>
      - <grounding:portType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
        http://www.daml.org/services/owl-s/1.1/CongoGrounding.wsdl#PutInCart_PortType
      </grounding:portType>
      - <grounding:operation rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
        http://www.daml.org/services/owl-s/1.1/CongoGrounding.wsdl#PutInCart_operation
      </grounding:operation>
    </grounding:WslOperationRef>
  </grounding:wslOperation>
- <grounding:wslInputMessage rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
  http://www.daml.org/services/owl-s/1.1/CongoGrounding.wsdl#PutInCart_Input
</grounding:wslInputMessage>
- <grounding:wslInput>
  - <grounding:WslInputMessageMap>
    <grounding:owlsParameter rdf:resource="http://www.daml.org/services/owl-s/1.1/CongoProcess.owl#PutInCa
    - <grounding:wslMessagePart rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
      http://www.daml.org/services/owl-s/1.1/CongoGrounding.wsdl#bookName
    </grounding:wslMessagePart>
  </grounding:WslInputMessageMap>
</grounding:wslInput>
- <grounding:wslReference rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
  http://www.w3.org/TR/2001/NOTE-wsdl-20010315
</grounding:wslReference>
- <grounding:wslDocument rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
  http://www.daml.org/services/owl-s/1.1/CongoGrounding.wsdl
</grounding:wslDocument>
</grounding:WsdAtomicProcessGrounding>

```

Congo WSDL

```

<portType name="CreateAcct_PortType">
  <operation name="CreateAcct_operation" owl-s-wsdl:owl-s-process="Congo:#CreateAcct">
    <input message="tns:CreateAcct_Input"/>
    <output message="tns:CreateAcct_Output"/>
  </operation>
</portType>

<portType name="PutInCart_PortType">
  <operation name="PutInCart_operation" owl-s-wsdl:owl-s-process="Congo:#PutInCart">
    <input message="tns:PutInCart_Input"/>
  </operation>
</portType>

<message name="PutInCart_Input">
  <part name="bookName" owl-s-wsdl:owl-s-parameter="Congo:#PutInCartBookISBN"/>
</message>

<binding name="PutInCart_SoapBinding" type="tns:PutInCart_PortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="PutInCart_operation">
    <soap:operation soapAction="tns:PutInCart"/>
    <input>
      <soap:body
        parts="bookName"
        use="encoded"
        namespace="http://www.daml.org/services/owl-s/1.0/Congo-Service.daml"/>
    </input>
  </operation>
</binding>

```

Production cycle of OWL-S WS

1. Developer generates Java code
2. Semi-automated transformation of Java into partial OWL-S description
 - WSDL is generate as by-product
3. User friendly OWL-S editor is used to complete the OWL-S description
4. UDDI client can be used for automatic advertisement in UDDI
5. Verification tools are available for correctness checking
6. Automatic client generation: integration with the OWL-S VM

Acknowledgements

- Semantic Web Services Tutorials developed by the OWL-S Coalition

END