02267: Software Development of Web Services

Week 9

Hubert Baumeister
huba@dtu.dk

Department of Applied Mathematics and Computer Science
Technical University of Denmark

Fall 2015

RESTful Web Services

Concepts

- Resource: Identified by URI: `http://localhost:8080/sr/resources/students/123`
- Representation: Mime types, e.g. application/xml, application/json, text/plain
- Uniform Interface: HTTP Verbs: GET, POST, PUT (PATCH), DELETE
- Error: HTTP Error codes
- Hypermedia: Data + Links

Contents

Error Handling

- Use of HTTP codes
- `http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html`
- 1xx Informational, e.g. 100 Continue
- 2xx Successful, e.g. 200 OK
- 3xx Redirection, e.g. 301 Moved Permanently
- 4xx Client Error, e.g. 404 Not Found
- 5xx Server Error, e.g. 500 Internal Server Error

Java and HTTP codes: service

No error reporting

```
@Path{"orders/{id}"}
public class OrderResource {
   @GET
   @Produces("application/xml")
   public Order getOrder(@PathParam("id") id) {
      return retrieveOrder(id);
   }
}
```

Control over HTTP response, e.g. status, headers, . . . use class `javax.ws.rs.core.Response`

```
@Path{"orders/{id}"}
public class OrderResource {
   @GET
   @Produces("application/xml")
   publc Response getOrder(@PathParam("id") id) {
      ...
      if (!orderExsists(id)) {
         return Response.
                  status(Response.Status.NOT_FOUND).
                  build();
      }
      Order order = retrieveOrder(id);
      return Response.ok(order).build();
   }
...
}
```

Java and HTTP codes: service

With an error message

```
@GET
@Produces("application/xml")
publc Response getOrder(@PathParam("id") id) {
   if (!correctlyFormed(id)) {
      return Response.
               status(Response.Status.BAD_REQUEST).
               entity("Id is not correctly formed","text/plain").
               build();
   }
   ...
}
```

Java and HTTP status codes: Jersey client

Catching an error

```
Client client = ClientBuilder.newClient();
WebTarget  target =
      client.target("http://localhost:8080/op/orders/45"};


orderRequest = target.path("45").request();
Response response = orderRequest.Response.cse.class);
assertEquals(404,response.getStatusCode());
```

Alternative Version

```
orderRequest = target.path("45").request();
try {
 Order response = orderRequest.get(Order.class);
 fail();
} catch (NotFoundException e) {
 assertEquals(404,e.getResponse().getStatusCode());
}
```

- General exception: javax.ws.rs.WebApplicationException
- Special subclasses for specific exceptions like NotFundException for status code 404

Java and HTTP status codes: Jersey client

Access to the HTTP response body (aka Entity)

```
Client client = ClientBuilder.newClient();
WebTarget  orders =
      client.target("http://localhost:8080/op/orders/some_id"};
Response response = orders.request().get(Response.class);
assertEquals(400,response.getStatusCode());
String errorMsg = response.getEntity(String.class);
assertEquals("Id is not correctly formed",errorMsg);
```

Contents

RESTful Web Services

- Up to now: services returned a set of data
- Now: data + links
  - links: What can I do next?

  → Next URI to call

  → Possible next step in the business process
  - Advantage: No guessing of URL's to achieve something

Example Order Process

Get an order

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<orderRepresentation>
   <link uri="http://localhost:8070/op/resources/orders/23/1"
      rel="http://orderprocess.ws/relations/self"/>
   <link uri="http://localhost:8070/op/resources/orders/23/1"
      rel="http://orderprocess.ws/relations/update"/>
   <link uri="http://localhost:8070/op/resources/orders/23/1/status"
      rel="http://orderprocess.ws/relations/status"/>
   <link uri="http://localhost:8070/op/resources/orders/23/1/payment"
      rel="http://orderprocess.ws/relations/payment"/>
   <link uri="http://localhost:8070/op/resources/orders/23/1/status"
      rel="http://orderprocess.ws/relations/cancel"/>
   <order>
      <id>1</id>
      <items>
         <amount>2</amount>
         <product>chair</product>
      </items>
      <status>ordered</status>
   </order>
</orderRepresentation>
```

Link Attributes

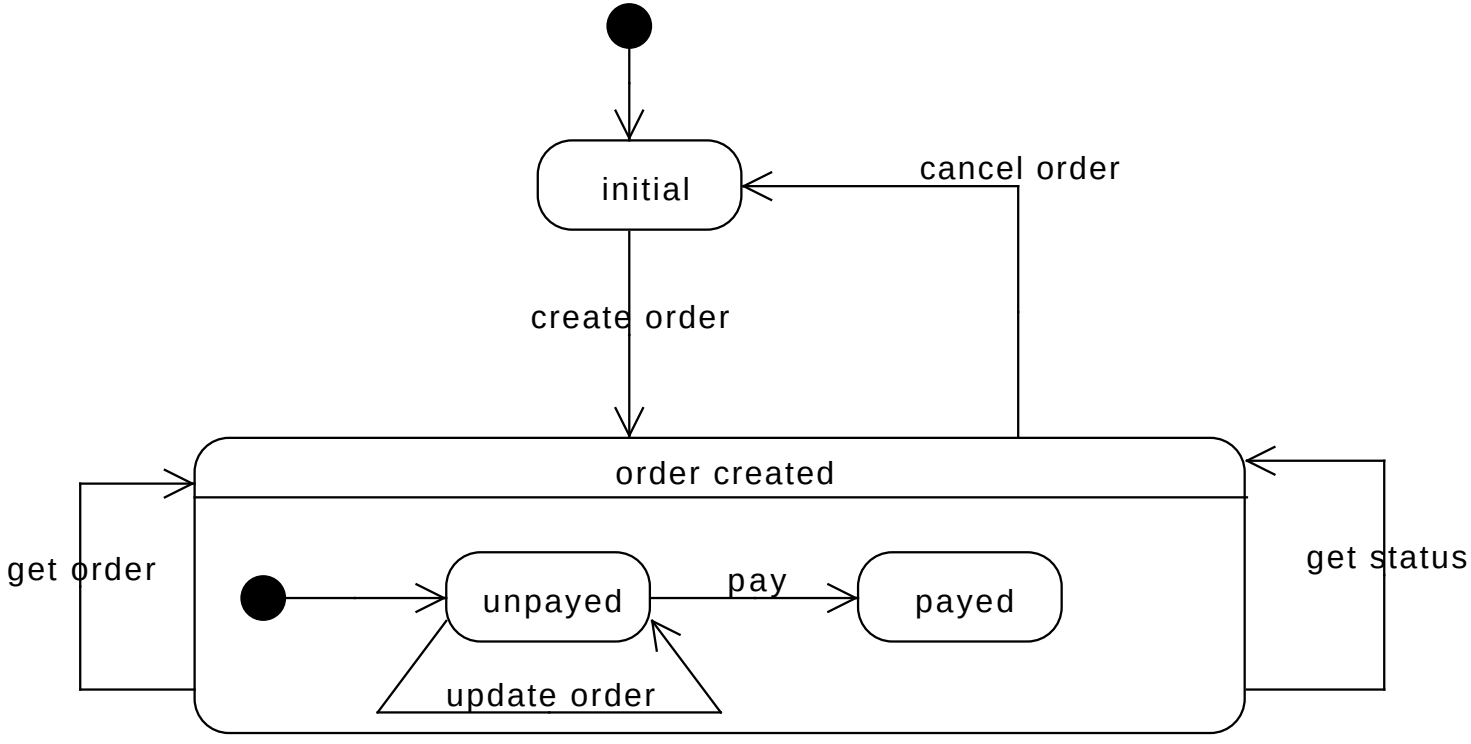
- No standard representation yet
- Possible: XHTML or domain specific XML
- Domain specific XML (cf. REST in Practice, Jim Webber et al., O'Reilly 2010)
  - URI: of a resource, e.g. `http://localhost:8080/op/resources/24/1/payment`
  - Relation: Meaning of a resource, e.g. `http://orderprocess.ws/relations/payment`

  → Which HTTP method to use with which arguments

Example: Order Process

- Operations:
  - create, update orders
  - pay for an order
  - access the status of orders
  - cancel orders

Order Process: Business process



Coordination with the client for the order process using UML state machines

1st step: Order Process: Resources

URI Path	Resource Class	HTTP Methods
orders/{cid}/{oid}	OrderResource	GET, PUT
orders/{cid}/{oid}/payment	OrderPaymentResource	PUT
orders/{cid}/{oid}/status	OrderStatusResource	GET, PUT
orders/{cid}/{oid}/reset	OrderResetResource	PUT

- Order resource @Path(”orders/cid/oid”)
  - Identification via customer id (cid) and order id (oid)
- Create order: PUT

  → Why not POST?

1st step: Order Process: Resources

URI Path	Resource Class	HTTP Methods
orders/{cid}/{oid}	OrderResource	GET, PUT
orders/{cid}/{oid}/payment	OrderPaymentResource	PUT
orders/{cid}/{oid}/status	OrderStatusResource	GET, PUT
orders/{cid}/{oid}/reset	OrderResetResource	PUT

- Order resource @Path(”orders/cid/oid”)
  - Identification via customer id (cid) and order id (oid)
- Create order: PUT
  - → Why not POST?
  - → Name is given by the client, not the service
- Update order: PUT
  - → can only be done if not already payed for

What about paying for an order?

- Uniform interface: GET, PUT, POST, DELETE

What about paying for an order?

- Uniform interface: GET, PUT, POST, DELETE
- Transform actions into resources that can be manipulated by the uniform interface
  - E.g. Payment
    - Payment resource @Path(”orders/cid/oid/payment”)
    - PUT payment information to this resource (i.e. creditcard information) for payment

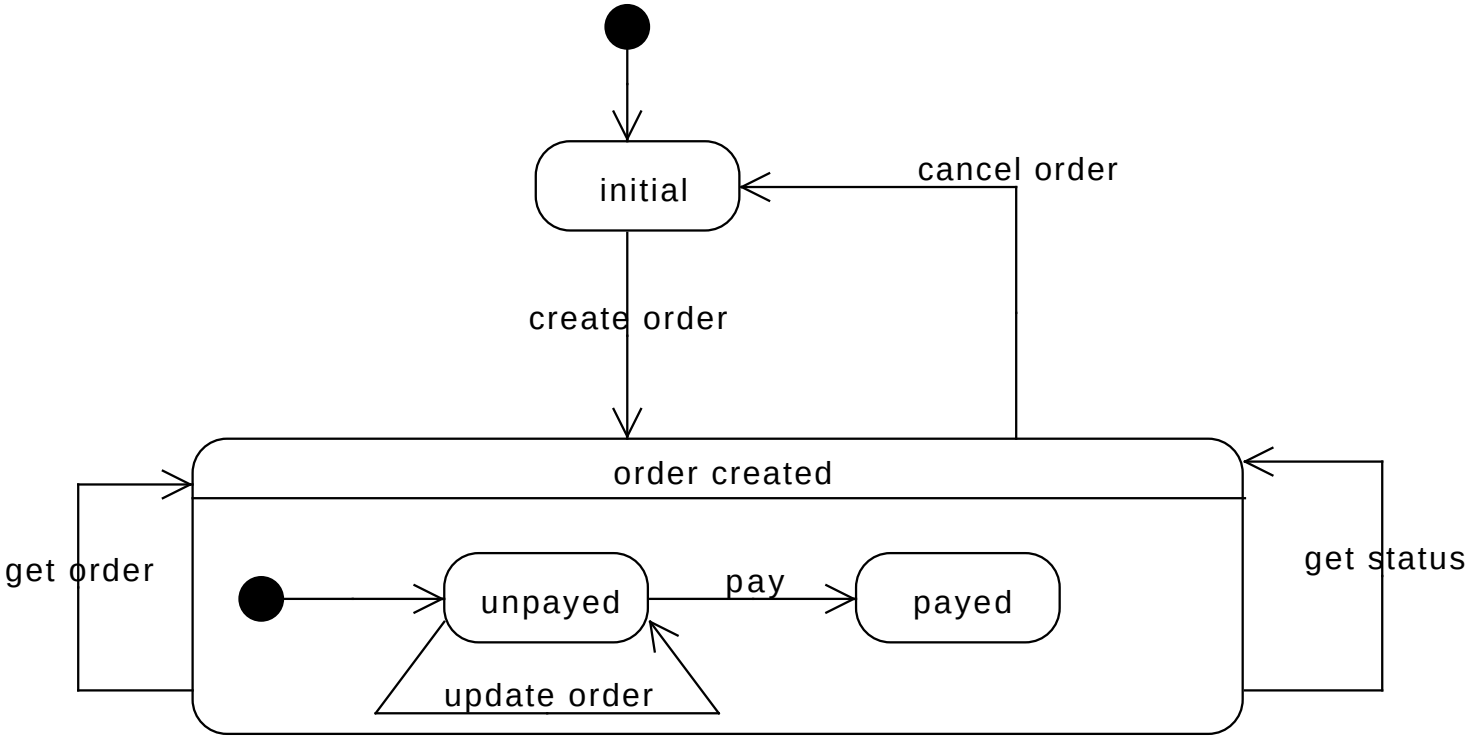
Accessing the status and cancelling

- Access to the status is again a resource @Path(”orders/cid/oid/status”)
- GET: returns the current status of the order
- PUT: cancels order if the new status is ”cancelled”; ignored otherwise

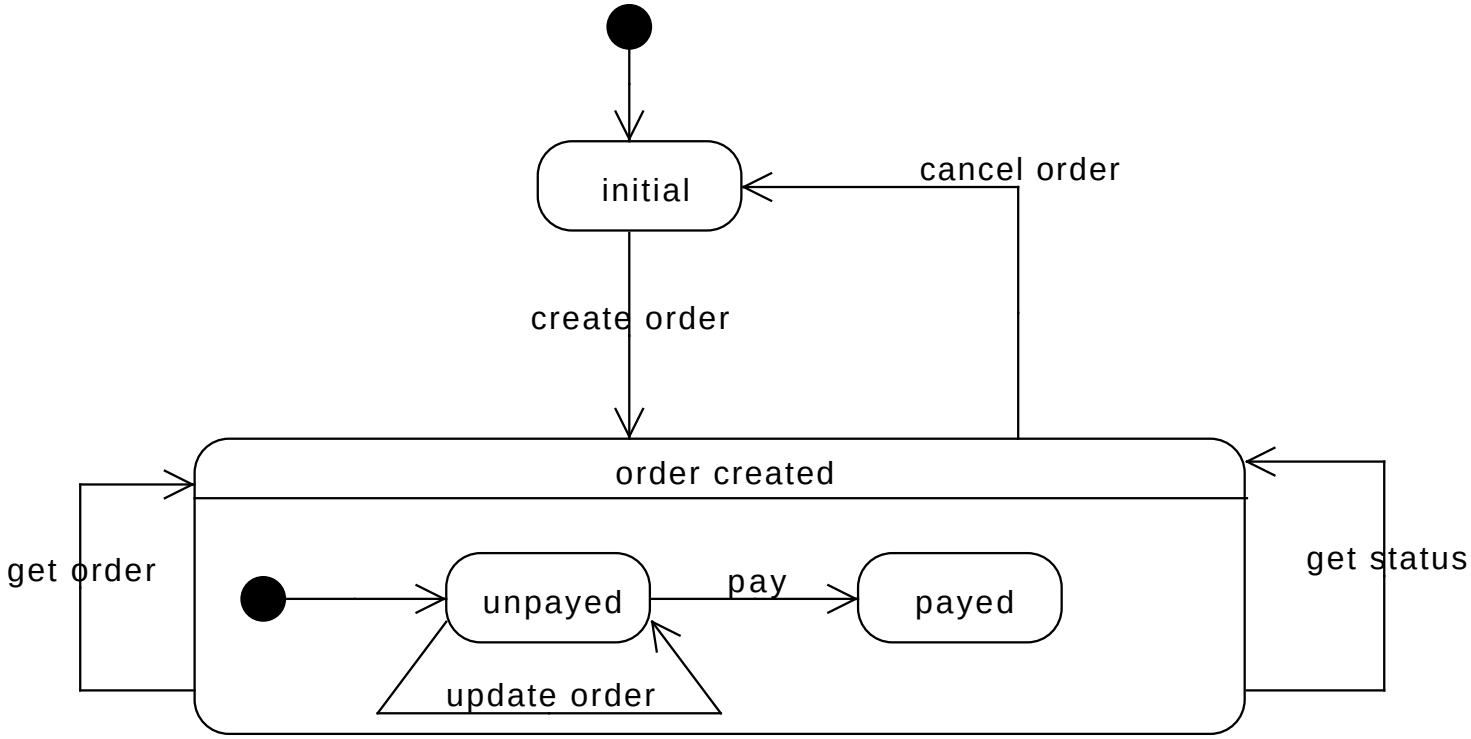
Additional resource for testing

- PUT on @Path(”orders/reset”)
  - resets the orders database
  - called before each test method is run using @Before annotation of JUnit

2nd Step: Define the next steps

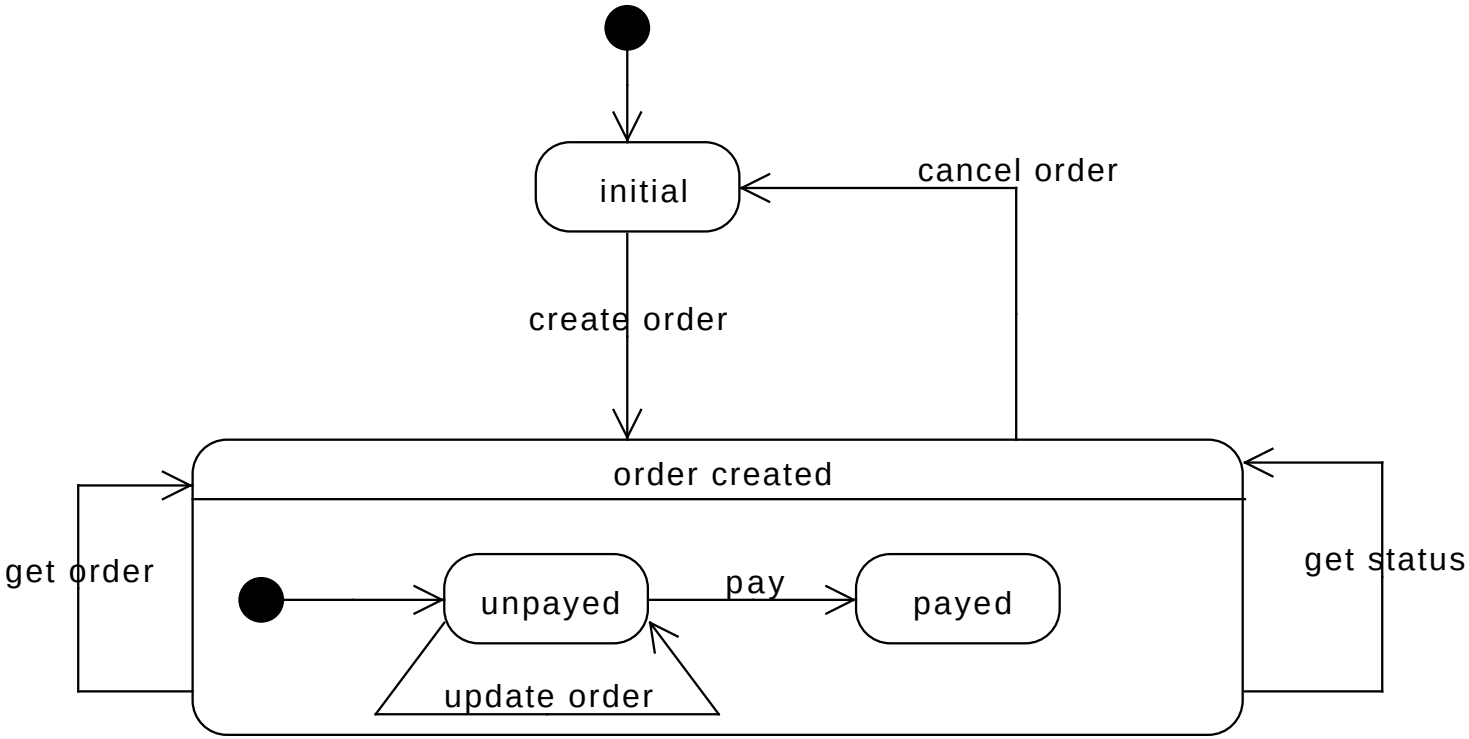


2nd Step: Define the next steps



- After create order one can:
  - access the status
  - get the order
  - update the order
  - pay
  - cancel the order

Links: Next states in the process



- After payment one can
  - access the status
  - get the order
  - cancel the order
  - Not possible anymore: pay and update order

Implementation in Java:

- Abstract class Representation: manages links
- Special representations: data + links
  - StatusRepresentation: Links + status information
  - OrderRepresentation: Links + order object
- E.g. getOrder returns a OrderRepresentation; getStatus a StatusRepresentation

Following links in Java:

```
OrderRepresentation result =
      client.resource("http://localhost:8080/op/resources/23/1).
         get(OrderRepresentation.class);

Link link = result.
      getLinkByRelation("http://orderprocess.ws/relations/cancel");

OrderRepresentation orderresult = client.resource(link.getUri()).
                accept(MEDIATYPE_ORDERPROCESS).
                put(OrderRepresentation.class, "cancelled");
```

- What one can do with a links URI is determined by its relation

→ Part of the (informal) documentation of a REST service

3rd Step: Define relations in order process

- ”http://orderprocess.ws/relations/self”: GET on URI returns the order
- ”http://orderprocess.ws/relations/update”: PUT with a new order updates the order
- ”http://orderprocess.ws/relations/status”: GET on URI returns only the status of the order
- ”http://orderprocess.ws/relations/payment”: PUT with credit card information submits payment for the order
- ”http://orderprocess.ws/relations/cancel”: PUT with ”cancelled” cancels the order

Use of a special mime type:

- @Produces/@Consumes(”application/orderprocess+xml)
- Mime type: application/orderprocess+xml instead of application/xml

→ New mime type indicates that messages have a specific form:

  - contain links of a specific form
  - use relationship attributes with an application specific semantics

Contents

WADL Web Application Definition Language

- Abstract description of REST services
  - → same function as WSDL for Web services based on SOAP
    - WSDL 2.0 has also support for RESTful Web services
- `http://www.w3.org/Submission/wadl/`

Order Process WADL

Generated by Jersey

`http://localhost:8080/op/webresources/application.wadl`

```
- <application>
    <doc jersey:generatedBy="Jersey: 1.0.3.1 08/14/2009 04:21 PM"/>
  - <resources base="http://127.0.0.1:8070/op/webresources/">
    + <resource path="orders/reset"></resource>
    + <resource path="orders/{cid}/{oid}/payment"></resource>
    - <resource path="orders/{cid}/{oid}">
        <param name="oid" style="template" type="xs:string"/>
        <param name="cid" style="template" type="xs:string"/>
      - <method id="createOrder" name="PUT">
        - <request>
            <representation mediaType="application/orderprocess+xml"/>
          </request>
        - <response>
            <representation mediaType="application/orderprocess+xml"/>
          </response>
        </method>
      - <method id="getOrder" name="GET">
        - <response>
            <representation mediaType="application/orderprocess+xml"/>
          </response>
        </method>
      </resource>
    + <resource path="orders/{cid}/{oid}/status"></resource>
    </resources>
  </application>
```

Client and Server side datatypes

- For REST and SOAP based Web services, domain specific datatypes need to be represented both on the server and the client side
- WSDL
  - Client (and server) versions of datatypes are generated from WSDL with wsimport
- RESTful Web services
  - WADL
    - can contain information about representations (i.e. type information) but does not require it
    - $\rightarrow$ need to add datatypes by hand for the client (e.g. generated from XML Schema)
  - No WADL
    - $\rightarrow$ need to add datatypes by hand for the client (e.g. generated from XML Schema)

Difference to SOAP based Web services

- REST Web services
  - use the concept of resources
  - rely and use the semantics of the HTTP protocol (HTTP verbs, media types, HTTP status codes)
  - are tailored to HTTP and Web architecture: e.g. caching using proxies
  - different types of resource representations: text XML, JSON, HTML, binary, . . .
- SOAP based Web services
  - use the concept of services
  - HTTP is one possible transport protocol, but SOAP does not rely on HTTP
  - HTTP verbs are irrelevant; the semantic of an operation is found in the SOAP message (e.g. register student)
  - One representation for data: XML defined using XML schema

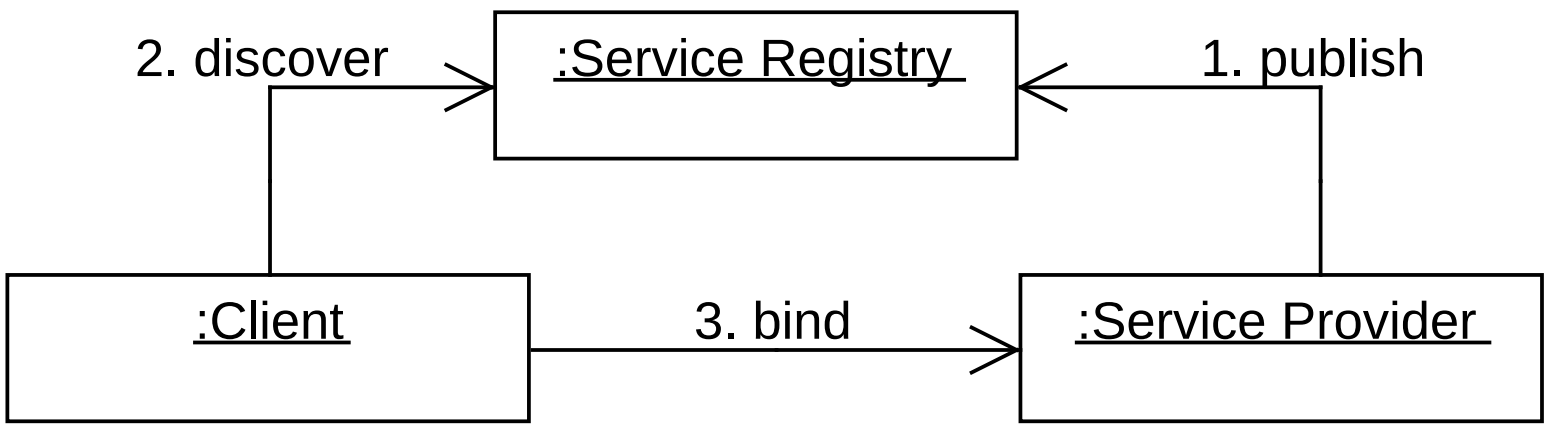
Contents

Problem

- Loose coupling
  - $\rightarrow$ Loose coupling of Web services
  - $\rightarrow$ Searching for the best service for a task
- Interaction across businesses but also within businesses

$\rightarrow$ Need for Web service discovery

- Manually: Design time discovery or static binding
- Programmatically: Run time or dynamic binding



Solutions

- UDDI
  - Global solution
  - Based on a API (defined through WSDL)
- WSIL
  - Distributed solution
  - Based on XML files

Universal Description, Discovery and Integration (UDDI)

- Basic idea: Universal Business Registry (UBR)
  - One global registry
  - Companies register themselves with the UBR
  - Companies register services they offer
    - Web services and other types of services
    - → Does not store WSDL file or other service descriptions only URL's where to find them
- Information stored in a UDDI registry:
  - White Pages: organisations / business entities
  - Yellow Pages: search by these categories
  - Green Pages: search by service description
- OASIS specification (`http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm`)
  - UDDI 1.0 (2000), UDDI 2.0 (2002), UDDI 3.0 (2005)

UDDI API

- API is accessible via SOAP
- Two types / two endpoints
  - Inquiry API
  - Publisher API

Inquiry API: Drill-down pattern

- Find Operations
  - Business, related_businesses, service, binding, tModel
  - Can be searched by name, CategoryBag etc.

  $\rightarrow$ Browse pattern
    - Look for certain information
    - Business, service, template, tModel, related businesses
- Get Operations
  - BindingDetail, businessDetail, businessDetailExt, serviceDetail, tModelDetail
  - Is accessed by a unique key
    - Use get details on some entity
    - business detail, service detail, binding detail, tModel detail, ...

Example: Find Business Request

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <SOAP-ENV:Body>
    <find_business generic="2.0" xmlns="urn:uddi-org:api_v2">
    <name>ms.com</name>
    </find_business>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example: Find Business Response

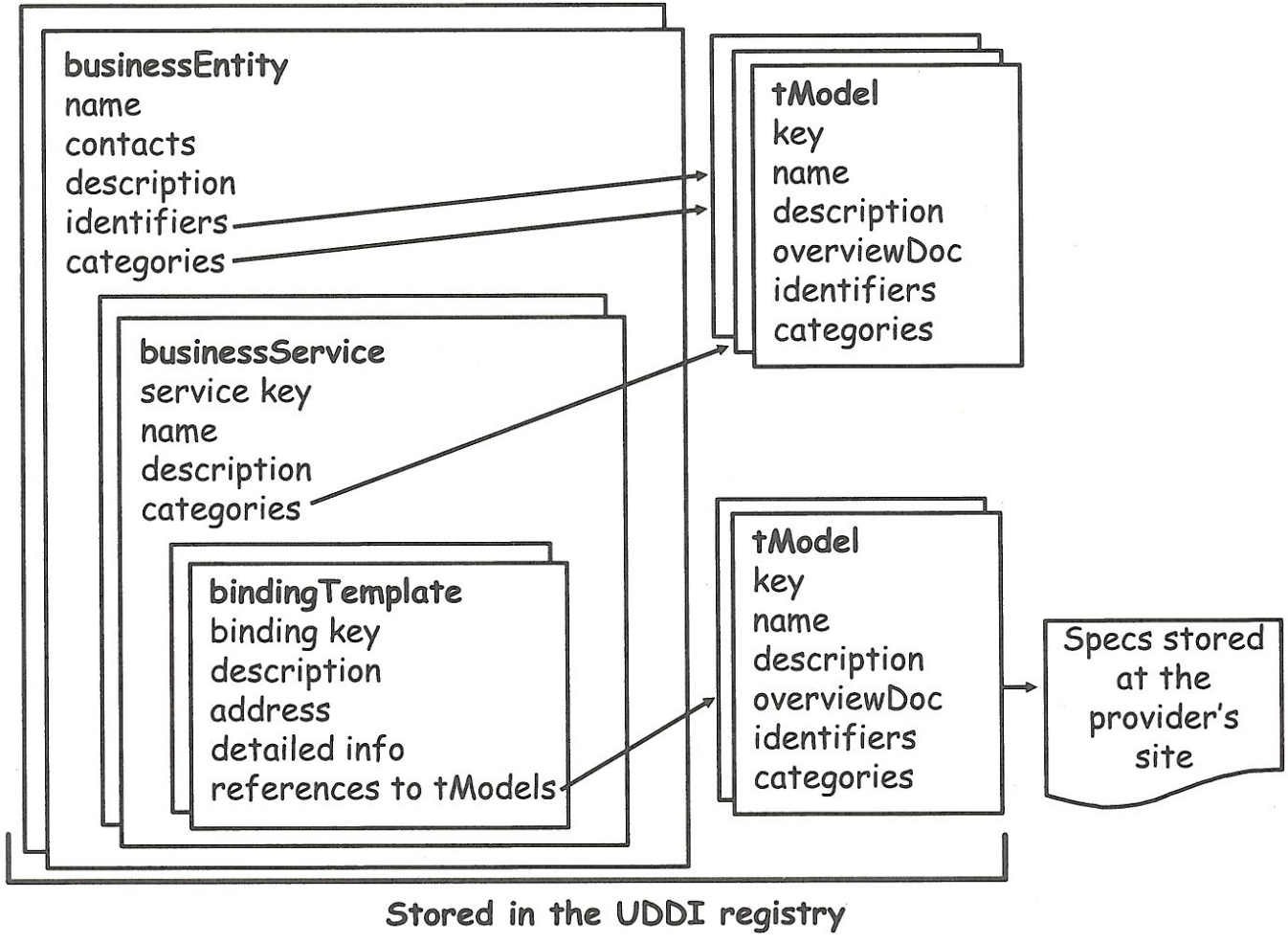
```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <soap:Body>
  <businessList generic="2.0" operator="ms.com"
    truncated="false" xmlns="urn:uddi-org:api_v2">
   <businessInfos>
    <businessInfo businessKey="309d717f-1d74-4334-8188-9c069dda0c76">
     <name xml:lang="en">ms.com</name>
     <description xml:lang="en">
         A Microsoft UDDI Services site.
     </description>
     <serviceInfos>
      <serviceInfo serviceKey="8bf635e2-1b0e-4e18-ab8b-aeb89ca08abd"
       businessKey="309d717f-1d74-4334-8188-9c069dda0c76">
       <name xml:lang="en">UDDI Services</name>
      </serviceInfo>
     </serviceInfos>
    </businessInfo>
   </businessInfos>
  </businessList>
 </soap:Body>
</soap:Envelope>
```

Example: Get Business Detail

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <SOAP-ENV:Body>
  <get_businessDetail generic="2.0" xmlns="urn:uddi-org:api_v2">
   <businessKey>309d717f-1d74-4334-8188-9c069dda0c76</businessKey>
  </get_businessDetail>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- This returns a business entity

Datastructures overview



Alonso et al, Web Services

Problems

- UDDI = Universal Business Registry (UBR)
  - UDDI Universal Business Registries hosted by IBM, Microsoft, and SAP (replicated); registry is discontinued from 2006
- Problems of a global registry
  - How does one trust these services? (e.g. credit card payment service)
  - What actually do these services?
  - Use of services often require a contract between users and suppliers
  - Quality of the registered information
  - Old information
- Instead: private / semi-private / business internal UDDI registries
  - UDDI 3.0 introduces a distributed registry
  - Use for intranet and business to business applications
- Alternative approach: distributed registry
  - $\rightarrow$ WSIL (Web Service Inspection Language)

Contents

Web-Service Inspection Language (WSIL)

- A decentralized approach for managing and discovery of Web services
- Local scope of the service definitions
  - But links to other service registries (WSIL and UDDI) are possible
- Local management $\rightarrow$ improved quality of the information
- Based on an XML format by IBM and Microsoft
  - Client has to search the XML directly; no *API*
- WSIL 1.0:
  `http://www-128.ibm.com/developerworks/library/specification/ws-wsilspec/`

Example: WSIL

```
<inspection xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/"
  xmlns:wsiluddi=
         "http://schemas.xmlsoap.org/ws/2001/10/inspection/uddi/">
 <service>
  <abstract>A stock quote service with two descriptions</abstract>
  <description referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
         location="http://example.com/stockquote.wsdl"/>
  <description referencedNamespace="urn:uddi-org:api">
    <wsiluddi:serviceDescription
       location="http://www.example.com/uddi/inquiryapi">
     <wsiluddi:serviceKey>
       4FA28580-5C39-11D5-9FCF-BB3200333F79
     </wsiluddi:serviceKey>
    </wsiluddi:serviceDescription>
  </description>
 </service>
 <service>
  <description referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
         location="ftp://anotherexample.com/tools/calculator.wsdl"/>
 </service>
 <link
    referencedNamespace=
      "http://schemas.xmlsoap.org/ws/2001/10/inspection/"
    location="http://example.com/moreservices.wsil"/>
</inspection>
```

WSIL Document

- Services
  - Format independent information
    - Short description
    - Where to reach
    - Where is additional information about the service
  - Format dependent information (bindings)
    - UDDI, WSDL, other bindings possible
- Links to other registries (WSIL, UDDI, . . .)

<wsil:link>

”Links” contain references to other service registries

- Link to WSIL registry

```
<link
 referencedNamespace=
        "http://schemas.xmlsoap.org/ws/2001/10/inspection/"
 location="http://example.com/moreservices.wsil"/>
```

- Link to UDDI registry

```
<link referencedNamespace="urn:uddi-org:api">
 <wsiluddi:businessDescription
      location="http://www.example.com/uddi/inquiryapi">
 <wsiluddi:businessKey>3C9CADD0-5C39-11D5-9FCF-BB3200333F79
 </wsiluddi:businessKey>
 <wsiluddi:discoveryURL useType="businessEntity">
   http://www.example.com/uddi?3C9CADD0-5C39-11D5-9FCF-BB3200333F79
 </wsiluddi:discoveryURL>
 </wsiluddi:businessDescription>
</link>
```

Where to find WSIL Documents

- Inspection.wsil at standard location
  - `inspection.wsil` at the root of the Web server
  - e.g. `http://www.example.com/inspection.wsil`
  - Using the META tag in an HTML document, e.g. on the homepage of a company

```
<HTML>
 <HEAD>
  <META name="serviceInspection"
    content="localservices.wsil">
  <META name="serviceInspection"
    content="http://www.example.com/calculators.wsil">
  <META name="serviceInspection"
    content="ftp://www.anotherexample.com/translators.wsil">
 </HEAD>
 <BODY>
...
 </BODY>
</HTML>
```

Summary

- RESTful Web services
  - Use of HTTP Status Codes
  - Implementing Business Logic
  - WADL
- SOAP based Web services
  - Web Service Discovery: UDDI, WSIL