

Middleware and Web Services

Lecture 9: Web Service Description Language

doc. Ing. Tomáš Vitvar, Ph.D.

tomas@vitvar.com • @TomasVitvar • <http://vitvar.com>



Czech Technical University in Prague

Faculty of Information Technologies • Software and Web Engineering • <http://vitvar.com/courses/mdw>



Modified: Sat Nov 22 2014, 09:27:30
Humla v0.3

Overview

- **Web Service Description Language**
 - *Elements, Types and Messages Definitions*
 - *Interface and Operations*
 - *Binding*
 - *Service and Endpoint*
 - *Description*
- WS-Addressing

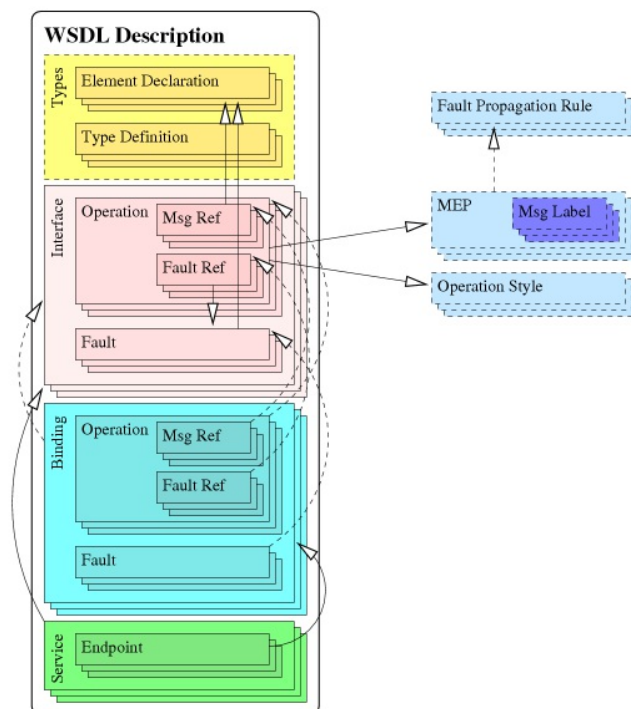
Specifications

- WSDL = Web Service Description Language
 - *A standard that allows to describe Web services explicitly (main aspects)*
 - *A contract between a requester and a provider*
- Specifications
 - *WSDL 1.1 – still widely used*
 - *Web Service Description Language 1.1* [🔗](#)
 - *WSDL 2.0 – An attempt to address several issues with WSDL 1.1*
 - *SOAP vs. REST, naming, expressivity*
 - *WSDL 2.0 Primer (part 0)* [🔗](#)
 - *WSDL 2.0 Core Language (part 1)* [🔗](#)

WSDL Overview and WSDL 1.1 Syntax

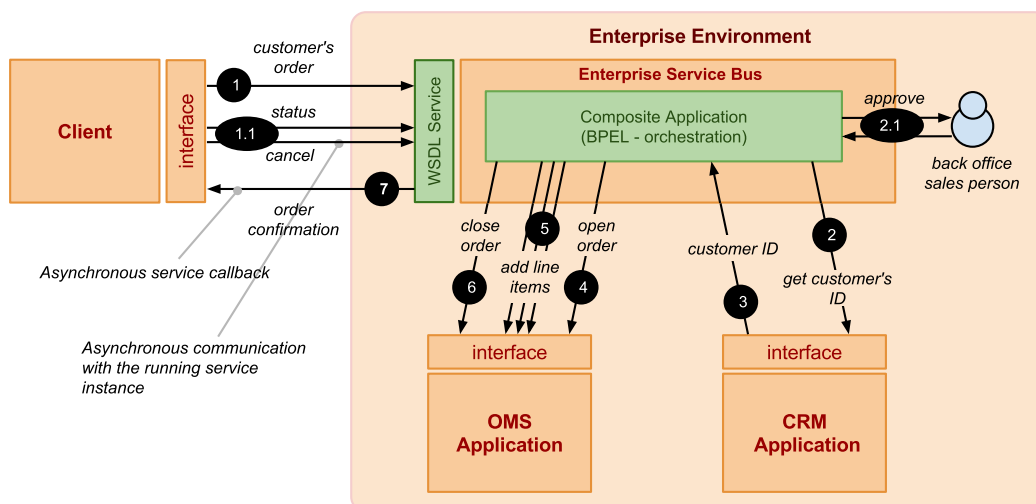
- Components of WSDL
 - *Information model (**types**)*
 - *Element types, message declarations (XML Schema)*
 - *Set of operations (**portType**)*
 - *A set of operations is "interface" in the WSDL terminology*
 - *operation name, input, output, fault*
 - *Binding (**binding**)*
 - *How messages are transferred over the network using a concrete transport protocol*
 - *Transport protocols: HTTP, SMTP, FTP, JMS, ...*
 - *Endpoint (**service**)*
 - *Where the service is physically present on the network*
- Types of WSDL documents
 - **Abstract WSDL** – *only information model and a set of operations*
 - **Concrete WSDL** – *everything, a concrete service available in the environment*

WSDL Components and Dependencies



Example

- Simple order process example



Top-level Element and Namespaces

- Example

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <definitions
3    xmlns="http://schemas.xmlsoap.org/wsdl/"
4    targetNamespace="http://mimdw.fit.cvut.cz/mdw_examples/ProcessOrder/OrderProc
5    xmlns:om="xmlns:om="http://mimdw.fit.cvut.cz/mdw-examples/cdm/order"
6    xmlns:tns="http://mimdw.fit.cvut.cz/mdw_examples/ProcessOrder/OrderProcess">
7    name="OrderProcess"
8    ...
9  </definitions>
```

- **definitions** is a top-level element
- Any un-prefixed elements will be WSDL 2.1 elements (line 3)
- Target namespace (line 4)
 - a vocabulary for the order service's parts that this WSDL document defines (it is not a XML namespace declaration!)
 - **xmlns:tns** (line 5) is a prefix definition for this namespace

Overview

- Web Service Description Language
 - *Elements, Types and Messages Definitions*
 - *Interface and Operations*
 - *Binding*
 - *Service and Endpoint*
 - *Description*
- WS-Addressing

Type Definitions

- XML Schema
 - Only XML Schema in WSDL 1.1
 - WSDL 2.0 allows to use other languages too
- Types for input, output, fault messages
- Single elements at the topmost level
 - may contain arbitrary structure
- Could be defined
 - directly inside the **types** element
 - externalally by using XML Schema **import** mechanism
 - Types to be reused in multiple WSDLs
 - Definitions of Common Data Model

Elements and Types Definitions Example

```
1  <types>
2    <schema xmlns="http://www.w3.org/2001/XMLSchema"
3      attributeFormDefault="unqualified"
4      elementFormDefault="qualified"
5      xmlns:op="http://mimdw.fit.cvut.cz/mdw-examples/cdm/order"
6      targetNamespace="http://.../mdw_examples/ProcessOrder/OrderProcess">
7
8      <import
9        namespace="http://mimdw.fit.cvut.cz/mdw-examples/cdm/order"
10       schemaLocation="http://.../soa-infra/services/mdw-examples/ProcessOrder/apps/MDW
11       <element name="OrderProcessRequest" type="op:Order"/>
12       <element name="OrderProcessResponse" type="string"/>
13       <element name="StatusRequest" type="op:StatusRequestType"/>
14       <element name="FaultMessage" type="string"/>
15       <complexType name="StatusRequestType">
16         <sequence>
17           <element name="process-id" type="string"/>
18         </sequence>
19       </complexType>
20       <element name="StatusResponse" type="op:StatusResponseType"/>
21       <complexType name="StatusResponseType">
22         <sequence>
23           <element name="process-id" type="string"/>
24           <element name="status" type="string"/>
25         </sequence>
26       </complexType>
27       <!-- [snip] -->
28     </schema>
29 </types>
```

Messages Definitions

- Definitions of messages
 - Messages flow between a client and a service
 - They can be request, response or fault
 - Each message has one or more "parts"
 - A message part references a single element defined in **types**
- Example messages definitions for order process
 - A name can be arbitrary name but a commonly agreed convention is:
{ServiceName}{Request|Response|Fault}Message

```
1 <message name="OrderProcessRequestMessage">
2   <part name="order" element="op:OrderProcessRequest"/>
3 </message>
4 <message name="OrderStatusResponseMessage">
5   <part name="status" element="op:StatusResponse"/>
6 </message>
7 <message name="OrderProcessFaultMessage">
8   <part name="text" element="op:FaultMessage"/>
9 </message>
10 <message name="OrderProcessResponseMessage">
11   <part name="result" element="op:OrderProcessResult"/>
12 </message>
```

Overview

- Web Service Description Language
 - Elements, Types and Messages Definitions
 - **Interface and Operations**
 - Binding
 - Service and Endpoint
 - Description
- WS-Addressing

Defining Interface

- Interface
 - *abstract description of a service*
 - *separation of*
 - *abstract functionality (interface definition)*
 - *and concrete details on how and where the functionality is offered*
- WSDL Interface
 - *A set of operations implemented by a service or a client*
 - *The client may implement the interface for asynchronous callbacks*
- Each operation specifies
 - *references to messages (input, output, fault)*
 - *Exchange pattern*
 - **Request-response** – *the most commonly used pattern*
 - **One-way** (also called *fire-and-forget*) – *only request from the client*
 - **Solicit-response** – *response-request*
 - **Notification** – *response-only, used in asynchronous callbacks*

Interface Example (1)

- Order process complex conversation
 1. The client invokes **processOrder**.
 2. The service responds back **synchronously** with order status.
 3. The client gets the status of order processing by invoking synchronous **getStatus** operation (this can be invoked several times).
 4. The service responds back **asynchronously** by invoking **processOrderResponse** – callback on client's interface
- Interface implemented by the order process service
 - **getStatus** operation must be executed in the same **conversation** as **processOrder** operation

```
1  <portType name="OrderProcess">
2    <operation name="processOrder">
3      <input message="op:OrderProcessRequestMessage"/>
4      <output message="op:OrderStatusResponseMessage"/>
5    </operation>
6    <operation name="getStatus">
7      <input message="op:OrderStatusRequestMessage"/>
8      <output message="op:OrderStatusResponseMessage"/>
9    </operation>
10 </portType>
```

Interface Example (2)

- Interface implemented by the client

```
1 <portType name="OrderProcessCallback">
2   <operation name="processOrderResponse">
3     <input message="op:OrderProcessResponseMessage"/>
4     <fault message="op:OrderProcessFaultMessage"/>
5   </operation>
6 </portType>
```

Overview

- Web Service Description Language
 - *Elements, Types and Messages Definitions*
 - *Interface and Operations*
 - *Binding*
 - *Service and Endpoint*
 - *Description*
- WS-Addressing

Defining a Binding

- Binding is part of a concrete WSDL
 - A WSDL exposed by a service deployed to application server or ESB
 - One binding for one interface (**portType**)
- "How" messages can be exchanged
 - one interface may have one or more bindings
 - such as one for SOAP over HTTP and one for SOAP over SMTP
- Binding specifies
 - details for every operation and fault in the interface
 - concrete message format and transmission protocol
 - rules for SOAP headers
 - (policies, e.g. ws-addressing for asynchronous communication)
- SOAP binding styles
 - RPC/encoded
 - RPC/literal
 - Document/encoded – nobody uses this one
 - Document/literal
 - Document/literal wrapped

Binding Example – HTTP Transport

- Binding to SOAP over HTTP

```
1  <binding name="OrderProcessBinding" type="op:OrderProcess">
2    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
3    <PolicyReference xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/polic
4      URI="#wsaddr_policy" wsdl:required="false"/>
5    <operation name="processOrder">
6      <soap:operation style="document" soapAction="processOrder"/>
7      <input>
8        <soap:body use="literal"/>
9      </input>
10     <output>
11       <soap:body use="literal"/>
12     </output>
13   </operation>
14   <!-- snip -->
15 </binding>
```

- HTTP transport defined by
 - URI **http://schemas.xmlsoap.org/soap/http** of **transport** attribute on **binding** element (line 2)
- A style **document/literal** defined by:
 - **style** attribute of **soap:operation** (line 6)
 - **use** attribute of **soap:body** (lines 8,11)

SOAP Binding – RPC/encoded

- WSDL

```
1 <message name="OrderStatusRequestMessage">
2   <part name="process-id" type="xsd:string"/>
3 </message>
4
5 <portType name="OrderProcess">
6   <operation name="getStatus">
7     <input message="OrderStatusRequestMessage"/>
8     <output message="..." />
9   </operation>
10 </portType>
```

- SOAP Request

```
1 <soap:envelope>
2   <soap:body>
3     <getStatus>
4       <process-id xsi:type="xsd:string">5</process-id>
5     </getStatus>
6   </soap:body>
7 </soap:envelope>
```

SOAP Binding – RPC/literal

- WSDL

```
1 <message name="OrderStatusRequestMessage">
2   <part name="process-id" type="xsd:string"/>
3 </message>
4
5 <portType name="OrderProcess">
6   <operation name="getStatus">
7     <input message="OrderStatusRequestMessage"/>
8     <output message="..." />
9   </operation>
10 </portType>
```

- SOAP Request

```
1 <soap:envelope>
2   <soap:body>
3     <getStatus>
4       <process-id>5</process-id>
5     </getStatus>
6   </soap:body>
7 </soap:envelope>
```

SOAP Binding – Document/literal

- WSDL

```
1 <types>
2   <schema>
3     <element name="process-id" type="xsd:string"/>
4   </schema>
5 </types>
6
7 <message name="OrderStatusRequestMessage">
8   <part name="payload" element="process-id"/>
9 </message>
10
11 <portType name="OrderProcess">
12   <operation name="getStatus">
13     <input message="OrderStatusRequestMessage"/>
14     <output message="..." />
15   </operation>
16 </portType>
```

- SOAP Request

```
1 <soap:envelope>
2   <soap:body>
3     <process-id>5</process-id>
4   </soap:body>
5 </soap:envelope>
```

SOAP Binding – Document/literal wrapped

- WSDL

```
1 <types>
2   <schema>
3     <element name="StatusRequest" type="StatusRequestType">
4       <complexType name="StatusRequestType">
5         <element name="process-id" type="xsd:string"/>
6       </complexType>
7     </schema>
8 </types>
9
10 <message name="OrderStatusRequestMessage">
11   <part name="payload" element="StatusRequest"/>
12 </message>
13
14 <portType name="OrderProcess">
15   <operation name="getStatus">
16     <input message="OrderStatusRequestMessage"/>
17     <output message="..." />
18   </operation>
19 </portType>
```

- SOAP Request

```
1 <soap:envelope>
2   <soap:body>
3     <StatusRequest>
4       <process-id>5</process-id>
5     </StatusRequest>
6   </soap:body>
7 </soap:envelope>
```

Overview

- Web Service Description Language
 - *Elements, Types and Messages Definitions*
 - *Interface and Operations*
 - *Binding*
 - *Service and Endpoint*
 - *Description*
- WS-Addressing

Service

- "Where" the service can be accessed
- Service element specifies
 - *always one interface*
 - *list of endpoint locations to access the service*
 - *each endpoint references a binding*
 - *protocols and transmission formats it supports*
- Multiple endpoints
 - *different protocols for the same service*
 - *different security requirements, for example*
 - *SOAP over HTTPS endpoint*
 - *SOAP over HTTP endpoint*

Service Element and SOAP Message

- Service element definition

```
1 <service name="orderprocess_client_ep">
2   <port name="OrderProcess_pt" binding="op:OrderProcessBinding">
3     <soap:address location="http://mimdw.fit.cvut.cz/soa-infra/services/mdw-examples"
4   </port>
5 </service>
```

- Corresponding HTTP request SOAP message example

```
1 > POST /soa-intra/services/mdw-examples/ProcessOrder/orderprocess_client_ep HTTP/1.1
2 > User-Agent: curl/7.30.0
3 > Host: mimdw.fit.cvut.cz
4 > Accept: */*
5 > Accept-Encoding: gzip,deflate
6 > Content-Type: text/xml; charset=UTF-8
7 > SOAPAction: "getStatus"
8 > Content-Length: 810
9
10 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
11   <soap:Header>
12     <!-- snip -->
13   </soap:Header>
14   <soap:Body>
15     <ns1:StatusRequest
16       xmlns:ns1="http://mimdw.fit.cvut.cz/mdw_examples/ProcessOrder/OrderProcess">
17       <ns1:process-id>5</ns1:process-id>
18     </ns1:StatusRequest>
19   </soap:Body>
20 </soap:Envelope>
```

Overview

- Web Service Description Language
 - *Elements, Types and Messages Definitions*
 - *Interface and Operations*
 - *Binding*
 - *Service and Endpoint*
 - *Description*
- WS-Addressing

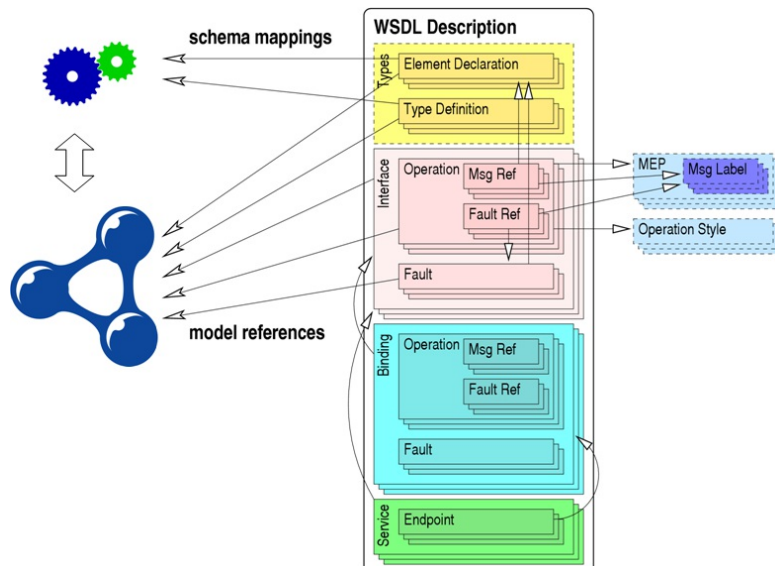
Description

- WSDL does not cover all service descriptions
 - *functionality only as a set of operations*
 - *no order of operations (e.g., a state diagram)*
- Solution
 - *Semantic Annotations for WSDL and XML Schema (SAWSDL)*
 - *defines links to rich semantic models*
 - *can be attached to any WSDL component*
 - *Textual description in WSDL documentation element*

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <definitions>
3  ...
4  <documentation>
5      This document describes an order service. Additional
6      application-level requirements for use of this service --
7      beyond what WSDL is able to describe -- are available
8      at http://company.com/doc/order-documentation.html
9  </documentation>
10 ...
11 </definitions>
```

- *Additional WS-* specifications, for example*
 - *WS-Addressing, WS-Reliability, WS-CDL, etc.*

SAWSDL



- SAWSDL extension attributes
 - **modelReference**, **schemaMappingLowering**, **schemaMappingLifting**
 - *SAWSDL W3C Recommendation, 2007* [🔗](#)

Example Interface Annotation

- Service classification at <http://example.org/classification>

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <root xmlns="http://example.org/service-classification-schema">
3   <order>
4     <book>
5       <adventure/>
6       <travel/>
7     </book>
8     <electronics>
9       <TV/>
10      <computer/>
11    </electronics>
12  </order>
13  <shipment><!-- shipment services --></shipment>
14 </root>
```

- Example annotation

– *prefix* **wsdl** is <http://www.w3.org/ns/wsdl>

– *prefix* **sawSDL** is <http://www.w3.org/ns/sawSDL>

```
1 <portType name="ProcessOrder"
2   sawSDL:modelReference="http://example.org/classification/root/order/electronics"/>
3   ...
4 </portType>
```

Overview

- Web Service Description Language
- WS-Addressing

Overview

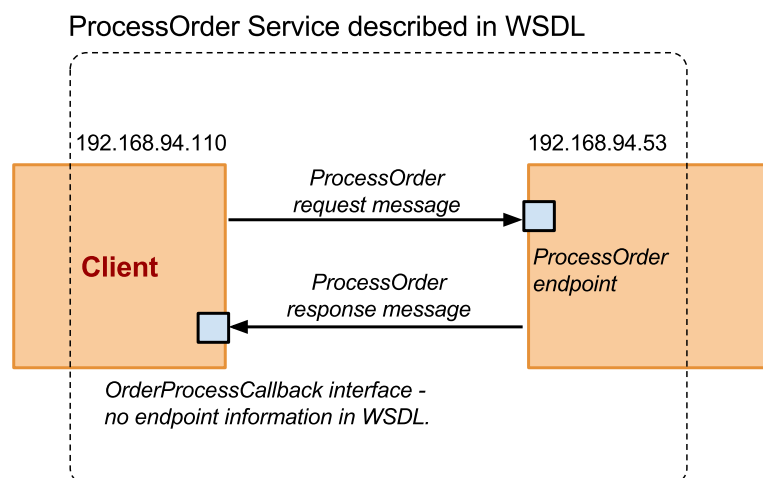
- WS-Addressing
 - W3C Recommendation, May 2006 [🔗](#)
 - A transport-independent mechanisms for web services to communicate addressing information
 - WSDL describes WS-Addressing as a policy attached to a WSDL binding

```
1 <binding name="OrderProcessBinding" type="op:OrderProcess">
2   <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
3   <PolicyReference xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/polic
4     URI="#wsaddr_policy" wsdl:required="false"/>
```

- Two main purposes
 1. Asynchronous communication
 - Client sends an endpoint where the server should send a response asynchronously
 2. Relating interactions to a conversation
 - Client and service communicate conversation ID

Order Processing Example

- Asynchronous communication via callback, steps:
 - Client submits an order request
 - Service processes the order (CRM, OMS, back-office)
 - Service responds asynchronously with an order response message



ProcessOrder Request Message

- Client sends process order request – **processOrder**
 - it sends addressing information where the client listens for the callback
 - it sends conversation ID (message ID) to start the conversation on the server

```
1 > POST /soa-infra/services/mdw-examples/ProcessOrder/orderprocess_client_ep HTTP/1.1
2 > Host: mimdw.fit.cvut.cz
3 > Content-Type: text/xml; charset=UTF-8
4 > SOAPAction: "processOrder"
5 > Content-Length: 810
6
7 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
8   xmlns:ord="http://mimdw.fit.cvut.cz/mdw-examples/cdm/order">
9   <soap:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
10     <wsa:ReplyTo>
11       <wsa:Address>http://192.168.94.110:2233/path/to/service</wsa:Address>
12     </wsa:ReplyTo>
13     <wsa:MessageID>urn:AXYYBA00531111E3BFACA780A7E5AF64</wsa:MessageID>
14   </soap:Header>
15   <soap:Body>
16     <ord:Order>
17       <ord:CustomerId>1</ord:CustomerId>
18       <ord:LineItems>
19         <ord:item>
20           <ord:label>Apple MacBook Pro</ord:label>
21           <ord:action>ADD</ord:action>
22         </ord:item>
23       </ord:LineItems>
24     </ord:Order>
25   </soap:Body>
26 </soap:Envelope>
```

GetStatus Request Message

- Client sends get status request – **getStatus**
 - after it invokes **processOrder** with conversation ID (message ID)
 - it uses the same conversation ID for get status request too
 - the request will be processed by the running service instance

```
1 > POST /soa-intra/services/mdw-examples/ProcessOrder/orderprocess_client_ep HTTP/1.1
2 > Host: mimdw.fit.cvut.cz
3 > Content-Type: text/xml; charset=UTF-8
4 > SOAPAction: "getStatus"
5 > Content-Length: 472
6
7 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
8   <soap:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
9     <wsa:RelatesTo>urn:AXYYBA00531111E3BFACA780A7E5AF64</wsa:RelatesTo>
10   </soap:Header>
11   <soap:Body>
12     <ns1:StatusRequest
13       xmlns:ns1="http://mimdw.fit.cvut.cz/mdw-examples/ProcessOrder/OrderProces
14       <ns1:process-id>18a9baec2d5ac0a2:64d155de:1425c4185f1:-7ff2</ns1:process-
15     </ns1:StatusRequest>
16   </soap:Body>
17 </soap:Envelope>
```