2. Web Services

Jiří Vokřínek

Agent Technology Center
Department of Computer Science
Faculty of Electrical Engineering, Czech Technical University in Prague

jiri.vokrinek@fel.cvut.cz http://agents.fel.cvut.cz
BOSS, WE'VE JUST FINISHED THE WEB SERVICE

NO, IT'S JUST A WEB SERVICE

GREAT!! DO WE NOW HAVE A SOA?

OK! OK! WE'LL CALL "HIM" SOA

HOW TO GET A SOA
Web Services

- W3C definition – a software system designed to support interoperable machine-to-machine interaction over a network, that has
  - Application programming interfaces accessed via HTTP
  - Interface described in a machine-processable format (WSDL)
  - Interaction using SOAP messages using HTTP with XML
Web Services

RESTful Web services
- Primary purpose is to manipulate XML representations of Web resources
- Uniform set of "stateless" operations

‘Big’ Web services
- Expose an arbitrary set of operations
- Stateful
Web Services
**RPC WS**

- Distributed **method call** interface
- The first WS usages/tools
- Often implemented by mapping services directly to language-specific functions (not loosely coupled)
SOA WS

- Implemented according to SOA
- Basic unit of communication is a message – **message-oriented** services
- Loosely coupled
- Focus to WSDL ‘contract’
- More business oriented / event-driven SOA
REST SOA

Use HTTP or similar protocols by constraining the interface to a set of well-known, standard operations.

The focus is on interacting with stateful resources.
What about content encoding?

Don’t you like the XML???
Content Encoding Features

- Human readability
- Binary efficiency
- Availability of interface description language
- Platform independence
- Availability of implementations
- Standardization
- e.g. XML, XML-RPC, JSON, YAML, etc.
Content Encoding Example

- XML
  - Human readable, not efficient
  - Strong typing, XSD, DTD

```xml
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value><i4>40</i4></value>
    </param>
  </params>
</methodCall>
```
Content Encoding Example

- JSON
  - Human readable, partially efficient
  - Strong typing, IDL partially supported
  - Derived from JavaScript
  - Efficient XML alternative for services
Content Encoding Example

JSON message example

```json
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"
    },
    "phoneNumber": [
        {
            "type": "home",
            "number": "212 555-1234"
        },
        {
            "type": "fax",
            "number": "646 555-4567"
        }
    ]
}
```
Content Encoding Example

Same message in XML

```xml
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber type="home">212 555-1234</phoneNumber>
    <phoneNumber type="fax">646 555-4567</phoneNumber>
  </phoneNumbers>
</person>
```
Content Encoding Example

Same message in XML

```xml
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber type="home">212 555-1234</phoneNumber>
    <phoneNumber type="fax">646 555-4567</phoneNumber>
  </phoneNumbers>
</person>

<person firstName="John" lastName="Smith" age="25">
  <address streetAddress="21 2nd Street" city="New York" state="NY" postalCode="10021" />
  <phoneNumbers>
    <phoneNumber type="home" number="212 555-1234"/>
    <phoneNumber type="fax" number="646 555-4567"/>
  </phoneNumbers>
</person>
```
Content Encoding Example

- YAML
  - Human readable
  - Not document oriented, but data oriented
  - Strong typing, IDL partially supported
  - Programming languages inspired (C, Perl, Python)
Content Encoding Example

YAML message example

```yaml
receipt: Oz-Ware Purchase Invoice
date: 2007-08-06
customer:
  given: Dorothy
  family: Gale

items:
- part_no: A4786
descr: Water Bucket (Filled)
  price: 1.47
  quantity: 4

- part_no: E1628
descr: High Heeled "Ruby" Slippers
  size: 8
  price: 100.27
  quantity: 1
```
Content Encoding Example

Protocol Buffers by Google

- Binary efficient, not human readable
- Strong typing, IDL

```protobuf
define Protocol Buffers
  message Point {
    required int32 x = 1;
    required int32 y = 2;
    optional string label = 3;
  }

  message Line {
    required Point start = 1;
    required Point end = 2;
    optional string label = 3;
  }

  message Polyline {
    repeated Point point = 1;
    optional string label = 2;
  }
```

This is IDL not the message. The message is variable length binary compressed chunk of data.
Web Service Design

Got it! Let’s make them!
**WS Design**

- **Bottom up** – first write the implementing class, then generate WSDL
  - Considered as simpler
  - Language/platform dependence/influence risk
**WS Design**

- **Top down** – first write the WSDL document, then generate class skeleton
  - Considered as more difficult
  - Produce cleaner designs
Bottom-up Service


Code

```java
package hello;

public class CircleFunctions {
    public double getArea(double r) {
        return java.lang.Math.PI * (r * r);
    }

    public double getCircumference(double r) {
        return 2 * java.lang.Math.PI * r;
    }
}
```
package hello;

import javax.jws.WebService;

@WebService
public class CircleFunctions {
    public double getArea(double r) {
        return java.lang.Math.PI * (r * r);
    }
    public double getCircumference(double r) {
        return 2 * java.lang.Math.PI * r;
    }
}
Bottom-up Service

Deploy

```java
package hello;

import javax.ws.rs.WebService;
import javax.xml.ws.Endpoint;

@WebService
public class CircleFunctions {
    public double getArea(double r) {
        return java.lang.Math.PI * (r * r);
    }

    public double getCircumference(double r) {
        return 2 * java.lang.Math.PI * r;
    }

    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8080/WebServiceExample/circlefunctions",
                        new CircleFunctions());
    }
}
```
Bottom-up Service

- Code – CircleFunctions.java

- Compile:
  ```
  > javac hello\CircleFunctions.java
  ```

- Generate service:
  ```
  > wsgen -cp hello.CircleFunctions
  ```
Bottom-up Service

Deploy:

> java hello.CircleFunctions

Enjoy:

http://localhost:8080/WebServiceExample/circlefunctions?WSDL

http://localhost:8080/WebServiceExample/circlefunctions?xsd=1
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
 xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://hello/
 name="CircleFunctionsService">
  - <types>
    - <xsd:schema>
      <xsd:import namespace="http://hello/"
        schemaLocation="http://localhost:8080/WebServiceExample/circlefunctions?
        xsd=1" />
      </xsd:schema>
    </types>
  - <message name="getArea">
    <part name="parameters" element="tns:getArea" />
  </message>
  - <message name="getAreaResponse">
    <part name="parameters" element="tns:getAreaResponse" />
  </message>
  - <message name="getCircumference">
    <part name="parameters" element="tns:getCircumference" />
  </message>
  - <message name="getCircumferenceResponse">
    <part name="parameters" element="tns:getCircumferenceResponse" />
  </message>
  - <message name="receive">
    <part name="parameters" element="tns:receive" />
  </message>
</definitions>
- <xs:schema xmlns:tns="http://hello/">
  xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0"
  targetNamespace="http://hello/">
  <xs:element name="getArea" type="tns:getArea" />
  <xs:element name="getAreaResponse" type="tns:getAreaResponse" />
  <xs:element name="getCircumference" type="tns:getCircumference" />
  <xs:element name="getCircumferenceResponse" type="tns:getCircumferenceResponse" />
  <xs:element name="receive" type="tns:receive" />
  <xs:element name="receiveResponse" type="tns:receiveResponse" />
- <xs:complexType name="receive">
  <xs:sequence>
    <xs:element name="arg0" type="xs:anyType" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
- <xs:complexType name="receiveResponse">
  <xs:sequence />
</xs:complexType>
- <xs:complexType name="getCircumference">
  <xs:sequence>
    <xs:element name="arg0" type="xs:double" />
  </xs:sequence>
</xs:complexType>
- <xs:complexType name="getCircumferenceResponse">
  <xs:sequence>
    <xs:element name="return" type="xs:double" />
  </xs:sequence>
</xs:complexType>
Bottom-up Service

Client stub:

```bash
> wsimport.exe http://localhost:8080/WebServiceExample/circlefunctions?wsdl
```
Bottom-up Service

... even “easier” with IDE
Bottom-up Service

... even “easier” with IDE
Top-down Service

- Corresponds to SOA model
- System design phase
- Various modeling tools
- XSD for data structures
- WSDL generated from model
- Supported by selected technologies/frameworks
Top-down Service

- Get WSDL
- Generate and implement
Client

- Always top-down
- Generated stub from WSDL (wsdl2java)
- Really simple in IDE ...