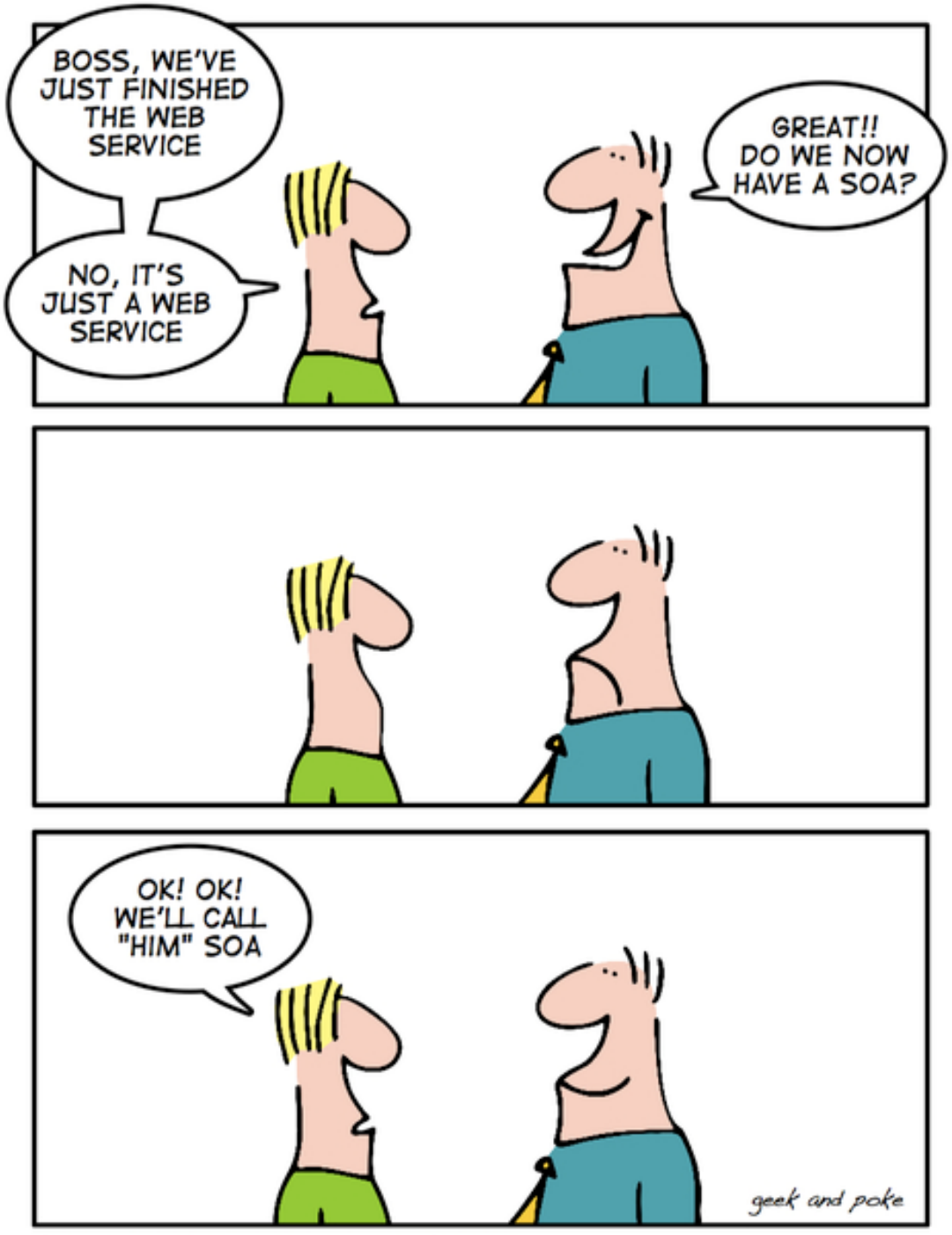
A4M36AOS – Architektury orientované na služby

2. Web Services

Jiří Vokřínek

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

jiri.vokrinek@fel.cvut.cz http://agents.felk.cvut.cz



HOW TO GET A SOA

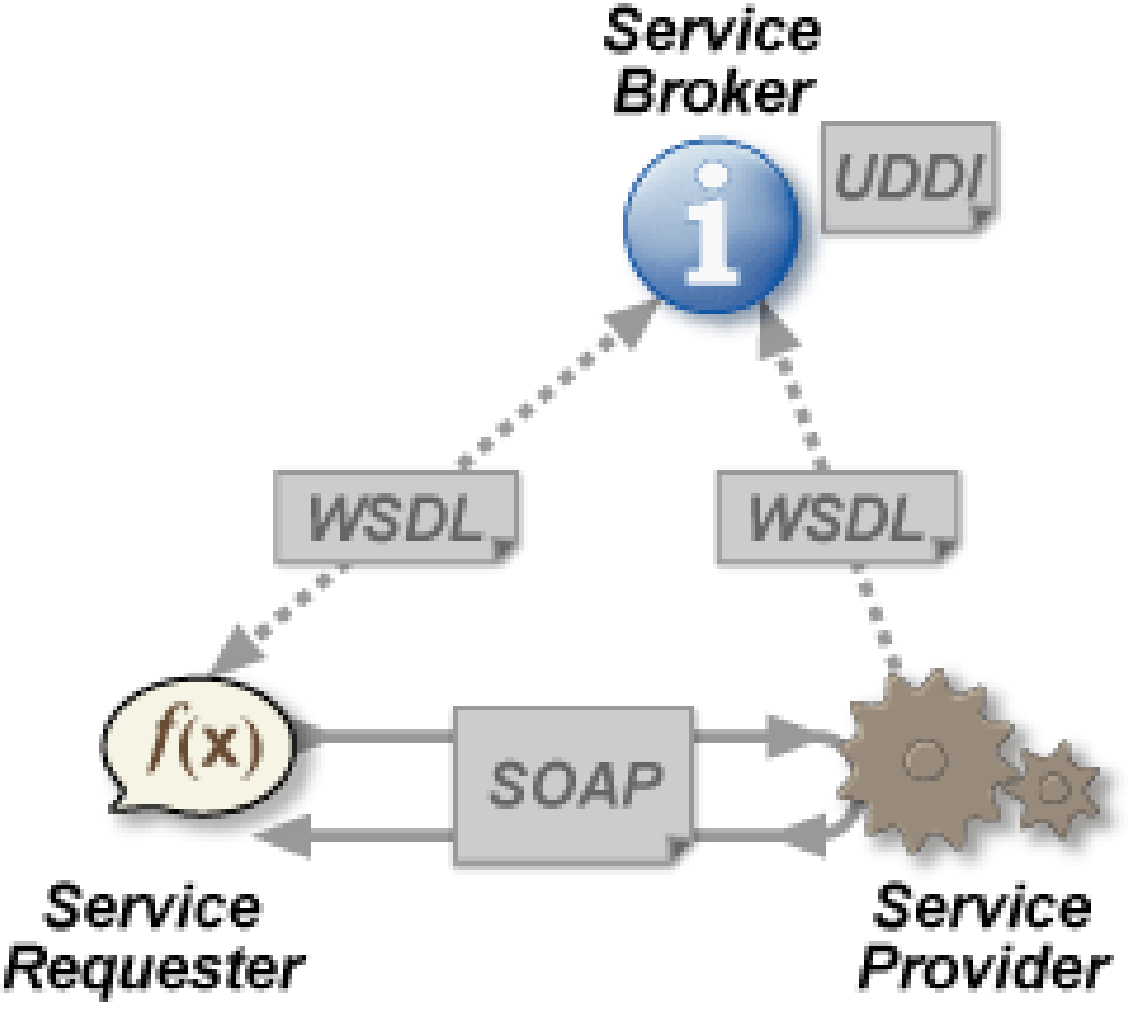
Web Services

- Application programming interfaces accessed via HTTP
- W3C definition – a software system designed to support interoperable machine-to-machine interaction over a network
- 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
  - Statefull

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**

Content Encoding

- 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

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

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

Content Encoding

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

Content Encoding

- JSON message example

```
{
    "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

- Same message in 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

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

Content Encoding

- YAML message example

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

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

    - part_no:   E1628
      descrip:   High Heeled "Ruby" Slippers
      size:      8
      price:     100.27
      quantity:  1
```

Content Encoding

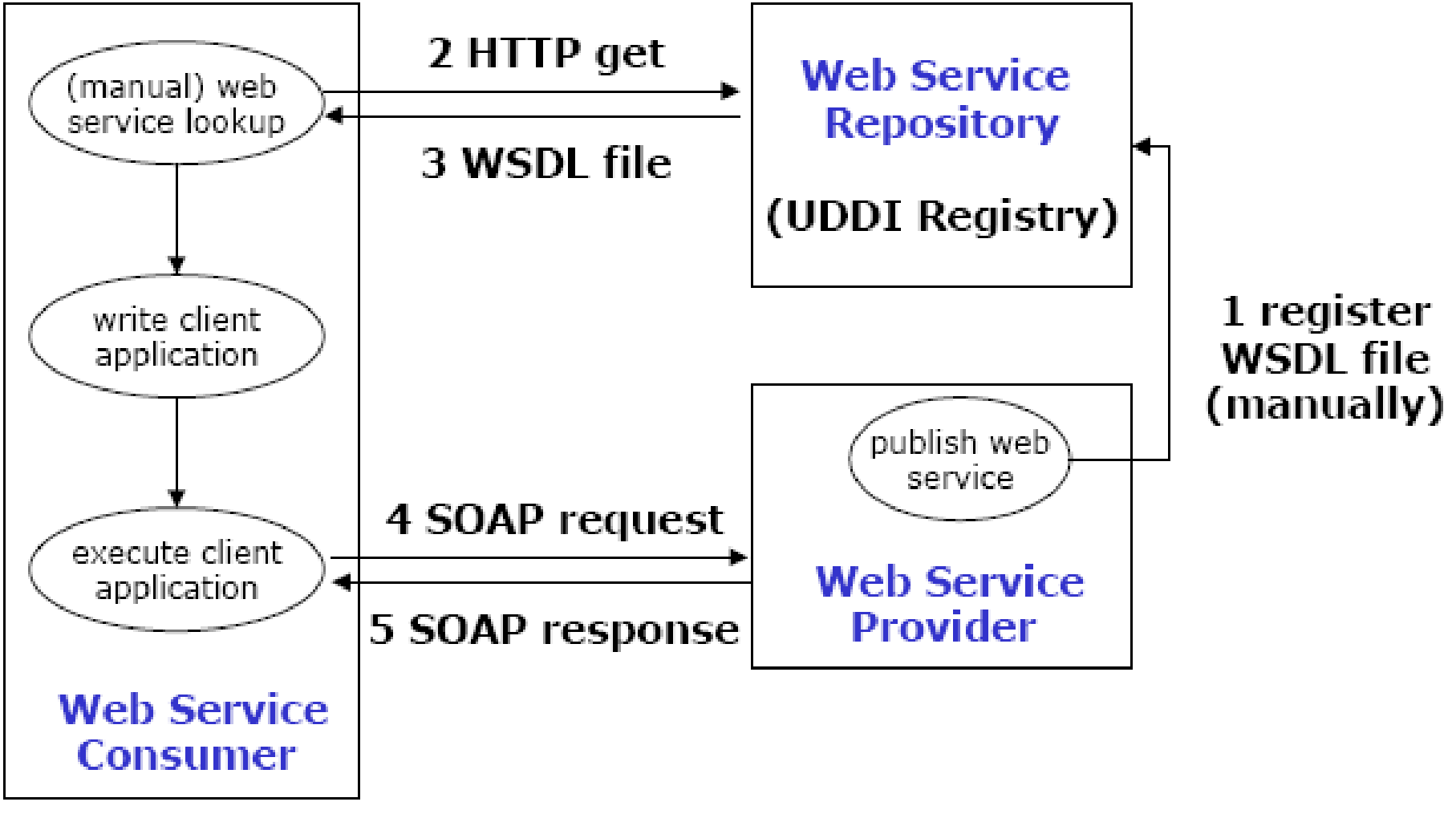
- Protocol Buffers
  - Binary efficient, not human readable
  - Strong typing, IDL

```
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;
}
```

WS Design

- Bottom up – first write the implementing class, then generate WSDL
  - Considered as simpler
  - Language/platform dependence/influence risk
- Top down – first write the WSDL document, then generate class skeleton
  - Considered as more difficult
  - Produce cleaner designs

WS Design


(manual) web service lookup
write client application
execute client application
Web Service Consumer
2 HTTP get
3 WSDL file
4 SOAP request
5 SOAP response
Web Service Repository
(UDDI Registry)
publish web service
Web Service Provider
1 register WSDL file (manually)

Bottom-up Service

http://www.oracle.com/us/technologies/java/jax-ws-2-141894.html

- **Code**

```
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;
    }
}
```

Bottom-up Service

- **Annotate**

```
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**

```
package hello;

import javax.jws.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/WebServiceExampl
e/circlefunctions?WSDL
```

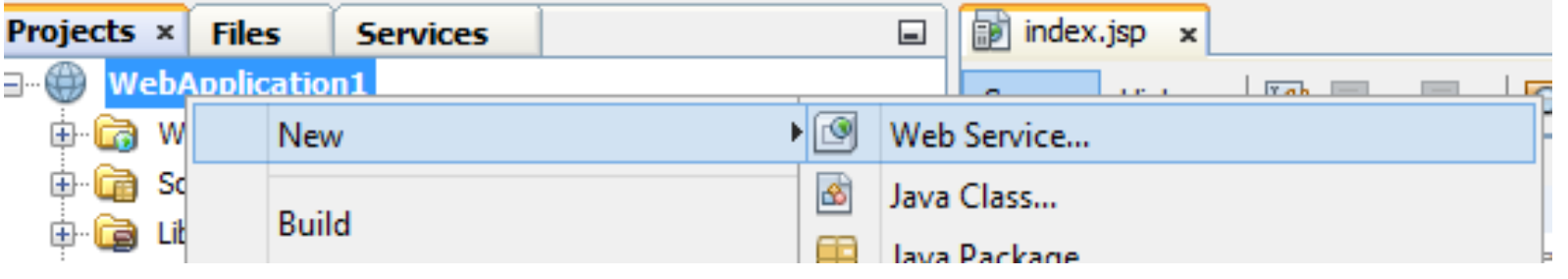
```
http://localhost:8080/WebServiceExampl
e/circlefunctions?xsd=1
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--  Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version
is JAX-WS RI 2.1.6 in JDK 6.   -->
<!--  Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version
is JAX-WS RI 2.1.6 in JDK 6.   -->
- <definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://hello/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    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>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--  Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version
is JAX-WS RI 2.1.6 in JDK 6.   -->
- <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>
```

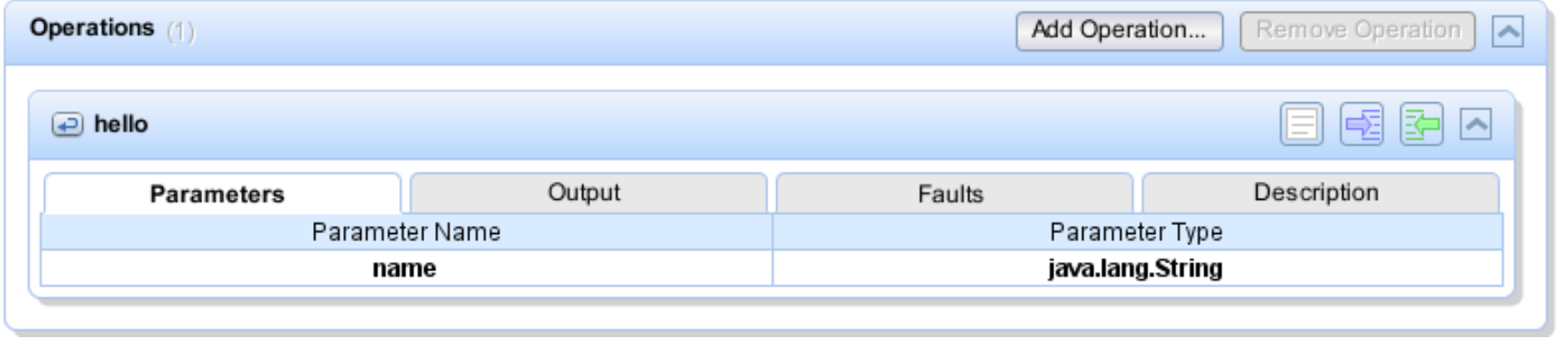
Bottom-up Service

... even easier with IDE



Bottom-up Service

... even easier with IDE



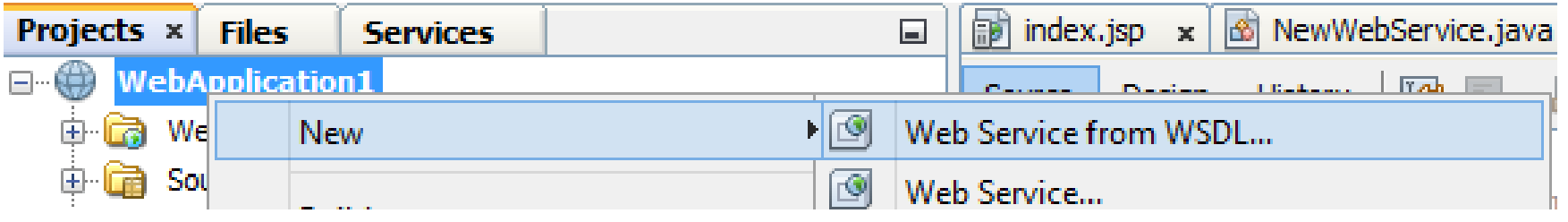
Endpoint	Information
Service Name: {http://service/}NewWebService Port Name: {http://service/}NewWebServicePort	Address: http://localhost:8084/WebApplication1/NewWebService WSDL: http://localhost:8084/WebApplication1/NewWebService?wsdl Implementation class: service.NewWebService

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 …