XML-базирани уеб услуги

въведение

Петьо Димитров

27.09.2013

Съдържание

Същност и характеристики

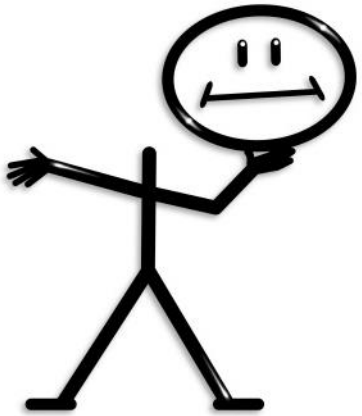
История

SOAP протокол и WSDL интерфейс

Java имплементация: SAAJ и JAX-WS

Разширения към уеб услугите

Какво са уеб услуги?

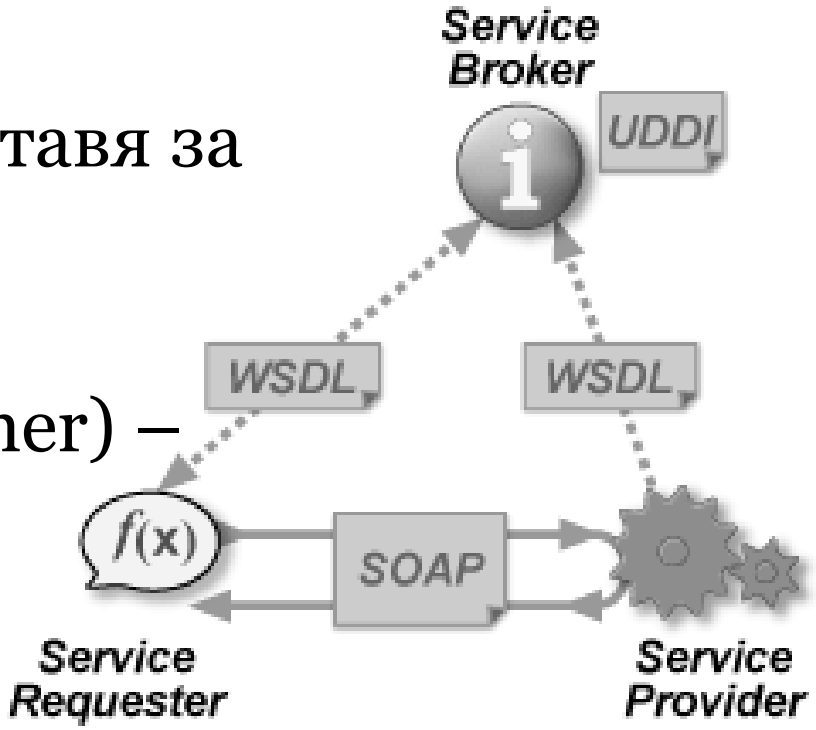


Характеристики на уеб услугите

- XML-базирани
- отворени за широк кръг участници
- ясно дефиниран интерфейс
- ориентирани към обмен на съобщения
- асинхронни и синхронни извиквания
- едрозърнести

Страни в комуникацията

- *Service Broker* (registry) – спомага за откриване на уеб услугата
- *Service Provider* – предоставя за използване услугата
- *Service Requester* (consumer) – използва услугата

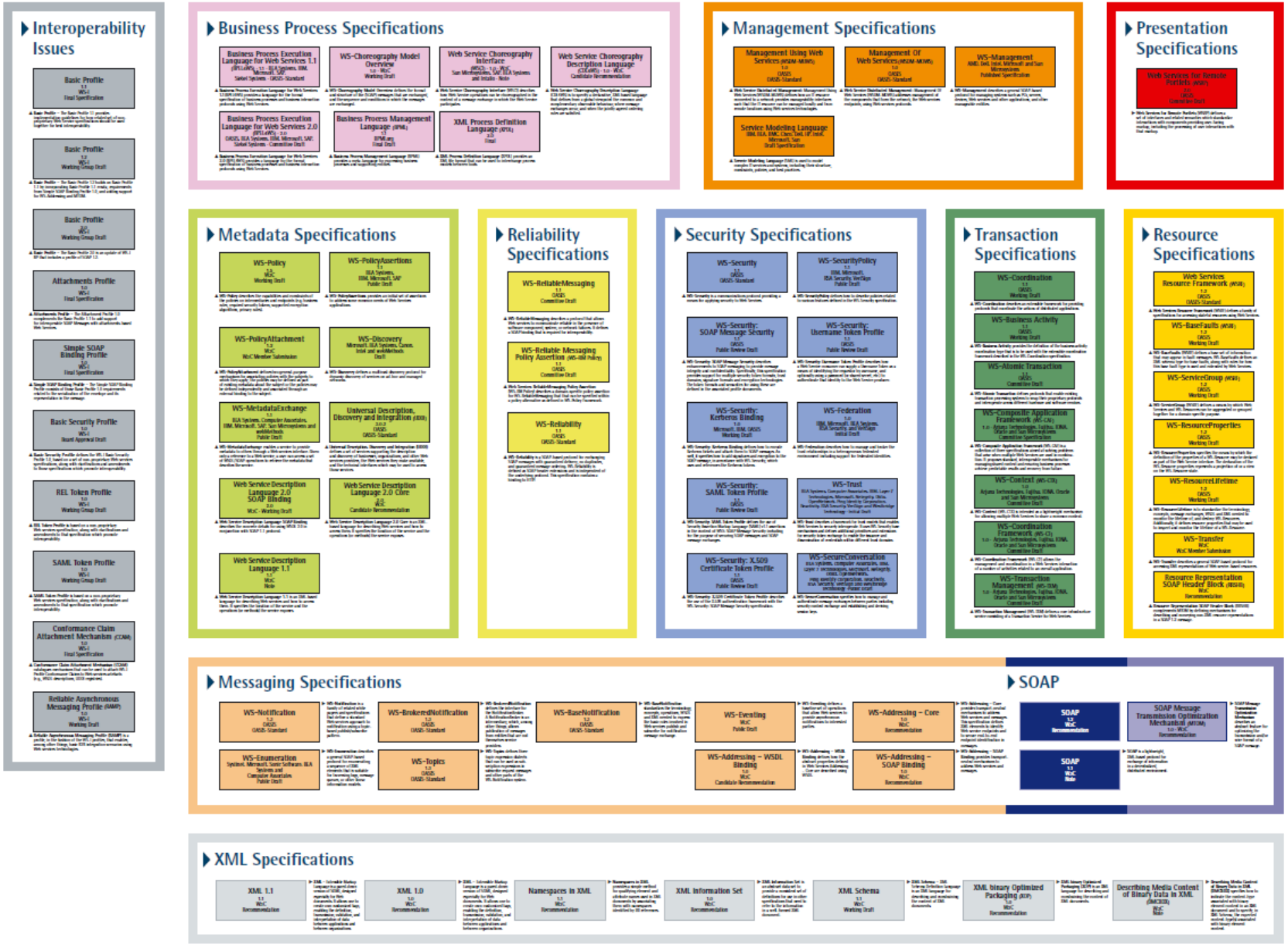


История на уеб услугите

- 2000г., Майкрософт разработва SOAP (a.k.a. Simple Object Access Protocol) за трансфер на данни, като алтернатива на съществуващи RPC решения (надделява над XML-RPC)
- бързо се включват други компании и в следващата година се появяват WSDL и UDDI
- това е първото поколение от стандарти за уеб услуги

История (продължение)

- SOAP еволюира към пренос на съобщения вместо отдалечени извиквания и абревиатурата отпада (не е особено прост и не дава достъп до обекти???)
- UDDI “bites the dust”
- под натиска на бизнеса започва разширяване и допълване на спецификациите



HTTP(S)

has counted to infinity... twice

- основен транспортен протокол в Интернет
- добре познат
- надежден
- подържан от съвременните инфраструктури
- появява се 1996г.

- една от опциите за уеб услуги

XML

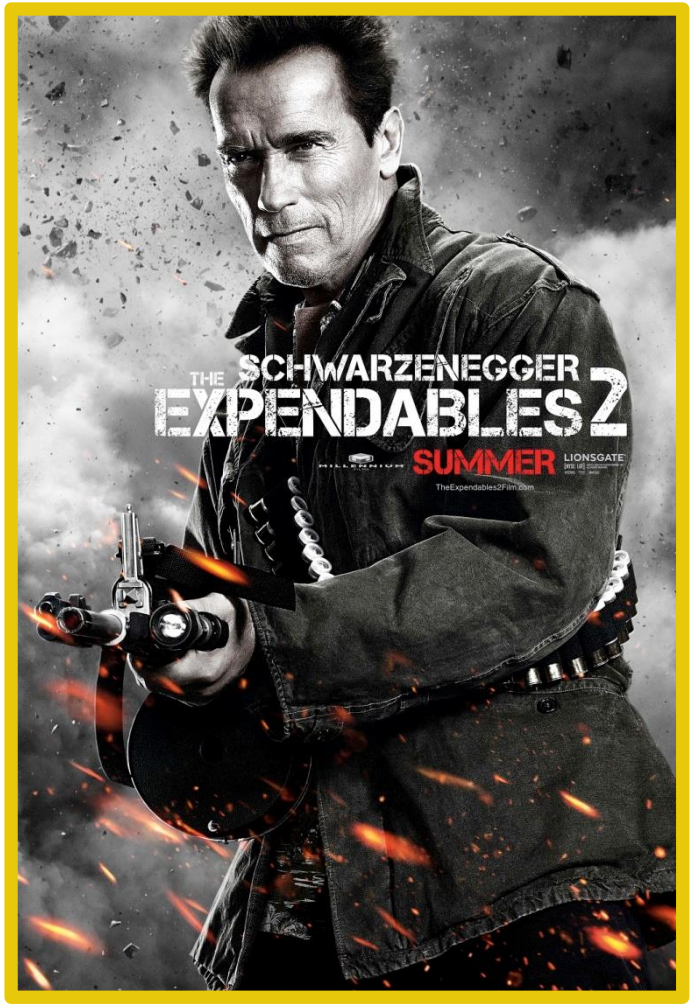
they drew first blood, not me (Rambo)

- Extensible Markup Language
- extensible – няма предефинирани тагове
- markup – текстов формат използващ тагове
- language – самоописателен (XMLSchema описва XML)
- фокус върху структура и данни (vs. HTML)
- GML→SGML→HTML→XML

SOAP

I'll be back
(Terminator)

- дефинира стандартен формат за съобщения
- прост, но лесно разширяем
- езиково и платформено независим
- базиран на XML



Структура

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

SOAP хедъри

- съдържат мета информация, която може да е необходима за съобщението:
  - инструкции за обработка или рутиране на съобщението
  - условия за сигурност
  - информация за корелиране на съобщения
- правят съобщението независимо
- имат незадължителен характер
- стоят в основата на много от WS-* спецификациите

SOAP + Java = SAAJ

- SOAP with Attachments API for Java
- API от по-ниско ниво позволяващо работа със SOAP съобщения
- стои в основата на JAX-RPC и JAX-WS
- поддържа SOAP и SwA спецификациите

SAAJ: създаване на съобщение

```
SOAPMessage msg =
MessageFactory.newInstance().createMessa
ge();

SOAPPart part = msg.getSOAPPart();
SOAPEnvelope env = part.getEnvelope();
SOAPBody body = env.getBody();

QName title = new
QName("http://opa/main", "title");
body.addBodyElement(title)
   .setValue("Avengers");
```

SAAJ: подготовка за изпращане

```
String ns = "http://opa/main";
URL wsdl = new URL("http://opa/Cinema?wsdl");
QName serviceName = new QName(ns, "Cinema");

Service service = Service.create(wsdl,
serviceName);

String portName = new QName(ns,
"CinemaPort");
Dispatch<SOAPMessage> dispatch =
service.createDispatch(portName,
SOAPMessage.class, Service.Mode.Message);
```

SAAJ: заявка и отговор

```
msg.writeTo(System.out);
SOAPMessage response = dispatch.invoke(msg);
response.writeTo(System.out);
```

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="...">
 <soap:Header/>
 <soap:Body>
  <title xmlns="http://opa/main">
   Avengers
  </title>
 </soap:Body>
</soap:Envelope>
```

SOAP и бинарни данни

- SOAP съобщенията са текст, но често пренасят бинарни данни (картинки, документи)

- base64
  - конвертира бинарните данни до ASCII текст
  - увеличава размера ~30%
  - пример:

```
<image imageType="jpg"
xsi:type="base64binary">4f3E9b0…</image>
```

SOAP и бинарни данни

- SwA (SOAP with Attachments) и DIME
  - използват `href` за да свържат съдържание извън XML съобщението
  - същата идея като при e-mail-ите
  - прикачените данни не се считат за част от съобщението

  - пример:

```
<image href="cid:image@test.com"/>
…
--MIME_boundary
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <image@test.com>
...binary JPG image...
--MIME_boundary--
```

SOAP и бинарни данни

- MTOM (Message Transmission Optimization Mechanism)
  - прехвърляните данни са същите като при SwA
  - прикачените данни са част от съобщението (SOAP infoset)
  - пример:

```
<image xop-mime:content-type='image/jpeg'>
   <xop:Include href="cid:image@test.com"/>
</image>
...
--MIME_boundary
...
```

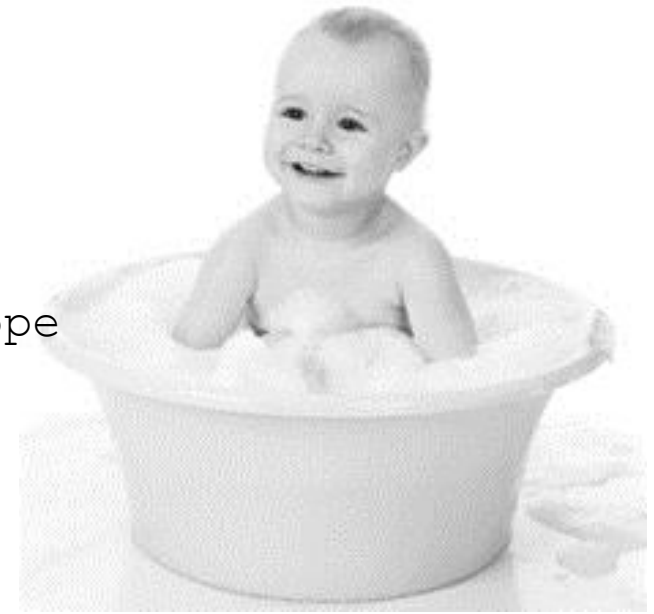
MTOM и Java

```
@WebService
@MTOM
public class Opa implements IOpa {
  public int test(int number) { ... }
}

IOpa opa = (IOpa)
service.getPort(IOpa.class);
BindingProvider bp = (BindingProvider) opa;
SOAPBinding binding = (SOAPBinding)
bp.getBinding();
binding.setMTOMEnabled(true);
```

SOAP 1.2

- позволява използването на други транспортни протоколи освен HTTP
- нов content type
  `application/soap+xml`
- нов namespace
  `http://www.w3.org/2003/05/soap-envelope`



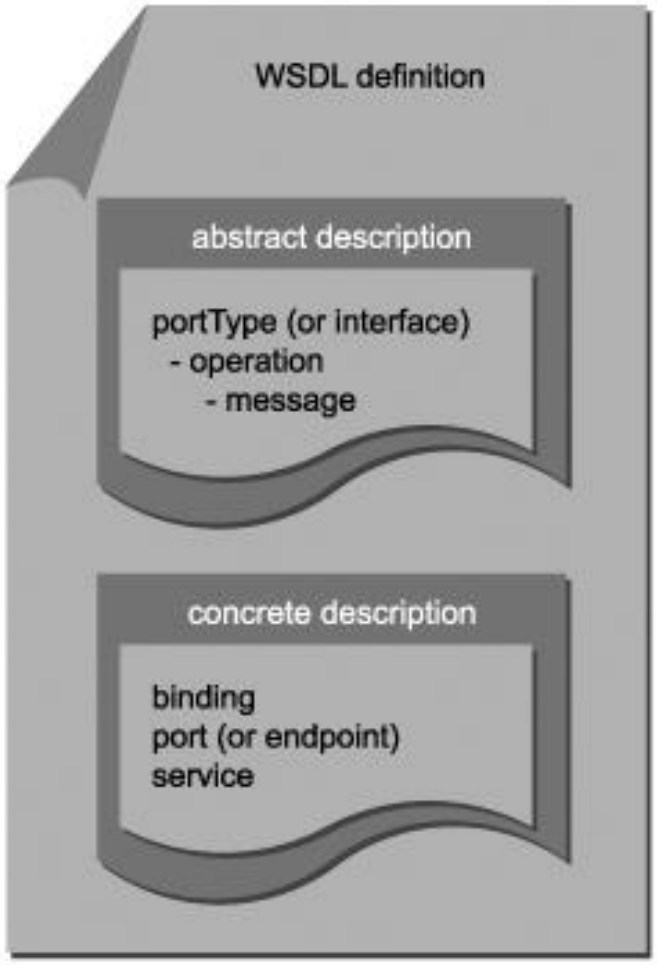
WSDL

Yippee ki-yay, motherfucker
(Die Hard)

- позволявана услугите да са независими (loosely coupled)
- всяка услуга има описание
- за уеб услугата това е договор, който изпълнява
- за клиента това е наръчник как да комуникира с услугата
- базиран на XML

Структура на WSDL документа

- абстрактна част – описва интерфейса на уеб услугата без инфо за използваната технология
- конкретна част – съдържа имплементационни детайли за уеб услугата
- допълва се от XSD дефиниции и policy-та



Структура - definitions

- корен на всяка дефиниция на уеб услуга
- важни са дефинираните namespace-и

```
<definitions name="Employee"
targetNamespace="http://opa/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:act="http://opa/schema/accounting/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://opa/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
...
</definitions>
```

Структура - types

- незадължителен елемент (може да се ползва messages)
- xsd дефиниции и/или import-и на xsd документи

```
<types>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace= "http://opa/schema/">
 <complexType name="ReturnCodeType">
  <sequence>
   <element name="Code" type="xsd:integer"/>
   <element name="Message" type="xsd:string"/>
  </sequence>
 </complexType>
</schema>
</types>
```

Структура - messages

- message се създава за всяко обменяно съобщение
- всяко съобщение има един или няколко part елемента
- всеки part елемент има type или element атрибут

```
<message
name="getEmployeeHoursRequestMessage">
 <part name="RequestParameter"
element="act:EmployeeHoursRequestType"/>
</message>
<message
name="getEmployeeHoursResponseMessage">
 <part name="ResponseParameter"
element="act:EmployeeHoursResponseType"/>
</message>
```

Структура - portType

- интерфейса на уеб услугата дефиниращ операции
- позицията на input и output определят MEP

```
<portType name="EmployeeInterface">
 <operation name="GetHoursLimit">
  <input
message="tns:getHoursRequestMessage"/>
  <output
message="tns:getHoursResponseMessage"/>
 </operation>
</portType>
```

Структура - binding

- имплементационна информация
- задава транспортния протокол и структурата на SOAP съобщенията

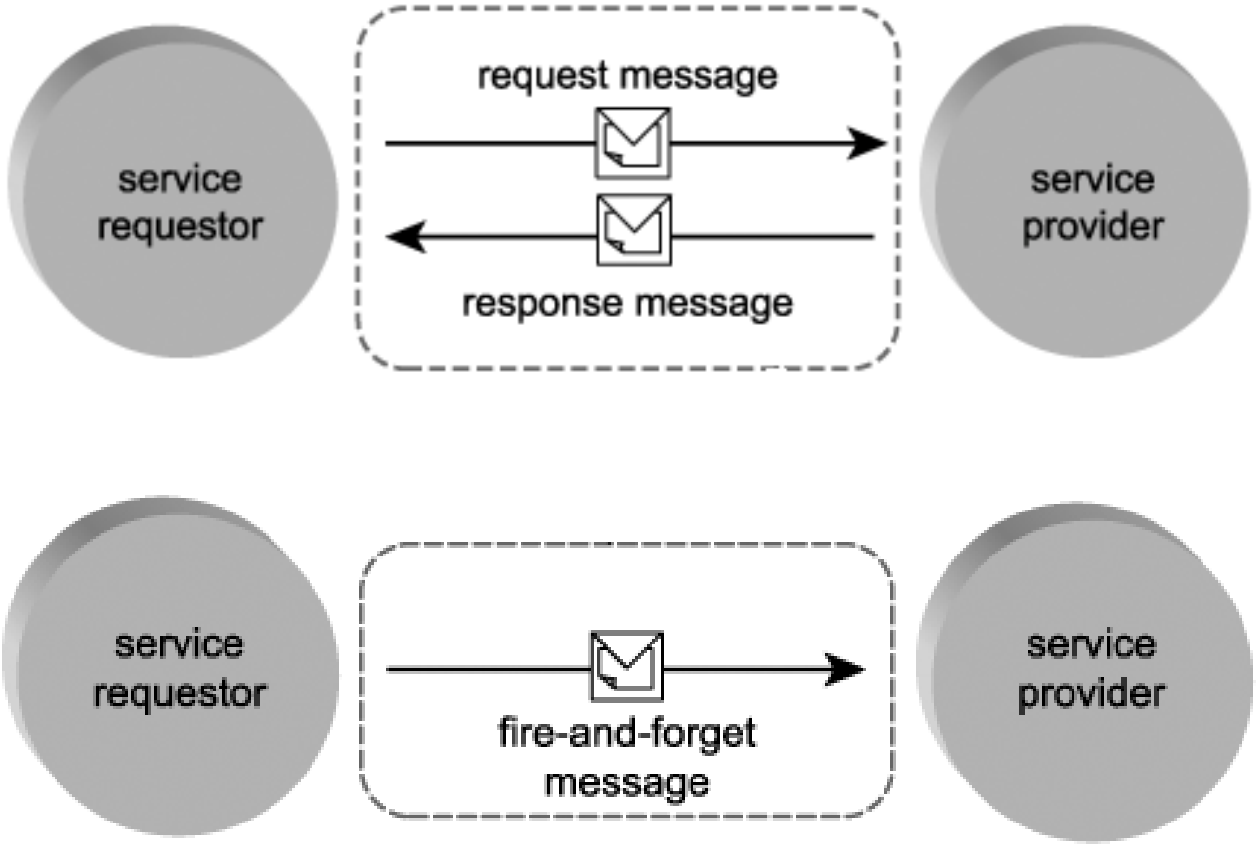
```
<binding name="EmployeeBinding" type="tns:EmployeeInterface">
 <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="GetWeeklyHoursLimit">
  <soap:operation soapAction="..."/>
  <input>
   <soap:body use="literal"/>
  </input>
  <output>
   <soap:body use="literal"/>
  </output>
 </operation>
</binding>
```

Структура - service

- оказва endpoint-а на услугате, тоест адреса където е достъпна
- други елементи: `import`, `documentation`

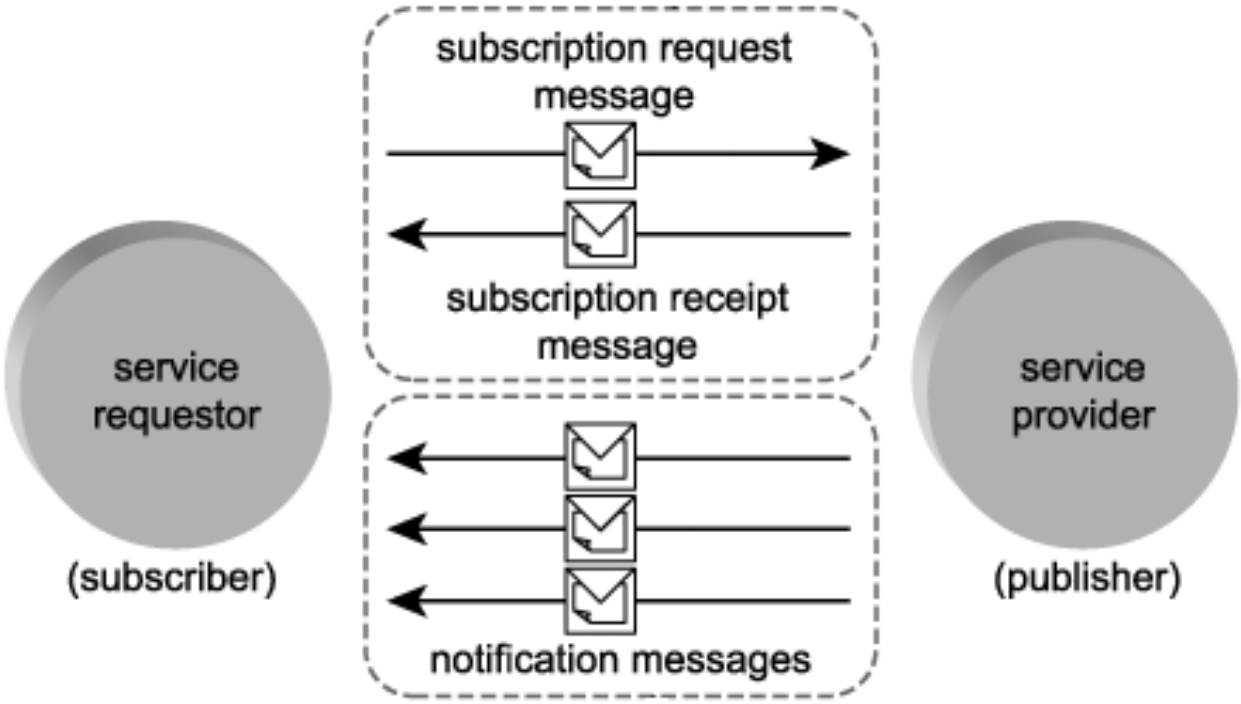
```
<service name="EmployeeService">
 <port binding="tns:EmployeeBinding"
name="EmployeePort">
  <soap:address
location="http://opa/employee/"/>
 </port>
</service>
```

MEP = Message Exchange Pattern



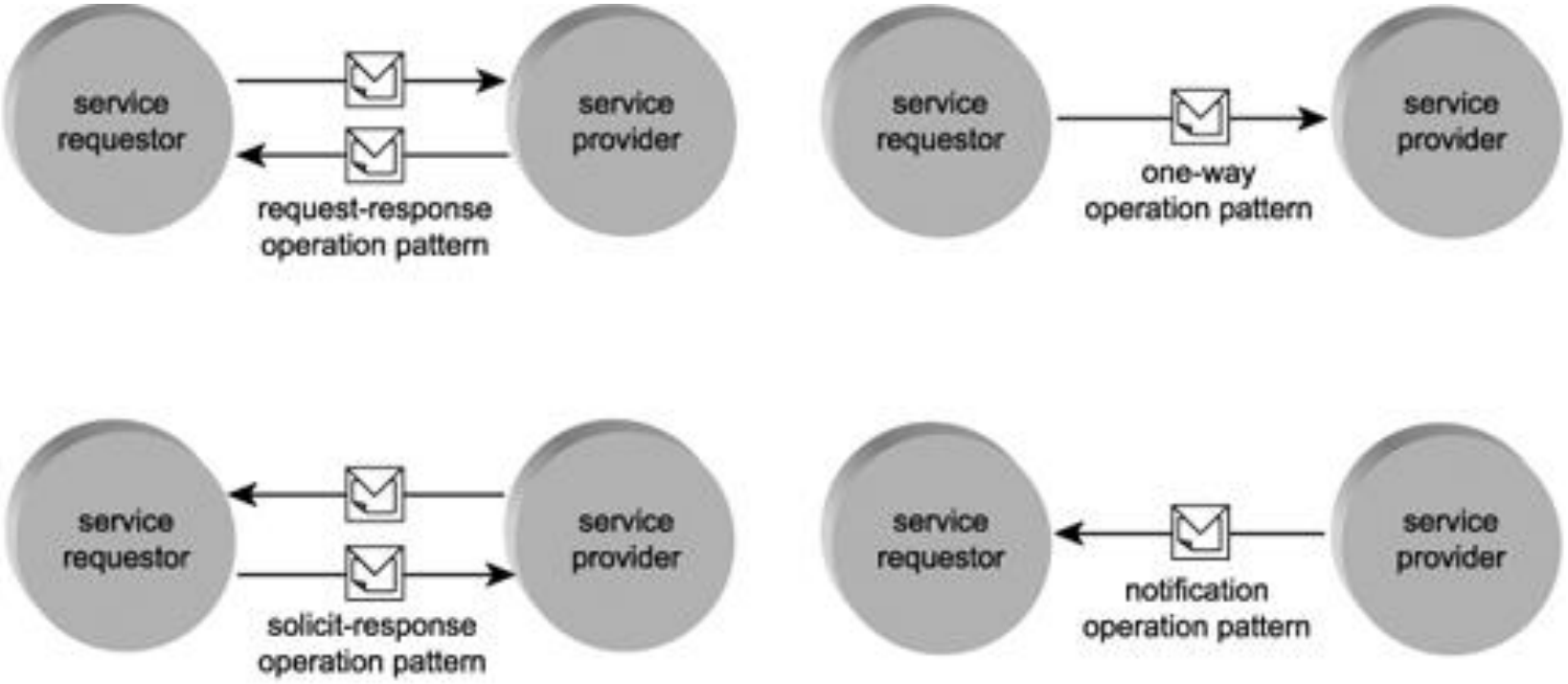
(single-destination, multi-cast, broadcast)

MEP = Message Exchange Pattern



MEP и WSDL

- определят се от реда на input и output елементите в WSDL-а



Document, literal, RPC, encoded

- style бива:
  - rpc (remote procedure call)
  - document (message centric)
- use бива:
  - encoded (xsd в SOAP съобщенията)
  - literal
- комбинации: rpc/encoded, document/literal, rpc/literal (рядко), document/encoded (няма) + 1

RPC/encoded: WSDL

```
<message name="myMethodRequest">
 <part name="x" type="xsd:int"/>
 <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>
<portType name="PT">
 <operation name="myMethod">
  <input message="myMethodRequest"/>
  <output message="empty"/>
 </operation>
</portType>
```

RPC/encoded: SOAP

```
<soap:envelope>
 <soap:body>
  <myMethod>
   <x xsi:type="xsd:int">5</x>
   <y xsi:type="xsd:float">5.0</y>
  </myMethod>
 </soap:body>
</soap:envelope>
```

RPC/encoded особености

+ прост WSDL
+ името на операцията присъства в съобщението

- не отговаря на изискванията на WSI-I профила за съвместимост
- типовата информация в SOAP увеличава размера на съобщението (и е често ненужна)
- трудно за валидация понеже само x и y елементите имат XSD

Document/literal: WSDL

```
<types>
 <schema>
  <element name="xElement" type="xsd:int"/>
  <element name="yElement" type="xsd:float"/>
 </schema>
</types>
<message name="myMethodRequest">
 <part name="x" element="xElement"/>
 <part name="y" element="yElement"/>
</message>
<message name="empty"/>
<portType name="PT">
 <operation name="myMethod">
  <input message="myMethodRequest"/>
  <output message="empty"/>
 </operation>
</portType>
```

Document/literal: SOAP

```
<soap:envelope>
 <soap:body>
  <xElement>5</xElement>
  <yElement>5.0</yElement>
 </soap:body>
</soap:envelope>
```

- няма име на операция
- няма типова информация

Document/literal особености

+ няма типова информация
+ цялото съдържание на body-то може да се валидира с xsd
+ донякъде съвместим е с WS-I

- WSDL-а е по-сложен
- липсва името на операцията, което затруднява разпределянето на съобщенията
- WS-I изисква да има само един елемент в тялото на съобщението (тук са 2)

Document/literal wrapped: WSDL

```
<types>
 <schema>
  <element name="myMethod">
   <complexType>
    <sequence>
     <element name="x" type="xsd:int"/>
     <element name="y" type="xsd:float"/>
    </sequence>
   </complexType>
  </element>
  <element name="myMethodResponse">
   <complexType/>
  </element>
 </schema>
</types>
<message name="methodRequest">
 <part name="parameters" element="myMethod"/>
</message>
<message name="methodResponse">
 <part name="parameters" element="myMethodResponse"/>
</message>
```

```
<portType name="PT">
 <operation name="myMethod">
  <input message="methodRequest"/>
  <output message="methodResponse"/>
 </operation>
</portType>
```

Document/literal wrapped: SOAP

```
<soap:envelope>
 <soap:body>
  <myMethod>
   <x>5</x>
   <y>5.0</y>
  </myMethod>
 </soap:body>
</soap:envelope>
```

- съобщението има само един part, който е елемент
- елемента има същото име като операцията и няма атрибути

Document/literal wrapped особености

+ няма типова информация
+ цялото съобщение може да се валидира
+ името на операцията присъства в съобщението
- съвместим е с WS-I профила (само един елемент в SOAP body-то)

- WSDL-а е много по-сложен
- има ли други проблеми?

WSDL + Java = JAX-WS

- JAVA API FOR XML WEB SERVICES
- анотации обогатяват класа с инфо за уеб услугата
- проектиране върху договор (WSDL) или код
- команда wsimport – за генерация на JAX-WS артефакти (SEI и SIB) и JAXB класове

```
wsimport -keep -verbose http://localhost:9999/ws?wsdl
```

- команда wsgen – за генерация на JAXB класове и WSDL и XSD дефиниции

Пример с Java (SEI и SIB)

```
@WebService
@SOAPBinding(style = Style.DOCUMENT,
use=Use.LITERAL)
public interface ICinema {
 @WebMethod
 String getMovieInfo(String title);
}

@WebService(endpointInterface="test.ICinema")
public class Cinema implements ICinema {
 public String getMovieInfo(String title) {
  return "Info for " + title;
 }
}
```

Пример с Java (publisher)

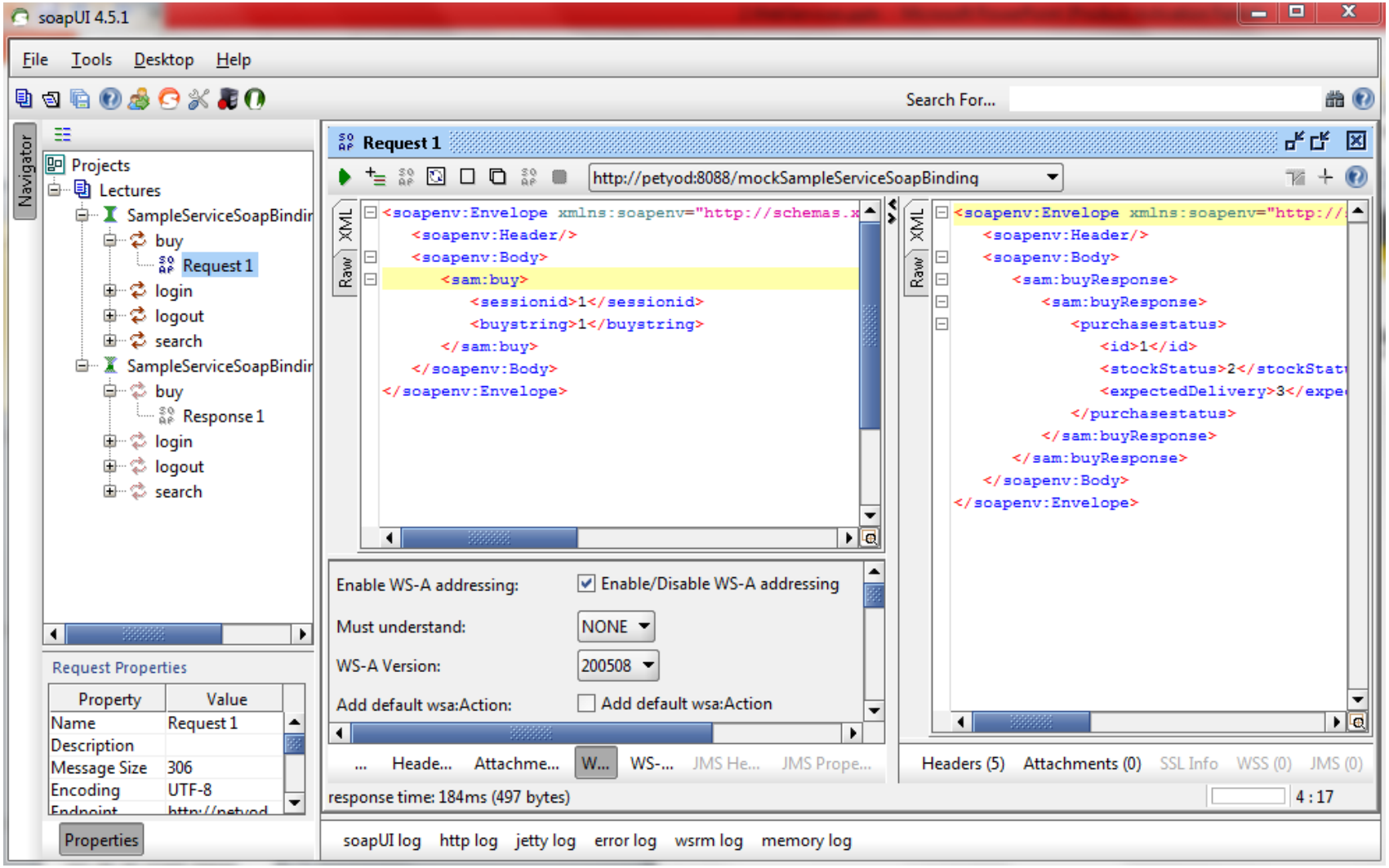
- позволява тестване на JAX-WS уеб услугата локално без сървър
- автоматично генерира WSDL и XSD дефиниции

```
public class CinemaPublisher {
 public static void main(String[] a) {
   Endpoint.publish("http://localhost:9999/cinema",
new Cinema());
 }
}
```

WSDL 2.0

- *definitions* → *description*
- *portType* → *interface*
- *port* → *endpoint*
- *messages* е премахнат заради ограничената си *RPC* употреба (part е предназначен за параметри) , за *document* винаги се ползва XSD
- *interface* има *extends* атрибут (от ООП)
- *operation* има *pattern* атрибут (in-out, out-only) и *wsdlx:safe* атрибут за идемпотентни операции
- *fault* се дефинира на ниво операция

Инструмент за работа с уеб услуги



Демонстрация

UDDI

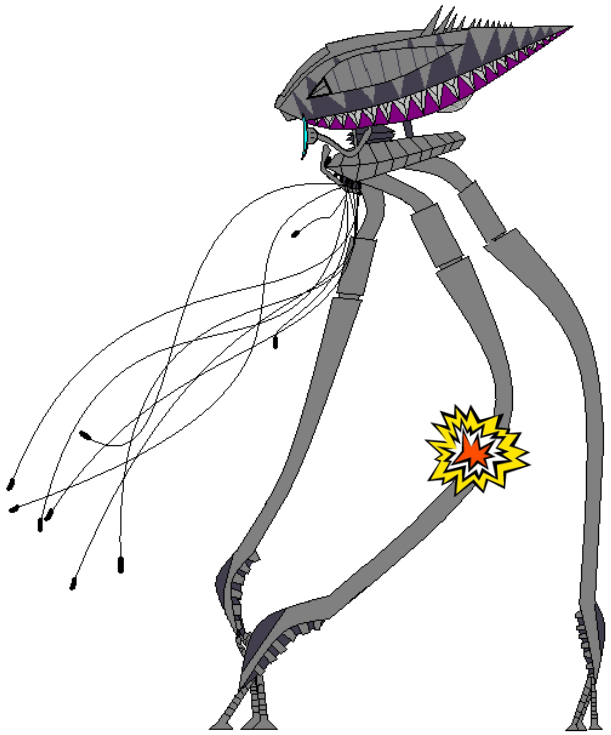
Are you man enough to fight with me?
(Street Fighter)

- Universal Description, Discovery, and Integration
- UBR = UDDI Business Registry = публични "жълти страници" за уеб услуги
- позволяващ регистриране и автоматично търсене на уеб услуги

UBR: Dead man walking

- 2006 г. IBM, SAP и Microsoft прекратяват поддръжката на публичните си UDDI регистри

- причини за провала:
  - сложен модел на данните (различни бизнеси)
  - ограничено търсене
  - сигурност
  - липса на поддръжка на регистрите



Наследство

- JAX-R: Java API for XML Registries
- ръчна настройка (http://www.xmethods.net)
- специфични решения (база данни, LDAP)
- вътрешни регистри (Apache jUDDI, WSRR)

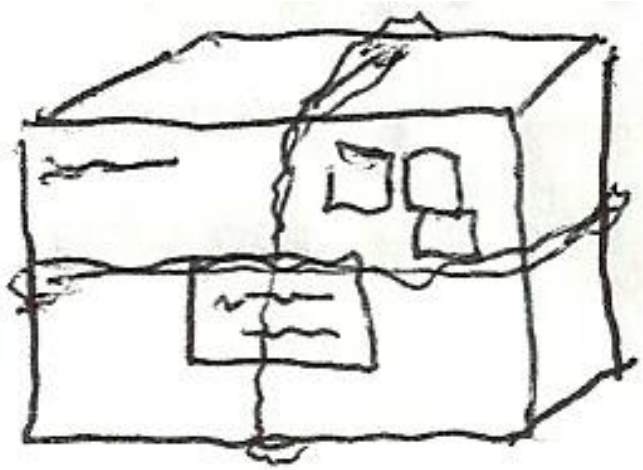
WS-Addressing

I cannot be defeated!
(Rocky IV)

- WS-Addressing – позволява информацията за изпращане да е част от съобщението

WS-Addressing информация

- откъде идва съобщението
- за къде е съобщението
- на кого трябва да се отговори
- кой трябва да го получи
- къде да иде при проблем с доставянето



WS-Addressing концепции

- endpoint reference – адрес позволяващ да се идентифицира конкретна инстанция на уеб услуга и нейни метаданни:
  - **URL до уеб услугата**
  - reference properties – за идентификация на инстанция
  - reference parameters – за обменяна на прости данни с услугата
  - port type – мястото на описанието на услугата на извикващата страна, полезна при отговора
  - policy – мета информация за обмена

Сапунена имплементация: EndpointReference

```
<wsa:EndpointReference>
<wsa:Address>http://opa</wsa:Address>
<wsa:ReferenceProperties>
 <app:id>unn:AFJK323llws</app:id>
 </wsa:ReferenceProperties>
 <wsa:ReferenceParameters>
  <app:sesno>12345</app:sesno>
 </wsa:ReferenceParameters>
</wsa:EndpointReference>
```

- други: `PortType, ServiceType, ServiceName, Policy`

WS-Addressing концепции

- message information headers – хедърите добавяни към едно съобщение:
  - дестинация (url)
  - източник (endpoint reference)
  - адрес за отговор (endpoint reference)
  - адрес за грешка (endpoint reference)
  - идентификатор на съобщението (id)
  - корелация между заявка и отговор (id)
  - action – показва целта на съобщението (url)

Сапунена имплементация: хедъри

```
</Header>
 <wsa:Action>http://opa/submit</wsa:Action>
 <wsa:To>http://opa</wsa:To>
 <wsa:From>
  <wsa:Address>http://men</wsa:Address>
  <wsa:ReferenceProperties>
   <app:id>unn:AFJK323llws</app:id>
  </wsa:ReferenceProperties>
  <wsa:ReferenceParameters>
   <app:sesno>22322447</app:sesno>
  </wsa:ReferenceParameters>
</wsa:From>
 <wsa:MessageID>uuid:2432</wsa:MessageID>
 <wsa:ReplyTo>...ER...</wsa:ReplyTo>
 <wsa:FaultTo>...ER...</wsa:FaultTo>
</Header>
```

RelatesTo се използва при отговори и ще съдържа стойността на **MessageID**

WS-Addressing и Java

```
@WebService
@Addressing(enabled=true, required=true)
public class Opa implements IOpa {
    @Action(input="http://opa/input",
output="http://opa/output")
    public int test(int number1, int number2)
{ ... }
}

IOpa opa = (IOpa) service.getPort(IOpa.class,
new AddressingFeature());
opa.test(1, 2);
```

WS-I Basic Profile спецификация

- *профил за съвместимост между уеб услуги* (версия 2.0 е публикувана на ноември 2010)
- WSDL 1.1
- SOAP 1.2
- MTOM
- WS-Addressing
- UDDI 3.0
- други изисквания: HTTP протокол (POST), rpc/literal или document/literal (wrapped), само request/response и one-way

Въпроси

Благодаря за вниманието