UNIVERSITÉ CÔTE D'AZUR

POLYTECH NICE SOPHIA

Web Service Description and Contracts

Jean-Yves Tigli - Benjamin Vella

From Weak Contracts …

Weak contracts
(unformal web service description)

- An unstructured web service description
- Therefore, without formal language and without grammar
- Cannot be processed to generate a piece of code
- Thus only describes specifications to help developers write client code by hand.

… to Strong Contracts

Strong contracts

(formal web service description)

- A structured web service description
- Using a formal language and with a grammar
- Can be processed to generate a piece of code with tools

WS-SOAP and WSDL
(Web Service Description Language)

<types>

- *Contient les définitions des types (utilise un système de typage comme XSD)*

<message>

- *Décrit les noms et types d'un ensemble de champs à transmettre*
- *Paramètres d'une invocation, valeur du retour, …*

<portType>

- *Décrit un ensemble d'opérations et les messages impliqués (0 ou 1 en entrée, 0 ou n en sortie). Partie la plus importante*

<binding>

- *Spécifie une liaison d'un <porttype> à un protocole concret (SOAP1.1, HTTP1.1, MIME, …).*
- *Un portType peut avoir plusieurs liaisons !*

<port>

- *Spécifie un point d'entrée (endpoint) comme la combinaison d'un <binding> et d'une adresse réseau*

<service>

- *Pour agréger un ensemble de ports*

Sample : HelloService.wsdl

```
<definitions name = "HelloService"
   targetNamespace = "http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns = "http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns = "http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns:xsd = "http://www.w3.org/2001/XMLSchema">

   <message name = "SayHelloRequest">
      <part name = "firstName" type = "xsd:string"/>
   </message>

   <message name = "SayHelloResponse">
      <part name = "greeting" type = "xsd:string"/>
   </message>

   <portType name = "Hello_PortType">
      <operation name = "sayHello">
         <input message = "tns:SayHelloRequest"/>
         <output message = "tns:SayHelloResponse"/>
      </operation>
   </portType>
```

```
   <binding name = "Hello_Binding" type = "tns:Hello_PortType">
      <soap:binding style = "rpc"
         transport = "http://schemas.xmlsoap.org/soap/http"/>
      <operation name = "sayHello">
         <soap:operation soapAction = "sayHello"/>
         <input>
            <soap:body
               encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
               namespace = "urn:examples:helloservice"
               use = "encoded"/>
         </input>

         <output>
            <soap:body
               encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
               namespace = "urn:examples:helloservice"
               use = "encoded"/>
         </output>
      </operation>
   </binding>

   <service name = "Hello_Service">
      <documentation>WSDL File for HelloService</documentation>
      <port binding = "tns:Hello_Binding" name = "Hello_Port">
         <soap:address
            location = "http://www.examples.com/SayHello/" />
      </port>
   </service>
</definitions>
```

Sample : HelloService.wsdl

```
<definitions name = "HelloService"
   targetNamespace = "http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns = "http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns = "http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns:xsd = "http://www.w3.org/2001/XMLSchema">

   <message name = "SayHelloRequest">
      <part name = "firstName" type = "xsd:string"/>
   </message>

   <message name = "SayHelloResponse">
      <part name = "greeting" type = "xsd:string"/>
   </message>

   <portType name = "Hello_PortType">
      <operation name = "sayHello">
         <input message = "tns:SayHelloRequest"/>
         <output message = "tns:SayHelloResponse"/>
      </operation>
   </portType>
```

Definitions – HelloService

Type – Using built-in data types and they are defined in XML Schema.

Message

- sayHelloRequest – firstName parameter
- sayHelloResponse – greeting return value

Port Type – sayHello operation that consists of a request and a response service.

Sample : HelloService.wsdl

Binding – Direction to use the SOAP HTTP transport protocol.

Service – Service available at http://www.examples.com/SayHello/

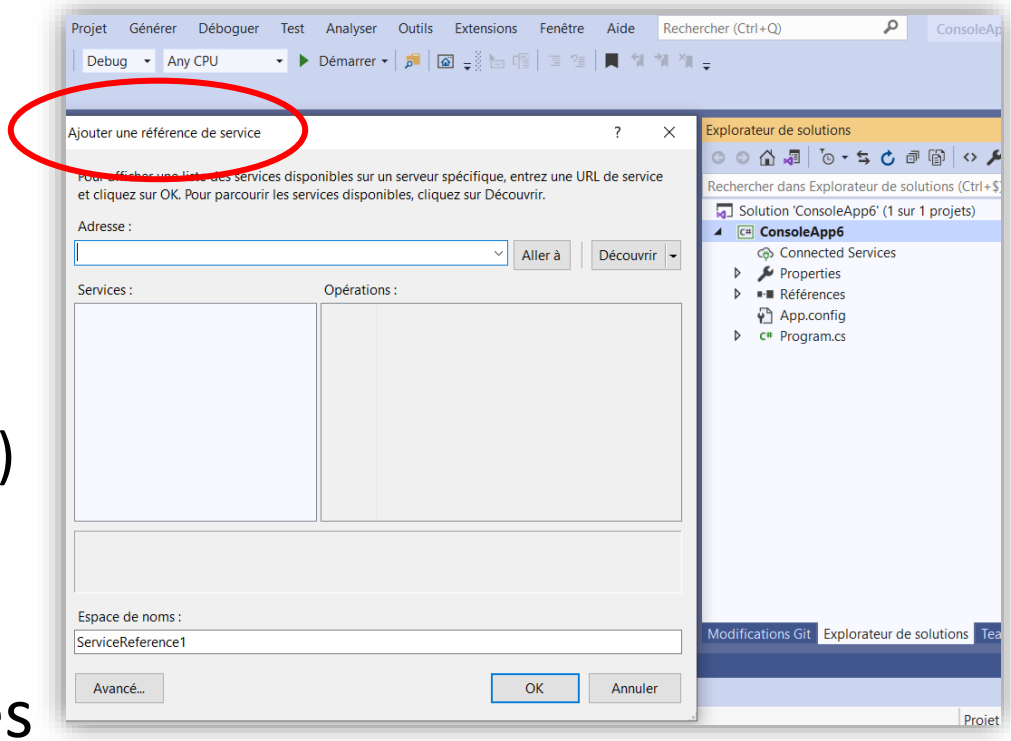
Port – Associates the binding with the URI http://www.examples.com/SayHello/ where the running service can be accessed.

```
<binding name = "Hello_Binding" type = "tns:Hello_PortType">
   <soap:binding style = "rpc"
      transport = "http://schemas.xmlsoap.org/soap/http"/>
   <operation name = "sayHello">
      <soap:operation soapAction = "sayHello"/>
      <input>
         <soap:body
            encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
            namespace = "urn:examples:helloservice"
            use = "encoded"/>
      </input>

      <output>
         <soap:body
            encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
            namespace = "urn:examples:helloservice"
            use = "encoded"/>
      </output>
   </operation>
</binding>

<service name = "Hello_Service">
   <documentation>WSDL File for HelloService</documentation>
   <port binding = "tns:Hello_Binding" name = "Hello_Port">
      <soap:address
         location = "http://www.examples.com/SayHello/" />
   </port>
</service>
</definitions>
```

WSDL and code generation of WS-SOAP Client

- Integrated tool in Visual Studio

  Projet / Add a web reference / …



- External tools for C#

  ServiceModel Metadata Utility Tool (Svcutil.exe)

  *svcutil *.wsdl *.xsd /language:C#*

- Other tools for other targets and languages

  Example to a javascript WS-SOAP Client

  wsdl2js -d javascript hello_world.wsdl

… and for WS-REST, what is available?

1. WS-REST and WADL

(Web Application Description Language)

- XML based
- Not popular enough currently

2. WS-REST and WSDL 2.0

(Web Services Description Language v 2.0)

- XML based
- Add `whttp:method` in the `wsdl:binding`

```
<wsdl:binding name="BookListHTTPBinding"
    type="http://www.w3.org/ns/wsdl/http"
    interface="">
  <wsdl:documentation>
     The REST HTTP binding for the book list service.
  </wsdl:documentation>
  <wsdl:operation ref="" whttp:method="GET"/>
</wsdl:binding>
```

… and for WS-REST, what is available?

3. WS-REST and Swagger and OpenAPI
   - *OpenAPI* Specification, originally known as the *Swagger* Specification, is a specification for machine-readable interface files for describing, producing, consuming, and visualizing RESTful web services
   - *Swagger* and some other tools can generate code, documentation and test cases given an interface file.
   - Thus, *OpenAPI* refers to the specification and *Swagger* refers to the family of open-source and commercial products for working with the *OpenAPI* Specification.

YAML and JSON OpenAPI description

- An OpenAPI document that conforms to the OpenAPI Specification is itself a JSON object, which may be represented either in JSON or YAML format
- YAML is a superset of JSON
- Swagger UI can read the openapi.json or openapi.yaml files equivalently

Object containing an array in JSON:

```json
{
"children": ["Avery","Callie","lucy","Molly"],
"hobbies": ["swimming","biking","drawing","horseplaying"]
}
```

Same object with an array in YAML:

```yaml
children:
  - Avery
  - Callie
  - lucy
  - Molly
hobbies:
  - swimming
  - biking
  - drawing
  - horseplaying
```

WS-REST and Swagger

- Swagger Tools and Integration : Swagger Codegen

- *Swagger Codegen* generates server stubs and client SDKs for any API, defined with the OpenAPI