Informatique Repartie

Chapitre 4 : SOAP

Cecilia Zanni-Merk

cecilia.zanni-merk@insa-rouen.fr

Bureau BO B R1 04

References

- Architectures réparties en Java
  Annick Fron
  ISBN 9782100738700
  Ed Dunod
- Java Web Services Up and Running
  Martin Kalin
  ISBN 9780596521127
  O’Reilly

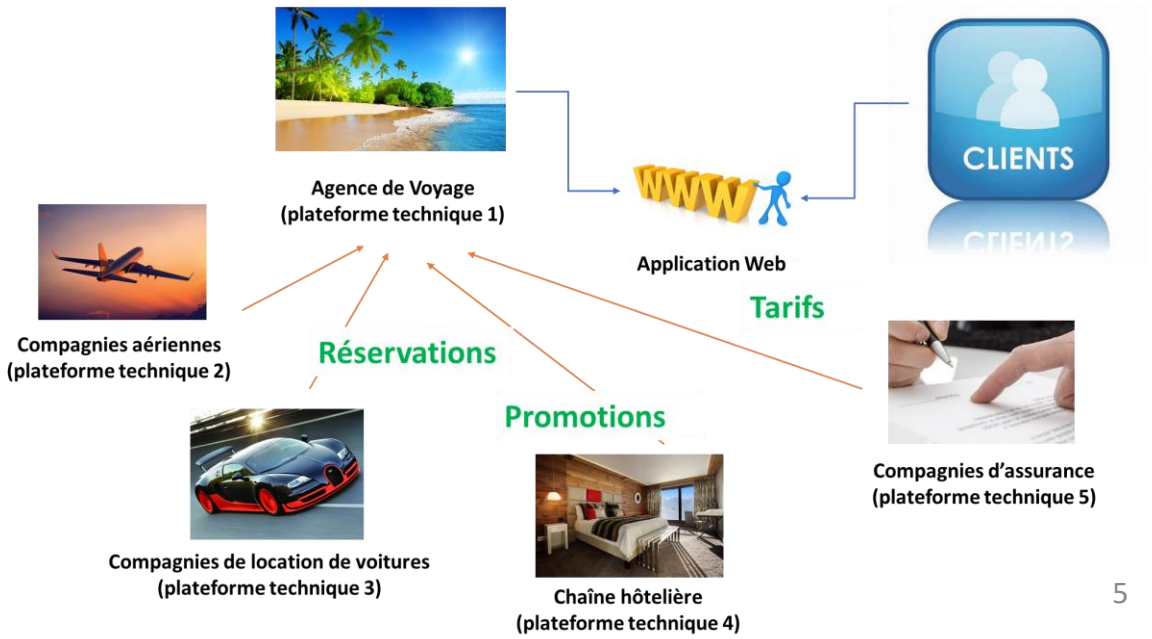
References in the Web

- https://www.w3schools.com/xml/xml_soap.asp
- http://apiacoa.org/publications/teaching/webservices/SOAP.pdf
- http://mbaron.developpez.com/cours/soa/soap/
- https://members.loria.fr/OPerrin/LProCisii/fichiers/SOAP.pdf

Introduction

Motivation

- Communication between applications on the Internet
  - B to B : carry out transactions or data exchanges between companies (business to business)
  - Communicate large amounts of data in loose coupling, with minimal knowledge of the other party and without being able to impose any service constraints.

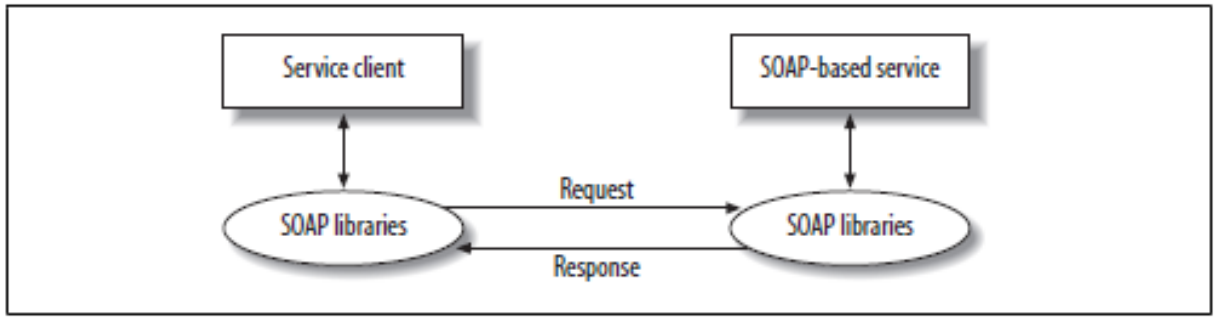


What is a Web Service ?

- Informal definition
  - It is a kind of *webified application*, that is, an application typically delivered over HTTP.
  - A WS is thus a distributed application whose components can be deployed and executed on distinct devices
- Formal definition (from the W3C)
  - A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format. Other systems interact with the Web service in a manner prescribed by its description using messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. https://www.w3.org/TR/ws-arch/#whatis

Types of Web Services

- Two types
  - SOAP-based
  - REST-based



- SOAP-based WS
  - Initially, Simple Object Access Protocol
  - Now, Service Oriented Architecture (SOA) Protocol
  - For now, SOAP is just an XML dialect in which documents are messages
  - In a typical SOAP-based web service, a client transparently sends a SOAP document as a request to a web service, which transparently returns another SOAP document as a response.

Types of Web Services

- REST-based WS
  - Representational State Transfer
  - Less standardized than SOAP, REST has few toolkits and “skinny” software libraries
  - The REST style is often seen as an antidote to the complexity of SOAP-based web services.
  - In a REST-style service, a client might send a standard HTTP request to a web service and receive an appropriate XML document as a response.

Key Features of WS

- Distinguishing them from other distributed software systems
- *Open infrastructure*
  - Web services are deployed using industry-standard, vendor-independent protocols such as HTTP and XML, which are ubiquitous and well understood.
- *Language transparency*
  - Web services and their clients can interoperate even if written in different programming language, that provide libraries, utilities, and even frameworks in support of web services.
  - There should be an intermediary to handle the differences in data types between the languages … they are the XML technologies
- *Modular design*
  - Web services are meant to be modular in design so that new services can be generated through the integration and layering of existing services

Standardization of Web Services

- Several norms
  - W3C
  - The OASIS consortium (http://www.oasis-open.org)
- Serveral actors
  - Microsoft .Net
  - Apache : Axis, CXF
  - Sun : JAX-WS and Metro
  - JBoss/WildFly
  - Other open-source implementations
- Interoperability
  - The idea is that the services can be used by everybody
  - WS-I.org : Web Service Interoperability organization (http://www.ws-i.org)

Why use Web Services?

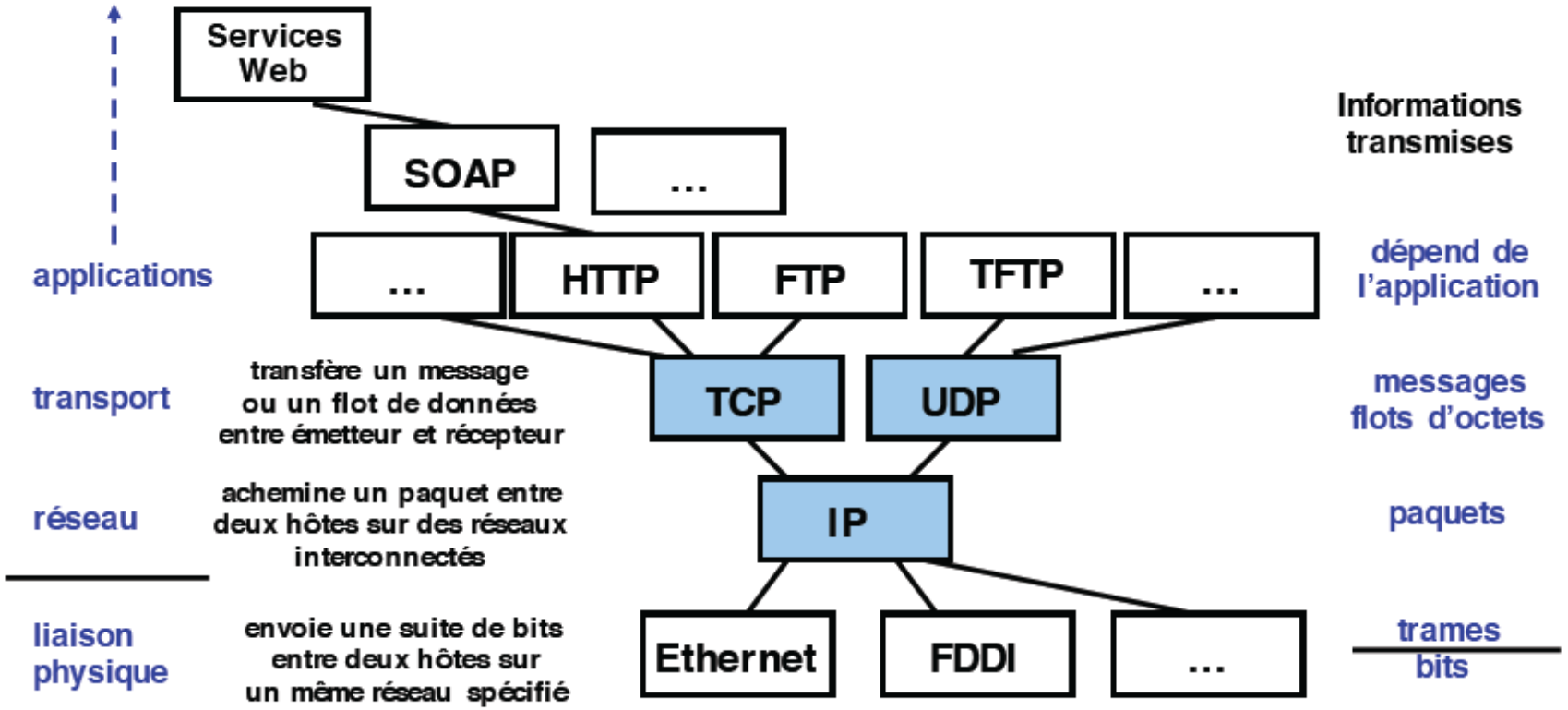
- Modern software systems are written in a variety of languages. These software systems will continue to be hosted on a variety of platforms.
- Companies generally have significant investment in legacy software systems whose functionality is useful and perhaps mission critical
- Additionally, interoperability is not just a long-term challenge but also a current requirement of production software
- Web services address these issues directly because such services are, first and foremost, language- and platform-neutral.
  - If a legacy COBOL system is exposed through a web service, the system is thereby interoperable with service clients written in other programming languages.

SOAP

Introduction – A first example – A word about WSDL – The hidden SOAP - A richer example

SOAP

- SOAP is an RPC protocol that uses XML to serialize methods and their arguments, as well as their return values. With SOAP, XML transits over HTTP
- W3C references
  - **XML** (Extensible markup Language)
    http://www.w3.org/XML/
    http://www.w3.org/XML/Schema
  - **HTTP** (Hypertext Transfer Protocol)
    http://www.w3.org/Protocols/
  - **SOAP** (Simple Object Access Protocol)
    https://www.w3.org/TR/soap12



HTTP : *HyperText Transfer Protocol* : protocole du Web
TFTP, FTP : (*Trivial*) *File Transfer Protocol*) : transfert de fichiers
TCP : *Transmission Control Protocol* : transport en mode connecté
UDP : *User Datagram Protocol* : transport en mode non connecté
IP : *Internet Protocol* : Interconnexion de réseaux, routage

Cours Intro aux Réseaux – Olivier Dalle – Université de Nice

A first SOAP example

- All the examples can be compiled and deployed using core Java SE (Java Standard 6 or greater) without any additional software, until Java version 8.
  - Afterwards, an application server (such as Apache Tomcat, is necessary)
- All of the libraries required to compile, execute, and consume web services are available in core Java, which supports *JAX-WS* (Java API for XML-Web Services).
- JAX-WS supports SOAP-based and REST-style services and is commonly shortened to *JWS* for Java Web Services.

A note about compilation and execution

- All the development of SOAP web services will be done by hand during this course, meaning that we will need to use Java version 8 (normally available in the school network)
- To do so :
  1. First, look for the installation folder of Java version 8 in your system. Normally, it is `/opt/jre-XXX/`
  2. Update your PATH environment variable doing `export PATH=<the folder you have found in 1>:$PATH` in a terminal. You should have something like this: `export PATH=/opt/jre-XYZ/bin/:$PATH`.
  3. To compile you do `javac --release 8 <files to compile>`
  4. To execute you that normally: `java <class to execute>`

A note about compilation and execution

- As indicated in the previous slide, we will use **Java version 8** to facilitate the implementation of the source code to develop
- Please install ***OpenJDK version 8*** in your personal laptops
  - A step by step tutorial done by an ITI student in 2021 is available on Moodle
- You can also visit ***https://adoptopenjdk.net/index.html***

A first SOAP example

- Generally, a Java-based web service consists of an interface and an implementation.
  - The interface declares the methods, which are the web service operations. The interface is called the *SEI*: Service Endpoint Interface.
  - The implementation defines the methods declared in the interface. The implementation is called the *SIB*: Service Implementation Bean.
- The SIB can be either a POJO (a Plain Old Java Object) or a Stateless Session *EJB* (Enterprise Java Bean).
  - For the moment, the SOAP-based web services will be implemented as POJOs, that is, as instances of regular Java classes.

A first SOAP example

- Implement a web service that returns the current time as either a string or as the elapsed milliseconds from the Unix epoch, midnight January 1, 1970 GMT.
- Procedure
  1. Develop the SEI and the SIB
  2. Implement an application to publish the web service
  3. Develop the client to consume the service

The SEI and the SIB

TimeServer.java

```java
package examples.ts; // time server

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;

/**
* The annotation @WebService signals that this is the
* SEI (Service Endpoint Interface). @WebMethod signals
* that each method is a service operation.
*
* The @SOAPBinding annotation impacts the under-the-hood
* construction of the service contract, the WSDL
* (Web Services Definition Language) document. Style.RPC
* simplifies the contract and makes deployment easier.
*/

@WebService
@SOAPBinding(style = Style.RPC) // more on this later
public interface TimeServer {
@WebMethod String getTimeAsString();
@WebMethod long getTimeAsElapsed();
}
```

TimeServerImpl.java

```java
package examples.ts; // time server

import java.util.Date;
import javax.jws.WebService;

/**
* The @WebService property endpointInterface links the
* SIB (this class) to the SEI (examples.ts.TimeServer).
* Note that the method implementations are not annotated
* as @WebMethods.
*/

@WebService(endpointInterface = "examples.ts.TimeServer")
public class TimeServerImpl implements TimeServer {
public String getTimeAsString() { return new Date().toString(); }
public long getTimeAsElapsed() { return new Date().getTime(); }
}
```

JWS annotations

- The API for the Web services annotations is : `javax.jws`
  - `@WebService`
  - `@SOAPBinding`
  - `@WebParam`
  - `@WebResult`
  - …
- Java 6 provides a mini web server with the class `Endpoint.publish()`

A word about publication …

- Once the SEI and SIB have been compiled, the web service is ready to be published.
- To compile the SEI and the SIB, from the working directory, that has the `examples` folder inside, do
  ```
  % javac --release 8./examples/ts/*.java
  ```
- In full production mode, a Java Application Server such as BEA WebLogic, GlassFish, JBoss, or WebSphere might be used; but in development and even light production mode, a simple Java application can be used.

The publisher application

TimeServerPublisher.java

```
1  package examples.ts;
2
3  import javax.xml.ws.Endpoint;
4  /**
5  * This application publishes the web service whose SIB is examples.ts.TimeServerImpl. For now, the
6  * service is published at network address 127.0.0.1., * which is localhost, and at port number 9876,
7  * as this port is likely available on any desktop machine. The * publication path is /ts, an arbitrary name.
8  *
9  * The Endpoint class has an overloaded publish method. In this two-argument version, the first argument is the
10 * publication URL as a string and the second argument is an instance of the service SIB, in this case
11 * examples.ts.TimeServerImpl.
12 *
13 * The application runs indefinitely, awaiting service requests. It needs to be terminated at the command prompt
14 * with control-C * or the equivalent.
15 *
16 * Once the applicatation is started, open a browser to the URL * http://127.0.0.1:9876/ts?wsdl
17 * to view the service contract, the WSDL document. This is an easy test to determine whether the service has
18 * deployed successfully. If the test succeeds, a client then can be executed against the service.
19 */
20
21 public class TimeServerPublisher {
22 public static void main(String[ ] args) {
23 // 1st argument is the publication URL
24 // 2nd argument is an SIB instance
25 Endpoint.publish("http://127.0.0.1:9876/ts", new TimeServerImpl());
26 }
27 }
```

The publisher application

TimeServerPublisher.java

```
1  package examples.ts;
2
3  import javax.xml.ws.Endpoint;
4  /**
5  * This application publishes the web service whose SIB is examples.ts.TimeServerImpl. For now, the
6  * service is published at network address 127.0.0.1., * which is localhost, and at port number 9876,
7  * as this port is likely available on any desktop machine. The * publication path is /ts, an arbitrary name.
8  *
9  * The Endpoint
10 * publication U
11 * examples.ts.T
12 *
13 * The applicati
14 * with control-
15 *
16 * Once the appl
17 * to view the s
18 * deployed succ
19 */
20
21 public class TimeServerPublisher {
22 public static void main(String[ ] args) {
23 // 1st argument is the publication URL
24 // 2nd argument is an SIB instance
25 Endpoint.publish("http://127.0.0.1:9876/ts", new TimeServerImpl());
26 }
27 }
```

- Compile from the working directory as you did before for the SEI and the SIB
- Execute

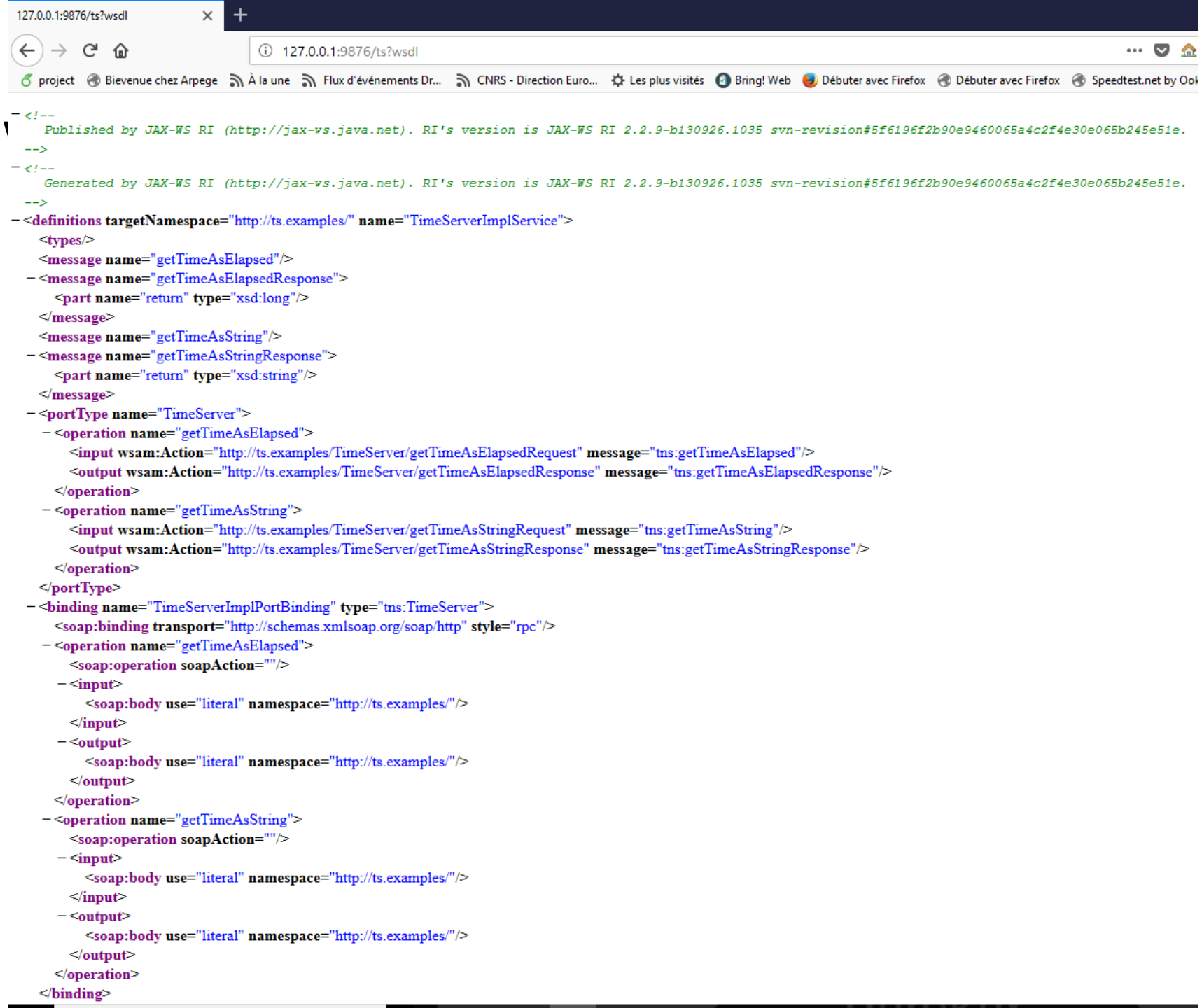
```
% java examples.ts.TimeServerPublisher
```

Testing the Web Service with a Browser

- We can test the deployed service by opening a browser and viewing the *WSDL* (Web Service Definition Language) document, which is an automatically generated service contract.
- The browser is opened to a URL that has two parts. The first part is the URL published in the Java `TimeServerPublisher` application: *http://127.0.0.1:9876/ts*.
- Appended to this URL is the query string *?wsdl*
- The result is *http://127.0.0.1:9876/ts?wsdl*.

Testing the Web Service with a Browser

Testing the


127.0.0.1:9876/ts?wsdl
127.0.0.1:9876/ts?wsdl
project
Bievenue chez Arpege
À la une
Flux d'événements Dr...
CNRS - Direction Euro...
Les plus visités
Bring! Web
Débuter avec Firefox
Débuter avec Firefox
Speedtest.net by Ook
<!--
Published by JAX-WS RI (http://jax-ws.java.net). RI's version is JAX-WS RI 2.2.9-b130926.1035 svn-revision#5f6196f2b90e9460065a4c2f4e30e065b245e51e.
-->
<!--
Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is JAX-WS RI 2.2.9-b130926.1035 svn-revision#5f6196f2b90e9460065a4c2f4e30e065b245e51e.
-->
<definitions targetNamespace="http://ts.examples/" name="TimeServerImplService">
<types/>
<message name="getTimeAsElapsed"/>
<message name="getTimeAsElapsedResponse">
<part name="return" type="xsd:long"/>
</message>
<message name="getTimeAsString"/>
<message name="getTimeAsStringResponse">
<part name="return" type="xsd:string"/>
</message>
<portType name="TimeServer">
<operation name="getTimeAsElapsed">
<input wsam:Action="http://ts.examples/TimeServer/getTimeAsElapsedRequest" message="tns:getTimeAsElapsed"/>
<output wsam:Action="http://ts.examples/TimeServer/getTimeAsElapsedResponse" message="tns:getTimeAsElapsedResponse"/>
</operation>
<operation name="getTimeAsString">
<input wsam:Action="http://ts.examples/TimeServer/getTimeAsStringRequest" message="tns:getTimeAsString"/>
<output wsam:Action="http://ts.examples/TimeServer/getTimeAsStringResponse" message="tns:getTimeAsStringResponse"/>
</operation>
</portType>
<binding name="TimeServerImplPortBinding" type="tns:TimeServer">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
<operation name="getTimeAsElapsed">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal" namespace="http://ts.examples/"/>
</input>
<output>
<soap:body use="literal" namespace="http://ts.examples/"/>
</output>
</operation>
<operation name="getTimeAsString">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal" namespace="http://ts.examples/"/>
</input>
<output>
<soap:body use="literal" namespace="http://ts.examples/"/>
</output>
</operation>
</binding>

A word about WSDL …

- Web services interfaces are described in WSDL.
  - https://www.w3.org/TR/wsdl20/
- This description is sufficient to use the service without knowing its implementation.
- The purpose is to enable an application to communicate on the Internet with the service it needs and to exchange data with it
  - The implementation infrastructure is heavier than for RMI (we need a web server); but it is essential if we want to pass information through a firewall.

A word about WSDL …

- Two sections deserve a quick look
- The `portType` section groups the operations that the web service delivers, in this case the operations `getTimeAsString` and `getTimeAsElapsed`, which are the two Java methods declared in the SEI and implemented in the SIB.

```
− <portType name="TimeServer">
  − <operation name="getTimeAsString">
      <input wsam:Action="http://ts.examples/TimeServer/getTimeAsStringRequest" message="tns:getTimeAsString"/>
      <output wsam:Action="http://ts.examples/TimeServer/getTimeAsStringResponse" message="tns:getTimeAsStringResponse"/>
    </operation>
  − <operation name="getTimeAsElapsed">
      <input wsam:Action="http://ts.examples/TimeServer/getTimeAsElapsedRequest" message="tns:getTimeAsElapsed"/>
      <output wsam:Action="http://ts.examples/TimeServer/getTimeAsElapsedResponse" message="tns:getTimeAsElapsedResponse"/>
    </operation>
  </portType>
```

A word about WSDL …

- The other WSDL section of interest is the `service` section, and in particular the service location, in this case the URL *http://localhost:9876/ts*.
- The URL is called the *service endpoint* and it informs clients about where the service can be accessed

```
− <service name="TimeServerImplService">
  − <port name="TimeServerImplPort" binding="tns:TimeServerImplPortBinding">
      <soap:address location="http://localhost:9876/ts"/>
    </port>
  </service>
```

A word about WSDL …

- The WSDL document is useful for both creating and executing clients against a web service.
- The core Java utility for generating client-support code from a WSDL document is called *wsimport*.
- At runtime, a client can consume the WSDL document associated with a web service in order to get critical information about the data types associated with the operations bundled in the service.
  - For example, a client could determine from our first WSDL that the operation `getTimeAsElapsed` returns an integer and expects no arguments.

A word about WSDL …

- The WSDL document is useful for both creating and executing clients against a web service.
- The core Java utility for generating client-support code from a WSDL document is called *wsimport*.
- At runtime, a client can consume the WSDL document associated with a web service in order to get critical information about the data types associated with the operations bundled in the service.
  - For example, a client could determine from our first WSDL that the operation `getTimeAsElapsed` returns an integer and expects no arguments.
  - Where in the WSDL document? To do later!

A Java client for the Time server

TimeClient.java

```
1  package examples.ts;
2
3  import javax.xml.namespace.QName;
4  import javax.xml.ws.Service;
5  import java.net.URL;
6
7  class TimeClient {
8      public static void main(String args[ ]) throws Exception {
9      URL url = new URL("http://localhost:9876/ts?wsdl");
10
11     // Qualified name of the service:
12     // 1st arg is the service URI
13     // 2nd is the service name published in the WSDL
14     QName qname = new QName("http://ts.examples/", "TimeServerImplService");
15
16     // Create, in effect, a factory for the service.
17     Service service = Service.create(url, qname);
18
19     // Extract the endpoint interface, the service "port".
20     TimeServer eif = service.getPort(TimeServer.class);
21
22     System.out.println(eif.getTimeAsString());
23     System.out.println(eif.getTimeAsElapsed());
24     }
25 }
```

A Java client for the Time server

TimeClient.java

```java
package examples.ts;

import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import java.net.URL;

class TimeClient {
    public static void main(String args[ ]) throws Exception {
    URL url = new URL("http://localhost:9876/ts?wsdl");

    // Qualified name of the service:
    // 1st arg is the service URI
    // 2nd is the service name published in the WSDL
    QName qname = new QName("http://ts.examples/", "TimeServerImplService");

    // Create, in effect, a factory for the service.
    Service service = Service.create(url, qname);

    // Extract the endpoint interface, the service "port".
    TimeServer eif = service.getPort(TimeServer.class);

    System.out.println(eif.getTimeAsString());
    System.out.println(eif.getTimeAsElapsed());
    }
}
```

```
−<definitions targetNamespace="http://ts.examples/" name="TimeServerImplService">
   <types/>
   <message name="getTimeAsString"/>
  −<message name="getTimeAsStringResponse">
     <part name="return" type="xsd:string"/>
   </message>
   <message name="getTimeAsElapsed"/>
  −<message name="getTimeAsElapsedResponse">
     <part name="return" type="xsd:long"/>
   </message>
```

A Java client for the Time server

TimeClient.java

```java
package examples.ts;

import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import java.net.URL;

class TimeClient {
    public static void main(String args[ ]) throws Exception {
    URL url = new URL("http://localhost:9876/ts?wsdl");

    // Qualified name of the service:
    // 1st arg is the service URI
    // 2nd is the service name published in the WSDL
    QName qname = new QName("http://ts.examples/", "TimeServerImplService");

    // Create, in effect, a factory for the service.
    Service service = Service.create(url, qname);

    // Extract the endpoint interface, the service "port".
    TimeServer eif = service.getPort(TimeServer.class);

    System.out.println(eif.getTimeAsString());
    System.out.println(eif.getTimeAsElapsed());
    }
}
```

```xml
<definitions targetNamespace="http://ts.examples/" name="TimeServerImplService">
  <types/>
  <message name="getTimeAsString"/>
  <message name="getTimeAsStringResponse">
    <part name="return" type="xsd:string"/>
  </message>
  <message name="getTimeAsElapsed"/>
  <message name="getTimeAsElapsedResponse">
    <part name="return" type="xsd:long"/>
  </message>
```

Invite de commandes

```
C:\Users\Cecilia\enseignement INSA Rouen\2017-2018\Informatique Répartie\chapitre 4 - SOAP>java examples.ts.TimeClient
Tue Mar 13 17:16:07 CET 2018
1520957767497
```

A Java client for the Time server

- The Java client uses the URL with a query string (http://localhost:9876/ts?wsdl) and explicitly creates an XML *qualified name* for the service, which has the syntax *namespace URI:local name*.
  - The Java class `java.xml.namespace.QName` represents an XML-qualified name.
  - In this example, the *namespace URI* is provided in the WSDL, and the *local name* is the SIB class name `TimeServerImpl` with the word `Service` appended.
    - **The local name occurs in the service section, the last section of the WSDL document.**

A Java client for the Time server

- Once the `URL` and `QName` objects have been constructed and the `Service.create` method has been invoked, we have the statement of interest:
```
TimeServer port =
service.getPort(TimeServer.class)
```
- Recall that, in the WSDL document, the `portType` section describes, in the style of an interface, the operations included in the web service.
- The `getPort` method returns a reference to a Java object that can invoke the `portType` operations.
- The port object reference is of type `examples.ts.TimeServer`, which is the SEI type.

The hidden SOAP

- In SOAP-based web services, a client typically makes a remote procedure call against the service by invoking one of the web service operations.
- As mentioned earlier, this back and forth between the client and service is the request/response message exchange pattern, and the SOAP messages exchanged in this pattern allow the web service and a consumer to be programmed in different languages.
- Typically, a client generates an HTTP request, which is itself a formatted message whose *body* is a **SOAP message**.

The hidden SOAP

exempleRequeteHTTP.xml

```
POST http://127.0.0.1:9876/ts HTTP/ 1.1
Accept: text/xml
Accept: multipart/*
Accept: application/soap
User-Agent: Javaxxx
Content-Length: 434
Content-Type: text/xml; charset=utf-8
SOAPAction: ""

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
        soap:encodingStyle="http:// schemas.xmlsoap.org/soap/encoding/"
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:tns="http://ts.examples/"
        xmlns:xsd ="http://www.w3.org/2001/XMLSchema">
    <soap:Body>
        <tns:getTimeAsString xsi:nil="true" />
    </soap:Body>
</soap:Envelope>
```

- The SOAP document or or message is commonly called SOAP envelope.
- In this SOAP envelope, the SOAP body contains a single element whose *local name* is `getTimeAsString`, which is the name of the web service operation that the client wants to invoke.

The hidden SOAP

- On the web service side, the underlying Java libraries process the HTTP request, extract the SOAP envelope, determine the identity of the requested service operation, invoke the corresponding Java method `getTimeAsString`, and then generate the appropriate SOAP message to carry the method's return value back to the client.

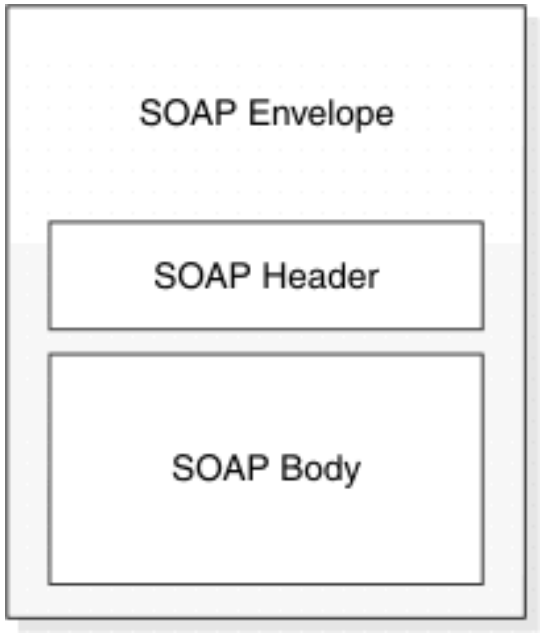
exempleReponseHTTP.xml

```
1  HTTP/1.1 200 OK
2  Content-Length: 323
3  Content-Type: text/xml; charset=utf-8
4  Client-Date: Sat, 24 Feb 2018 02:12:54 GMT
5  Client-Peer: 127.0.0.1:9876
6  Client-Response-Num: 1
7
8  <?xml version="1.0" ?>
9  <soapenv:Envelope
10         xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
11         xmlns:xsd="http://www.w3.org/2001/XMLSchema">
12     <soapenv:Body>
13         <ans:getTimeAsStringResponse xmlns:ans="http://ts.examples/">
14             <return>Sat Feb 24 14:12:54 CST 2018</return>
15         </ans:getTimeAsStringResponse>
16     </soapenv:Body>
17 </soapenv:Envelope>
```

The hidden SOAP

- Once again, the SOAP envelope is the body of an HTTP message, in this case the HTTP response to the client.
  - The HTTP *start line* now contains the status code as the integer 200 and the corresponding text OK, which signal that the client request was handled successfully.
- The SOAP envelope in the HTTP response's body contains the current time as a string named `return`.
- The Java SOAP library on the client's side extracts the SOAP envelope from the HTTP response and, because of information in the WSDL document, expects the desired return value from the web service operation to occur in the XML return element.

The SOAP envelope



- The SOAP envelope is a wrapper element that identifies the subordinate elements as a SOAP message and provides namespace declarations. Namespaces provide semantic context for elements within the SOAP body.
- The SOAP header is an optional element that can contain metadata, such as authentication information, localization support, and delivery routes.
- The SOAP body contains the payload of the message, which is either the web service request or web service response. The response can be a processing error, which is called a SOAP fault.

Error management

- <soap:fault> contained in the body.
- The error element is optional and appears only in response messages and only once.
- Four optional sub-tags
  - faultcode
  - faultstring
  - faultactor
  - detail
- Four types of error codes
  - soap:Server
  - soap:Client
  - soap:VersionMismatch
  - soap:MustUnderstand

A richer example

- The operations in the `TimeServer` service take no arguments and return simple types, a string and an integer.
- The `Teams` web service is a richer example with several differences
  - The Teams service is implemented as a single Java class rather than as a separate SEI and SIB. This is done simply to illustrate the possibility.
  - A more important difference is in the return types of the two `Teams` operations.
    - The operation `getTeam` is parameterized and returns an object of the programmer-defined type `Team`, which is a list of `Player` instances, another programmer-defined type.
    - The operation `getTeams` returns a `List<Team>`, that is, a `Java Collection`.

A richer example

Teams.java | TeamsUtility.java | Player.java | Team.java | TeamsPublisher.java

```
1  package examples.team;
2
3  import java.util.List;
4  import javax.jws.WebService;
5  import javax.jws.WebMethod;
6
7  @WebService
8  public class Teams {
9      private TeamsUtility utils;
10
11     public Teams() {
12         utils = new TeamsUtility();
13         utils.make_test_teams();
14     }
15
16     @WebMethod
17     public Team getTeam(String name) { return utils.getTeam(name); }
18
19     @WebMethod
20     public List<Team> getTeams() { return utils.getTeams(); }
21 }
22
```

A richer example

- The utility class `TeamsUtility` generates the data. In a production environment, this utility might retrieve a team or list of teams from a database.
- To keep this example simple, the utility instead creates the teams and their players on the fly.

A richer example

TeamsUtility.java | Teams.java | Player.java | Team.java | TeamsPublisher.java

```
5   import java.util.ArrayList;
6   import java.util.Map;
7   import java.util.HashMap;
8
9   public class TeamsUtility {
10      private Map<String, Team> team_map;
11
12      public TeamsUtility() {
13          team_map = new HashMap<String, Team>();
14      }
15
16      public Team getTeam(String name) { return team_map.get(name); }
17
18      public List<Team> getTeams() {
19          List<Team> list = new ArrayList<Team>();
20          Set<String> keys = team_map.keySet();
21          for (String key : keys)
22              list.add(team_map.get(key));
23          return list;
24      }
25
26      public void make_test_teams() {
27          List<Team> teams = new ArrayList<Team>();
28
29          Player chico = new Player("Leonard Marx", "Chico");
30          Player groucho = new Player("Julius Marx", "Groucho");
31          Player harpo = new Player("Adolph Marx", "Harpo");
32
33          List<Player> mb = new ArrayList<Player>();
34          mb.add(chico); mb.add(groucho); mb.add(harpo);
35
36          Team marx_brothers = new Team("Marx Brothers", mb);
37          teams.add(marx_brothers);
38          store_teams(teams);
39      }
40
41      private void store_teams(List<Team> teams) {
42          for (Team team : teams)
43          team_map.put(team.getName(), team);
44      }
```

Publishing the service and writing the client

- Recall that the SEI for the `TimeServer` service contains the annotation:
  `@SOAPBinding(style = Style.RPC)`
- This annotation requires that the service use only very simple types such as string and integer.
- By contrast, the `Teams` service uses richer data types, which means that `Style.DOCUMENT`, the default, should replace `Style.RPC`.
  - The document style requires more setup, to be explained later

Publishing the service and writing the client

- The steps to have the service deployed and a sample client written quickly are:

1. The source files are compiled in the usual way. From the working directory, which has *examples* as a subdirectory, the command is:
   `% javac --release 8 examples/team/*.java`
   In addition to the `@WebService`-annotated `Teams` class, the *examples/team* directory contains the `Team`, `Player`, `TeamsUtility`, and `TeamsPublisher` classes, that you will find in Moodle

Publishing the service and writing the client

2. In the working directory, invoke the `wsgen` utility, which comes with core Java 6:
   ```
   % wsgen -cp . examples.team.Teams
   ```
   This utility generates various *artifacts*; that is, Java types needed by the method `Endpoint.publish` to generate the service's WSDL.
3. Execute the `TeamsPublisher` application.
4. For preparing the development of the client, in its working directory, invoke the `wsimport` utility, which also comes with core Java 6:
   ```
   % wsimport -p teamsC -keep http://localhost:8888/teams?wsdl
   ```
5. Write the client

The client

TeamClient.java

```
1  package teamsC;
2
3  import teamsC.TeamsService;
4  import teamsC.Teams;
5  import teamsC.Team;
6  import teamsC.Player;
7  import java.util.List;
8
9  public class TeamClient {
10
11     public static void main(String[ ] args) {
12         TeamsService service = new TeamsService();
13         Teams port = service.getTeamsPort();
14         List<Team> teams = port.getTeams();
15         for (Team team : teams) {
16             System.out.println("Team name: " + team.getName() + " (roster count: " +
17                                                     team.getRosterCount() + ")");
18             for (Player player : team.getPlayers())
19             System.out.println(" Player: " + player.getNickname());
20         }
21     }
22 }
23
```

The clie

TeamClient.java

```
1  package teamsC;
2
3  import teamsC.TeamsSe
4  import teamsC.Teams;
5  import teamsC.Team;
6  import teamsC.Player;
7  import java.util.List;
8
9  public class TeamClient {
10
11     public static void main(String[ ] args) {
12         TeamsService service = new TeamsService();
13         Teams port = service.getTeamsPort();
14         List<Team> teams = port.getTeams();
15         for (Team team : teams) {
16             System.out.println("Team name: " + team.getName() + " (roster count: " +
17                                                     team.getRosterCount() + ")");
18             for (Player player : team.getPlayers())
19             System.out.println(" Player: " + player.getNickname());
20         }
21     }
22 }
23
```

C:\Windows\System32\cmd.exe

```
C:\Users\Cecilia\enseignement INSA Rouen\2017-2018\Informatique Répartie\chapitre 4 - SOAP\examples>java teamsC.TeamClie
nt
Team name: Abbott and Costello (roster count: 2)
 Player: Bud
 Player: Lou
Team name: Burns and Allen (roster count: 2)
 Player: George
 Player: Gracie
Team name: Marx Brothers (roster count: 3)
 Player: Chico
 Player: Groucho
 Player: Harpo
```

Some final remarks

- Developing a web service with JAX-WS requires several steps:
  - Code the class that encapsulates the service
  - Compile the class
  - Use the `wsgen` command to generate the files required for deployment (schemas, WSDL, classes, ...)
- To define an endpoint with JAX-WS, there are several constraints :
  - The class that encapsulates the endpoint must be `public`, not `static`, not `final`, not `abstract` and be annotated with `@WebService`
  - It must have a default constructor (without parameters)
  - It is recommended to explicitly define the SEI interface
  - Methods exposed by the web service must be `public`, not `static`, not `final` and annotated with `@WebMethod`
  - The parameter and return value types of these methods must be supported by JAXB

WSDL

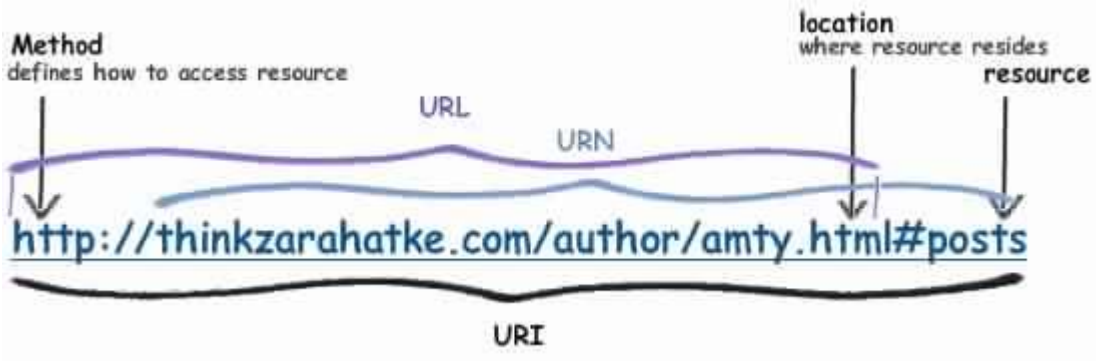
A few recalls

- Let's see again the code of the client against the `TimeServer` service

```
URL url = new URL("http://localhost:9876/ts?wsdl");
QName qname = new QName("http://ts.examples/", "TimeServerImplService");
Service service = Service.create(url, qname);
```

- The client invokes the `Service.create` method with two arguments:
  - a URL, which provides the endpoint at which the service can be accessed, and
  - an XML-qualified name (a Java `QName`), which in turn consists of the service's local name (in this case, `TimeServerImplService`) and a namespace identifier (in this case, the URI `http://ts.examples/`).

A few recalls



- Let's see again the code of the client against the `TimeServer` service

```
URL url = new URL("http://localhost:9876/ts?wsdl");
QName qname = new QName("http://ts.examples/", "TimeServerImplService");
Service service = Service.create(url, qname);
```

- The client invokes the `Service.create` method with two arguments:
  - a URL, which provides the endpoint at which the service can be accessed, and
  - an XML-qualified name (a Java `QName`), which in turn consists of the service's local name (in this case, `TimeServerImplService`) and a namespace identifier (in this case, the URI `http://ts.examples/`).

Generating Client-Support Code from a WSDL

- As seen before, the use of `wsimport` can ease the writing of a client to consume a SOAP web service
- Let's analyse in detail what happens
- After the `examples.ts.TimeServerPublisher` application has been started, the command:
  `% wsimport -keep -p client http://localhost:9876/ts?wsdl`
  generates two source and two compiled files in the subdirectory *client*.

Generating Client-Support Code from a WSDL

TimeServer.java

```
1  package client;
2
3  import javax.jws.WebMethod;
4  import javax.jws.WebResult;
5  import javax.jws.WebService;
6  import javax.jws.soap.SOAPBinding;
7
8  @WebService(name = "TimeServer", targetNamespace = "http://ts.examples/")
9
10 @SOAPBinding(style = SOAPBinding.Style.RPC)
11 public interface TimeServer {
12     @WebMethod
13     @WebResult(partName = "return")
14     public String getTimeAsString();
15
16     @WebMethod
17     @WebResult(partName = "return")
18     public long getTimeAsElapsed();
19 }
20
```

Generating Client-Support Code from a WSDL

TimeServer.java

```
1  package client;
2
3  import javax.jws.WebMethod;
4  import javax.jws.WebResult;
5  import javax.jws.WebService;
6  import javax.jws.soap.SOAPBinding;
7
8  @WebService(name = "TimeServer", targetNamespace = "http://ts.examples/")
9
10 @SOAPBinding(style = SOAPBinding.Style.RPC)
11 public interface TimeServer {
12     @WebMethod
13     @WebResult(partName = "return")
14     public String getTimeAsString();
15
16     @WebMethod
17     @WebResult(partName = "return")
18     public long getTimeAsElapsed();
19 }
20
```

Generating Client-Support Code from a WSDL

TimeServer.java

```
1  package client;
2
3  import javax.jws.WebMethod;
4  import javax.jws.WebResult;
5  import javax.jws.WebService;
6  import javax.jws.soap.SOAPBinding;
7
8  @WebService(name = "TimeServer", targetNamespace = "http://ts.examples/")
9
10 @SOAPBinding(style = SOAPBinding.Style.RPC)
11 public interface TimeServer {
12     @WebMethod
13     @WebResult(partName = "return")
14     public String getTimeAsString();
15
16     @WebMethod
17     @WebResult(partName = "return")
18     public long getTimeAsElapsed();
19 }
20
```

Generating Client-Support Code from a WSDL

G DL

TimeServerImplService.java

```
1  package client;
2
3  import java.net.MalformedURLException;
4  import java.net.URL;
5  import javax.xml.namespace.QName;
6  import javax.xml.ws.Service;
7  import javax.xml.ws.WebEndpoint;
8  import javax.xml.ws.WebServiceClient;
9
10 @WebServiceClient(name = "TimeServerImplService",
11 targetNamespace = "http://ts.examples/",
12 wsdlLocation = "http://localhost:9876/ts?wsdl")
13 public class TimeServerImplService extends Service {
14     private final static URL TIMESERVERIMPLSERVICE_WSDL_LOCATION;
15     static {
16         URL url = null;
17         try {
18             url = new URL("http://localhost:9876/ts?wsdl");
19             }
20         catch (MalformedURLException e) {
21                 e.printStackTrace();
22                 }
23         TIMESERVERIMPLSERVICE_WSDL_LOCATION = url;
24     }
25
26     public TimeServerImplService(URL wsdlLocation, QName serviceName) {
27         super(wsdlLocation, serviceName);
28     }
29
30     public TimeServerImplService() {
31         super(TIMESERVERIMPLSERVICE_WSDL_LOCATION,
32             new QName("http://ts.examples/", "TimeServerImplService"));
33     }
34
35     @WebEndpoint(name = "TimeServerImplPort")
36     public TimeServer getTimeServerImplPort() {
37         return (TimeServer)super.getPort(new QName("http://ts.examples/", "TimeServerImplPort"), TimeServer.class);
38     }
39 }
```

G DL

TimeServerImplService.java

```java
package client;

import java.net.MalformedURLException;
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import javax.xml.ws.WebEndpoint;
import javax.xml.ws.WebServiceClient;

@WebServiceClient(name = "TimeServerImplService",
targetNamespace = "http://ts.examples/",
wsdlLocation = "http://localhost:9876/ts?wsdl")
public class TimeServerImplService extends Service {
    private final static URL TIMESERVERIMPLSERVICE_WSDL_LOCATION;
    static {
        URL url = null;
        try {
            url = new URL("http://localhost:9876/ts?wsdl");
            }
        catch (MalformedURLException e) {
                e.printStackTrace();
                }
        TIMESERVERIMPLSERVICE_WSDL_LOCATION = url;
    }

    public TimeServerImplService(URL wsdlLocation, QName serviceName) {
        super(wsdlLocation, serviceName);
    }

    public TimeServerImplService() {
        super(TIMESERVERIMPLSERVICE_WSDL_LOCATION,
            new QName("http://ts.examples/", "TimeServerImplService"));
    }

    @WebEndpoint(name = "TimeServerImplPort")
    public TimeServer getTimeServerImplPort() {
        return (TimeServer)super.getPort(new QName("http://ts.examples/", "TimeServerImplPort"), TimeServer.class);
    }
}
```

G DL

TimeServerImplService.java

```java
package client;

import java.net.MalformedURLException;
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import javax.xml.ws.WebEndpoint;
import javax.xml.ws.WebServiceClient;

@WebServiceClient(name = "TimeServerImplService",
targetNamespace = "http://ts.examples/",
wsdlLocation = "http://localhost:9876/ts?wsdl")
public class TimeServerImplService extends Service {
    private final static URL TIMESERVERIMPLSERVICE_WSDL_LOCATION;
    static {
        URL url = null;
        try {
            url = new URL("http://localhost:9876/ts?wsdl");
            }
        catch (MalformedURLException e) {
                e.printStackTrace();
                }
        TIMESERVERIMPLSERVICE_WSDL_LOCATION = url;
    }

    public TimeServerImplService(URL wsdlLocation, QName serviceName) {
        super(wsdlLocation, serviceName);
    }

    public TimeServerImplService() {
        super(TIMESERVERIMPLSERVICE_WSDL_LOCATION,
            new QName("http://ts.examples/", "TimeServerImplService"));
    }

    @WebEndpoint(name = "TimeServerImplPort")
    public TimeServer getTimeServerImplPort() {
        return (TimeServer)super.getPort(new QName("http://ts.examples/", "TimeServerImplPort"), TimeServer.class);
    }
}
```

Generating Client-Support Code from a WSDL

- Together the two generated types, the interface `client.TimeServer` and the class `client.TimeServerImplService`, ease the task of writing a Java client against the web service.

Generating Client-Support Code from a WSDL

- Together the two generated types, the interface `client.TimeServer` and the class `client.TimeServerImplService`, ease the task of writing a Java client against the web service.

Generating Client-Support Code from a WSDL

- Together tl
  `client.`
  `client.`
  Java client

TimeClientWSDL.java

```
1   package client;
2
3   class TimeClientWSDL {
4       ublic static void main(String[ ] args) {
5
6
7       // The TimeServerImplService class is the Java type bound to
8       // the service section of the WSDL document.
9       TimeServerImplService service = new TimeServerImplService();
10
11      // The TimeServer interface is the Java type bound to
12      // the portType section of the WSDL document.
13      TimeServer eif = service.getTimeServerImplPort();
14
15      // Invoke the methods.
16      System.out.println(eif.getTimeAsString());
17      System.out.println(eif.getTimeAsElapsed());
18
19      }
20  }
21
```

Generating Client-Support Code from a WSDL

- Troublesome details such as the appropriate `QName` and service endpoint now are hidden in the *wsimport*-generated class, `client.TempServerImplService`

Generating Client-Support Code from a WSDL

- Troublesome details such as the appropriate `QName` and service endpoint now are hidden in the *wsimport*-generated class, `client.TempServerImplService`
- **Steps for writing clients with the help from WSDL-based artifacts**
  - First, construct a `Service` object using one of two constructors in the *wsimport* generated class. The no-argument constructor is preferable because of its simplicity.
  - Invoke the `get...Port` method on the constructed `Service` object. The method returns an object that encapsulates the web service operations, declared in the original SEI.

The structure of a WSDL document

- At a high level, a WSDL document is a contract between a service and its consumers.
- The contract provides such critical information as the service endpoint, the service operations, and the data types required for these operations.
- The service contract also indicates, in describing the messages exchanged in the service, the underlying service pattern.
- The outermost element (called the *document* or *root* element) in a WSDL is named `definitions` because the WSDL provides definitions grouped five sections:

The structure of a WSDL document

1. The `types` section, which is optional, provides data type definitions under some data type system such as XML Schema. A particular document that defines data types is an *XSD* (XML Schema Definition). The `types` section holds, points to, or imports an XSD.
   - If the types section is empty, the service uses only simple data types such as `xsd:string` and `xsd:long`.
2. The `message` section defines the messages that implement the service. Messages are constructed from data types either defined in the `types` section or, if empty, available as defaults. Recall the use of the directional properties `in` and `out` for the message order, that defines the service pattern

The structure of a WSDL document

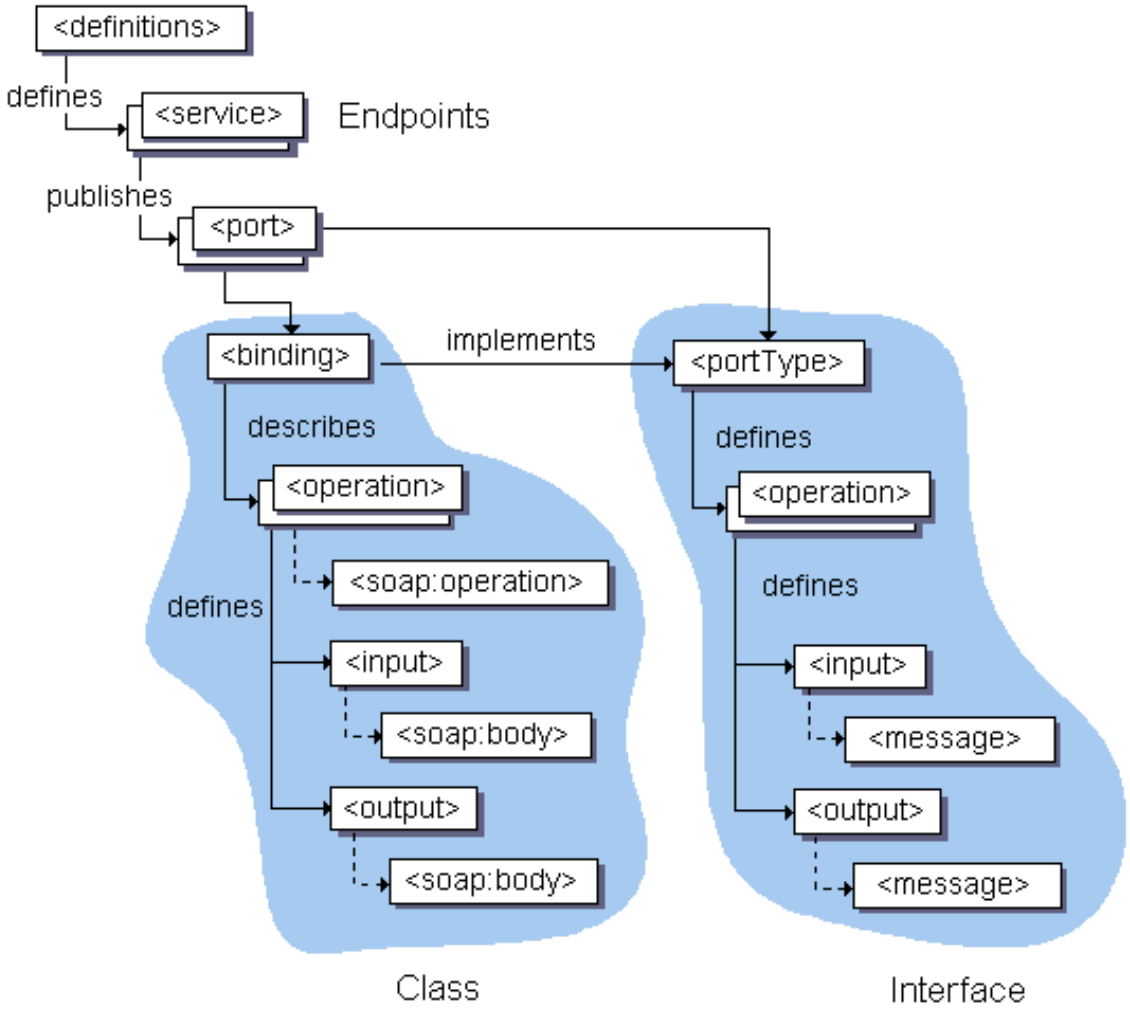
3. The `portType` section presents the service as named operations, with each operation as one or more messages.
   - Note that the operations are named after methods annotated as `@WebMethods`.
4. The `binding` section is where the WSDL definitions become concrete, it must specify implementation details
   - The transport protocol to be used in sending and receiving the underlying SOAP messages.
   - The `style` of the service takes either `rpc` or `document` as a value.
   - The data format to be used in the SOAP messages. There are two choices, `literal` and `encoded`.

The structure of a WSDL document

5. The `service` section specifies one or more endpoints at which the service's functionality is available.
   - The service section lists one or more `port` elements, where a `port` consists of a `portType` (interface) together with a corresponding `binding` (implementation).

```
− <service name="TimeServerImplService">
  − <port name="TimeServerImplPort" binding="tns:TimeServerImplPortBinding">
      <soap:address location="http://localhost:9876/ts"/>
    </port>
  </service>
```

The structure of a WSDL document



http://download.oracle.com/otn_hosted_doc/jdeveloper/1012/web_services/ws_wsdlstructure.html

A Closer Look at WSDL Bindings

- In the WSDL `binding` section, the `style` attribute has `rpc` and `document` as possible values, with `document` as the default.
- The `use` attribute has `literal` and `encoded` as possible values, with `literal` as the default.
- In theory, there are four possibilities

style	use
document	literal
document	encoded
rpc	literal
rpc	encoded

A Closer Look at WSDL Bindings

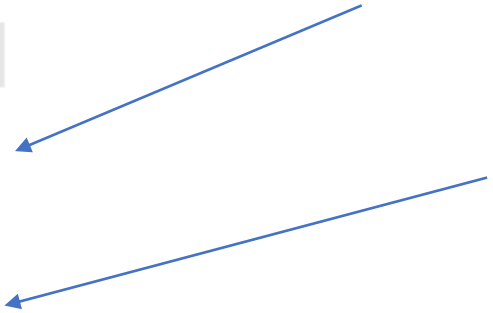
- In the WSDL `binding` section, the `style` attribute has `rpc` and `document` as possible values, with `document` as the default.
- The `use` attribute has `literal` and `encoded` as possible values, with `literal` as the default.
- In theory, there are four possibilities

style	use
document	literal
document	encoded
rpc	literal
rpc	encoded

A Closer Look at WSDL Bindings

- In the WSDL `binding` section, the `style` attribute has `rpc` and `document` as possible values, with `document` as the default.
- The `use` attribute has `literal` and `encoded` as possible values, with `literal` as the default.
- In theory, there are four possibilities

style	use
document	literal
document	encoded
rpc	literal
rpc	encoded



A Closer Look at WSDL Bindings

- The `document` style indicates that a SOAP-based web service's underlying messages contain full XML documents
- The `rpc` style indicates that the underlying SOAP messages contain parameters in the request messages and return values in the response messages
- The `TimeServer` WSDL in `rpc` style

```
<types></types>
<message name="getTimeAsString"></message>
<message name="getTimeAsStringResponse">
  <part name="time_response" type="xsd:string"></part>
</message>
<message name="getTimeAsElapsed"></message>
<message name="getTimeAsElapsedResponse">
  <part name="time_response" type="xsd:long"></part>
</message>
```

A Closer Look at WSDL Bindings

- The `TimeServer` **WSDL in** `document` **style**

```
<types>
  <xsd:schema>
    <xsd:import schemaLocation="http://localhost:9876/ts?xsd=1"
         namespace="http://ts.ch02/">
    </xsd:import>
  </xsd:schema>
</types>
<message name="getTimeAsString">
  <part element="tns:getTimeAsString" name="parameters"></part>
</message>
<message name="getTimeAsStringResponse">
  <part element="tns:getTimeAsStringResponse" name="parameters"></part>
</message>
<message name="getTimeAsElapsed">
  <part element="tns:getTimeAsElapsed" name="parameters"></part>
</message>
<message name="getTimeAsElapsedResponse">
  <part element="tns:getTimeAsElapsedResponse" name="parameters"></part>
</message>
```

A Closer Look at WSDL Bindings

- The `document` style deserves to be the default.
- This `style` can support services with rich, explicitly defined data types because the service's WSDL can define the required types in an XSD document.
- From an architectural perspective, the `document` style is the simpler of the two in that the body of a SOAP message is a self-contained, precisely defined document.
- The `rpc` style requires messages with the names of the associated operations (the `@WebMethods`) with parameters as sub-elements.

A Closer Look at WSDL Bindings

- The `document` style deserves to be the default.
- This `style` can support services with rich, explicitly defined data types because the service's WSDL can define the required types in an XSD document.
- From an architectural perspective, the `document` style is the simpler of the two in that the body of a SOAP message is a self-contained, precisely defined document.
- The `rpc` style requires messages with the names of the associated operations (the `@WebMethods`) with parameters as sub-elements.

Final Remarks

Limitations of the WSDL

- WSDL documents, as web service descriptions, should be publishable and discoverable. A *UDDI* (Universal Description Discovery and Integration) registry is one way to publish WSDLs so that potential clients can discover them and ultimately consume the services that the WSDLs describe.
- Once a WSDL has been located, critical questions remain about the service described in the WSDL.
  - The WSDL does not explain the service *semantics,* what the service is about. Figuring this out is left to the programmer.
  - The service providers usually give supplementary material such as documentation, tutorials, and sample code libraries

Limitations of the WSDL

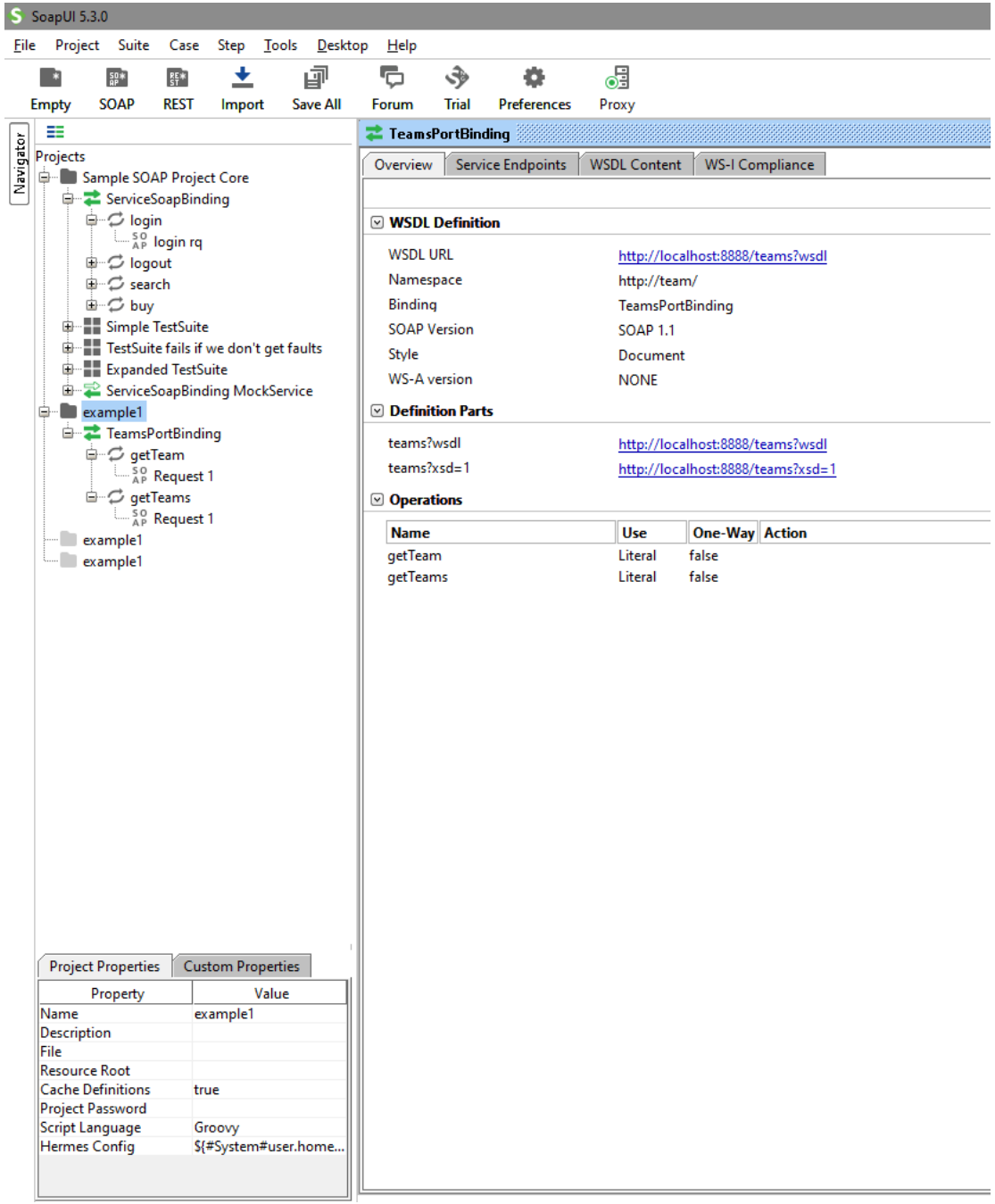
- The W3C is pursuing initiatives in web semantics under the rubric of WSDL-S (Semantics).
- As of now, a WSDL typically is useful only if the client programmer already understands what the service is about

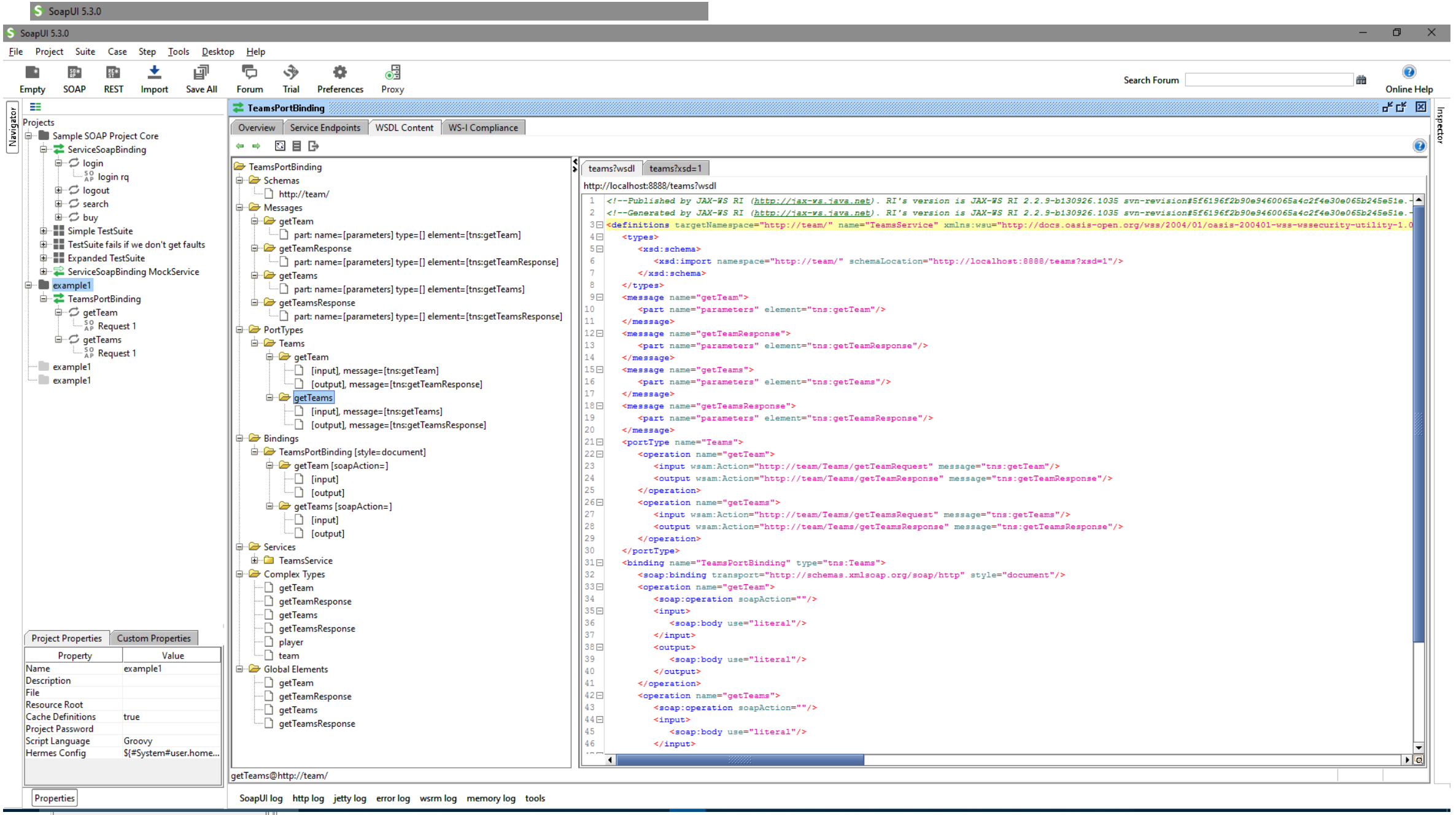
A tool to test: SOAP UI

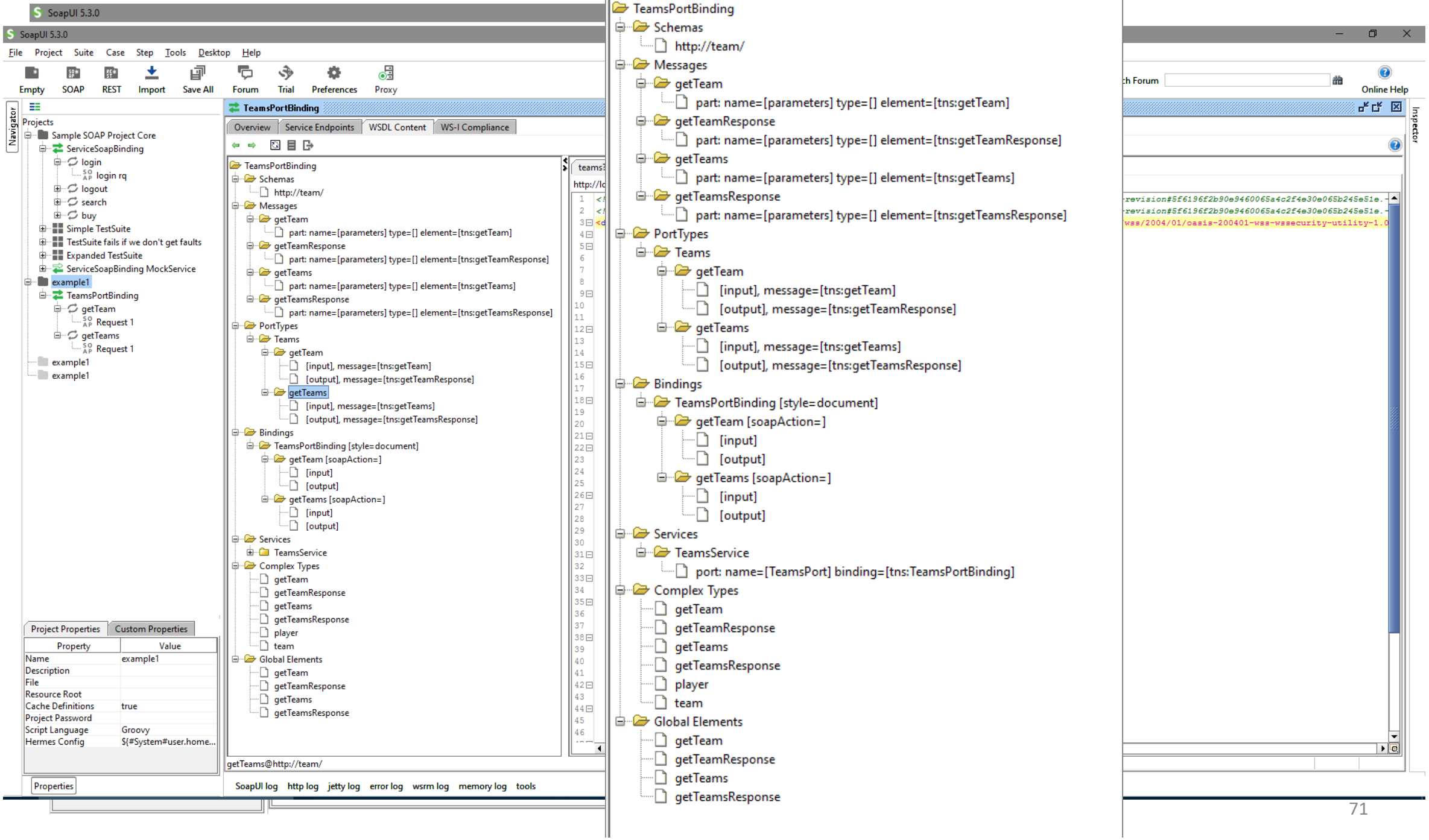
- SOAP UI is a graphical test tool for Web Services
  - www.soapui.org
  - Available as standalone or integrated in development environments (such as Eclipse)
  - Can be used on any development platform
- Main features
  - Supports Extended Web Services (WSDL + SOAP + UDDI) or REST
  - Inspect Web Services
  - Invoking Web Services
  - Develop Web Services
  - Simulate Web Services
  - Perform quality tests (response time, etc.)

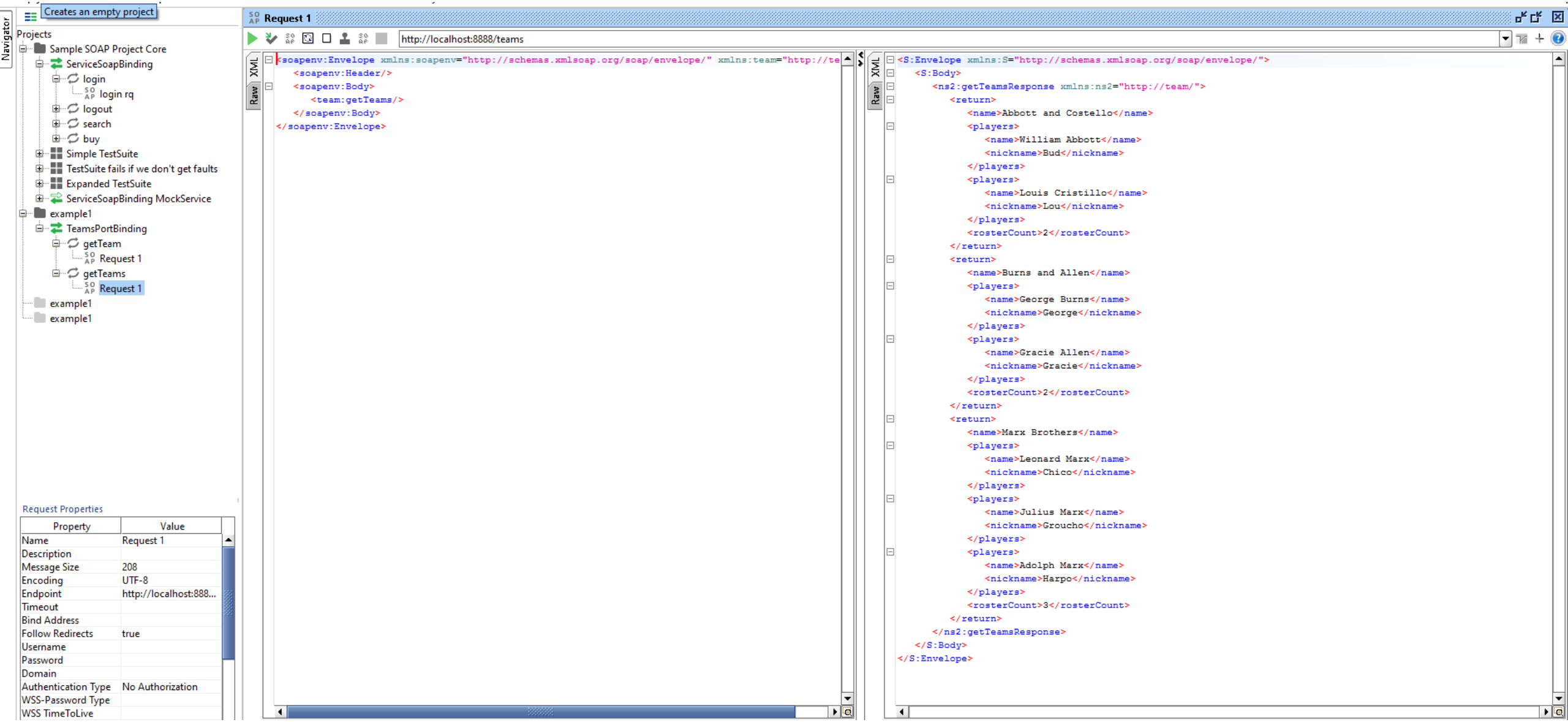
Inspecting an existing

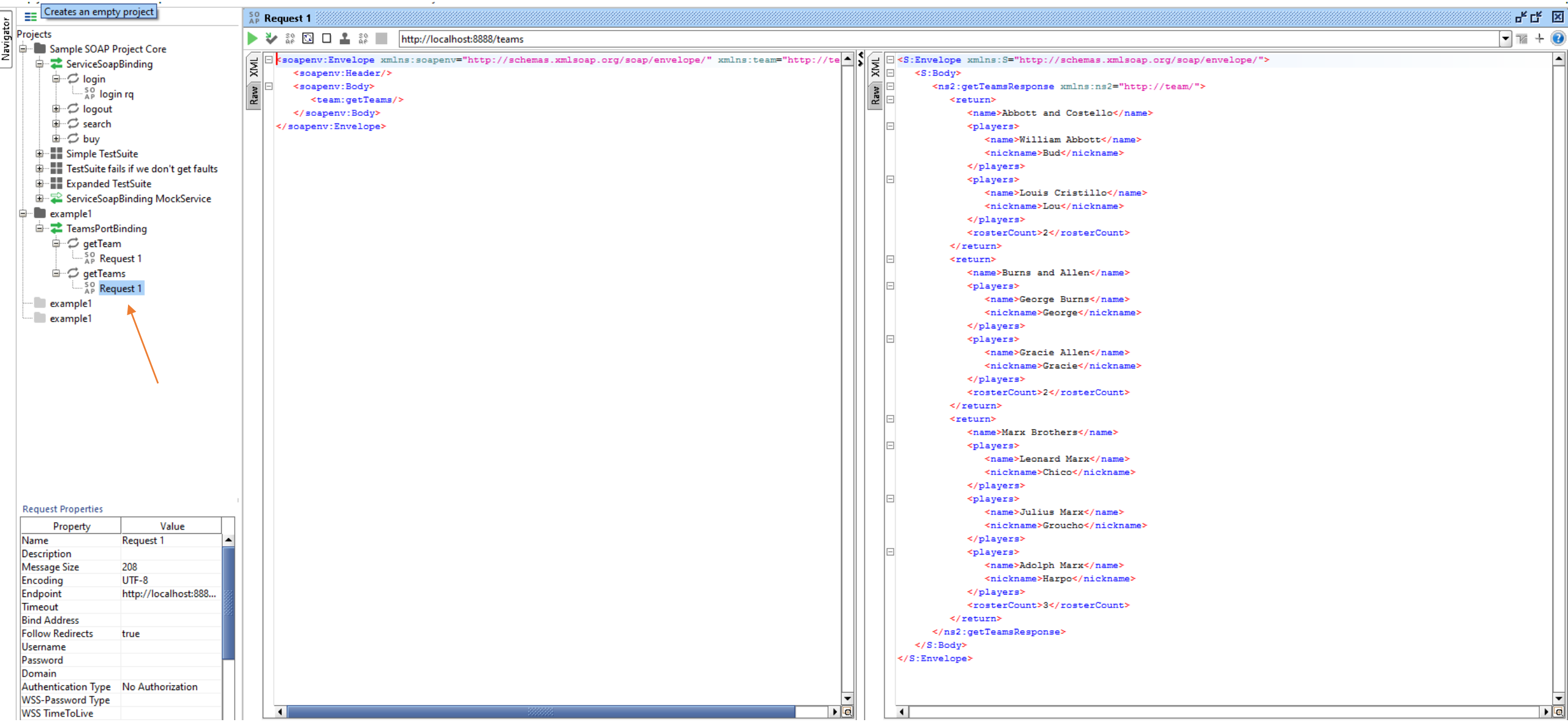
- We can create a new SoapUI project to test it.
- This is done by entering the URL of the wsdl file http://localhost:8888/teams?wsdl
- Once the project is created, the main window gives us several possibilities
  - The endpoints
  - The wsdl file
  - WS-I compliance tests


SoapUI 5.3.0
File Project Suite Case Step Tools Desktop Help
Empty SOAP REST Import Save All Forum Trial Preferences Proxy
Navigator
Projects
Sample SOAP Project Core
ServiceSoapBinding
login
login rq
logout
search
buy
Simple TestSuite
TestSuite fails if we don't get faults
Expanded TestSuite
ServiceSoapBinding MockService
example1
TeamsPortBinding
getTeam
Request 1
getTeams
Request 1
example1
example1
Project Properties
Custom Properties
Property
Value
Name
example1
Description
File
Resource Root
Cache Definitions
true
Project Password
Script Language
Groovy
Hermes Config
${#System#user.home...
TeamsPortBinding
Overview
Service Endpoints
WSDL Content
WS-I Compliance
WSDL Definition
WSDL URL
http://localhost:8888/teams?wsdl
Namespace
http://team/
Binding
TeamsPortBinding
SOAP Version
SOAP 1.1
Style
Document
WS-A version
NONE
Definition Parts
teams?wsdl
http://localhost:8888/teams?wsdl
teams?xsd=1
http://localhost:8888/teams?xsd=1
Operations
Name
Use
One-Way
Action
getTeam
Literal
false
getTeams
Literal
false


SoapUI 5.3.0
File Project Suite Case Step Tools Desktop Help
Empty SOAP REST Import Save All Forum Trial Preferences Proxy
Search Forum
Online Help
Navigator
Projects
Sample SOAP Project Core
ServiceSoapBinding
login
login rq
logout
search
buy
Simple TestSuite
TestSuite fails if we don't get faults
Expanded TestSuite
ServiceSoapBinding MockService
example1
TeamsPortBinding
getTeam
Request 1
getTeams
Request 1
example1
example1
Project Properties
Custom Properties
Property
Value
Name
example1
Description
File
Resource Root
Cache Definitions
true
Project Password
Script Language
Groovy
Hermes Config
${#System#user.home...
Properties
TeamsPortBinding
Overview
Service Endpoints
WSDL Content
WS-I Compliance
TeamsPortBinding
Schemas
http://team/
Messages
getTeam
part: name=[parameters] type=[] element=[tns:getTeam]
getTeamResponse
part: name=[parameters] type=[] element=[tns:getTeamResponse]
getTeams
part: name=[parameters] type=[] element=[tns:getTeams]
getTeamsResponse
part: name=[parameters] type=[] element=[tns:getTeamsResponse]
PortTypes
Teams
getTeam
[input], message=[tns:getTeam]
[output], message=[tns:getTeamResponse]
getTeams
[input], message=[tns:getTeams]
[output], message=[tns:getTeamsResponse]
Bindings
TeamsPortBinding [style=document]
getTeam [soapAction=]
[input]
[output]
getTeams [soapAction=]
[input]
[output]
Services
TeamsService
Complex Types
getTeam
getTeamResponse
getTeams
getTeamsResponse
player
team
Global Elements
getTeam
getTeamResponse
getTeams
getTeamsResponse
teams?wsdl
teams?xsd=1
http://localhost:8888/teams?wsdl
<!--Published by JAX-WS RI (http://jax-ws.java.net). RI's version is JAX-WS RI 2.2.9-b130926.1035 svn-revision#5f6196f2b90e9460065a4c2f4e30e065b245e51e.-
<!--Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is JAX-WS RI 2.2.9-b130926.1035 svn-revision#5f6196f2b90e9460065a4c2f4e30e065b245e51e.-
<definitions targetNamespace="http://team/" name="TeamsService" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0
<types>
<xsd:schema>
<xsd:import namespace="http://team/" schemaLocation="http://localhost:8888/teams?xsd=1"/>
</xsd:schema>
</types>
<message name="getTeam">
<part name="parameters" element="tns:getTeam"/>
</message>
<message name="getTeamResponse">
<part name="parameters" element="tns:getTeamResponse"/>
</message>
<message name="getTeams">
<part name="parameters" element="tns:getTeams"/>
</message>
<message name="getTeamsResponse">
<part name="parameters" element="tns:getTeamsResponse"/>
</message>
<portType name="Teams">
<operation name="getTeam">
<input wsam:Action="http://team/Teams/getTeamRequest" message="tns:getTeam"/>
<output wsam:Action="http://team/Teams/getTeamResponse" message="tns:getTeamResponse"/>
</operation>
<operation name="getTeams">
<input wsam:Action="http://team/Teams/getTeamsRequest" message="tns:getTeams"/>
<output wsam:Action="http://team/Teams/getTeamsResponse" message="tns:getTeamsResponse"/>
</operation>
</portType>
<binding name="TeamsPortBinding" type="tns:Teams">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
<operation name="getTeam">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="getTeams">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
getTeams@http://team/
SoapUI log
http log
jetty log
error log
wsrm log
memory log
tools
Inspector


SoapUI 5.3.0
File Project Suite Case Step Tools Desktop Help
Empty SOAP REST Import Save All Forum Trial Preferences Proxy
Online Help
Navigator
Inspector
Projects
Sample SOAP Project Core
ServiceSoapBinding
login
login rq
logout
search
buy
Simple TestSuite
TestSuite fails if we don't get faults
Expanded TestSuite
ServiceSoapBinding MockService
example1
TeamsPortBinding
getTeam
Request 1
getTeams
Request 1
example1
example1
TeamsPortBinding
Overview Service Endpoints WSDL Content WS-I Compliance
Schemas
http://team/
Messages
getTeam
part: name=[parameters] type=[] element=[tns:getTeam]
getTeamResponse
part: name=[parameters] type=[] element=[tns:getTeamResponse]
getTeams
part: name=[parameters] type=[] element=[tns:getTeams]
getTeamsResponse
part: name=[parameters] type=[] element=[tns:getTeamsResponse]
PortTypes
Teams
getTeam
[input], message=[tns:getTeam]
[output], message=[tns:getTeamResponse]
getTeams
[input], message=[tns:getTeams]
[output], message=[tns:getTeamsResponse]
Bindings
TeamsPortBinding [style=document]
getTeam [soapAction=]
[input]
[output]
getTeams [soapAction=]
[input]
[output]
Services
TeamsService
Complex Types
getTeam
getTeamResponse
getTeams
getTeamsResponse
player
team
Global Elements
getTeam
getTeamResponse
getTeams
getTeamsResponse
Project Properties
Custom Properties
Property
Value
Name
example1
Description
File
Resource Root
Cache Definitions
true
Project Password
Script Language
Groovy
Hermes Config
${#System#user.home...
Properties
getTeams@http://team/
SoapUI log
http log
jetty log
error log
wsrm log
memory log
tools
TeamsPortBinding
Schemas
http://team/
Messages
getTeam
part: name=[parameters] type=[] element=[tns:getTeam]
getTeamResponse
part: name=[parameters] type=[] element=[tns:getTeamResponse]
getTeams
part: name=[parameters] type=[] element=[tns:getTeams]
getTeamsResponse
part: name=[parameters] type=[] element=[tns:getTeamsResponse]
PortTypes
Teams
getTeam
[input], message=[tns:getTeam]
[output], message=[tns:getTeamResponse]
getTeams
[input], message=[tns:getTeams]
[output], message=[tns:getTeamsResponse]
Bindings
TeamsPortBinding [style=document]
getTeam [soapAction=]
[input]
[output]
getTeams [soapAction=]
[input]
[output]
Services
TeamsService
port: name=[TeamsPort] binding=[tns:TeamsPortBinding]
Complex Types
getTeam
getTeamResponse
getTeams
getTeamsResponse
player
team
Global Elements
getTeam
getTeamResponse
getTeams
getTeamsResponse


Navigator
Creates an empty project
Projects
Sample SOAP Project Core
ServiceSoapBinding
login
login rq
logout
search
buy
Simple TestSuite
TestSuite fails if we don't get faults
Expanded TestSuite
ServiceSoapBinding MockService
example1
TeamsPortBinding
getTeam
Request 1
getTeams
Request 1
example1
example1
Request Properties
Property
Value
Name
Request 1
Description
Message Size
208
Encoding
UTF-8
Endpoint
http://localhost:888...
Timeout
Bind Address
Follow Redirects
true
Username
Password
Domain
Authentication Type
No Authorization
WSS-Password Type
WSS TimeToLive
Request 1
http://localhost:8888/teams
XML
Raw
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:team="http://te
<soapenv:Header/>
<soapenv:Body>
<team:getTeams/>
</soapenv:Body>
</soapenv:Envelope>
XML
Raw
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body>
<ns2:getTeamsResponse xmlns:ns2="http://team/">
<return>
<name>Abbott and Costello</name>
<players>
<name>William Abbott</name>
<nickname>Bud</nickname>
</players>
<players>
<name>Louis Cristillo</name>
<nickname>Lou</nickname>
</players>
<rosterCount>2</rosterCount>
</return>
<return>
<name>Burns and Allen</name>
<players>
<name>George Burns</name>
<nickname>George</nickname>
</players>
<players>
<name>Gracie Allen</name>
<nickname>Gracie</nickname>
</players>
<rosterCount>2</rosterCount>
</return>
<return>
<name>Marx Brothers</name>
<players>
<name>Leonard Marx</name>
<nickname>Chico</nickname>
</players>
<players>
<name>Julius Marx</name>
<nickname>Groucho</nickname>
</players>
<players>
<name>Adolph Marx</name>
<nickname>Harpo</nickname>
</players>
<rosterCount>3</rosterCount>
</return>
</ns2:getTeamsResponse>
</S:Body>
</S:Envelope>


Navigator
Creates an empty project
Projects
Sample SOAP Project Core
ServiceSoapBinding
login
login rq
logout
search
buy
Simple TestSuite
TestSuite fails if we don't get faults
Expanded TestSuite
ServiceSoapBinding MockService
example1
TeamsPortBinding
getTeam
Request 1
getTeams
Request 1
example1
example1
Request Properties
Property
Value
Name
Request 1
Description
Message Size
208
Encoding
UTF-8
Endpoint
http://localhost:888...
Timeout
Bind Address
Follow Redirects
true
Username
Password
Domain
Authentication Type
No Authorization
WSS-Password Type
WSS TimeToLive
Request 1
http://localhost:8888/teams
XML
Raw
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:team="http://te
<soapenv:Header/>
<soapenv:Body>
<team:getTeams/>
</soapenv:Body>
</soapenv:Envelope>
XML
Raw
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body>
<ns2:getTeamsResponse xmlns:ns2="http://team/">
<return>
<name>Abbott and Costello</name>
<players>
<name>William Abbott</name>
<nickname>Bud</nickname>
</players>
<players>
<name>Louis Cristillo</name>
<nickname>Lou</nickname>
</players>
<rosterCount>2</rosterCount>
</return>
<return>
<name>Burns and Allen</name>
<players>
<name>George Burns</name>
<nickname>George</nickname>
</players>
<players>
<name>Gracie Allen</name>
<nickname>Gracie</nickname>
</players>
<rosterCount>2</rosterCount>
</return>
<return>
<name>Marx Brothers</name>
<players>
<name>Leonard Marx</name>
<nickname>Chico</nickname>
</players>
<players>
<name>Julius Marx</name>
<nickname>Groucho</nickname>
</players>
<players>
<name>Adolph Marx</name>
<nickname>Harpo</nickname>
</players>
<rosterCount>3</rosterCount>
</return>
</ns2:getTeamsResponse>
</S:Body>
</S:Envelope>

Request 1

http://localhost:8888/teams

XML Raw

```
POST http://localhost:8888/teams HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: text/xml;charset=UTF-8
SOAPAction: ""
Content-Length: 208
Host: localhost:8888
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
```

XML Raw

```
HTTP/1.1 200 OK
Date: Mon, 27 Feb 2017 12:02:02 GMT
Transfer-encoding: chunked
Content-type: text/xml; charset=utf-8

<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><ns2:getTeamsResponse xmlns:ns2="http://tea
```

SSL Info WSS (0)

A word to finish

- The big difficulty of SOAP is not to implement the code, but to choose one of the multiple ways to do it.
- There are several SOAP implementations in Java since Java 6
- Oracle has its own with JAX-WS, which we have just explored
- Axis (http://ws.apache.org/axis2/) in the Apache world; and its new version 2 which does not strictly impose the data model
- CXF (http://cxf.apache.org), an alternative to Axis2 on Apache