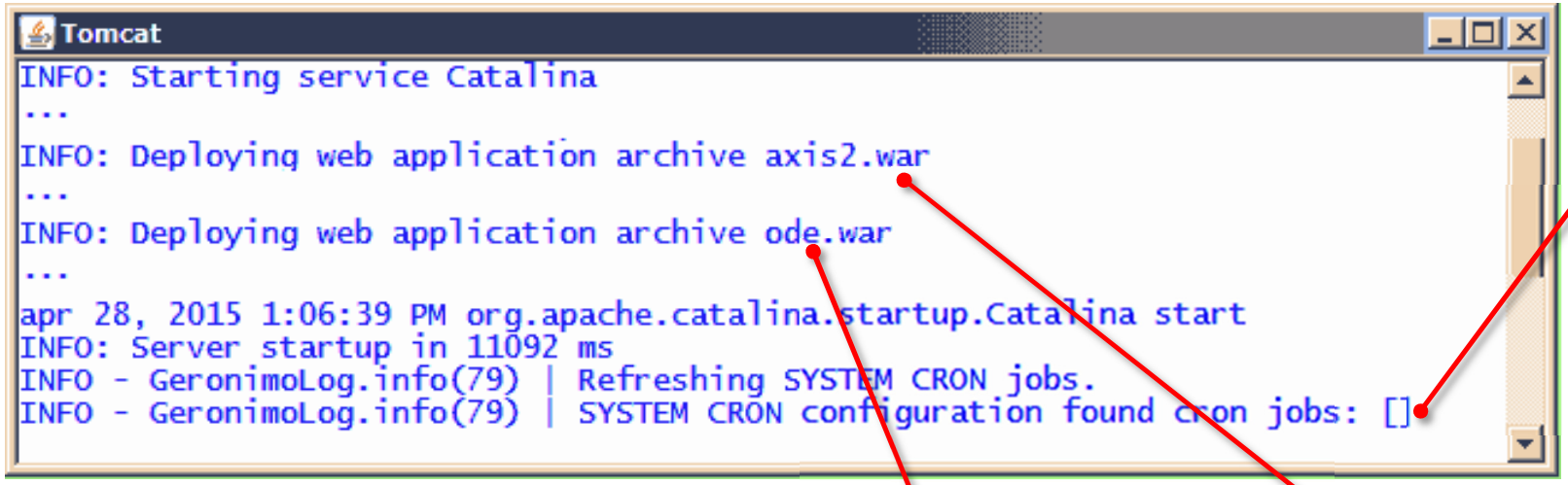
SERVER-SIDE WEB SERVICES

Mario G.C.A. Cimino

Department of Information Engineering

Apache Axis and Apache Ode

- ✓ Download the *all-in-one* package and extract it to C:
  http://www.iet.unipi.it/m.cimino/sse/res/sse_extract_to_C.zip
  - Apache6 Tomcat is a Java servlet[1] container. To start/stop the web server use *c:\sse\hutdownApache* and *c:\sse\startupApache*



- ✓ The package contains the servlets *Apache Ode* and *Apache Axis2*, deployed as a WAR[2]
- ✓ After startup, point the browser to *http://localhost:8080* to test the web server
- ✓ **Apache Axis2** is a Web Services / SOAP / WSDL engine
  http://localhost:8080/axis2/ as a first test.
  Click on *Services* to see the available ones.

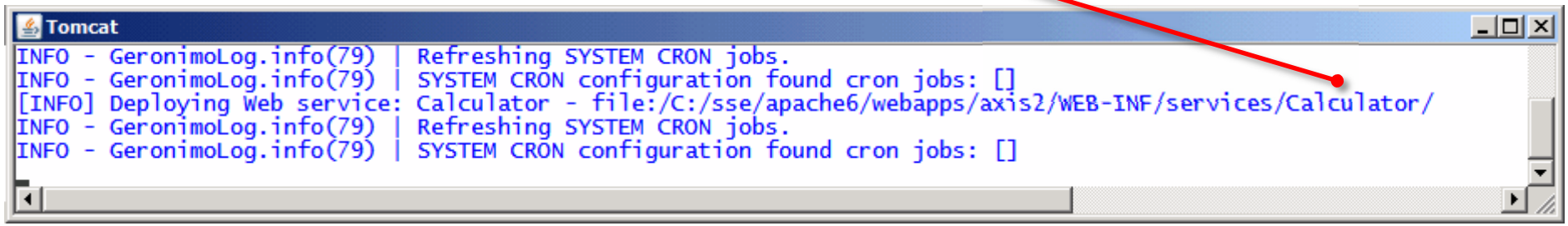
[1] Java server side components to extend the capabilities of an application server.
[2] A WAR file (or Web application ARchive) is a JAR file used to distribute Java-based web applications. In general, to deploy a war, put it into the webapps folder of tomcat.

✓ Open the c:\myapp\Calculator\Calculator.java file

```
public class Calculator {
    public int add(int i1, int i2) {
        return i1 + i2;
    }

    public b subtract(int i1, int i2) {
        return i1 - i2;
    }
}
```

✓ Click on *make.bat* to compile it. Copy the *c:\myapp\Calculator* folder to *C:\sse\apache6\webapps\axis2\WEB-INF\services*

The Tomcat console will show some deployment info



✓ Test the Calculator service on http://localhost:8080/axis2/ → click on *Services*.

✓ Click on the *Calculator* link:

http://localhost:8080/axis2/services/Calculator?wsdl

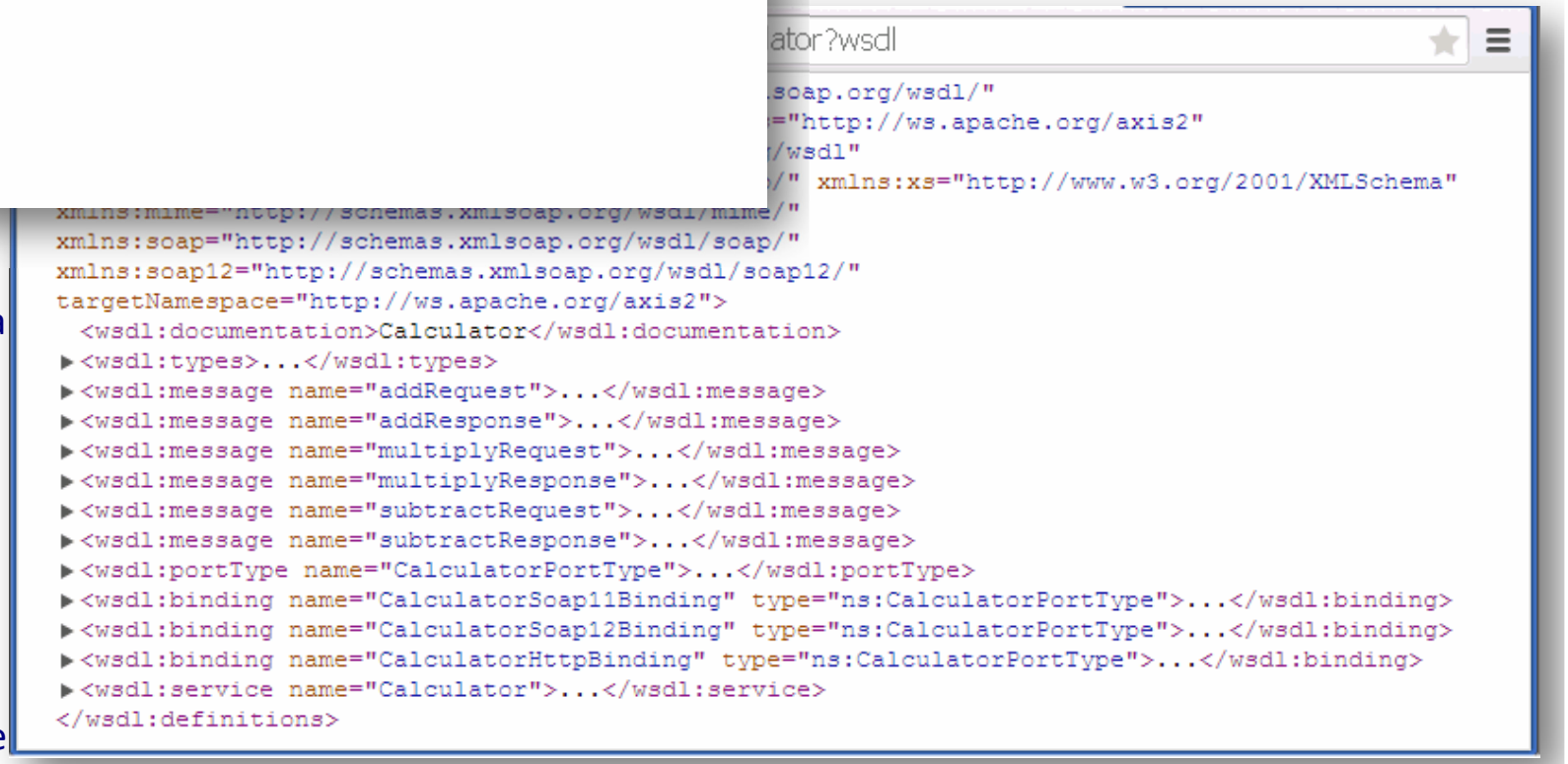
You will get the WSDL description of the Calculator service.

wsdl:definition : root element, with the target namespace and some other fundamental attributes

wsdl:types: data types used by operations, as input, output, and fault types. Often data types are specified using **XML schema**

wsdl:message: defines the messages to exchange and the related types (wsdl:types)

wsdl:portType (interface) – defines operations with input and output which refer to the messages (wsdl:message)

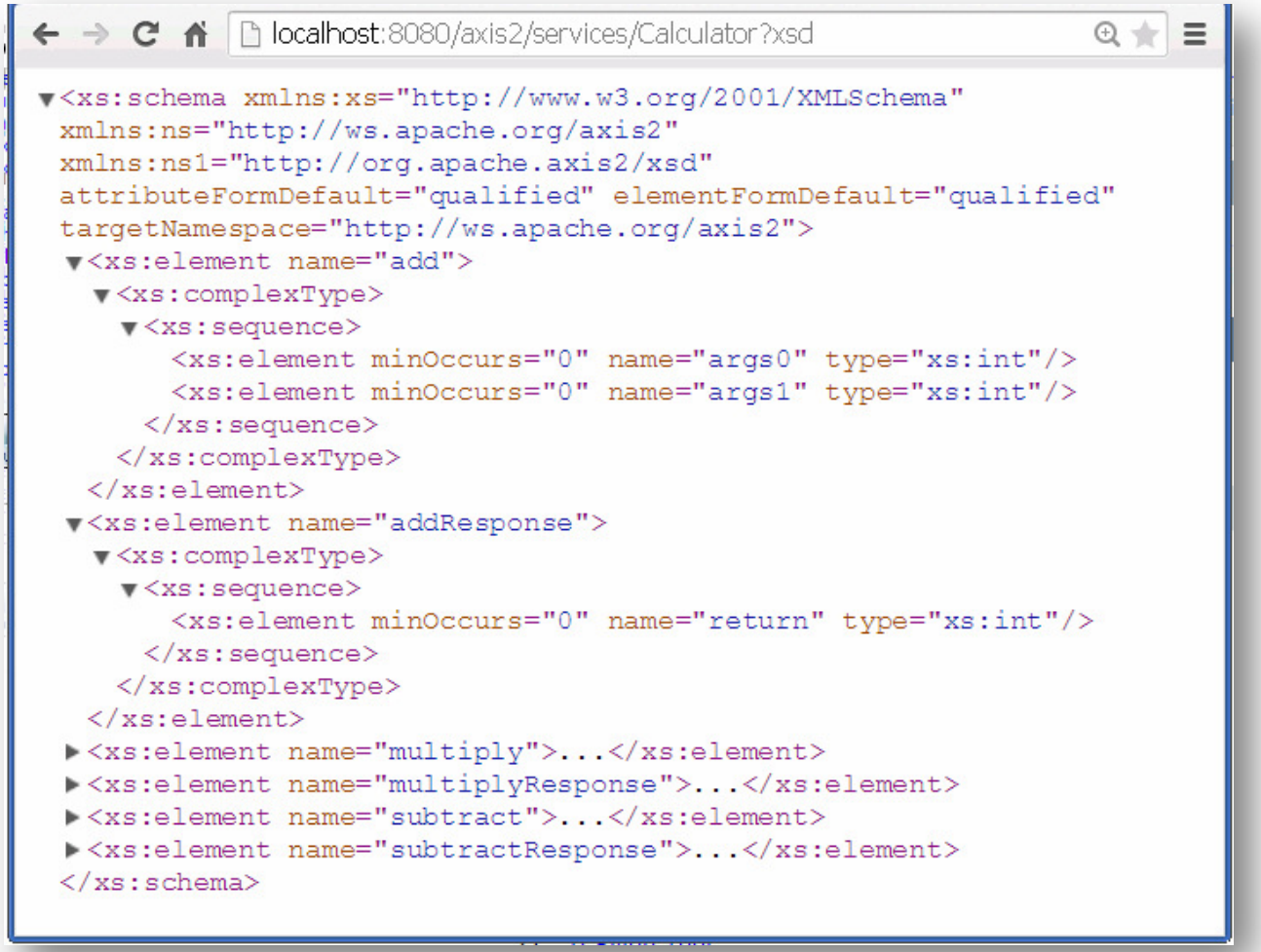


wsdl:binding: network protocols and soap data format specification (document-centric or rpc)

wsdl:service: defines the endpoints (addresses) associated to the binding (wsdl:binding), with one or more logical port name

https://www.w3schools.com/xml/xml_wsdl.asp

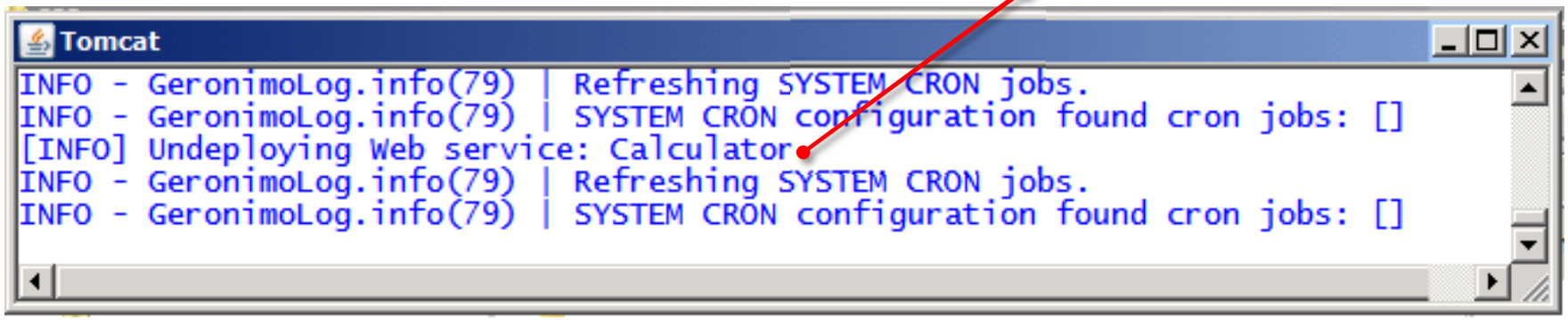
✓ Point the browser to *http://localhost:8080/axis2/services/Calculator?xsd* for data types



✓ Create a sample request by using SoapUI, as in the *Client-side WS* tutorial.

✓ Use it with sendsoap, as in the *Client-side WS* tutorial.

- ✓ Remove the *C:\sse\apache6\webapps\axis2\WEB-INF\services\Calculator* folder
- ✓ The Tomcat console will show some undeployment info



- ✓ Add a *divide* method to the *c:\myapp\Calculator\Calculator.java* file. Create and test the new web service.
- ✓ Create a web service with a new name, *DistilledWater*:

  (i) make a copy of the folder *c:\myapp\Calculator*

  (ii) rename the folder *Calculator* as *DistilledWater*

  (iii) in the file *META-INF\services.xml*, replace the two occurrences of the term *Calculator* with *DistilledWater*

  (iv) rename the file *Calculator.java* as *DistilledWater.java*

  (v) update the internal java implementation of *DistilledWater.java*

  (vi) Compile it by clicking on *make.bat*

  (vii) Move the *c:\myapp\DistilledWater* folder to to *C:\sse\apache6\webapps\axis2\WEB-INF\services*

✓ Emulation of DistilledWater

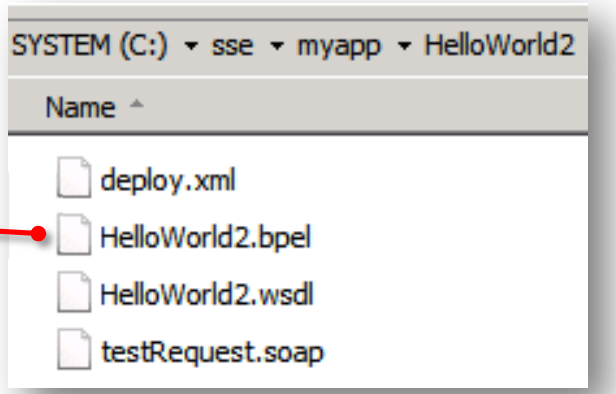
```
public class DistilledWater {

  public String order(int quantity) {
    double r = Math.random();
    if (r <= 0.05) return "service unavailable"; //(1)

    r = Math.random();
    if (r >= 0.3) return "order confirmed"; //(2)
    if (r >= 0.1 && r < 0.3) return "order confirmed but amount reduced"; //(3)
    else return "finished product"; //(4)
  }
}

/*
Footnotes
(1) 95% service availability
(2) 70% of cases: sufficient stock
(3) 20% of cases: insufficient stock
(4) 10% of cases: out of stock
*/
```

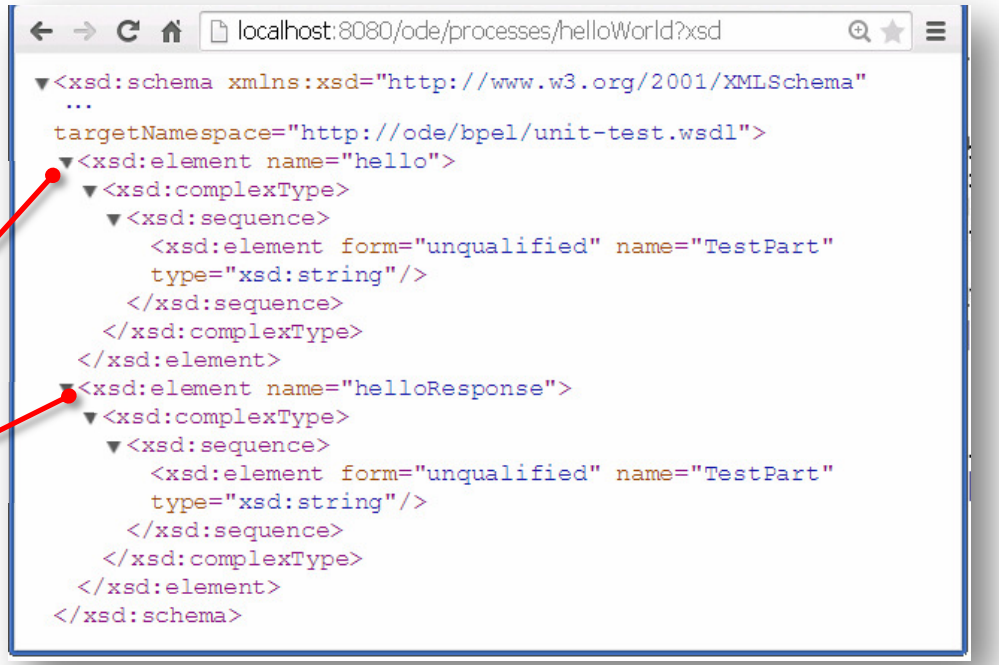
✓ In a real implementation, the result is a complex parameter, including a message and the amount ordered, defined via an **XSD**, validated/processed using the **DOM API**.

✓ Use emulation to test the orchestration logic: (i) emulate instances of complex I/O parameters with a mnemonic string for each case; (ii) the orchestrator can distinguish the resulting case by checking the substring if *resp.contains("…")*.

- ✓ **Apache Ode** (Orchestration Director Engine) is a BPEL[3] engine.

  Point the browser to *http://localhost:8080/ode/* as a first test.
  Click on *Processes* to see the available ones.
- ✓ Open the folder *c:\myapp\HelloWorld2*
- ✓ In the Apache ODE, business processes are written with BPEL standard.
- ✓ Copy the *c:\myapp\HelloWorld2* folder to *C:\sse\apache6\webapps\ode\WEB-INF\processes*
  The Tomcat console will show some deployment info
- ✓ Test the HelloWorld service on
  http://localhost:8080/ode/processes/helloWorld?wsdl
  and
  http://localhost:8080/ode/processes/helloWorld?xsd
- ✓ The service takes an input string (hello) and returns an output string (helloResponse)





[3] BPEL or WS-BPEL (Business Process Execution Language) is an XML standard to **orchestrate** web services.

Open the .bpel file

- ✓ BPEL has three basic components:
  - Input/Output (WSDL)
  - Data types (XSD)
  - Programming logic (BPEL)
- ✓ Import the WSDL od XSD
- ✓ A PartnerLink is a web service portType
- ✓ Variables declaration (programming)
- ✓ Message Exchange: receive
- ✓ Variable assignment (programming)
- ✓ XPath Expression (programming)

http://www.w3schools.com/xpath/xpath_functions.asp

- ✓ Message exchange: reply

http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html#_Toc164738506

```
<process name="HelloWorld2"
    targetNamespace="http://ode/bpel/unit-test"
    xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
    ... >
  <import location="HelloWorld2.wsdl"
     namespace="http://ode/bpel/unit-test.wsdl"
     importType="http://schemas.xmlsoap.org/wsdl/" />
  <partnerLinks>
     <partnerLink name="helloPartnerLink"
        partnerLinkType="test:HelloPartnerLinkType"
        myRole="me" />
  </partnerLinks>
  <variables>
    <variable name="myVar" messageType="test:HelloMessage"/>
    <variable name="tmpVar" type="xsd:string"/>
  </variables>
  ...
```

```
  ...
  <sequence>
     <receive
        name="start"
        partnerLink="helloPartnerLink"
        portType="test:HelloPortType"
        operation="hello"
        variable="myVar"
        createInstance="yes"/>
     <assign name="assign1">
        <copy>
           <from variable="myVar" part="TestPart"/>
           <to variable="tmpVar"/>
        </copy>
        <copy>
           <from>concat($tmpVar,' World')</from>
           <to variable="myVar" part="TestPart"/>
        </copy>
     </assign>
     <reply name="end"
        partnerLink="helloPartnerLink"
        portType="test:HelloPortType"
        operation="hello"
        variable="myVar"/>
  </sequence>
</process>
```

✓ The service concatenates the string ' World' to the input message. Modify the service:

(i) Remove the *C:\sse\apache6\webapps\ode\WEB-INF\processes\HelloWorld2* folder. The Tomcat console will show some undeployment info

(ii) Replace the string 'World' with the string 'Universe' in *c:\myapp\HelloWorld2\ HelloWorld2.bpel*

(iii) Copy the *c:\myapp\HelloWorld2 folder* to *C:\sse\apache6\webapps\ode\WEB-INF\processes*
The Tomcat console will show some deployment info

✓ Create a sample request by using SoapUI.