The Abdus Salam
International Centre for Theoretical Physics

United Nations
Educational, Scientific and
Cultural Organization

IAEA
International Atomic Energy Agency

2273-9

Second Workshop on Open Source and the Internet for Building Global Scientific Communities with Emphasis on Environmental Monitoring and Distributed Instrumentation

28 November - 16 December, 2011

Web services

C. Kavka
ESTECO Srl. Trieste
Italy

Web services

Carlos Kavka
Esteco SRL
Trieste, Italy

Second Workshop on Open Source and the Internet for Building Global Scientific Communities
The Abdus Salam ICTP, Trieste, Italy
28 November - 16 December 2011

www.esteco.com

Presentation outline

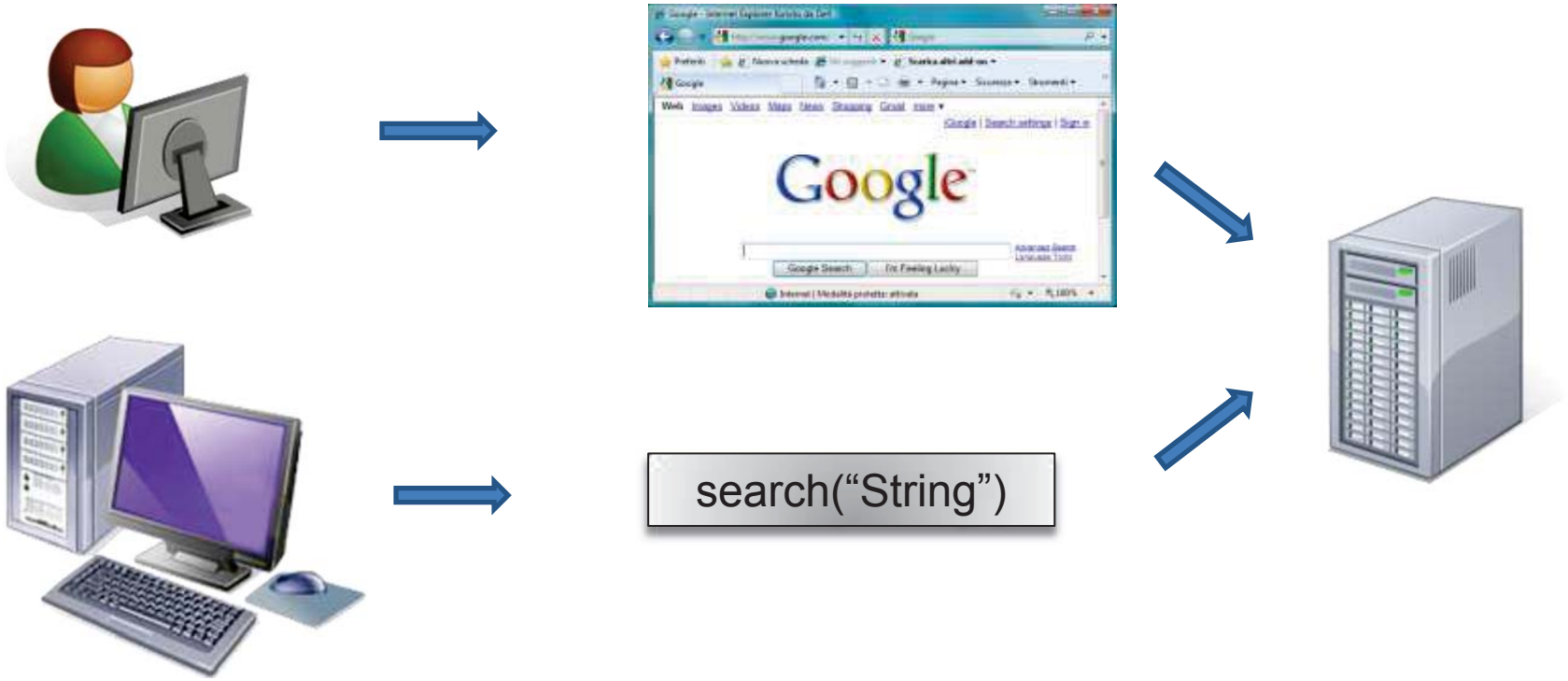
- Introduction to web services
- Basis of the XML language
- Web services architecture
- Web services specification
- Examples in PHP
- Advanced topics
- References

Web services

- A program-to-program communication model
- Based on open protocols
- Language independent: the consumer and the provider can use different languages
- XML language is used for the interaction
- Self-contained and self-describing
- Basic component of the Service Oriented Architecture

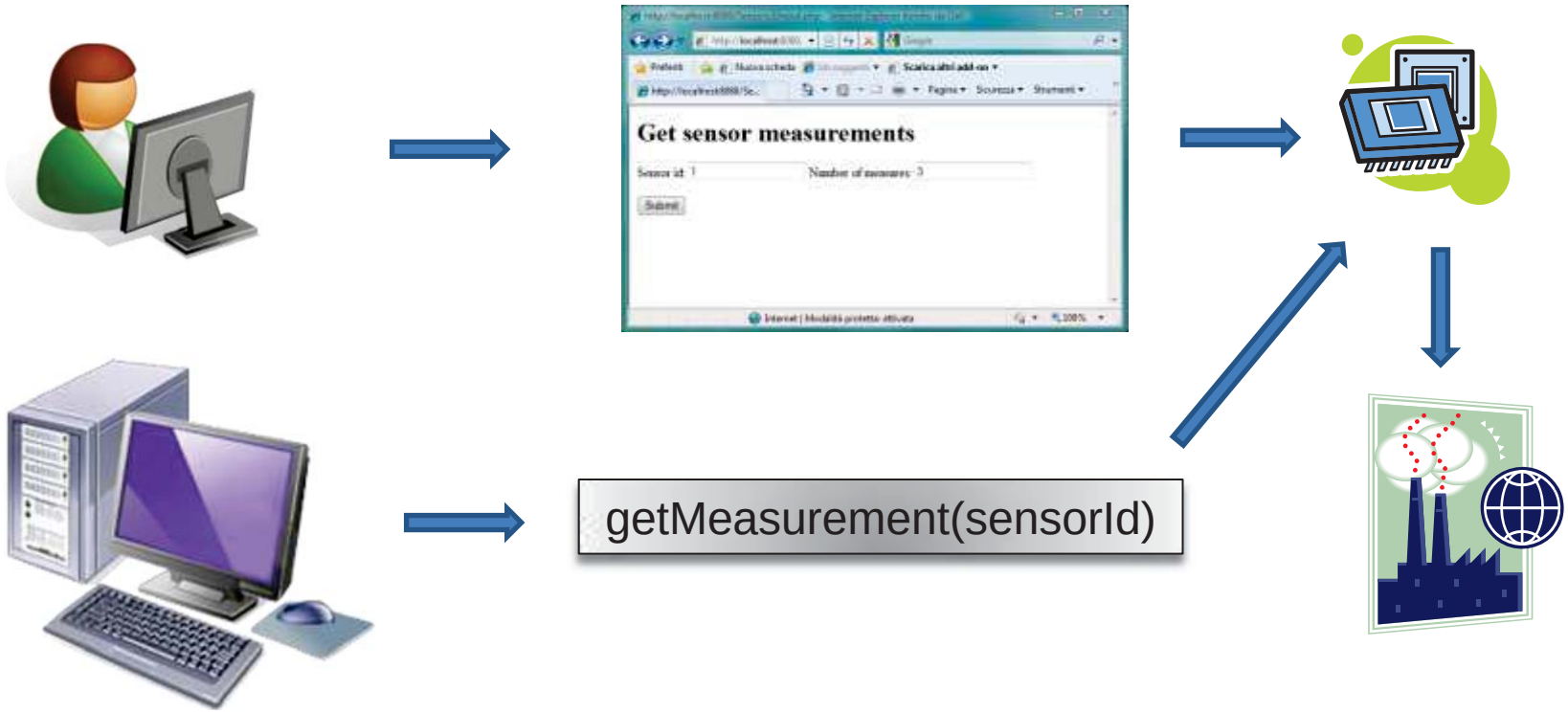
Web services – an example

- Google™ offers a web interface for humans and a web services based interfaces for programs.



Web services – our objective

- Provide not only a web-based interface to interact with devices, but also a program-to-program interface

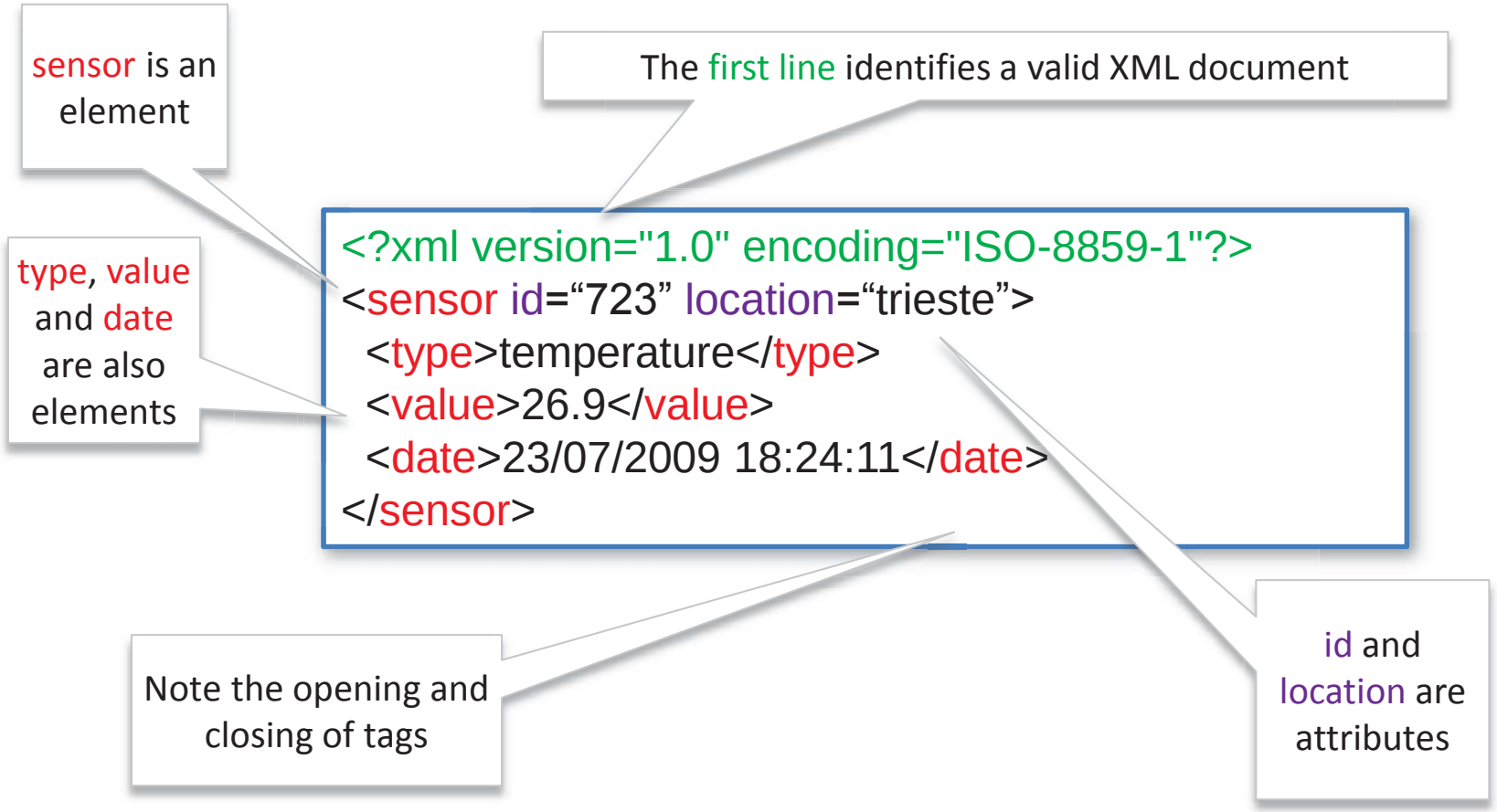


XML – eXtensible Markup Language

- Standard language designed to structure, transport and store data
- It is a text based markup language (like HTML)
- Centered about the concept of XML elements:
  - Represents a single piece of information
  - Can be nested
  - Can specify attributes

XML – eXtensible Markup Language

- Example: codification of a sensor measurement



XML – eXtensible Markup Language

- Tags are case sensitive:

<Sensor>…</Sensor> is different from <sensor>…</sensor>

- Comments can be used in the document:

<!–- This is a comment -->

- Some character needs to be replaced:

< ⇨ <	> ⇨ >	& ⇨ &	‘⇨ '	“⇨ "

XML – eXtensible Markup Language

- Conflicts can arise when different pieces of information are added together

codifies a measurement

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<sensor id="723" location="trieste">
  <type>temperature</type>
  <value>26.9</value>
  <date>23/07/2009 18:24:11</date>
</sensor>
```

codifies a sensor description

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<sensor manufacturer="Dallas">
  <code>DS-2431</code>
  <type>temperature sensor</type>
  <price>1.21</price>
</sensor>
```

Same tag names!

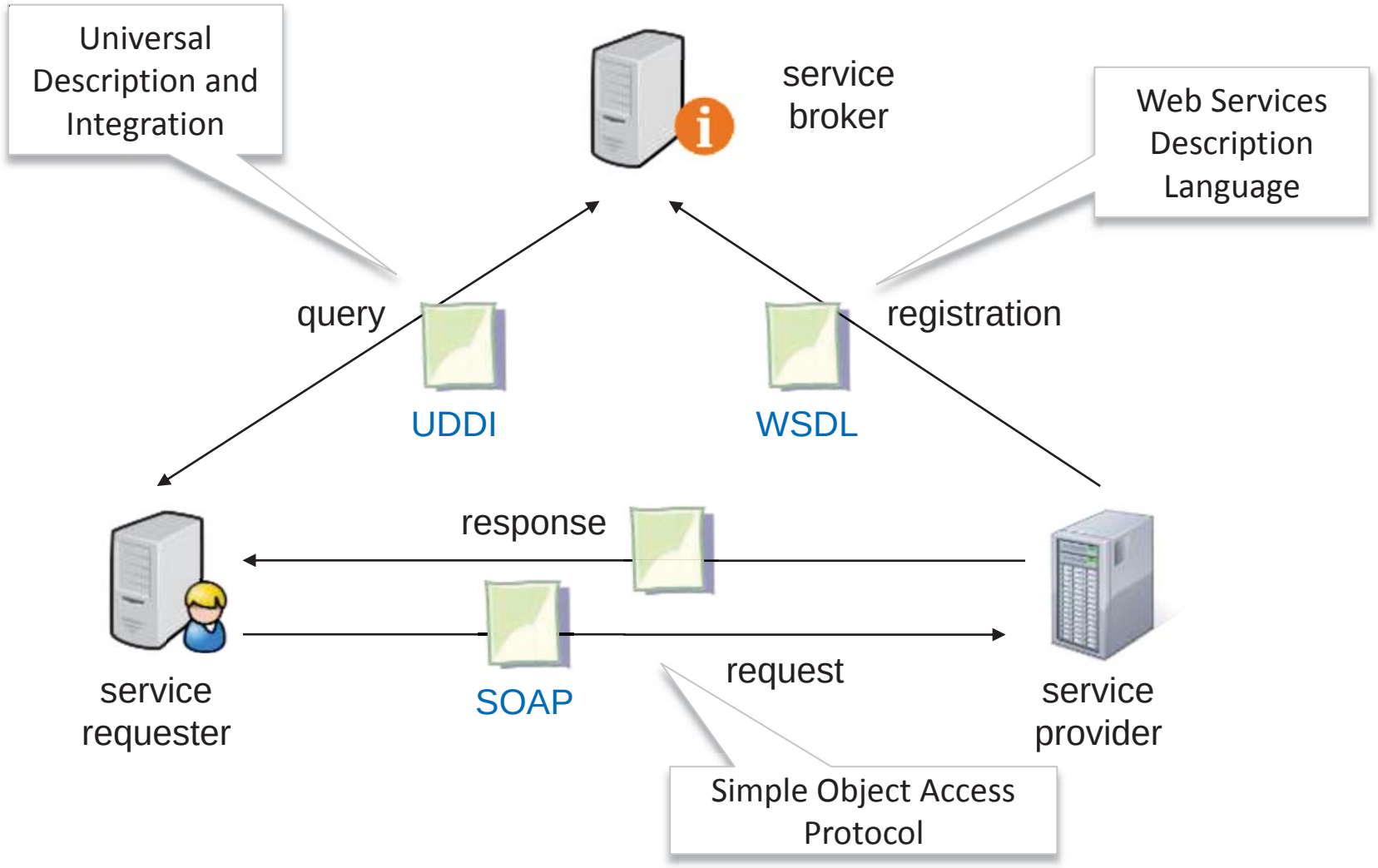
XML — eXtensible Markup Language

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<info xmlns:data="http://ictp.it/datasensors"
        xmlns:lib="http://ictp.it/devicelibrary">

 <data:sensor id="723" location="trieste">
  <data:type>temperature</lib:type>
  <data:value>26.9</lib:value>
  <data:date>23/07/2009 18:24:11</lib:date>
 </data:sensor>

 <lib:sensor manufacturer="Dallas">
  <lib:code>DS-2431</lib:code>
  <lib:type>temperature sensor</lib:type>
  <lib:price>1.21</lib:price>
 </lib:sensor>
</info>
```

Two namespaces: xmlns:data and xmlns:lib solve the conflict

both elements are qualified now

Web services architecture



WSDL – Web Service Definition Language

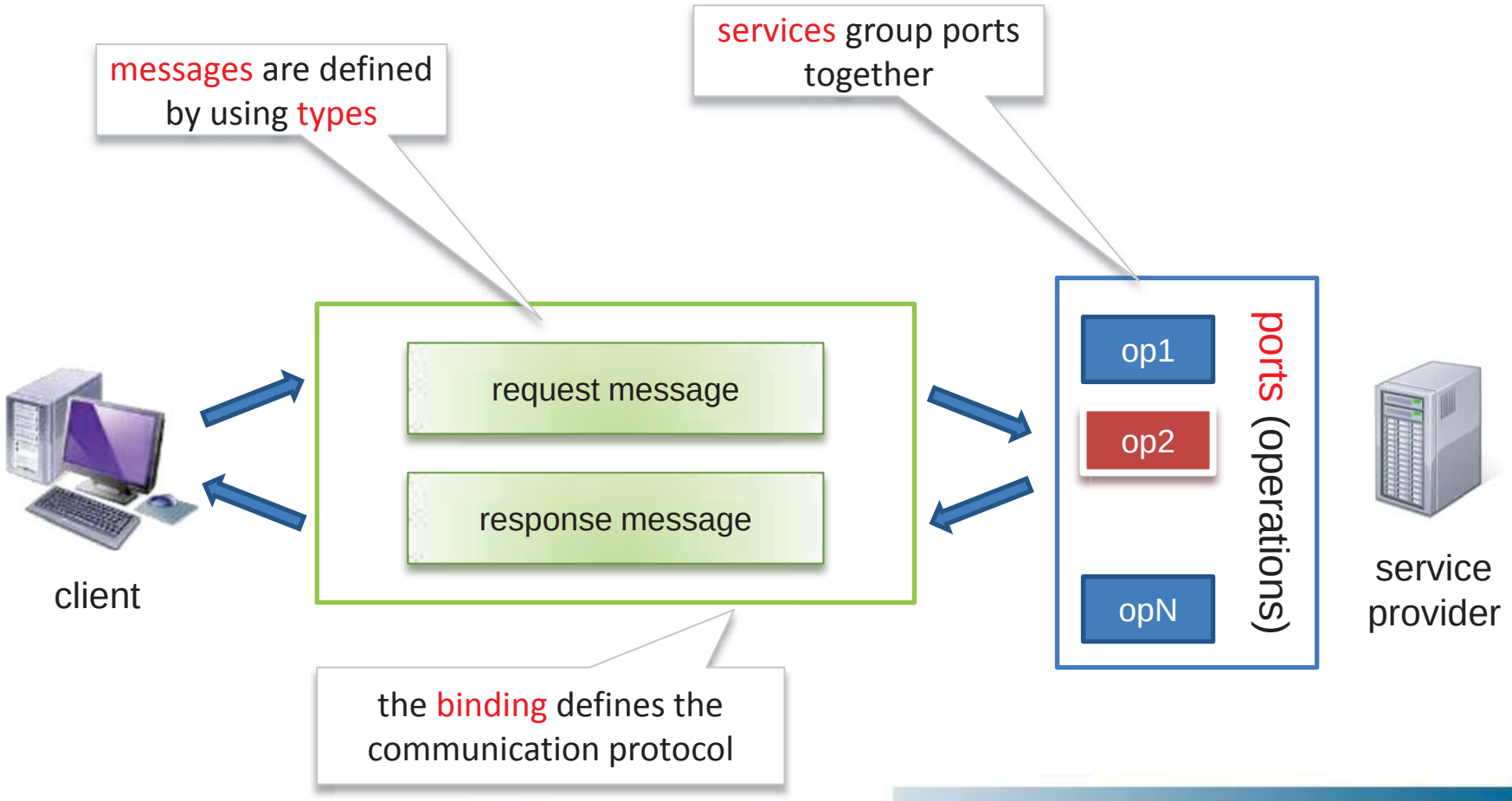
- WSDL is used to fully describe a web service

- A WS description in WSDL is an XML file that describes:
  - types (simple or complex),
  - messages (used to exchange information),
  - ports (define the messages associated to the operations)
  - bindings (specifies concrete protocol and data format specifications for the operations)
  - services (group a set of ports together)

- Next page shows an example!

WSDL – Web Service Definition Language

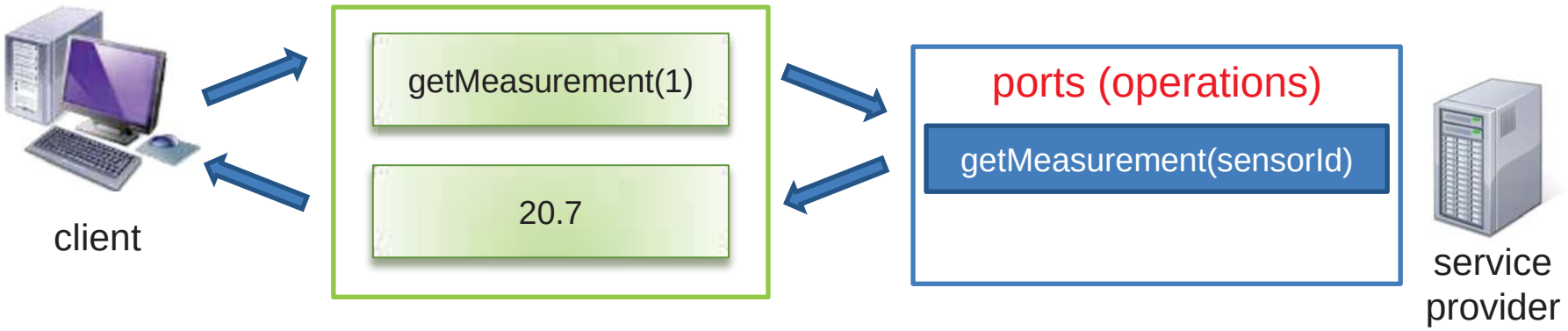
- Main components defined by a WSDL specification



WSDL – an example

- Since concepts are difficult to understand, the elements in the WSDL description file will be introduced with an example

- In this example, the client requests the last measurement of a specific sensor by passing its ID:



WSDL - file skeleton for sensor-services-1

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.ictp.it/sensor-services-1/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    name="sensor-services-1"
  targetNamespace="http://ictp.it/sensor-services-1/">
    TYPES
    MESSAGES
    OPERATIONS AND PORT TYPES
    BINDINGS AND SERVICES
</wsdl:definitions>
```

standard definitions

defines the namespace for the service

Example – type section

- Encloses data type definitions that are relevant for the exchanged messages.

```
<wsdl:types>
  <xsd:schema targetNamespace="http://ictp.it/sensor-services-1/">

    <xsd:element name="getMeasurement" type="xsd:int">
    </xsd:element>

    <xsd:element name="getMeasurementResponse" type="xsd:float">
    </xsd:element>

  </xsd:schema>
</wsdl:types>
```

int

float

define the types **int** and **float** to be used in the messages

Example – messages section

- Describes the logical abstract content of a message.
- It is structured in parts

```
<wsdl:message name="getMeasurementRequest">
  <wsdl:part element="tns:getMeasurement" name="parameters"/>
</wsdl:message>

<wsdl:message name="getMeasurementResponse">
  <wsdl:part element="tns:getMeasurementResponse" name="parameters"/>
</wsdl:message>
```

Request message: int

Response message: float

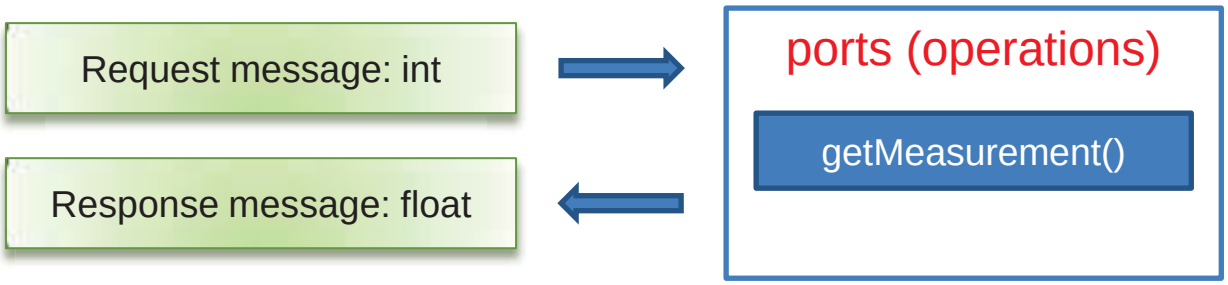
define the messages to be used for communcation

Example – port types/operations section

- **Port types** define a set of abstract operations. Each **operation** refers to an input message and output messages.

```
<wsdl:portType name="sensor-services-1">
   <wsdl:operation name="getMeasurement">
     <wsdl:input message="tns:getMeasurementRequest"/>
     <wsdl:output message="tns:getMeasurementResponse"/>
   </wsdl:operation>
 </wsdl:portType>
```

define the operations in terms of port types



Example – bindings section

- specifies concrete protocol and data format specifications for the operations and messages.

```
<wsdl:binding name="sensor-services-1SOAP" type="tns:sensor-services-1">
  <soap:binding style="document"
                transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="getMeasurement">
    <soap:operation
        soapAction="http://ictp.it/sensor-services-1/getMeasurement"/>
    <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

The communication will be Implemented using SOAP with literal (or concrete format)

getMeasurement(int) -> float

Example – services section

- Aggregate a set of related ports (for example with different bindings).

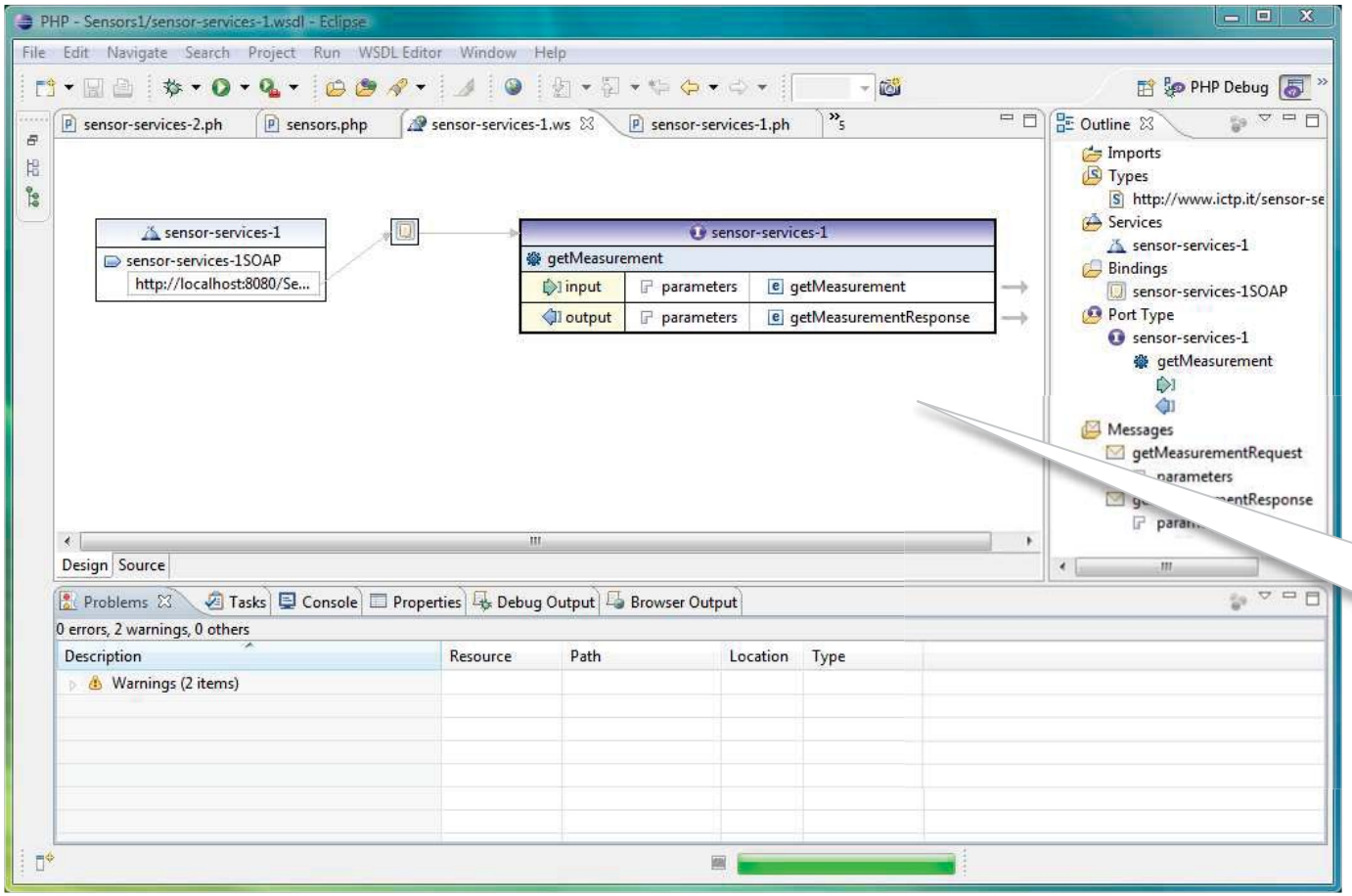
```
<wsdl:service name="sensor-services-1">
   <wsdl:port
         binding="tns:sensor-services-1SOAP" name="sensor-services-1SOAP">
     <soap:address
         location="http://localhost:8080/Sensors1/sensor-services-1.php"/>
   </wsdl:port>
 </wsdl:service>
```

getMeasurement(int) -> float

This is the service end point

WSDL – Design Tools

- Fortunately, many tools can help with WSDL definition!



We will use this tool in the laboratory exercises!

PHP server for the example

```php
<?php
function getMeasurement($sensorId) {

  include("../Sensors.sensors.php");
  $sensors = new Sensors;

  try {
    $data = $sensors->getLastMeasurement($sensorId);
    } catch(Exception $e) {
    return new SoapFault("service", "Cannot get measurement");
    }
    return $data;
  }

  ini_set("soap.wsdl_cache_enabled", "0");
  $server = new SoapServer("sensor-services-1.wsdl");
  $server->addFunction("getMeasurement");
  $server->handle();
?>
```

This is the function that implements the web service

Sensors is a support class that provides last measure for a sensor

SOAP faults can be generated in case of errors

Server creation

PHP client for the example

```php
<?php

$client = new SoapClient("http://localhost:8080/Sensors1/sensor-services-1.wsdl");

  $request = 2;
  try {
    $response = $client->getMeasurement($request);
    echo "Last measurement from sensor: " . $request .
          " is :" . $response;
  } catch (Exception $e) {
    echo "No measurements found for sensor : " . $request;
  }
?>
```

Create a **SOAP** client based on its WSDL specification

Call the service

This is the output of the execution of the program

- The output of the execution is:

```
Last measurement for sensor: 2 is: 20.5
```

WSDL – complex types

- Complex types can have element children
- Example of a two elements sequence:

```
<types>
  <xs:schema targetNamespace="http://ictp.it/dataformat/"
                  xmlns="http://ictp.it/dataformat/" >
  <xs:complexType name="MyInput">
    <xs:sequence>
      <xs:element name="x" type="xs:double"/>
      <xs:element name="y" type="xs:double"/>
    </xs:sequence>
  </xs:complexType>
  </xs:schema>
</types>
```

First element

Second element

WSDL – complex types

- Complex types can also have attributes

```
<types>
  <xs:schema targetNamespace="http://ictp.it/dataformat/"
                xmlns="http://ictp.it/dataformat/" >
  <xs:complexType name="MyInput">
    <xs:sequence>
      <xs:element name="x" type="xs:double"
                  minOccurs="0" maxOccurs="1">
      <xs:element name="y" type="xs:double"
                  minOccurs="1" maxOccurs="1">
      <xs:element name="z" type="xs:double"
                  maxOccurs="unbounded">
    </xs:sequence>
  </xs:complexType>
  </xs:schema>
</types>
```

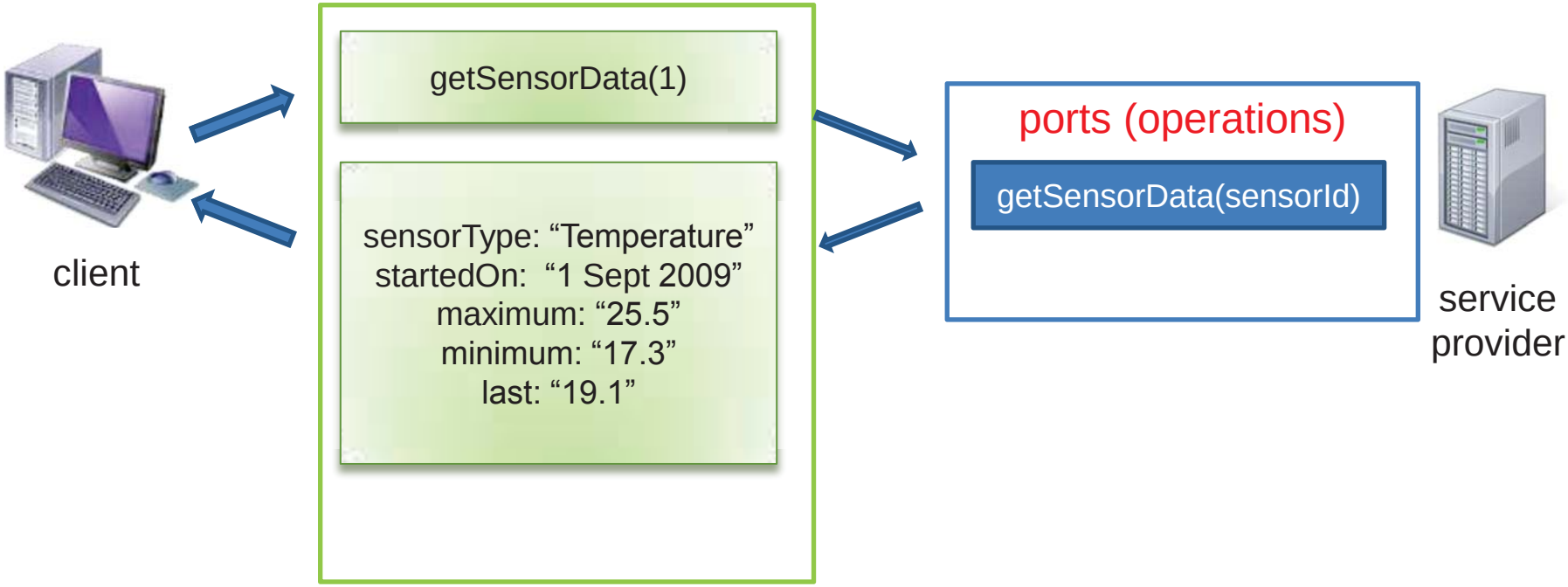
can be or not

must exist

any number of occurrences, equivalent to an array

WSDL – an example with complex types

- In this example, the client requests all sensor information by passing its ID:



Example 2 – type section

```
<wsdl:types>
  <xsd:schema targetNamespace="http://ictp.it/sensor-services-2/">
   <xsd:element name="getSensorData" type="xsd:int">
   </xsd:element>
   <xsd:element name="getSensorDataResponse">
    <xsd:complexType>
     <xsd:sequence>
     <xsd:element name="sensorType" type="xsd:string"/>
     <xsd:element name="last" type="xsd:float“/>
     <xsd:element name="minimum" type="xsd:float“/>
     <xsd:element name="maximum" type="xsd:float“/>
     <xsd:element name="startedOn" type="xsd:string“/>
    </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
  </xsd:schema>
 </wsdl:types>
```

getSensorData(int)

returns a sequence of data as a results

Example 2 – messages section

```
<wsdl:message name="getSensorDataRequest">
  <wsdl:part element="tns:getSensorData" name="parameters"/>
 </wsdl:message>
 <wsdl:message name="getSensorDataResponse">
  <wsdl:part element="tns:getSensorDataResponse" name="parameters"/>
 </wsdl:message>
```

Request message: int

Response message: complex data

define the messages to be used for communcation

PHP server for the example 2

```php
<?php
 function getSensorData($sensorId) {

 include("../Sensors/sensors.php");
 $sensors = new Sensors;

try {
   $data = $sensors->getSensorData($sensorId);
   } catch(Exception $e) {
   return new SoapFault("sensor-service“,  "Cannot get data”);
   }
   return $data;
 }

 ini_set("soap.wsdl_cache_enabled", "0");
 $server = new SoapServer("sensor-services-2.wsdl");
 $server->addFunction("getSensorData");
 $server->handle();
 ?>
```

This is the function that implements the web service

```
array("sensorType" => “temperature”,
      "startedOn" => “1 Sep 2009”,
      "maximum" => 22.1,
      "minimum" => 18.7,
      "last" => 19.2);
```

Server creation

PHP client for the example 2

```php
<?php
  $client = new SoapClient("http://localhost:8080/Sensors2/sensor-services-2.wsdl");
  $request = 1;
  try {
    $response = $client->getSensorData($request);
    echo "Type:    " . $response->sensorType;
    echo "Started: " . $response->startedOn;
    echo "Last:    " . $response->last;
    echo "Minimum: " . $response->minimum;
    echo "Maximum: " . $response->maximum;
  } catch (Exception $e) {
  echo "Sorry, no data found for sensor: \"" . $request .
          "\"<br>";
  }
?>
```

Create a **SOAP** client based on its WSDL specification

Call the service and gets (example):

```
array("sensorType" => “temperature”,
      "startedOn" => “1 Sep 2009”,
      "maximum" => 22.1,
      "minimum" => 18.7,
      "last" => 19.2);
```

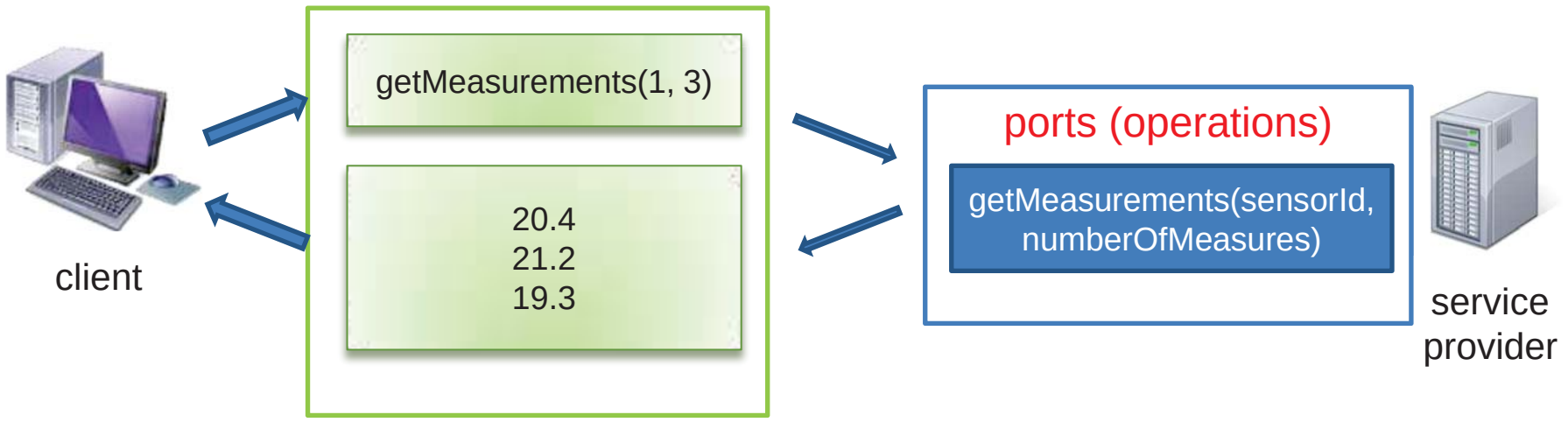
This is the output of the execution of the program

- The output of the execution is:

Type: temperature Started: 1 Sep 2009 Last: 19.2 Minimum: 18.7 Maximum: 22.1

WSDL – another example with complex types

- In this example, the client requests a specific number of measurements for a particular sensor:



Example 3 – type section

```
<wsdl:types>
  <xsd:schema targetNamespace="http://ictp.it/sensor-services-3/">
   <xsd:element name="getMeasurements">
    <xsd:complexType>
     <xsd:sequence>
     <xsd:element name="sensor" type="xsd:int"/>
     <xsd:element name="number" type="xsd:int">
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="getMeasurementsResponse">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="measurements" type="xsd:float“ maxOccurs="unbounded"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
  </xsd:schema>
 </wsdl:types>
```

note that the number of elements for **measurements** is not limited

Example 3 – messages section

```
<wsdl:message name="getMeasurementsRequest">
  <wsdl:part element="tns:getMeasurements" name="parameters"/>
</wsdl:message>
<wsdl:message name="getMeasurementsResponse">
  <wsdl:part element="tns:getMeasurementsResponse" name="parameters"/>
</wsdl:message>
```

Request message: complex data

Response message: complex data

define the messages to be used for communcation

PHP server for the example 3

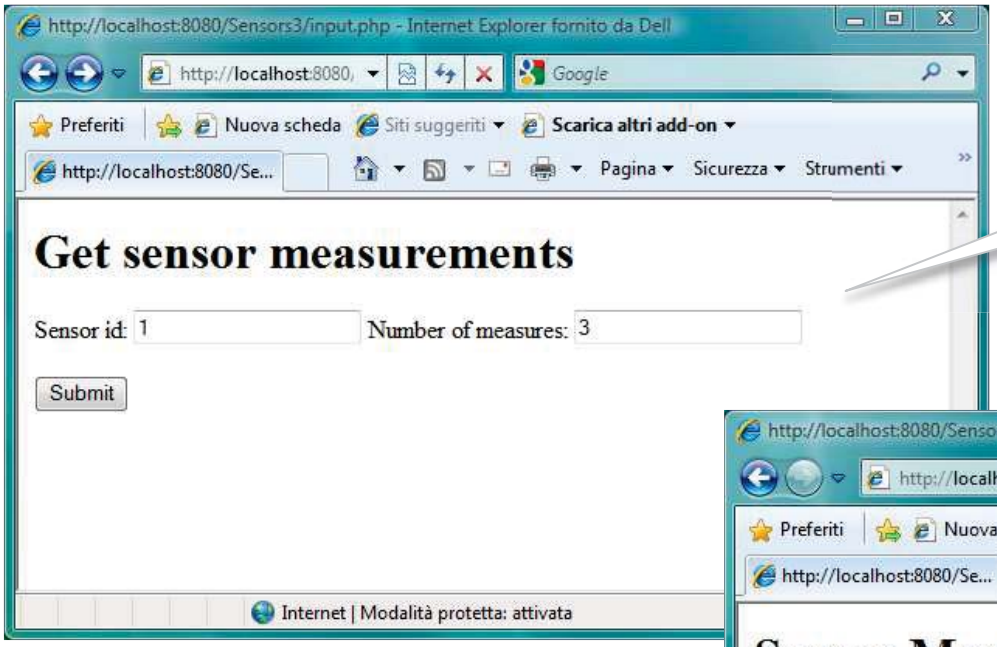
```php
<?php
  function getMeasurements($request) {

  include("../Sensors/sensors.php"); $sensors = new Sensors;

try {
    $temperature = $sensors->getAllMeasurements($request->sensor);
    } catch(Exception $e) {
    return new SoapFault("sensor-service“, "Cannot get measurements”);
    }
    return array_slice($temperature, 0, $request->number);
  }

  ini_set("soap.wsdl_cache_enabled", "0");
  $server = new SoapServer("sensor-services-3.wsdl");
  $server->addFunction("getMeasurements");
  $server->handle();
?>
```

This is the function that implements the web service

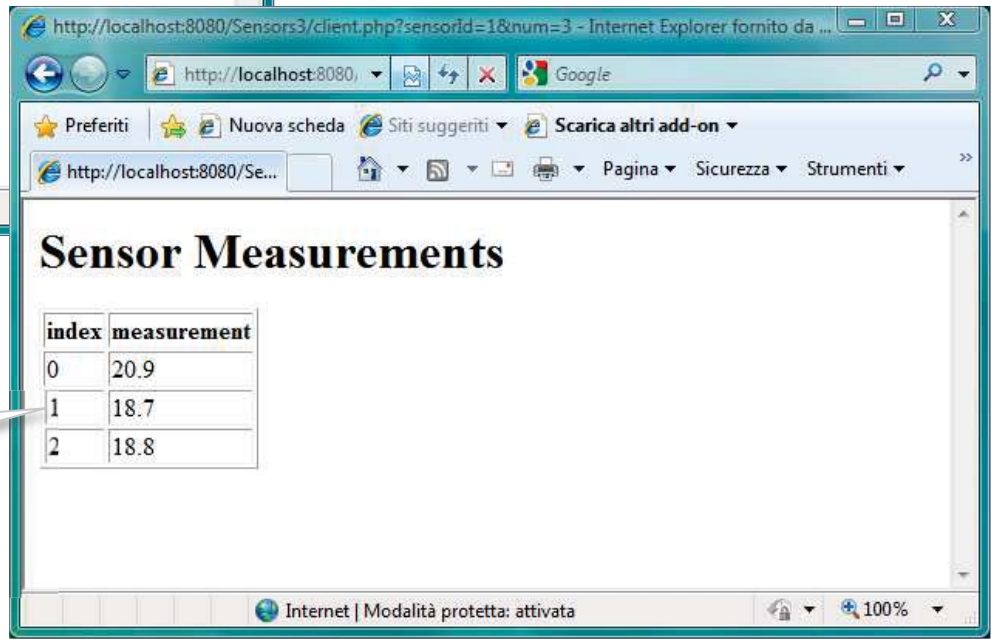
array(20.9, 18.7, 18.8)

Server creation

PHP web client for the example 3



The user can request a set of measurements of a specific sensor…



…which are obtained calling the previous web service and displayed

PHP web client for the example 3

```
<html>
  <body>
  <h1>Get sensor measurements</h1>
  <form name="input" action="client.php" method="get">

    Sensor id: <input type="text" name="sensorId"/><br>

    Number of measures: <input type="text" name="num"/><br>

    <input type="submit" value="Submit"/>

  </form>
  </body>
</html>
```

input form that calls **client.php** with input data

fields are **sensor id** and **number of measures**

execution is requested when the user clicks on the **submit** button

PHP web client for the example 3

```
<html>
  <body>
  <h1>Sensor Measurements</h1>
<?php
  $client = new SoapClient("http://localhost:8080/Sensors3/sensor-services-3.wsdl");
  $request = array("sensor" => $_GET["sensorId"], "number" => $_GET["num"]);
  $response = $client->getMeasurements($request);
  $measurements = $response->measurements;

  echo "<table border=\"1\"><tr><th>index</th><th>measurement</th></tr>";
  foreach ($measurements as $i => $value) {
    echo "<tr><td>" . $i . "</td><td>" . $value . "</td></tr>";
  }
  echo "</table>";
?>
  </body>
</html>
```

create a **SOAP** client based on its WSDL specification

call the service

prepare input parameters based on user request

prepare an HTML table with all data returned by the web service call

SOAP - Simple Object Access Protocol

- Communication protocol for program to program communication
- Platform independent
- Language independent
- Based on XML
- Simple and extensible
- Go over HTTP (allows to get around firewalls!)

SOAP - Simple Object Access Protocol

- This was the message sent on our last example:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                        xmlns:q0="http://ictp.it/sensor-services-3/"
                        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <q0:getMeasurements>
      <sensor>1</sensor>
      <number>3</number>
    </q0:getMeasurements>
  </soapenv:Body>
</soapenv:Envelope>
```

Here we can identify the parameters of the requested operation

SOAP - Simple Object Access Protocol

- This was the message recevied on our last example:

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
                                    xmlns:ns1="http://ictp.it/sensor-services-3/">
<SOAP-ENV:Body>
  <getMeasurementsResponse>
    <measurements>20.9</measurements>
    <measurements>18.7</measurements>
    <measurements>18.8</measurements>
  </getMeasurementsResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Here we can identify the results of the requested operation

Other topics not covered

- RPC binding
  - first approach to web services
  - maps services directly to language-specific functions or method calls
- Restful (REpresentational State Transfer) web services:
  - emerging as a popular alternative to using SOAP-based technologies
  - lightweight nature
  - transmit data directly over HTTP.

References

- Web Services (WSDL - SOAP)
  - http://www.w3.org/2002/ws/desc/
- XML
  - http://www.w3.org/TR/xmlbase/
- Book on PHP and web services:
  - **PHP Cookbook**. *Adam Trachtenberg, David Sklar* (2006). O'Reilly Media.
- Many resources on Internet (just Google for them)\