The WSDL Document Structure (1)

- `<portType>`
  - The operations performed by the web service and the involved messages
  - Interface of the service exposed to the client
  - Corresponding to a function library

- `<message>`
  - The data elements used by the web service
  - Consist of one or more parts; define the name and data type of each part
  - Corresponding to the parameters of a function

- `<types>`
  - The data types used by the web service
  - Address heterogeneity between clients and server
The WSDL Document Structure (2)

- `<binding>`: define the details of how to format and transport the messages

  - **Formatting messages**
    - Define which version of SOAP is used
    - Style of the message (document or RPC)
    - How the message appear inside the SOAP body (literal or encoded)

  - **Transporting messages**
    - Which communication protocols used to transport the message (e.g. HTTP or other transport protocols)
Layout of WSDL document

<definitions>

<types>
  definition of types........
</types>

<message>
  definition of a message....
</message>

<portType>
  definition of a port.......  
</portType>

<binding>
  definition of a binding....
</binding>

</definitions>
WSDL example

<message name="getTermRequest">
    <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
    <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
    <operation name="getTerm">
        <input message="getTermRequest"/>
        <output message="getTermResponse"/>
    </operation>
</portType>
Relation between service, operations, resource, messages, bindings

- **messages**
  - TCP
  - HTTP
  - SMTP

- **endpoints**
  - URI

- **bindings**
  - interface
    - operation
  - interface
    - operation

- **service**

- **resource**
An example of grid service

- simple *Math Web Service*, referred to as *MathService*.

- The service is associated to one resource, which contains two resource properties (RPs for short):
  - Value (integer)
  - LastOP: Last operation performed (string)

- The service allows users to perform the following operations:
  - addition
  - subtraction
  - getValueRP (access the Value RP)

- The addition and subtraction operations expect only *one integer parameter* and don’t return anything
Step 1: Defining the interface in WSDL – PortType

```xml
<definitions ... >

<portType name="MathPortType"
    wsdlpp:extends="wsrpw:GetResourceProperty"
    wsrp:ResourceProperties="tns:MathResourceProperties">

    <operation name="add">
        <input message="tns:AddInputMessage"/>
        <output message="tns:AddOutputMessage"/>
    </operation>

    <operation name="subtract">
        <input message="tns:SubtractInputMessage"/>
        <output message="tns:SubtractOutputMessage"/>
    </operation>

    <operation name="getValueRP">
        <input message="tns:GetValueRPInputMessage"/>
        <output message="tns:GetValueRPOutputMessage"/>
    </operation>

</portType>
</definitions>
```
Step 1: Defining the interface in WSDL – Messages

```xml
<definitions ... >

<message name="AddInputMessage">
  <part name="parameters" element="tns:add"/>
</message>

<message name="AddOutputMessage">
  <part name="parameters" element="tns:addResponse"/>
</message>

<!-- PortType -->

</definitions>
```
Step 1: Defining the interface in WSDL – Types

```xml
<definitions ... />
<types>
<xsd:schema ... >

<!-- REQUESTS AND RESPONSES -->
<xsd:element name="add" type="xsd:int"/>
<xsd:element name="addResponse">
  <xsd:complexType/>
</xsd:element>

<!-- more type definitions -->
</xsd:schema>
</types>
<!-- Messages -->
<!-- PortType -->
</definitions>
```
Step 1: Defining the interface in WSDL – Messages (alternatively)

```xml
<definitions ... >

<message name="AddInputMessage">
    <part name="parameters" type="xsd: int"/>
</message>

<message name="AddOutputMessage">
    <part name="parameters"
        element="tns:addResponse"/>
</message>

<!-- PortType -->

</definitions>
```
Step 1: Defining the interface in WSDL – Types
(add operation has two parameters)

```xml
<xsd:element name="add">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="a1" type="xsd:int"/>
      <xsd:element name="a2" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

→ Advantages of declaring the data type as an element

- The element can contain multiple sub-parameters
- The element is reusable by other services
Step 1: Defining the interface in WSDL – Types

```xml
<types>
<xsd:schema ...>

<!-- Requests and responses declarations-->

<xsd:element name="Value" type="xsd:int"/>
<xsd:element name="LastOp" type="xsd:string"/>

<xsd:element name="MathResourceProperties">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:Value" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="tns:LastOp" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
</types>
```
Using WSDL editor to write the WSDL file
Step 2: Implementing the service
Stub classes in web service invocation

1. The client application calls the client stub. The client stub will turn this ‘local invocation’ into a proper SOAP request (called *marshaling* or *serializing*).

2. The SOAP request is sent over a network using the HTTP protocol. The server receives the SOAP requests and hands it to the server stub for unmarshalling.

3. Once the SOAP request has been unmarshalled, the server stub invokes the service implementation.

4. The result of the requested operation is handed to the server stub, which will turn it into a SOAP response.

5. The SOAP response is sent over a network using the HTTP protocol. The client stub receives the SOAP response and turns it into something the client application can understand.

6. The application receives the result of the Web Service invocation.
Step 2: Implementing the service

- WSDL is language-neutral and there is no mention of the language in which the service is going to be implemented

- In our case, we are going to use Java to implement the services

- So, we need to map the WSDL namespaces to Java packages

  http://www.globus.org/namespaces/examples/core/MathService_instance=
  org.globus.examples.stubs.MathService_instance

- Then, stub classes are automatically generated from the WSDL file
  - Stub class generates, sends and receives the SOAP requests
Step 2: Implementing the Service – the QName interface

- An element related to a web service can be referred to by the Qualified Names (QNames)

  - QName includes a namespace and a local name – i.e. \{namespace\}local_name
  - The QName of the element Value is \{http://www.globus.org/namespaces/examples/core/MathService_instance\}Value
  - A qualified name is represented in Java using the QName class
  - It is a good practice to put all frequently used qualified names into a separate interface

```java
public interface MathQNames {
    public static final String NS = "http://www.globus.org/namespaces/examples/core/MathService_instance";
    public static final QName RP_VALUE = new QName(NS, "Value");
    public static final QName RP_LASTOP = new QName(NS, "LastOp");
    public static final QName RESOURCE_PROPERTIES = new QName(NS, "MathResourceProperties");
}
```
Step 2: Implementing the Service – Service Implementation

```java
public AddResponse add(int a) throws RemoteException {
    value += a; lastOp = "ADDITION";
    return new AddResponse();
}

public SubtractResponse subtract(int a) throws RemoteException {
    value -= a; lastOp = "SUBTRACTION";
    return new SubtractResponse();
}

public int getValueRP(GetValueRP params) throws RemoteException {
    return value;
}
```
Step 3: Configuring the deployment of the service
The Architecture of the Web Server

- **SOAP Engine**: handling SOAP requests and responses
  - Apache Axis

- **Application Server**: SOAP engine runs as an application in it
  - Jakarta Tomcat

- **HTTP Server**: handling HTTP messages
  - Apache HTTP Server

- The above three altogether are called “Web service container”
Step 3: Configuring the deployment in WSDD (1)

- Up to this point, we have written the service interface (WSDL) and the service implementation (Java).
- We need to make them available to the clients (through a web service container); this process is called the deployment.
- The deployment descriptor file is used to tell the Web Services container how the web service should be published.
- The deployment descriptor is written in WSDD format (Web Service Deployment Descriptor).
Step 3: Configuring the deployment in WSDD (2)

```xml
<deployment name="defaultServerConfig"
    xmlns="http://xml.apache.org/axis/wsdd/
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <service name="examples/core/first/MathService" … >
        <parameter name="className" value="org.globus.
            examples.services.core.first.impl.MathService"/>
        <wsdlFile>share/schema/examples/MathService_
            instance/Math_service.wsdl</wsdlFile>
        <parameter name="loadOnStartup" value="true"/>
    … 
    </service>
</deployment>
```
Step 4: Create a GAR file with Ant

Using those three files we wrote in the previous three pages we will generate a Grid Archive, or GAR file.

Creating a GAR file involves:
- Processing the WSDL file to add missing pieces
- Creating the stub classes from the WSDL
- Compiling the stubs classes
- Compiling the source code of service implementation
- Organize all the files into a specific directory structure

**Ant**, a java build tool, is used to create the GAR file:
- Ant can generate the executables from the source code, following a buildfile
- A build.xml file to direct Ant to perform other steps

In GT4, Ant is wrapped in a shell script, globus-build-service.sh.

So, creating a GAR file comes down to run:

```
./globus-build-service.sh -d <service base directory> -s <service's WSDL file>
```
Step 5: Deploy the service into a Web Services container

- Deployment is done with a GT4 tool that unpacks the GAR file and copies the files within into the corresponding locations in Web Service Container, i.e. run:

  \[ \text{globus-deploy-gar <GAR file>} \]

- Using the following command to undeploy the service:

  \[ \text{globus-undeploy-gar <service>} \]