BBM490

ENTERPRISE WEB ARCHITECTURE

Mert ÇALIŞKAN

Hacettepe University Spring Semester ’15

Recap of Week 04

- We defined what IoC / DI is and how it's defined in Spring Framework.
- We defined the base interfaces:
  BeanFactory & ApplicationContext.
- We created a Spring-ified app with the help of Maven. We configured the app with XML and with Java Code.

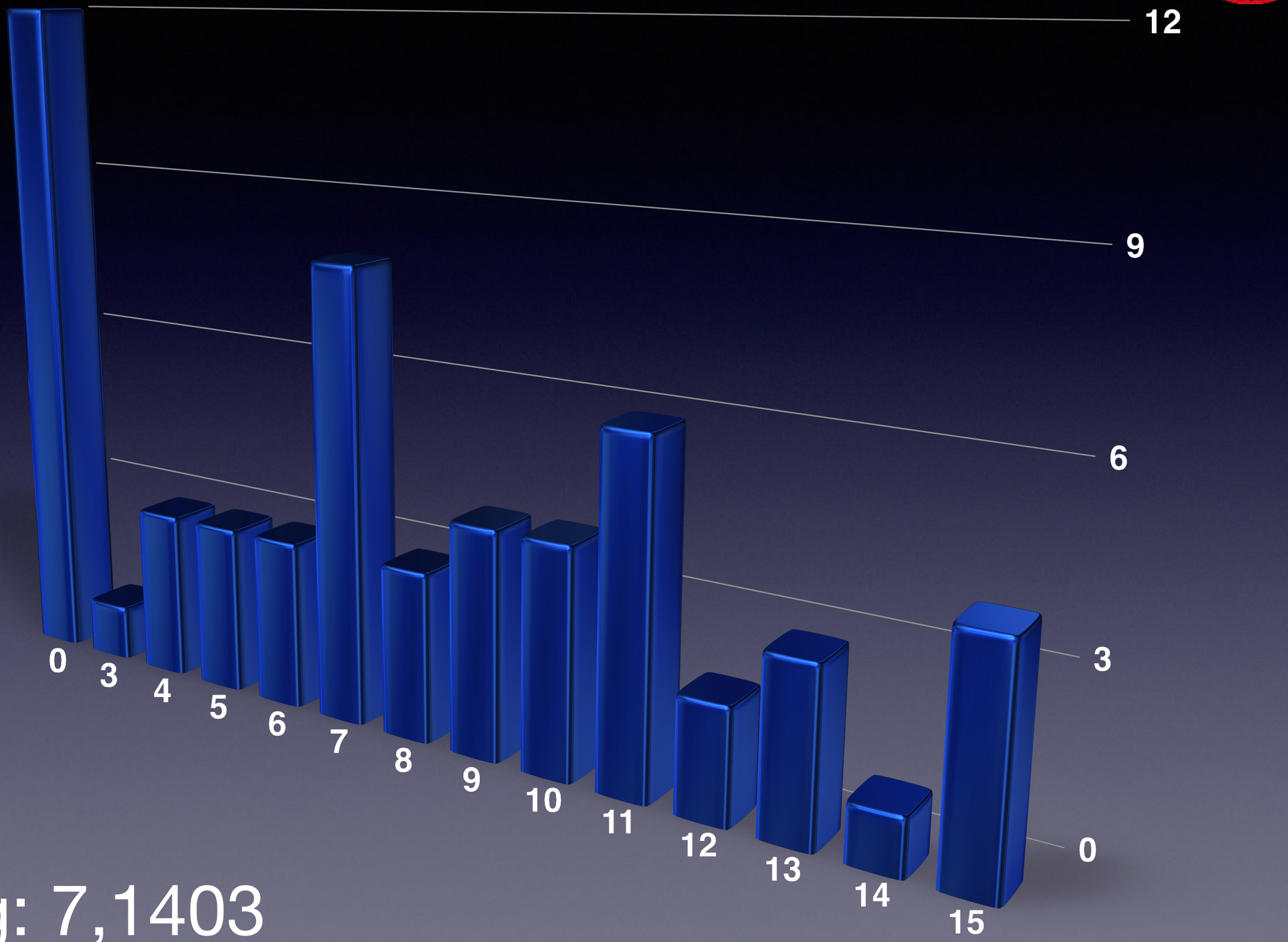
Recap of Week 04

- We detailed the <bean> tag appContext.xml.
- We defined injecting methods.
- We mentioned abstract beans, inner bean definitions and ways to injecting collections.
- We detailed how the auto-wiring applies. (byName | byType | constructor | no)
- We gave details on the annotation based configs.
- We created a Spring-ified web application with the help of Maven and get the spring context up & running.

Recap of Week 04

- We gave the overall picture for Spring Framework and tried to detail its components like,
  - Core Container
  - Web
  - Data Access / Integration
  - Testing
  - AOP
- We detailed AOP with its concepts and AspectJ.

Quiz Notes



Avg: 7,1403

Week 5

Web Services

We'll define the Web Services concept and will create services with JAX-WS and JAX-RS JavaEE Standards

The need of integration

- Everything is in need to communicate.
- With the advent of internet, machines got more powerful while they are communicating with each other.
- Applications now remove the human latency and point them to be a part of more creative ways.
- By nature, there can’t be only one application so there is a need for application-to-application (A2A) communication.

So,
How can we achieve this integration in a standardised way?

Web Service

- Web service is the answer..! It's a software system designed to support interoperable machine-to-machine interaction over a network.
- It's an application interface that conforms to specific standards, which enables other applications to communicate with it.
- As mentioned above Web Services are based on standards like XML, SOAP, WSDL and UDDI. We will cover them all.

Web Service / Advantages

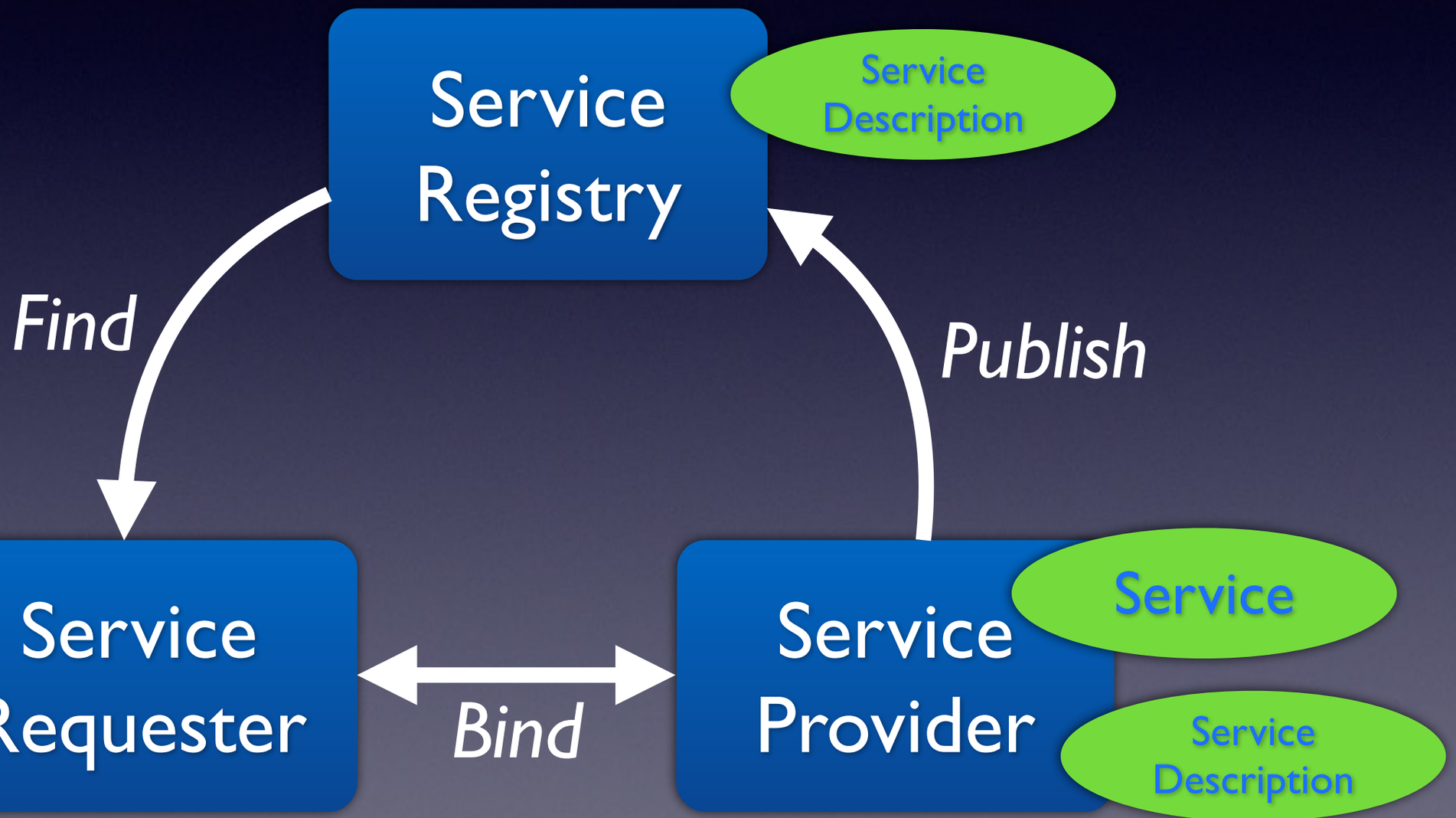
- Web Services operates regardless of the
  - programming language
  - hardware platform
  - operating system

- It has an interface described in a machine-processable format (specified with WSDL).

Disadvantages

- Performance:
  - Larger payload compared to proprietary mechanisms
  - Network Delay
  - Serialization/Deserialization cost
- Stateless &No Session Management
- Integration Effort:
  - XML/JSON are not equivalent to programming language constructs.
  - Domain Model- Integration Model do not always match.

How to consume a Web Service



Web Service Architecture: Roles, Operations and Artefacts

- Service Registry is the central registry that allows you to find the published services.
- Provider publishes its services on the registry while the requester finds and invokes (bind) it.
- Invocation of a service without using a registry is also possible and it's used commonly.

The Standards

4 main artefacts that are used to define Web Services

XML: Extensible Markup Language
It is used to define the data

SOAP: (Simple Object Access Protocol)
It is used to transfer the data

WSDL: (Web Services Description Language)
It is used to define the service

UDDI: (Universal Description, Discovery and Integration)
It is used to list the available services

XML

- Stands for EXtensible Markup Language
- It's used to define the data for transportation and storing (unlike HTML, which is just to display the data)
- It's human readable and machine readable.

- Created by W3C (World Wide Web Consortium)

123456, shipped, Mr, Mehmet, Turgut, Mustafa Kemal Cad., No 6/A, Cankaya, Ankara, 06300, Turkey

Any idea what this comma separated stuff is?

XML

When we format the data with some special text to make it to look more structured. Makes more sense now?

```
<order>
  <orderNo shipped=“true”>123456</orderNo>
  <shipTo>
    <name>Mr Mehmet Turgut</name>
    <address>Mustafa Kemal Cad. No 6/A, Cankaya,
Ankara, 06300, Turkey</address>
  </shipTo>
</order>
```

XML - order.xml

- What if we even make it more structured?

```
<order>
<orderNo shipped="true">123456</orderNo>
<shipTo>
        <name>
            <title>Mr</title>
            <firstName>Mehmet</firstName>
            <lastName>Turgut</lastName>
        </name>
        <address>
            <addressLine1>Mustafa Kemal Cad. No 6/A</addressLine1>
            <addressLine2></addressLine2>
            <county>Cankaya</county>
            <city>Ankara</city>
            <zip>06500</zip>
            <country>Turkey</country>
        </address>
</shipTo>
</order>
```

XML
Elements vs. Attributes

- In XML, there are no rules about when to use attributes, and when to use child elements.
- Data can be stored in attributes or child elements.

```
<note date="12/11/2002">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

date as attribute

date as element

```
<note>
  <date>12/11/2002</date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Which one to use?

- attributes cannot contain multiple values (child elements can)
- attributes are not easily expandable (for future changes)
- If you use attributes over elements, you end up with documents that are difficult to read and maintain.
- Elements should be used to describe the data wherever possible.

Where do we define this structure?

- XML can refer to a definition for this, named DTD.
- DTD stands for Document Type Definition.
- It defines the document structure with a list of elements and attributes.

- DTD can be declared inside an XML document or as an external interface.

defines the root element

```
<!DOCTYPE documentelement [definition]>
```

```
<!DOCTYPE documentelement SYSTEM "documentelement.dtd">
```

DTD of order.xml

The root element

```
<!DOCTYPE order [
      <!ELEMENT order (orderNo|shipTo)*>
      <!ELEMENT orderNo (#PCDATA)>
      <!ATTLIST orderNo
            shipped CDATA #REQUIRED>
      <!ELEMENT shipTo (name|address)*>
      <!ELEMENT name (title|firstName|lastName)*>
      <!ELEMENT title (#PCDATA)>
      <!ELEMENT firstName (#PCDATA)>
      <!ELEMENT lastName (#PCDATA)>
      <!ELEMENT address (addressLine1|addressLine2|county|city|zip|country)*>
      <!ELEMENT addressLine1 (#PCDATA)>
      <!ELEMENT addressLine2 (#PCDATA)>
      <!ELEMENT county (#PCDATA)>
      <!ELEMENT city (#PCDATA)>
      <!ELEMENT zip (#PCDATA)>
      <!ELEMENT country (#PCDATA)>
]>
```

Defines what elements that root contains

Defines the shipped attribute

defines the elements to be of type "#PCDATA"

Makes more sense now?

```
<order>
<orderNo shipped="true">123456</orderNo>
<shipTo>
        <name>
            <title>Mr</title>
            <firstName>Mehmet</firstName>
             <lastName>Turgut</lastName>
        </name>
        <address>
            <addressLine1>Mustafa Kemal Cad. No 6/A</addressLine1>
            <addressLine2></addressLine2>
            <county>Cankaya</county>
            <city>Ankara</city>
            <zip>06500</zip>
            <country>Turkey</country>
        </address>
</shipTo>
</order>
```

PCDATA

- PCDATA stands for Parsed Character Data.
- PCDATA is text that will be parsed by a parser. The text will be examined by the parser for entities and markup.
- Think of it as character data as the text found between the start tag and the end tag of an XML element.

```
<?xml version="1.0"?>
<foo>
<bar><test>content!</test></bar>
</foo>
```

- By default, everything is PCDATA.

CDATA

- The CDATA defines the text that will *not* be parsed by a parser. Tags inside the text will *not* be treated as markup and entities will not be expanded.

```
<?xml version="1.0"?>
<foo>
<bar><![CDATA[<test>content!</test>]]></bar>
</foo>
```

- In the example above how is the content of the <bar> tag will be interpreted by a parser?

Why use a definition, a la DTD?

- With a DTD, each of your XML files can carry a description of its own format.
- With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.
- Your application can use a standard DTD to verify that the data you receive from the outside world is valid.
- You can also use a DTD to verify your own data.

Drawbacks of DTD

- DTD's do not support namespaces (we will cover the concept of 'namespace' in a bit).
- With DTD you cannot validate the content to data types like,
  - The length of a field or a validation of an email address
  - You cannot define custom data types
- So with all these stated, of course there is a better alternative to define what can go inside an XML. And it's called XSD.

XSD

- Stands for XML Schema Definition.
- It describes what a given XML document can contain.
- It’s also an XML, it has the same notation.
- You can think of it as a database schema, which describes the data that can be contained inside a database.
- XSD is controlled by the World Wide Web Consortium (W3C), They are also managing HTML.

How do we define XSD

- XSD contains 4 types of data.
  - Elements
  - Types
    - Simple Types
    - Complex Types
  - Attributes
  - Groups
- We'll go into the detail of defining elements and types along with the attributes.

<xs:element>

- Elements are the main building block of all XML documents, containing the data and determine the structure of the instance document
- An element can be defined as follows:

```
<xs:element name="x" type="y" />
```

- Each element definition within the XSD must have a 'name' property, which is the tag name that will appear in the XML document.
- The 'type' property provides the description of what type of data can be contained within the element when it appears in the XML document.

<xs:element>

- There are a number of predefined simple types like,
  - xs:string
  - xs:integer
  - xs:boolean
  - xs:date
- To extend these built-in types you can do type definition with simple/complex types.

<xs:element> Example

- An example definition for <name> and <address> definition will be as follows:

```
<xs:element name="name" type="xs:string"/>
<xs:element name="address" type="xs:string"/>
```

<xs:simpleType>

- A simple type extends the built in data types such as xs:string, xs:integer, and xs:date, allowing you to create your own data types.
- Possible use cases for this could be,
- Defining an ID, this may be an integer with a maximum value limit.
- E-Mail address that complies with a regular expression.
- A field with a maximum length and so on.

<xs:simpleType>

- Examples of simple type definitions:

```
<xs:simpleType name="stringtype">
    <xs:restriction base="xs:string"/>
</xs:simpleType>
```

```
<xs:simpleType name="inttype">
    <xs:restriction base="xs:positiveInteger"/>
</xs:simpleType>
```

```
<xs:simpleType name="dectype">
    <xs:restriction base="xs:decimal"/>
</xs:simpleType>
```

```
<xs:simpleType name="orderidtype">
    <xs:restriction base="xs:string">
        <xs:pattern value="[0-9]{6}"/>
    </xs:restriction>
</xs:simpleType>
```

<xs:simpleType>

- Here is a simpleType definition for email addresses. We are extending the base type string and adding a regular expression validator as a pattern.

```
<simpleType name="emailAddress">
  <restriction base=“string">
   <pattern
    value="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
   </pattern>
  </restriction>
</simpleType>
```

- We are defining an element of this simpleType as,

```
<element name="contactMail" type="tns:emailAddress"></element>
```

<xs:complexType>

- A complex type is a container for other element definitions, this allows you to specify which child elements an element can contain. This allows you to provide some structure within your XML documents.

```
<complexType name="note">
 <sequence>
   <element name="from" type="string"></element>
   <element name="to" type="string"></element>
   <element name="subject" type="string"></element>
   <element name="content" type="string"></element>
 </sequence>
</complexType>
```

<xs:complexType>

- Complex type definition

```
<xs:simpleType name="stringtype">
    <xs:restriction base="xs:string"/>
</xs:simpleType>
```

```
<xs:complexType name="shiptotype">
    <xs:sequence>
        <xs:element name="name" type="stringtype"/>
        <xs:element name="address" type="stringtype"/>
        <xs:element name="city" type="stringtype"/>
        <xs:element name="country" type="stringtype"/>
    </xs:sequence>
</xs:complexType>
```

<xs:attribute>

- An attribute provides extra information within an element.

```
<xs:attribute name="x" type="y" />
```

- An attribute can be defined as follows:

```
<xs:element name="Order">
  <xs:complexType>
    <xs:attribute name="OrderID" type="int" use="optional" />
  </xs:complexType>
</xs:element>
```

- Both of the definitions given below are valid.

```
<Order OrderID="6" />   or <Order />
```

XML namespaces

- A namespace is a unique URI (Uniform Resource Identifier)
- The idea behind namespaces is,
  - To disambiguate between two elements that happen to share the same name.
  - To group elements relating to a common idea together.
- To define a namespace *xmlns* attribute is being used.

```
<tns:contactMail xmlns:tns="http://cs.hacettepe.edu.tr/bbm490"...
```

XML Parsers

- SAX
  - Parses node by node
  - Doesn't store the XML in memory
  - You can't insert or delete nodes, no updates.
  - Runs faster than DOM
- DOM
  - Parses whole XML document
  - Stores the entire XML document into memory before processing
  - You can insert or delete nodes
  - Runs slower than SAX

XML Parsers

- SAX parser: It's event based. When the tag <something> encountered *tagStarted* event gets triggered and when the tag </something> gets encountered *tagEnded* event gets triggered.
- So you need to handle each event and the data that is returned.
- DOM parser: The entire XML gets parsed and represented as a DOM tree. Once parsed user can navigate within the XML. No events gets fired while XML gets parsed.

Parsing with Java Code

```
Reader xml = new StringReader("<foo><bar>test</bar></
foo>");

DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document dom = db.parse(new InputSource(xml));

System.out.println("root element name = " +
dom.getDocumentElement().getNodeName());
```

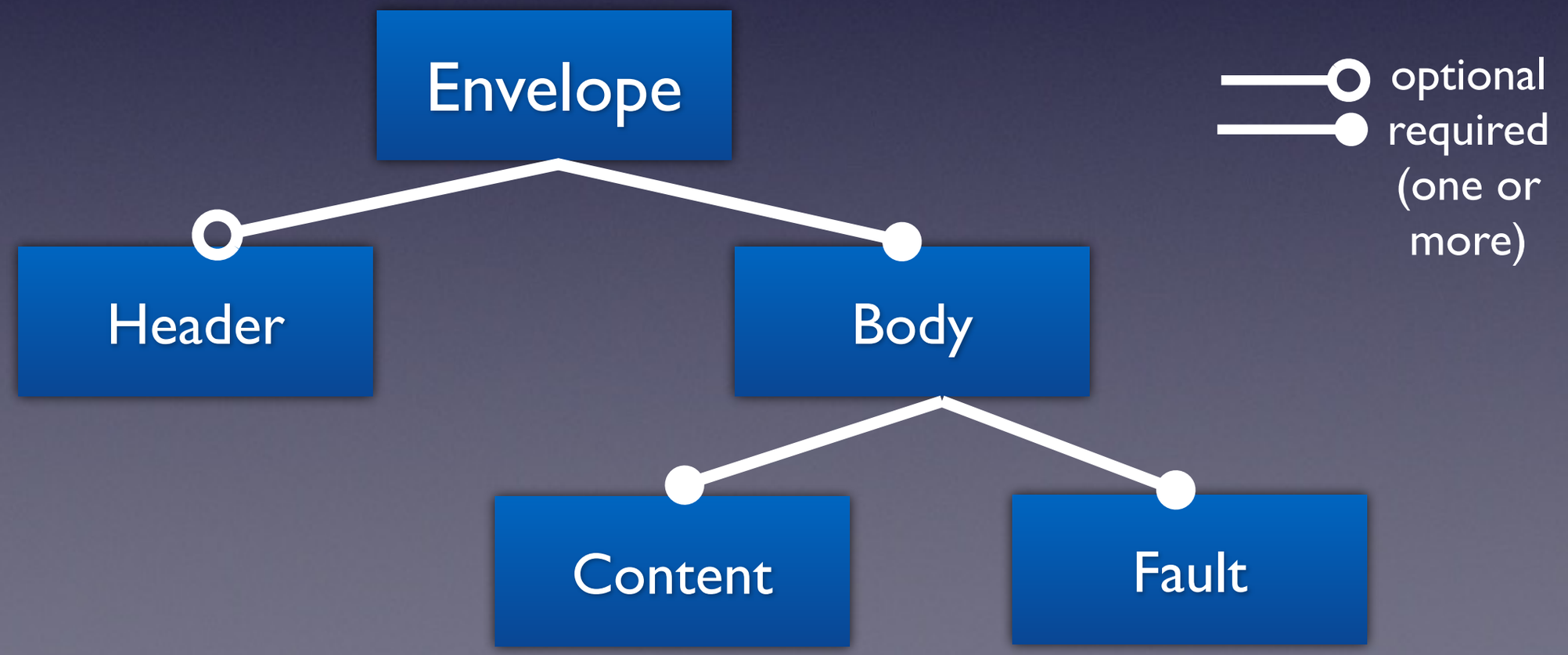
Is it DOM or is it SAX?

Parsing with Java Code

```
Reader xml = new StringReader("<foo><bar>test</bar></foo>");

SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser parser = factory.newSAXParser();

XMLReader reader = parser.getXMLReader();
reader.setContentHandler(new ContentHandler() {
   ...
});
reader.parse(new InputSource(xml));
```

SOAP

- Simple Object Access Protocol
- Cross platform (platform independent) remote calls (usually over HTTP) with XML messages
- It's protocol neutral, could be done also with SMTP (a.k.a. emailing)



Example SOAP Request

```
POST http://localhost:8080/SpringWebServicesProject/
services/countryService HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: text/xml;charset=UTF-8
SOAPAction: ""
Content-Length: 218
Host: localhost:8080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

<soapenv:Envelope xmlns:soapenv="http://
schemas.xmlsoap.org/soap/envelope/" xmlns:bbm="http://
bbm490.hacettepe.edu.tr/">
   <soapenv:Header/>
   <soapenv:Body>
      <bbm:getAll/>
   </soapenv:Body>
</soapenv:Envelope>
```

Example SOAP Response

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ns2:getAllResponse xmlns:ns2="http://bbm490.hacettepe.edu.tr/">
         <return>
            <id>01</id>
            <name>Turkey</name>
         </return>
         <return>
            <id>01</id>
            <name>United States</name>
         </return>
      </ns2:getAllResponse>
   </soap:Body>
</soap:Envelope>
```

Advantages / Disadvantages

- SOAP is versatile enough to allow for the use of different transport protocols. The standard stacks use HTTP as a transport protocol, but other protocols such as SMTP can also be used. JMS and Message Queues can also use SOAP.
- When using standard implementations and the default SOAP/HTTP binding, the XML infoset is serialised as XML.
- This may not be an issue when only small messages are sent but with heavy load of content this could be a problem.

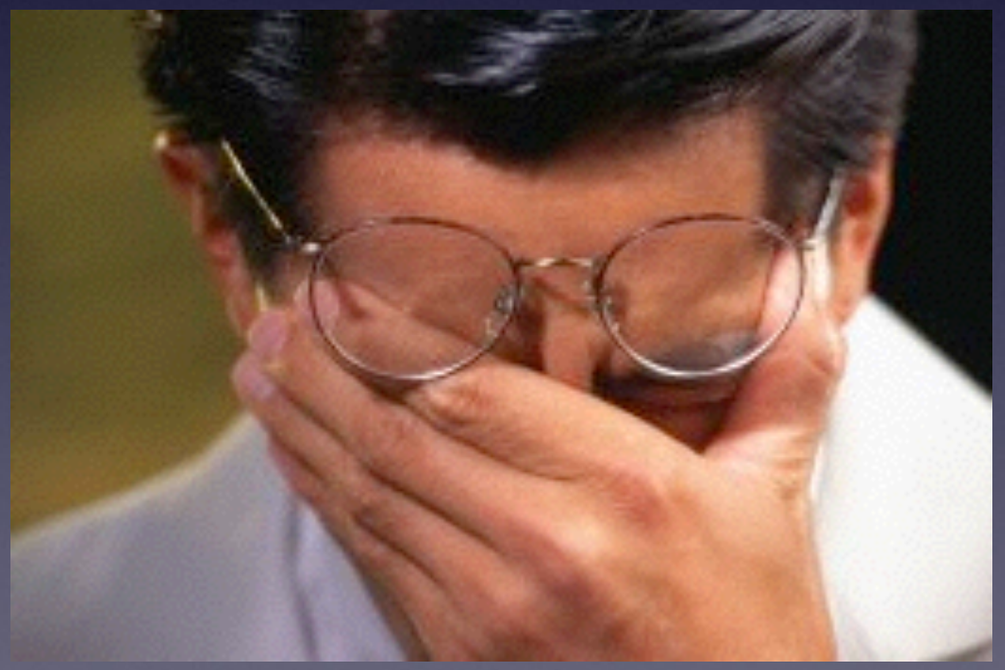
WSDL

- Stands for Web Service Definition Language
- It's XML Based.
- Describes the Point of Contact for Service Provider
- Defines
  - Structure of messages
  - Physical location of service

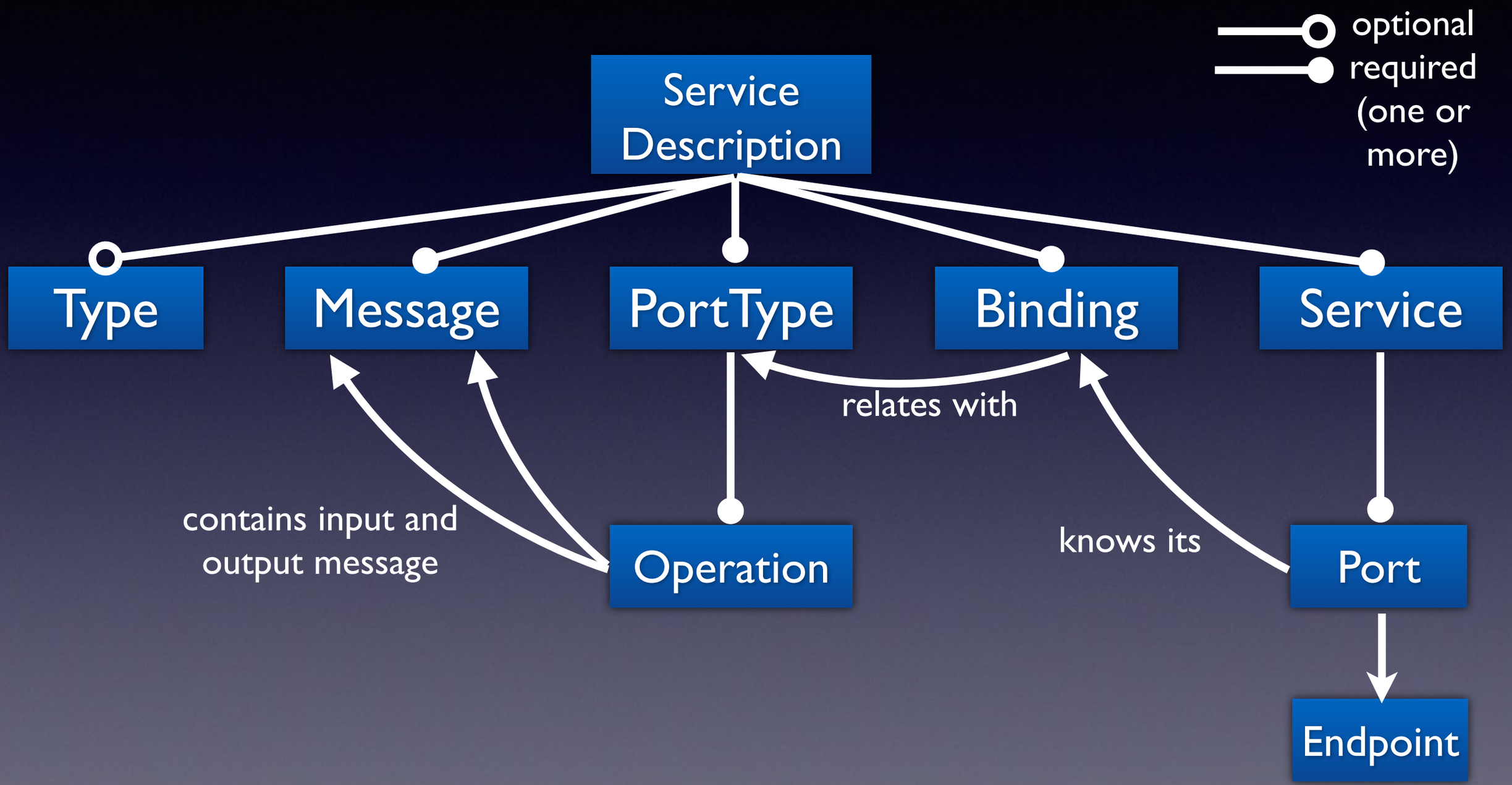
What Does WSDL Look Like?

```
<definitions name="EndorsementSearch"
  targetNamespace="http://namespaces.snowboard-info.com"
  xmlns:es="http://www.snowboard-info.com/EndorsementSearch.wsdl"
  xmlns:esxsd="http://schemas.snowboard-info.com/EndorsementSearch.xsd"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <!-- omitted types section with content model schema info -->
  <message name="GetEndorsingBoarderRequest">
    <part name="body" element="esxsd:GetEndorsingBoarder"/>
  </message>

  <message name="GetEndorsingBoarderResponse">
    <part name="body" element="esxsd:GetEndorsingBoarderResponse"/>
  </message>

  <portType name="GetEndorsingBoarderPortType">
    <operation name="GetEndorsingBoarder">
      <input message="es:GetEndorsingBoarderRequest"/>
      <output message="es:GetEndorsingBoarderResponse"/>
      <fault message="es:GetEndorsingBoarderFault"/>
    </operation>
  </portType>

  <binding name="EndorsementSearchSoapBinding"
           type="es:GetEndorsingBoarderPortType">
    <soap:binding style="document"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetEndorsingBoarder">
      <soap:operation
        soapAction="http://www.snowboard-info.com/EndorsementSearch"/>
      <input>
        <soap:body use="literal"
          namespace="http://schemas.snowboard-info.com/EndorsementSearch.xsd"/>
      </input>
      <output>
        <soap:body use="literal"
          namespace="http://schemas.snowboard-info.com/EndorsementSearch.xsd"/>
      </output>
      <fault>
        <soap:body use="literal"
          namespace="http://schemas.snowboard-info.com/EndorsementSearch.xsd"/>
      </fault>
    </operation>
  </binding>

  <service name="EndorsementSearchService">
    <documentation>snowboarding-info.com Endorsement Service</documentation>
    <port name="GetEndorsingBoarderPort"
          binding="es:EndorsementSearchSoapBinding">
      <soap:address location="http://www.snowboard-info.com/EndorsementSearch"/>
    </port>
  </service>
</definitions>
```

Your Eyes Hurt, aren't they?



What Does WSDL Look Like?



- Service: Contains a set of system functions that have been exposed to the Web-based protocols.
- Port: Defines the address or connection point to a Web service. It is typically represented by a simple HTTP URL string.
- Binding: Specifies the interface and defines the SOAP binding style.
- PortType: Defines a Web service, the operations that can be performed, and the messages that are used to perform the operation.

```
<wsdl:definitions name="PersonServiceImplService"
   targetNamespace="http://bbm490.hacettepe.edu.tr/">
   <wsdl:types>
      <xs:schema elementFormDefault="unqualified"
         targetNamespace="http://bbm490.hacettepe.edu.tr/" version="1.0">
         <xs:element name="getAll" type="tns:getAll" />
         <xs:element name="getAllResponse" type="tns:getAllResponse" />
         <xs:complexType name="getAll">
            <xs:sequence />
         </xs:complexType>
         <xs:complexType name="getAllResponse">
            <xs:sequence>
               <xs:element maxOccurs="unbounded" minOccurs="0" name="return"
                  type="tns:person" />
            </xs:sequence>
         </xs:complexType>
         <xs:complexType name="person">
            <xs:sequence>
               <xs:element minOccurs="0" name="age" type="xs:int" />
               <xs:element minOccurs="0" name="lastName" type="xs:string" />
               <xs:element minOccurs="0" name="name" type="xs:string" />
            </xs:sequence>
         </xs:complexType>
      </xs:schema>
   </wsdl:types>
```

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

```
 <wsdl:portType name="PersonService">
   <wsdl:operation name="getAll">
     <wsdl:input message="tns:getAll" name="getAll">
     </wsdl:input>
     <wsdl:output message="tns:getAllResponse"
name="getAllResponse">
     </wsdl:output>
   </wsdl:operation>
 </wsdl:portType>
```

```
<wsdl:binding name="PersonServiceImplServiceSoapBinding"
  type="tns:PersonService">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="getAll">
    <soap:operation soapAction="" style="document" />
    <wsdl:input name="getAll">
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output name="getAllResponse">
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

```
  <wsdl:service name="PersonServiceImplService">
    <wsdl:port binding="tns:PersonServiceImplServiceSoapBinding"
      name="PersonServiceImplPort">
      <soap:address
        location="http://localhost:8080/
SpringWebServicesProject/services/personService" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Code 1st vs. Contract 1st

- For generating a Web Service, 2 approaches exist.
- Code first:
  - Java Implementation first
    - WSDL generated automatically
- Contract first:
  - WSDL created first (manually)
  - Java Web Service implemented later

UDDI

- Universal Discovery and Directory Interface
- Started in early 2000s
- XML based Service Discovery Protocol.
- Think of it as yellow pages.
- Included in the WS-I standard.
- Not widely adopted as hoped, even Microsoft removed it from Windows Server OS.
- OSS implementation: http://juddi.apache.org

JAX-WS

- JAX-WS is Java API for XML Web Services.
- It provides ways to build web services and clients that communicate using XML.
- JAX-WS is the API and JAX-WS RI is the Reference Implementation.
- Apache CXF is compliant with JAX-RS and JAX-WS.
- We will create services using CXF framework.

Let's create a WS

- We'll create a standalone WS with JAX-WS, there will be no application server.

```
@WebService
public class StandAloneServiceImpl {

    @WebMethod
    public List<Country> getAll() {
        List<Country> countryList = new ArrayList<Country>();
        countryList.add(new Country(1, "TR"));
        countryList.add(new Country(2, "USA"));
        return countryList;
    }
}

public class Publisher {

    public static void main(String[] args) {
        Endpoint.publish("http://localhost:9999/ws/countryService", new
StandAloneServiceImpl());
    }
}
```

src:ws-standalone

WS annotations

- *@WebService*
  annotates classes as web services
- *@WebMethod*
  annotates methods web service operations
- *@WebParam*
  annotates web service parameters (optional)
- *@WebResult*
  annotates return values of web service annotations

Another WS impl.

```
@WebService
public class EchoWebService {

 @WebMethod
 public String echo() {
    return "echo";
 }

 @WebMethod
 @WebResult(partName = "return")
 public String echoName(@WebParam(name = "myname") String name) {
    return "echo " + name;
 }
}
```

Apache CXF

- Apache CXF is an open-source, fully featured Web services framework.
- It's integrated with Spring framework.
- It supports most of the web service standards.
- It supports Web Services and REST Services with JAX-WS and JAX-RS.

Let's create a Web Services Project with Apache CXF

- We'll create a Maven Web Application.
- We'll add Spring and CXF Framework dependencies.
- We'll create applicationContext.xml with CXF configuration.
- We'll modify the web.xml for CXFServlet definition.
- We'll create a domain object and Web Service class.

Project configurations

- CXF Framework dependencies

```
<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-frontend-jaxws</artifactId>
  <version>3.0.4</version>
</dependency>
<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-transports-http</artifactId>
  <version>3.0.4</version>
</dependency>
```

Project configurations

- applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jaxws="http://cxf.apache.org/jaxws"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans.xsd
          http://www.springframework.org/schema/context
          http://www.springframework.org/schema/context/spring-context.xsd
          http://cxf.apache.org/jaxws
          http://cxf.apache.org/schemas/jaxws.xsd">

    <import resource="classpath:META-INF/cxf/cxf.xml" />
    <import resource="classpath:META-INF/cxf/cxf-servlet.xml" />
```

Project configurations

- applicationContext.xml

```
<jaxws:endpoint id="personService"
implementor="tr.edu.hacettepe.bbm490.PersonServiceImpl" address="/
personService" />
```

- web.xml

```
    <servlet>
        <servlet-name>CXFServlet</servlet-name>
        <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
        <init-param>
            <param-name>hide-service-list-page</param-name>
            <param-value>false</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>CXFServlet</servlet-name>
        <url-pattern>/services/*</url-pattern>
    </servlet-mapping>
```

Project configurations

- Web Service interface definition

```
@WebService
public interface PersonService {

   @WebMethod
   List<Person> getAll();
}
```

- Web Service implementation

```
public class PersonServiceImpl implements PersonService {

   @Override
   public List<Person> getAll() {
      List<Person> personList = new ArrayList<Person>();
      …
      return personList;
   }
}
```

Person class

```java
@XmlRootElement
public class Person {

    private String name;
    private String lastname;

    public Person() {
    }

    public Person(String name, String lastname) {
        this.name = name;
        this.lastname = lastname;
    }

    @XmlAttribute
    public String getName() {
        return name;
    }

    @XmlElement
    public String getLastname() {
        return lastname;
    }
}
```

Person class

- JAXB — Java Architecture for XML Binding
- Allows Java developers to map Java classes to XML representations.
- Marshalling: Java to XML
- Unmarshalling: XML to Java
- Annotations are:
  - @XmlRootElement
  - @XmlElement
  - @XmlAttribute

So what about the RESTful services?

ReST

Representational
e
State
Transfer

- Resource Identification through URI
- Resource: Any "Thing", Anything you would want to point to.
- Representation: Serialisable description of resource could be images (jpg, png), XML or JSON data
- via HTTP Methods
  GET / PUT / POST / DELETE

ReST Example

Requirement: Give me the employee with id: 1234

Old School WS Method Invocation:

```
Employee employeeWS.getEmployee(Long id)
```

Cool ReST way:

```
GET /myApp/employees/1234   HTTP/1.1
```

ReST is just like CRUD

- You can expose your DB on WEB :)
- Simple & Stupid...

HTTP	SQL	Notation
POST	INSERT	Create
GET	SELECT	Read
PUT	UPDATE	Update
DELETE	DELETE	Delete

JAX-RS

- It's the API for creating REST services.
- It's so easy to create with annotations.
- REST is cool but,
  - It's not solving real world problems like Security, Transactions and etc. (WS-* are still in need)
  - Documentation is still needed for resources, representations... (no WSDL)

Let's create a REST service

- We'll use CXF Framework with Eclipse Kepler.
- First we need to add new Maven dependencies

```
<dependency>
   <groupId>org.apache.cxf</groupId>
   <artifactId>cxf-rt-frontend-jaxrs</artifactId>
   <version>3.0.4</version>
</dependency>
<dependency>
   <groupId>org.codehaus.jackson</groupId>
   <artifactId>jackson-jaxrs</artifactId>
   <version>1.9.0</version>
</dependency>
```

Project Configurations

```
<servlet>
  <servlet-name>CXFServlet</servlet-name>
  <servlet-class>
      org.apache.cxf.transport.servlet.CXFServlet
  </servlet-class>
  <init-param>
    <param-name>hide-service-list-page</param-name>
    <param-value>false</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>CXFServlet</servlet-name>
  <url-pattern>/services/*</url-pattern>
</servlet-mapping>
```

Project Configurations

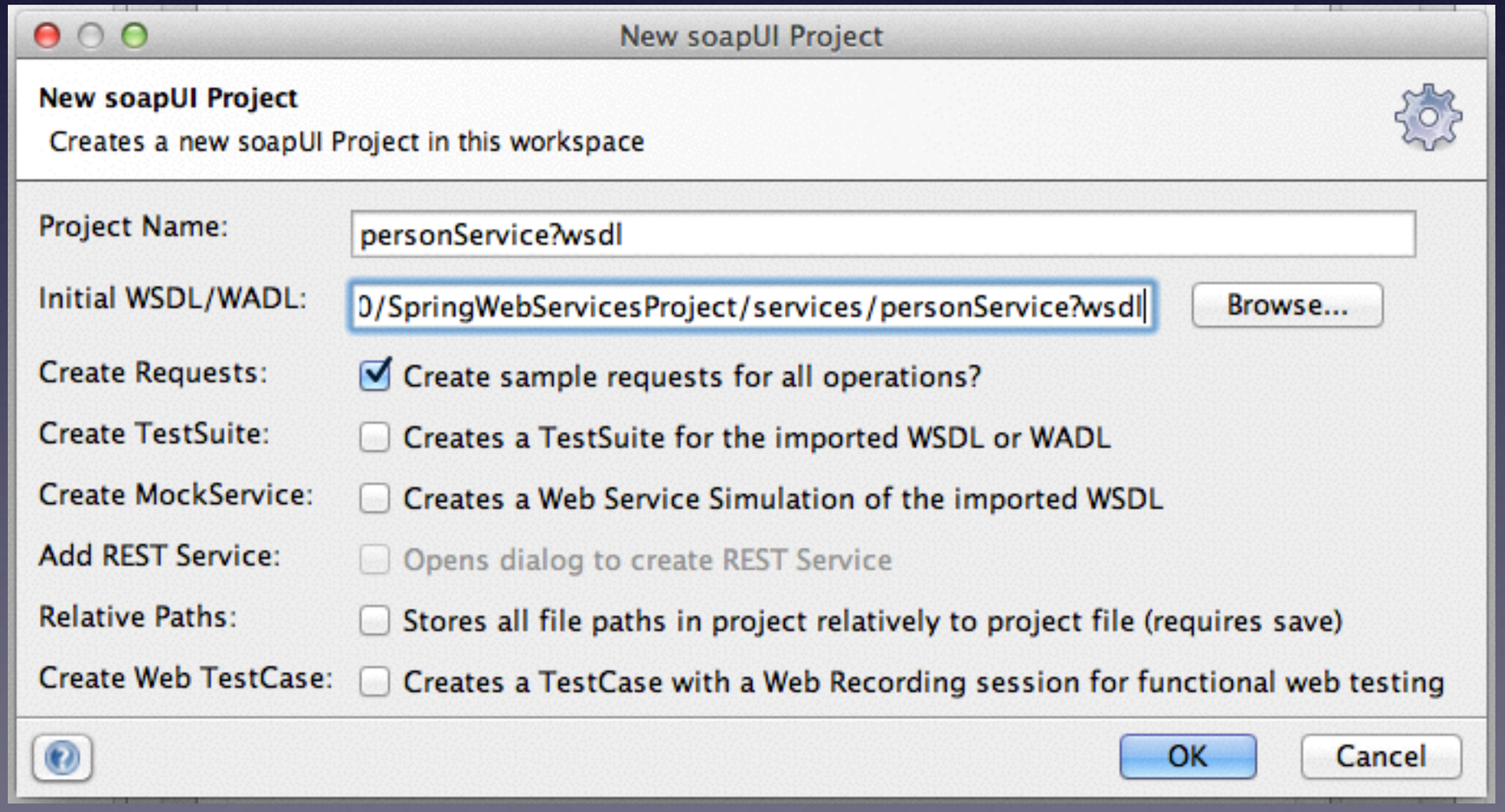
```
<jaxrs:server id="userRestService" address="/rest">
   <jaxrs:serviceBeans>
      <ref bean="userService" />
   </jaxrs:serviceBeans>
   <jaxrs:providers>
      <ref bean="jsonProvider" />
   </jaxrs:providers>
</jaxrs:server>

<bean id="jsonProvider"
class="org.codehaus.jackson.jaxrs.JacksonJsonProvider" />
```

Project Configurations

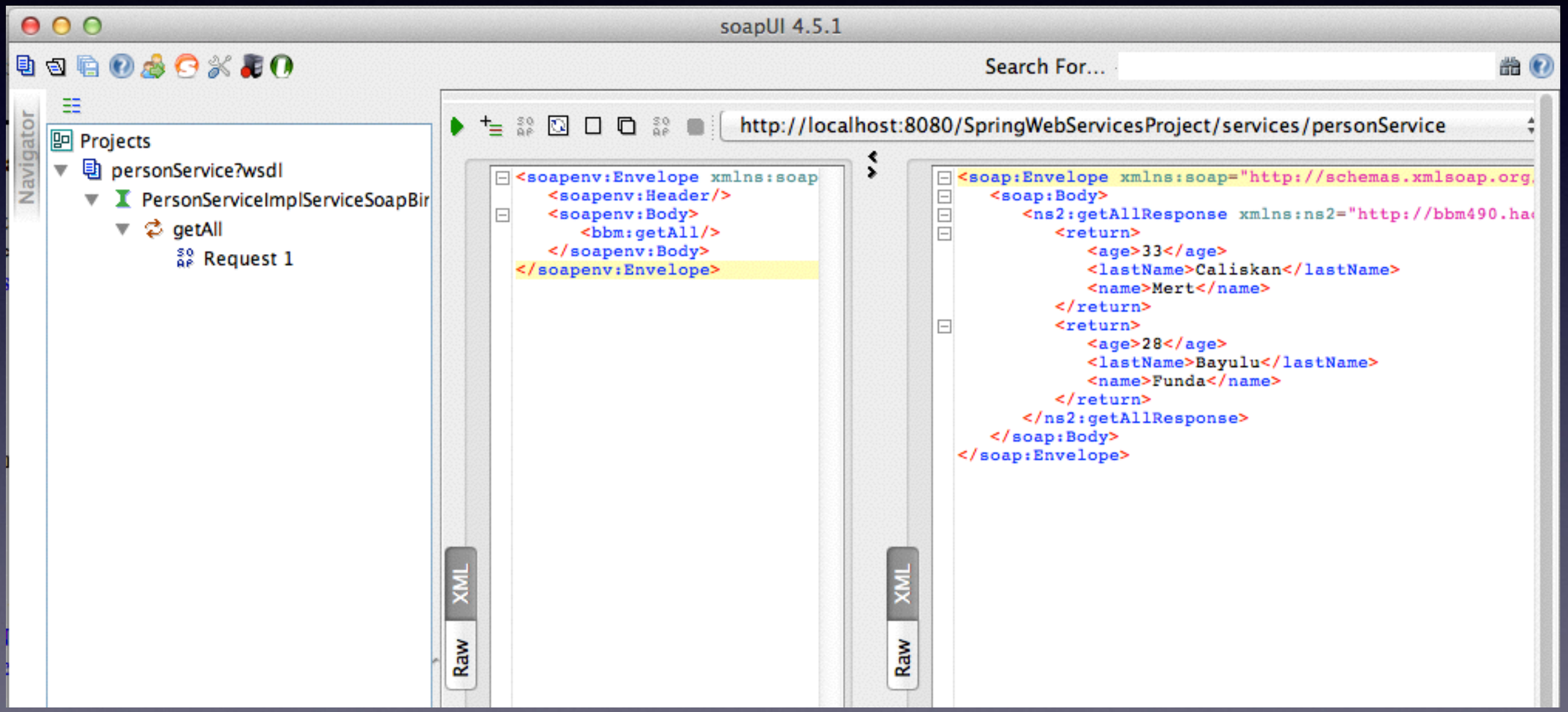
```
@Service
@Path("/userService/")
@Produces("application/json")
public class UserService {

    private static Map<Integer, User> users = new HashMap<Integer, User>();

    static {
        users.put(1, new User(1, "John"));
        users.put(2, new User(2, "Mike"));
        users.put(3, new User(3, "Mary"));
    }

    @GET
    @Path("/users")
    public List<User> getUsers() {
        return new ArrayList<>(users.values());
    }

    @GET
    @Path("/user/{id}")
    public User getUser(@PathParam("id") Integer id) {
        return users.get(id);
    }
}
```

soapUI

- SoapUI is an open source web service testing application.
- It can be downloadable at http://www.soapui.org.



soapUI


soapUI 4.5.1
Search For...
Navigator
Projects
personService?wsdl
PersonServiceImplServiceSoapBir
getAll
Request 1
http://localhost:8080/SpringWebServicesProject/services/personService
<soapenv:Envelope xmlns:soap
<soapenv:Header/>
<soapenv:Body>
<bbm:getAll/>
</soapenv:Body>
</soapenv:Envelope>
XML
Raw
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org
<soap:Body>
<ns2:getAllResponse xmlns:ns2="http://bbm490.ha
<return>
<age>33</age>
<lastName>Caliskan</lastName>
<name>Mert</name>
</return>
<return>
<age>28</age>
<lastName>Bayulu</lastName>
<name>Funda</name>
</return>
</ns2:getAllResponse>
</soap:Body>
</soap:Envelope>
XML
Raw

SOA
Service Oriented Architecture

Not SOAP...!

Ok, maybe it is...

WHAT SOA IS ?

- Stands for Service Oriented Architecture
- Software Architecture where the whole information system is regarded as a collection of services interoperated and used freely through a common interface (the service bus) independent of hardware.

WHAT SOA IS NOT ?

- It's NOT a technical standard
- It's NOT a technology
- It's NOT a concrete architecture,
  but it leads to one.

- An application that uses web services is SOA.
  This is Wrong…!

AIM OF SOA ?

- Too many aims for sure but,
- To INTEGRATE..!
  - Services should interact, they must exchange information...
- To make you obey the Architectural Principles...!

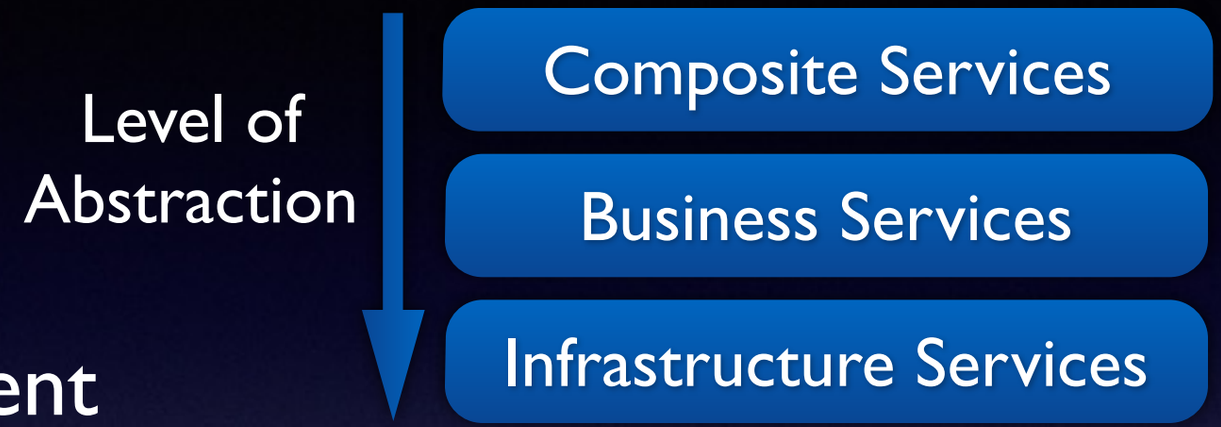
ARCHITECTURAL PRINCIPLES

- Loose Coupling
  Minimise dependencies between each other, just aware of each other...

- Service Contract
  A set of technical data along with business information

ARCHITECTURAL PRINCIPLES

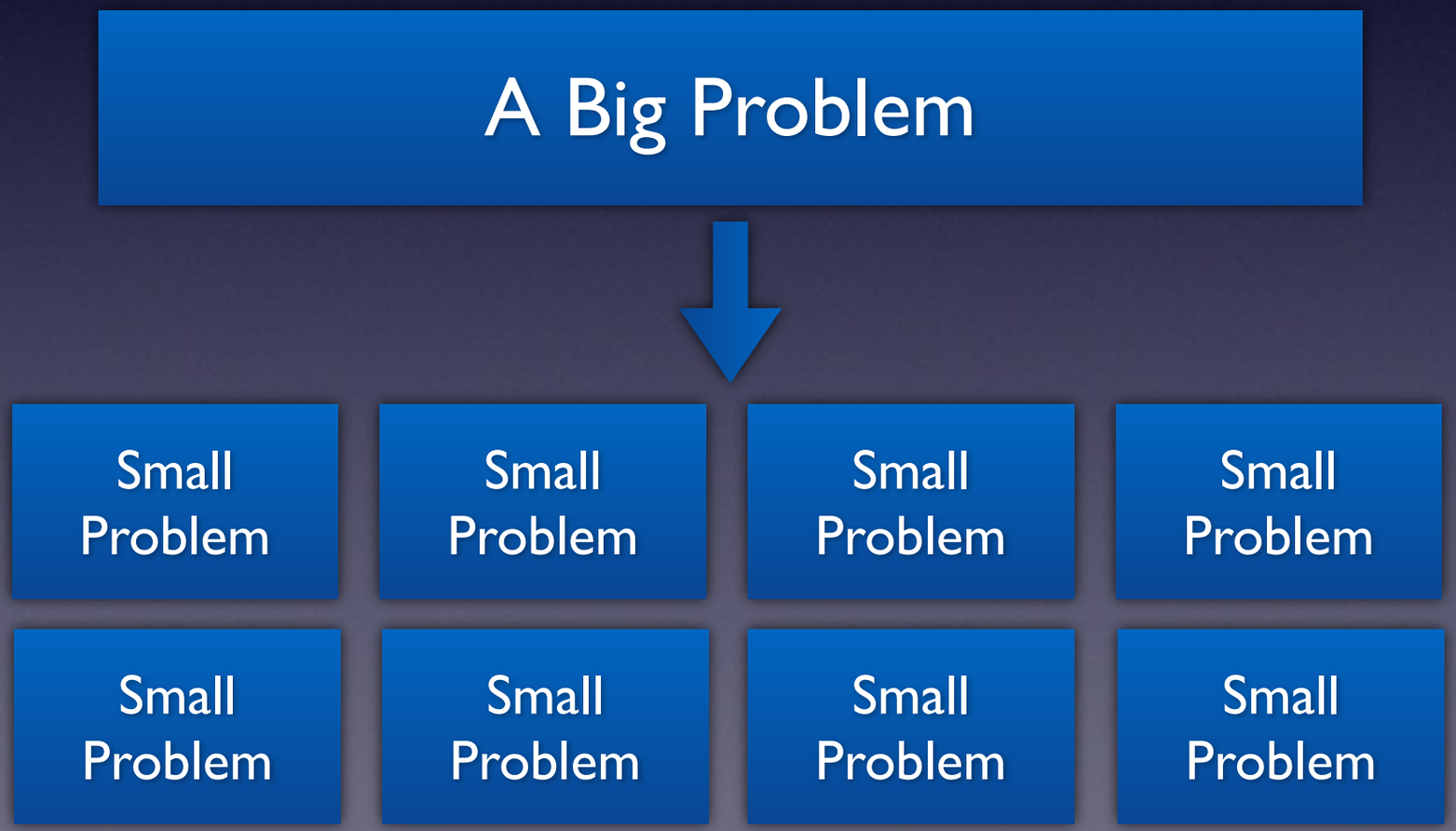
- Service Abstraction
  Service Blocks on top of each other to build Service Oriented Environment



- Service Reusability
  Positioning services as reusable enterprise resources

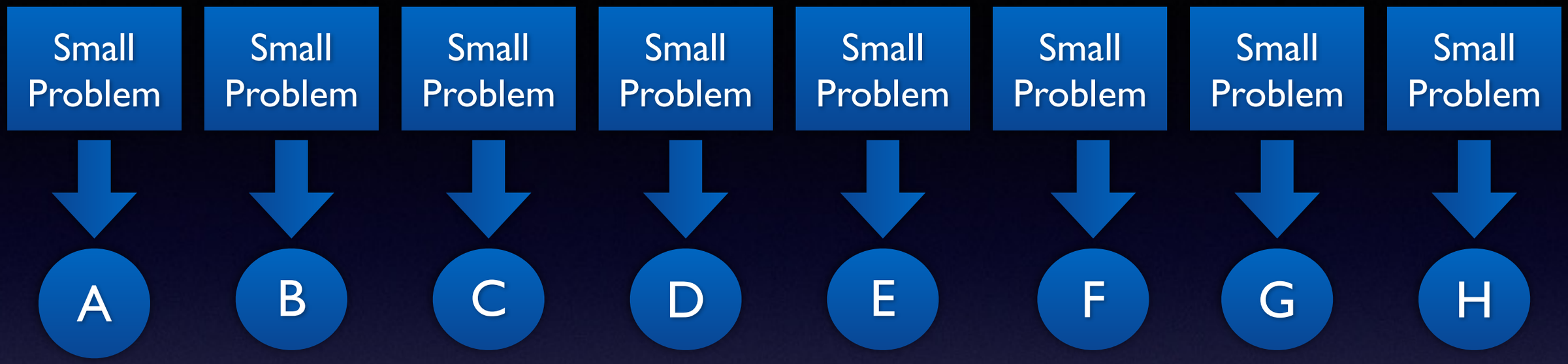
ARCHITECTURAL PRINCIPLES

- Service Composability
  To re-use the capability of a service to build up composed services

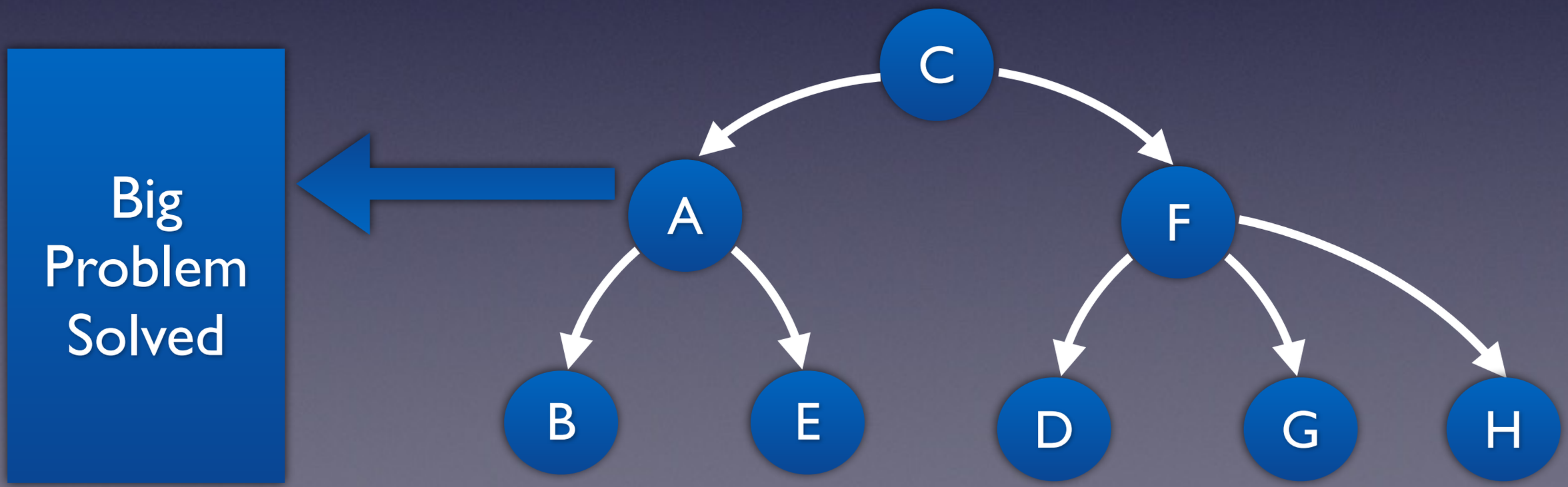


Small problems collectively represent a big problem

Service Composability



Solution Logic that each address to solve a problem



Service Discoverability

- Descriptive services that can be accessed via discovery mechanisms. (i.e. UDDI)
- Increase the service re-use...
- Services defined with WSDL and registered to services registry.