Advanced Programming 2 - Web Services

DR. ELIAHU KHALASTCHI

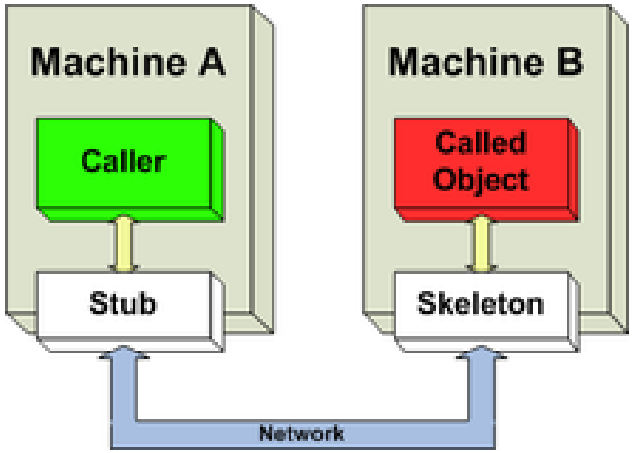
2016

Services Oriented Architecture

SOA

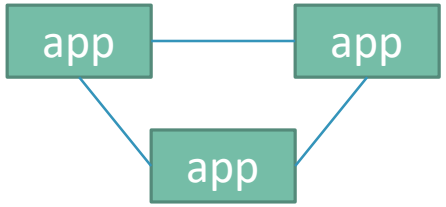
Evolution of SOA

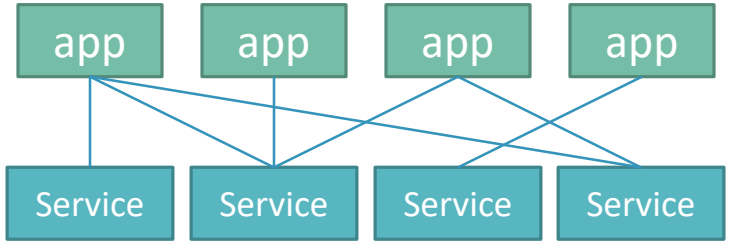
Monolithic Design (60s) → Object Oriented Design (80s) → Client-Server (two-tier) → Distributed Object Design (multi-tier)



Evolution of SOA

Monolithic Design (60s) → Object Oriented Design (80s) → Client-Server (two-tier) → Distributed Object Design (multi-tier) → Component Object Model Design

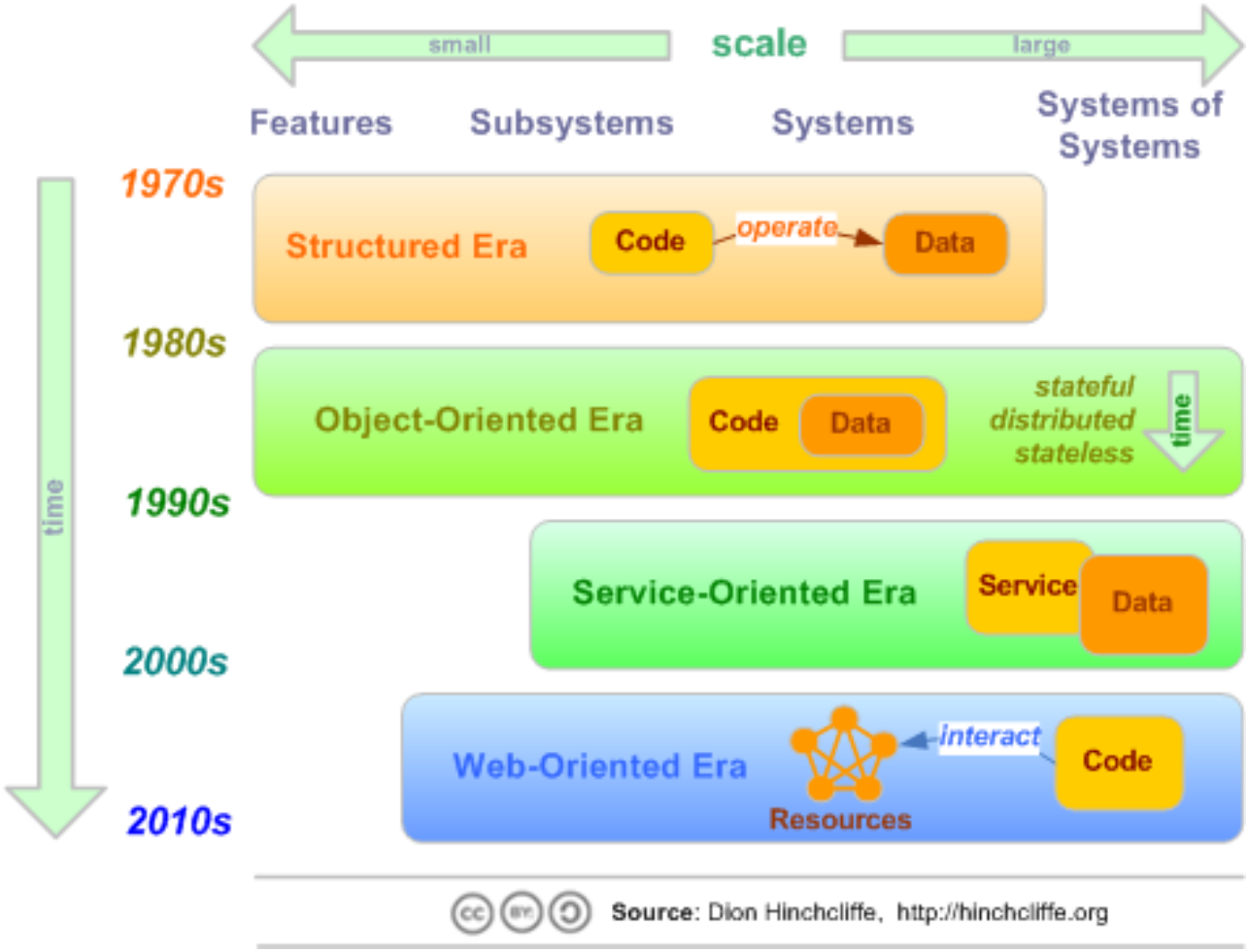




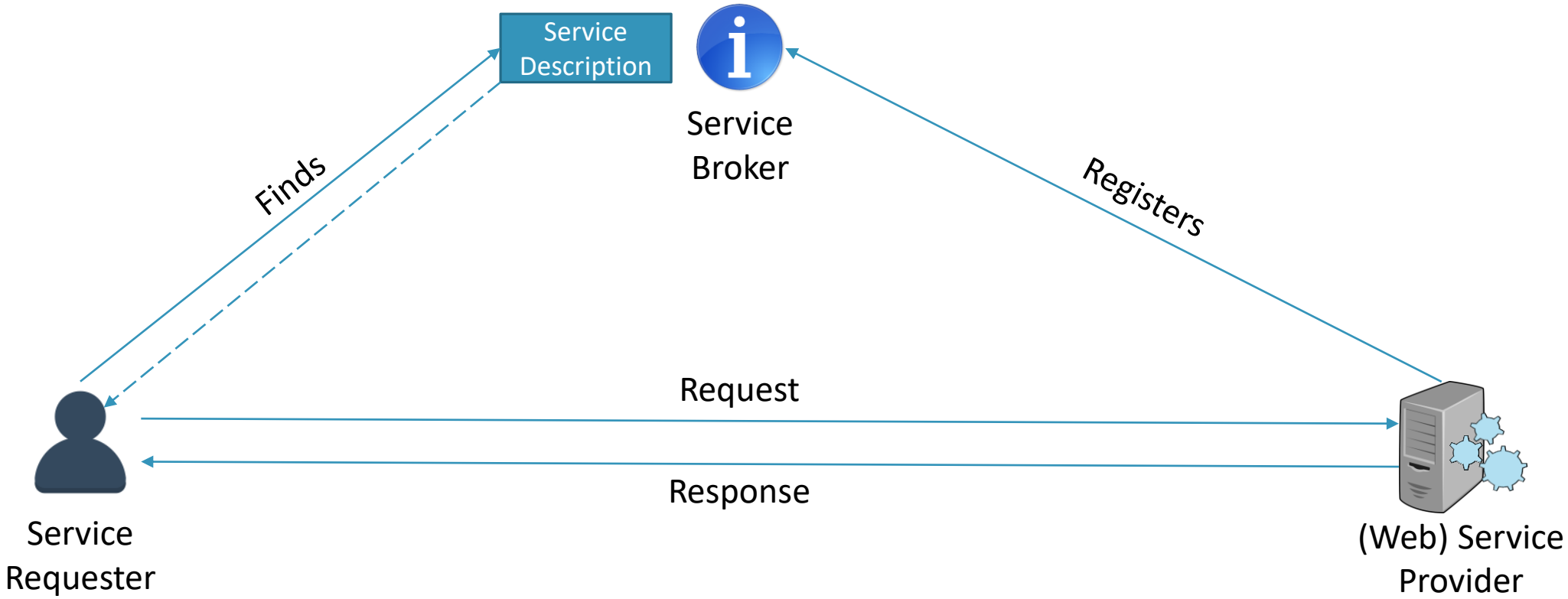
SOA – Service Oriented Architecture (2000)

WOA – Web Oriented Architecture

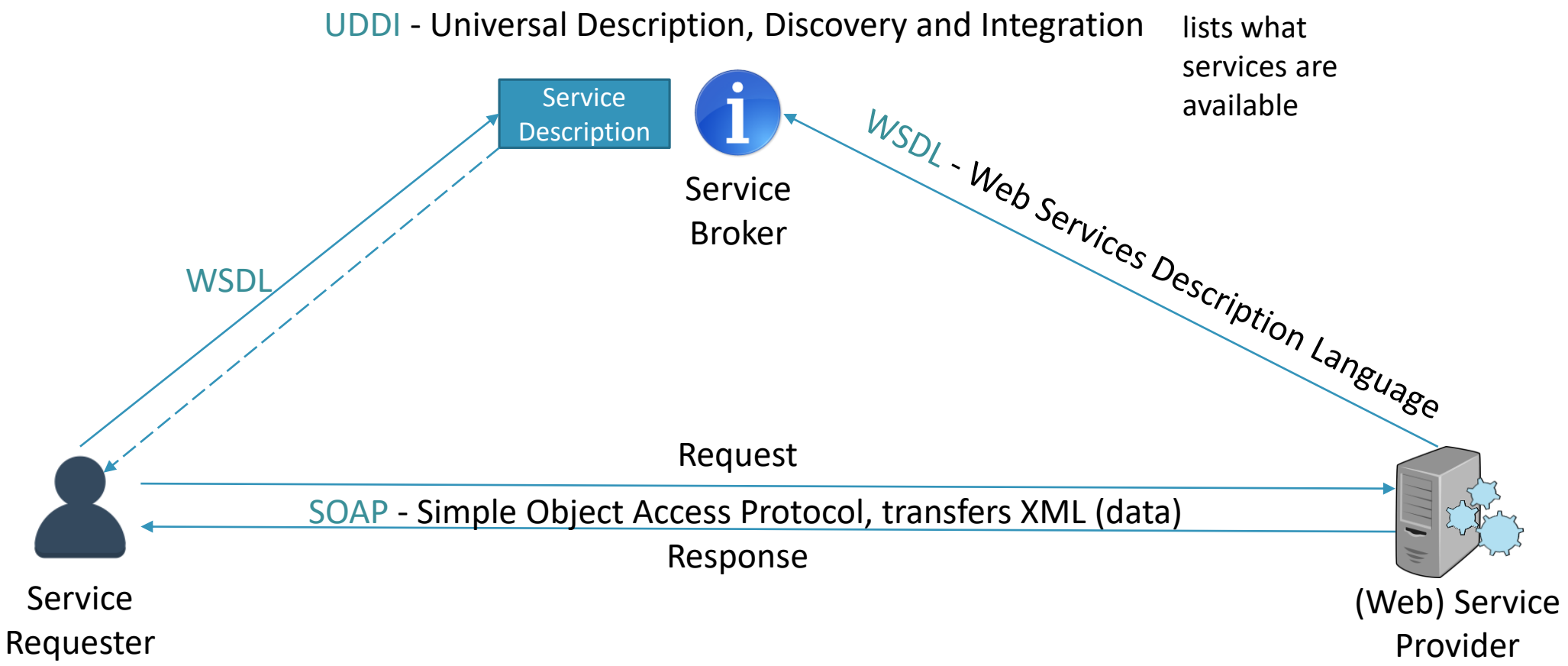
Popular Models for Developing and Integrating Software - 1970s to Now



Architecture and Terminology



Architecture and Terminology



SOA – Service Oriented Architecture

Service – a well defined function that does not depend on the state of other services

Web Service Example

JAVA

Web Service Interface & Implementation

```java
package com.myServices;

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

@WebService
@SOAPBinding(style=Style.RPC)
public interface MazeGeneratorService {
   public int[][] generate(int rows,int cols);
}
```

Remote Procedure Call (RPC)
Document

```java
@WebService(endpointInterface="com.myServices.MazeGeneratorService")
@SOAPBinding(style=Style.RPC)
public class MyMazeGeneratorService implements MazeGeneratorService{
 @Override
 public int[][] generate(int rows, int cols) {
  int[][] maze=new int[rows][cols];
  //...
  return maze;
 }
}
```

Publishing the Service

```
import javax.xml.ws.Endpoint;

public class Publisher {

  public static void main(String[] args) {
    Endpoint endpoint=Endpoint.publish("http://localhost:8080/maze",
                                       new MyMazeGeneratorService());
    System.out.println(endpoint.isPublished()); // true
  }
}
```

Check the published WSDL

```
<definitions>

 <types>
   data type definitions..
 </types>

 <message>
  definition of the data being communicated...
 </message>

 <portType>
   set of operations...
 </portType>

 <binding>
   protocol and data format specification....
 </binding>

</definitions>
```

localhost:8080/maze?wsdl

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▶<!--...-->
▶<!--...-->
▼<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="h
 xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns=
 targetNamespace="http://myServices.com/" name="MyMazeGeneratorServiceService">
  ▼<types>
    ▼<xsd:schema>
       <xsd:import namespace="http://jaxb.dev.java.net/array" schemaLocation="http://localhost:8080/maze?xsd=1"/>
     </xsd:schema>
   </types>
  ▼<message name="generate">
     <part name="arg0" type="xsd:int"/>
     <part name="arg1" type="xsd:int"/>
   </message>
  ▼<message name="generateResponse">
     <part xmlns:ns1="http://jaxb.dev.java.net/array" name="return" type="ns1:intArrayArray"/>
   </message>
  ▼<portType name="MazeGeneratorService">
    ▼<operation name="generate" parameterOrder="arg0 arg1">
       <input wsam:Action="http://myServices.com/MazeGeneratorService/generateRequest" message="tns:generate"/>
       <output wsam:Action="http://myServices.com/MazeGeneratorService/generateResponse" message="tns:generateResponse"/>
     </operation>
   </portType>
  ▼<binding name="MyMazeGeneratorServicePortBinding" type="tns:MazeGeneratorService">
     <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    ▼<operation name="generate">
       <soap:operation soapAction=""/>
      ▼<input>
         <soap:body use="literal" namespace="http://myServices.com/"/>
       </input>
      ▼<output>
         <soap:body use="literal" namespace="http://myServices.com/"/>
       </output>
     </operation>
   </binding>
  ▼<service name="MyMazeGeneratorServiceService">
    ▼<port name="MyMazeGeneratorServicePort" binding="tns:MyMazeGeneratorServicePortBinding">
       <soap:address location="http://localhost:8080/maze"/>
     </port>
   </service>
 </definitions>
```

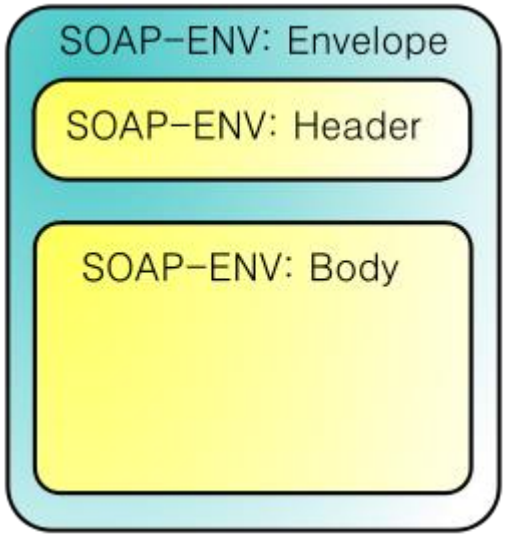
Service Requester

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

public class MazeRequester {
 public static void main(String[] args) throws Exception{
   URL url=new URL("http://localhost:8080/maze?wsdl");
   QName qName = new QName("http://myServices.com/","MyMazeGeneratorServiceService");
   Service service=Service.create(url,qName);
   MazeGeneratorService mazeGenerator=service.getPort(MazeGeneratorService.class);
   int[][] maze = mazeGenerator.generate(10, 10);

   for(int i=0;i<maze.length;i++){
      for(int j=0;j<maze[i].length;j++)
         System.out.print(maze[i][j]);
      System.out.println();
   }
 }
}
```

```
1011111111
1000100011
1011111011
1000000011
1111010111
1000010001
1111011011
1000010011
1101010011
1111111011
```

SOAP – Simple Object Access Protocol

- An XML structured communication protocol over HTTP
- + Instructions are decupled from the operating system / application




```
public int[][] generate(int rows, int cols);
```

o.generate(10, 5);

Service Requester

```
<soap:envelope>
   <soap:body>
      <generate>
         <rows>10</rows>
         <cols>5</cols>
      </generate >
   </soap:body>
</soap:envelope>
```

(Web) Service Provider

Representational State Transfer

REST

REST – Representational State Transfer

- We have resources (documents, services, etc.)
- Each resource has a Representation
- e.g., http://he.wikipedia.org/wiki/REST
- Which presents its State,
- and links that Transfer the client onto another resource

- *Roy Fielding

REST

- Uses HTTP
  - GET, POST, PUT, DELETE

- Architectural Style
  - Recognize your resources and give them URIs (Uniform Resource Identifier)
  - Decide if the client affects the state or just views it (GET vs. POST)
  - A resource accessed by GET should not change its state
  - Place links to other representations
  - Information should be provided gradually

REST Benefits

- Exploits the benefits of the HTTP architecture – the architecture of the web
  - Request-response mechanism
  - Layered intermediates – proxies, gateways, caching
- Resource distribution = users requests distribution
- No need for envelops
- Efficiency
- Scalability
- User perceived performance

SOAP

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:body pb="http://www.acme.com/phonebook">
  <pb:GetUserDetails>
   <pb:UserID>12345</pb:UserID>
  </pb:GetUserDetails>
 </soap:Body>
</soap:Envelope>
```

REST

```
http://www.acme.com/phonebook/UserDetails/12345
```

REST Examples

JAVA + JERSEY

A Hello World! Resource

http://localhost:8080/RESTexample/helloWorld

```
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;
// The Java class will be hosted at the URI path "/helloWorld"
@Path("/helloWorld")
public class HelloWorldResource {
    // The Java method will process HTTP GET requests
    @GET
    // The Java method will produce content identified by the MIME Media
    // type "text/plain"
    @Produces("text/plain")
    public String sayHello() {
      return "Hello World"; // Return some textual content
    }
}
```

Using Path Parameters

http://localhost:8080/RESTexample/users/Eli

```
@Path("/users/{username}")
public class UserResource {
    @GET
    @Produces("text/html")
    public String getUser(@PathParam("username") String userName) {
   return  "<html><body>"+
           "<h1>Welcome "+userName+"</h1>"+
           "</body></html>";
    }
}
```

Using Query Parameters

```
@Path("/table")
public class SizeResource {
    @GET
    @Produces("text/html")
    public String getSize(@QueryParam("size") @DefaultValue("50") int size,
                          @QueryParam("n") @DefaultValue("10") int n){
      String ret="<html><body><table width="+size+"% border=2>";
      for(int i=0;i<n;i++){
        ret+="<tr>";
        for(int j=0;j<n;j++)
          ret+="<td>"+(i+1)*(j+1)+"</td>";
        ret+="</tr>";
      }
      return  ret+"</table></body></html>";
    }
}
```

http://localhost:8080/RESTexample/table?size=40&n=10

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Form Processing

HTML

```html
<form action="http://localhost:8080/RESTexample/form" method="post">
 <input name="name" type="text">
 <input type="submit" value="send">
</form>
```

Servlet code

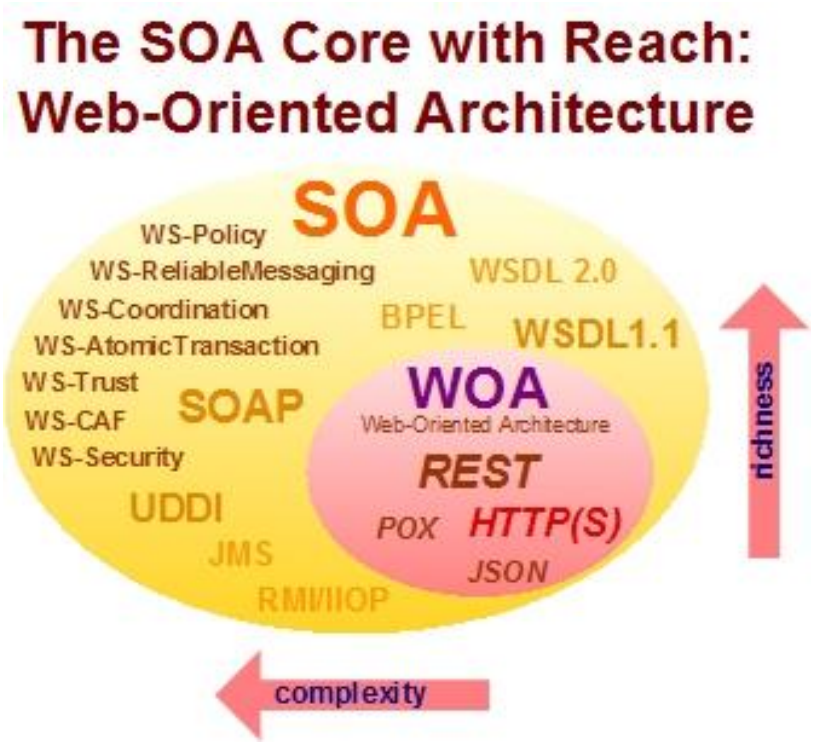
```java
@Path("/form")
public class FormResource {
    @POST
    @Consumes("application/x-www-form-urlencoded")
    @Produces("text/plain")
    public String post(@FormParam("name") String name) {
     // save the name
     return "welcome "+name;
    }
}
```

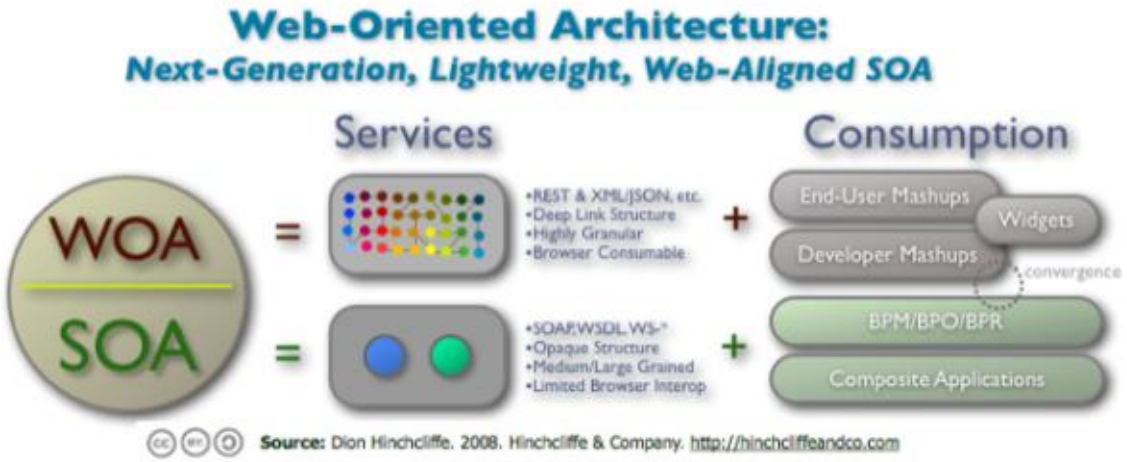
REST vs. SOAP

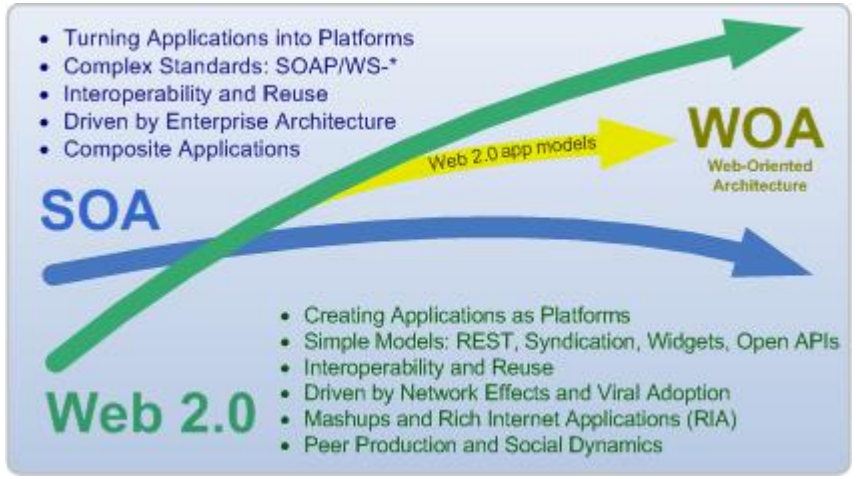
REST vs. SOAP

REST	SOAP
▪Architecture	▪Protocol
▪Exposes resources which represents data	▪Exposes operations which represent logic
▪Uses HTTP commands (GET / POST/ DELETE)	▪Uses HTTP POST
▪Simple point-to-point communication	▪Loosely coupled distributed messaging
▪Supports multiple data formats (XML, JSON,…)	▪Supports only XML
▪Stateless communication	▪Stateless or stateful (conversational) operations

WOA vs. SOA


The SOA Core with Reach:
Web-Oriented Architecture
SOA
WS-Policy
WS-ReliableMessaging
WSDL 2.0
WS-Coordination
BPEL
WSDL1.1
WS-AtomicTransaction
WS-Trust
WOA
Web-Oriented Architecture
SOAP
WS-CAF
WS-Security
REST
UDDI
POX
HTTP(S)
JMS
JSON
RMI/IIOP
richness
complexity


Web-Oriented Architecture:
Next-Generation, Lightweight, Web-Aligned SOA
Services
Consumption
WOA
SOA
•REST & XML/JSON, etc.
•Deep Link Structure
•Highly Granular
•Browser Consumable
•SOAP, WSDL, WS-*
•Opaque Structure
•Medium/Large Grained
•Limited Browser Interop
End-User Mashups
Widgets
Developer Mashups
convergence
BPM/BPO/BPR
Composite Applications
Source: Dion Hinchcliffe. 2008. Hinchcliffe & Company. http://hinchcliffeandco.com


• Turning Applications into Platforms
• Complex Standards: SOAP/WS-*
• Interoperability and Reuse
• Driven by Enterprise Architecture
• Composite Applications
Web 2.0 app models
WOA
Web-Oriented Architecture
SOA
• Creating Applications as Platforms
• Simple Models: REST, Syndication, Widgets, Open APIs
• Interoperability and Reuse
• Driven by Network Effects and Viral Adoption
• Mashups and Rich Internet Applications (RIA)
• Peer Production and Social Dynamics
Web 2.0