



July 2019  
IPT Training XML Technologies

# XML Web Services: SOAP, WSDL, REST, JSON

Trayan Iliev  
[tiliev@ipproduct.org](mailto:tiliev@ipproduct.org)  
<http://ipproduct.org>

Copyright © 2003-2019 IPT - Intellectual  
Products & Technologies

# За лектора



## Trayan Iliev

- CEO of IPT – Intellectual Products & Technologies (<http://ipproduct.org/>)
- Oracle® certified programmer 15+ Y
- end-to-end reactive fullstack apps with Java, ES6/7, TypeScript, Angular, React and Vue.js
- SOA & BPM with XML, WSDL, WS-\*, BPEL, BPMN, UML
- 12+ years IT trainer
- Voxxed Days, jPrime, jProfessionals, BGOUG, BGJUG, DEV.BG speaker
- Organizer RoboLearn hackathons and IoT enthusiast (<http://robolearn.org>)

# Agenda (1)

1. *JavaScript* a multi-paradigm, fullstack application development language of the Web. Versions. Main features
2. *VS Code* and *VS Code* extensions. Linting with *ESLint*
3. Running and debugging programs in browser and *NodeJS*
4. *JavaScript* basic language constructs and data types
5. Object-oriented *JavaScript* – object literals, *new* with constructors, prototypes, *Object.create()*, using ***this***.
6. Defining, enumerating and deleting properties
7. *JavaScript Object Notation (JSON)*
8. Prototypal inheritance, polymorphism and method overriding, classes and constructors, classical inheritance, *instanceof*

# Agenda (2)

9. Arrays - creating, reading, writing, adding and deleting array elements, array length, sparse arrays. Iterating arrays.
10. Array methods – *join()*, *concat()*, *slice()*, *splice()*, *push()*, *pop()*, *shift()*, *unshift()*, *forEach()*, *map()*, *filter()*, *every()*, *some()*, *reduce()*, *reduceRight()*, *indexOf()*, *lastIndexOf()*. Array-like obj.
11. Function declaration and expressions. Invoking functions. Self-invoking functions. Anonymous functions.
12. Function arguments – passing by value and by reference. Default values. Functions as values.
13. Using *call()*, *apply()*, *bind()*. Closures and callbacks.
14. Functions as namespaces – *IIFE* and *Module* design pattern.
15. Novelties in ES 6-9: classes, lambdas, Promises, async, etc.



# Where is The Code?

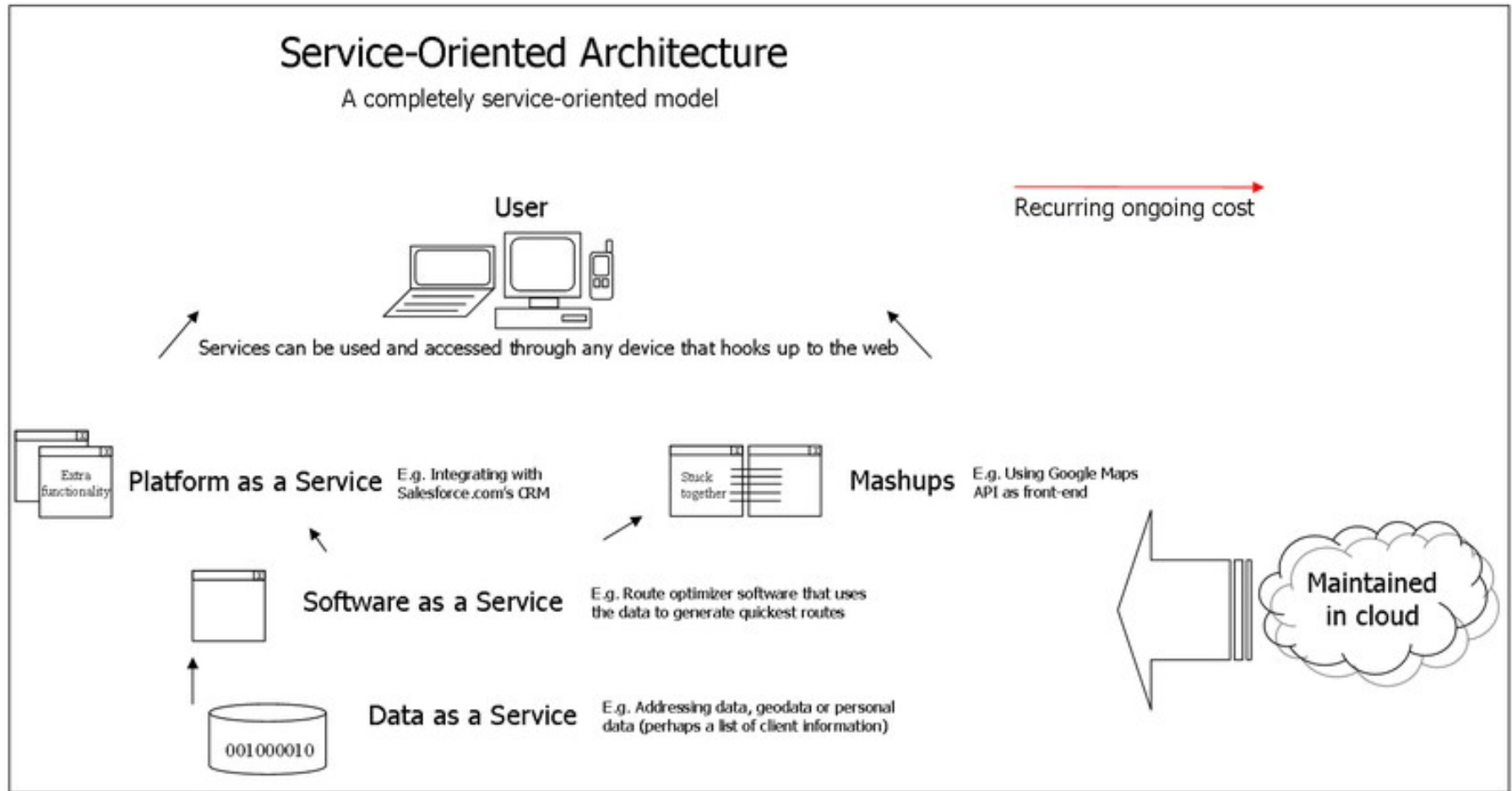
**XML Technologies** – projects and examples  
@ GitHub:

<https://github.com/iproduct/course-xml>

# Agenda

- Service Oriented Architecture (SOA) & REST
- Web Standards and Protocols (HTTP, URI, MIME, HTML, XML, JSON)
- REpresentational State Transfer (REST) architectural style – advantages and main constraints
- RESTful services + JSON – lightweight and efficient way for building platform independent and loosely-coupled applications
- Java API for RESTful Web Services – JAX-RS
- Hypermedia As The Engine Of Application State (HATEOAS)
- Let's try it (IPT Course Manager demo explained)

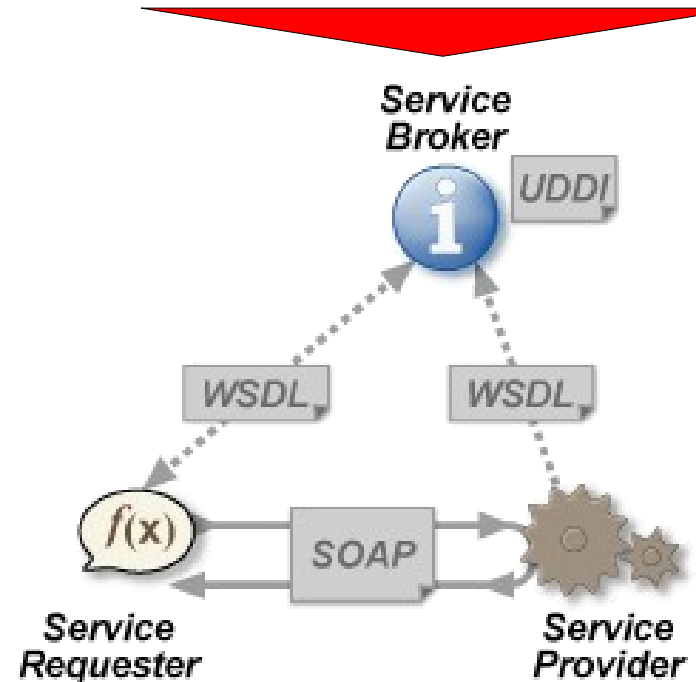
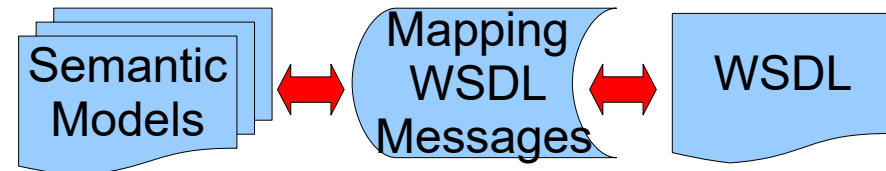
# Service Oriented Architecture (SOA)



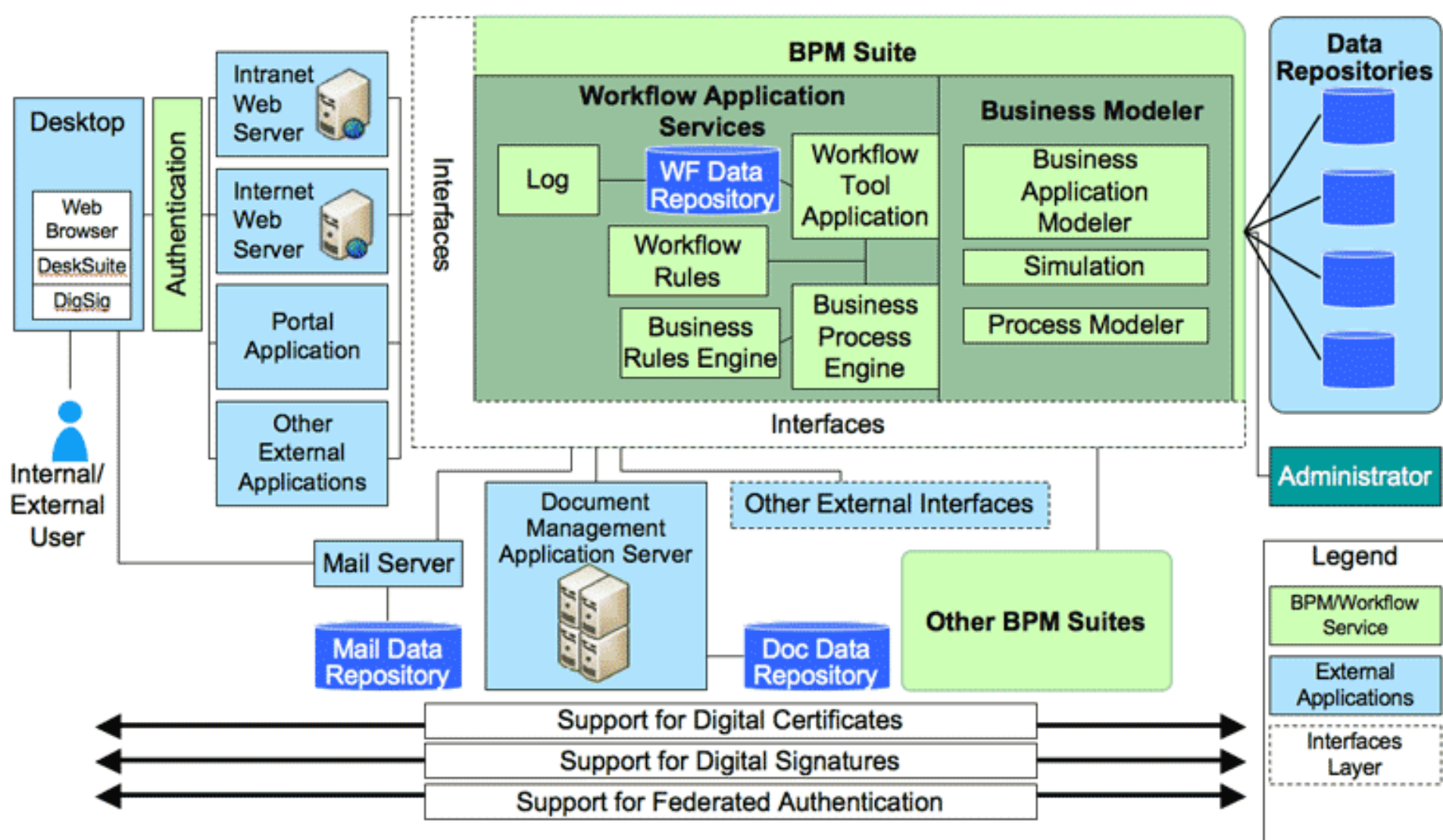
# SOA == Web Services ?

Web Services are:

- components for building distributed applications in SOA architectural style
- communicate using open protocols
- are self-descriptive and self-content
- can be searched and found using UDDI or ebXML registries (and more recent specifications – WSIL & **Semantic Web Services**)







Sample business portal architecture that demonstrates how using Business Process Modeling (BPM) instruments can be built new executable business processes through orchestration & choreography of both human and software automated activities

**Source: National Institute of Health (2007). Business Process Management (BPM) Service Pattern – <http://enterprisearchitecture.nih.gov/ArchLib/AT/TA/WorkflowServicePattern.htm>**

# Service Oriented Architecture (SOA) - Definitions

- **OASIS** Reference Model for Service Oriented Architecture 1.0 Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.
- **Wikipedia:** Service-oriented architecture (SOA) is a flexible set of design principles used during the phases of systems development and integration in computing. A system based on a SOA will package functionality as a suite of interoperable services that can be used within multiple, separate systems from several business domains.

# Service Oriented Architecture (SOA) – Definitions

**Webopedia:** Service-oriented architecture (SOA) – an application architecture in which all functions, or services, are defined using a description language and have invocable interfaces that are called to perform business processes. Each interaction is independent of each and every other interaction and the interconnect protocols of the communicating devices (i.e., the infrastructure components that determine the communication system do not affect the interfaces). Because interfaces are platform-independent, a client from any device using any operating system in any language can use the service.

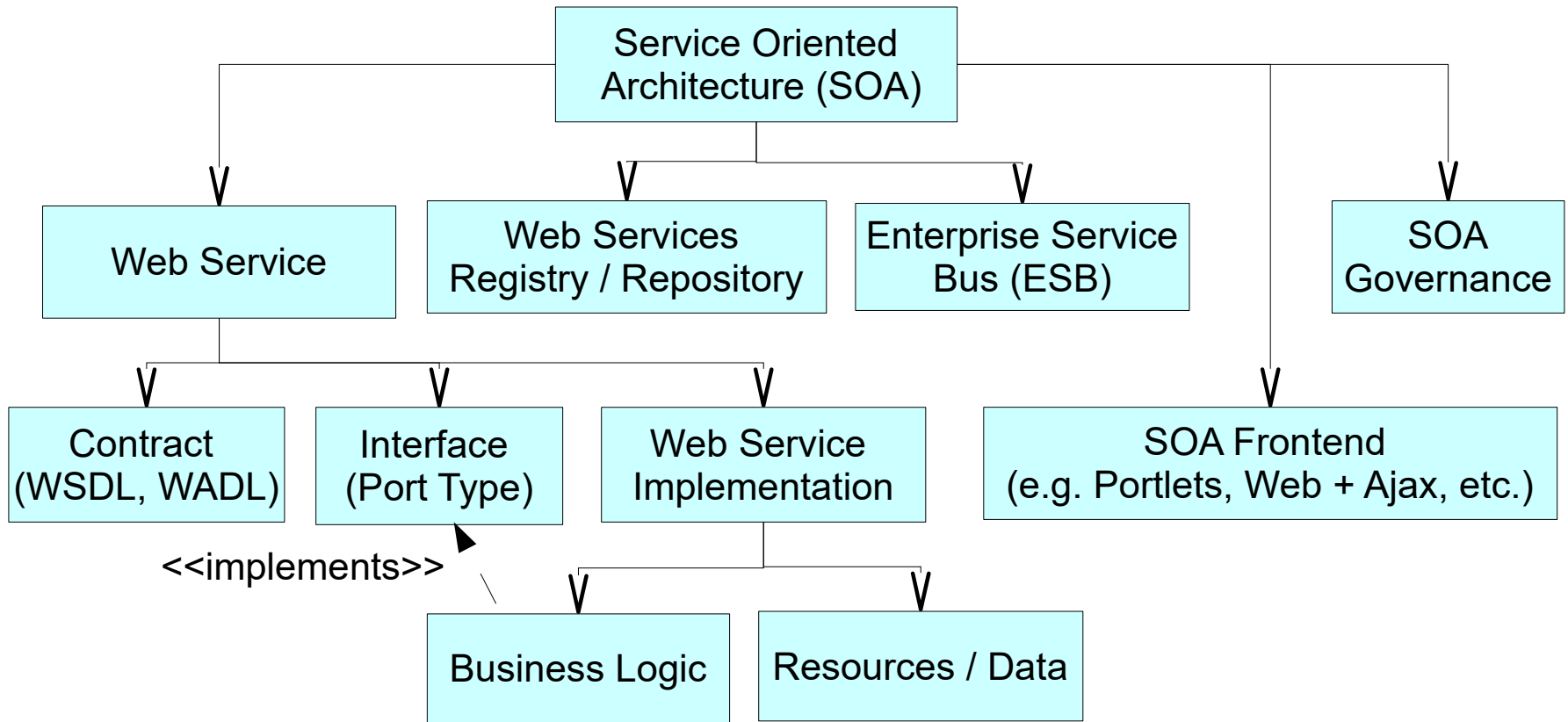
Though built on similar principles, SOA is not the same as Web services, which indicates a collection of technologies, such as SOAP and XML. SOA is more than a set of technologies and runs independent of any specific technologies.

# Service Oriented Architecture (SOA) – Definitions

**Thomas Erl:** SOA represents an open, agile, extensible, federated, composable architecture comprised of autonomous, QoS-capable, vendor diverse, interoperable, discoverable, and potentially reusable services, implemented as Web services. SOA can establish an abstraction of business logic and technology, resulting in a loose coupling between these domains. SOA is an evolution of past platforms, preserving successful characteristics of traditional architectures, and bringing with it distinct principles that foster service-orientation in support of a service-oriented enterprise. SOA is ideally standardized throughout an enterprise, but achieving this state requires a planned transition and the support of a still evolving technology set

References: Erl, Thomas. [Serviceorientation.org](http://Serviceorientation.org) – About the Principles, 2005–06

# SOA – Main Concepts



By idea from: Dirk Krafzig, Karl Banke, and Dirk Slama. Enterprise SOA. Prentice Hall, 2005



# Web Services Support

- XML -based web services:
  - Simple Object Access Protocol (SOAP)
    - XML-based envelope
    - XML-based encoding rules
    - XML-based request and response convention
  - Web Services Description Language (WSDL)
  - Universal Description, Discovery and Integration (UDDI) and ebXML Registries integration

# Example 1: SOAP (1)

## ❖ SOAP Request:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
```

```
  <S:Header/>
```

```
  <S:Body>
```

```
    <ns2:add xmlns:ns2="http://calculator.me.org/">
```

```
      <i>5</i>
```

```
      <j>12</j>
```

```
    </ns2:add>
```

```
  </S:Body>
```

```
</S:Envelope>
```

# Example 1: SOAP (2)

## ❖ SOAP Response:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
```

```
  <S:Body>
```

```
    <ns2:addResponse xmlns:ns2="http://calculator.me.org/">
```

```
      <return>17</return>
```

```
    </ns2:addResponse>
```

```
  </S:Body>
```

```
</S:Envelope>
```

# Example 2: SOAP Request

POST http://localhost:8080/ws HTTP/1.1

Accept-Encoding: gzip,deflate

Content-Type: text/xml; charset=UTF-8

SOAPAction: "http://iproduct.org/course/spring-web-service/GetCountryRequest"

Content-Length: 319

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:spr="http://iproduct.org/course/spring-web-service">
```

```
  <soapenv:Header/>
```

```
  <soapenv:Body>
```

```
    <spr:getCountryRequest>
```

```
      <spr:name>Bulgaria</spr:name>
```

```
    </spr:getCountryRequest>
```

```
  </soapenv:Body>
```

```
</soapenv:Envelope>
```

# Example 2: SOAP Response

HTTP/1.1 200

Accept: text/xml, text/html, image/gif, image/jpeg, \*; q=.2, \*/\*; q=.2

SOAPAction: ""

Content-Type: text/xml;charset=utf-8

Content-Length: 415

Date: Sun, 21 Jul 2019 17:23:56 GMT

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:getCountryResponse xmlns:ns2="http://iproduct.org/course/spring-web-service">
      <ns2:country>
        <ns2:name>Bulgaria</ns2:name>
        <ns2:population>46704314</ns2:population>
        <ns2:capital>Sofia</ns2:capital>
        <ns2:currency>BGN</ns2:currency>
      </ns2:country>
    </ns2:getCountryResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



# Basic Structure of SOAP Messages

	<code>&lt;?xml version="1.0"?&gt;</code>
SOAP Envelope	<code>&lt;soap:Envelope xmlns:soap= "http://www.w3.org/2001/12/soap-envelope" soap:encodingStyle="http://www.w3.org/ 2001/12/soap-encoding"&gt;</code>
SOAP Header (optional)	<code>&lt;soap:Header&gt; ... &lt;/soap:Header&gt;</code>
SOAP Body	<code>&lt;soap:Body&gt; ...</code>
SOAP Fault	<code>&lt;soap:Fault&gt; ... &lt;/soap:Fault&gt;</code>
	<code>&lt;/soap:Body&gt;</code>
	<code>&lt;/soap:Envelope&gt;</code>

# SOAP Binding Styles

- **Remote procedure call (RPC)**: the structure of an RPC style `<soap:Body>` element needs to comply with the rules specified in detail in Section 7 of the **SOAP 1.1** specification: the `<soap:Body>` element may contain only **one element that is named after the operation**, all parameters must be represented as sub-elements of this wrapper element. Typically used with SOAP encoding, which is not compliant to **WS-I** specification requirements.
- **Document Style**: the `<soap:Body>` content is specified by **XML Schema** defined in the `<wsdl:type>` section. It **doesn't follow the SOAP conventions** – **SOAP message in `<soap:Body>` is sent as one document** without specific formatting. Document style is preferred because it is **WS-I** compliant.

# SOAP Message Encodings

- **SOAP encoding (use="encoded")** – allows serialization / deserialization of data types according to rules defined by the SOAP 1.1 standard (referred by the encodingStyle attribute). This style is not **WS-I** compliant.
- **Literal XML encoding (use="encoded")** – designates the fact that the document should be read as-is and not encoded. The document serializes as XML according to XML schema definition (following the XML schema definitions literally). Each message references a concrete schema definition. Literal encoding is **WS-I** compliant.

# Web Services Description Language (WSDL)

- Web Services Description Language (WSDL) is a W3C standard for XML-based description of web services, allowing their discovery and automatic generation of program code for their (runtime) access by the client
- WSDL specifies the following information:
  - The name of the web service and address information
  - The underlying communication protocol and message/operation encoding style which are used to access the public operations of the service
  - Information about operations, parameters, and data-types representing the web service interface

# WSDL 1.1 Basic Document Structure

```
<definitions>
  <types>    types definition</types>
  <message>  message definition </message>
  <portType>
    <operation>
      <input> ... </input>
      <output> ... </output>
    </operation>
  </portType>
  <binding>  binding definition  </binding>
  <service>
    <port>
      <soap:address ... />
    </port>
  </service>
</definitions>
```



# Example: WSDL 1.1 Web Service Description (1)

```
<definitions
  targetNamespace="http://endpoint.basicauth.helloservice/"
  name="HelloService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://endpoint.basicauth.helloservice/"
        schemaLocation="http://localhost:8080/helloservice-
          basicauth/HelloService?xsd=1" />
    </xsd:schema>
  </types>
  <message name="sayHello">
    <part name="parameters" element="tns:sayHello"/>
  </message>
  ...
```

# Example: WSDL 1.1 Web Service Description (2)

```
<message name="sayHelloResponse">
  <part name="parameters" element="tns:sayHelloResponse"/>
</message>
<portType name="Hello">
  <operation name="sayHello">
    <input message="tns:sayHello"/>
    <output message="tns:sayHelloResponse"/>
  </operation>
</portType>
<binding name="HelloPortBinding" type="tns:Hello">
...

```

# Example: WSDL 1.1 Web Service Description (3)

```
<soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
  <operation name="sayHello">
    <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  ...
```

# Example: WSDL 1.1 Web Service Description (4)

...

```
<service name="HelloService">
```

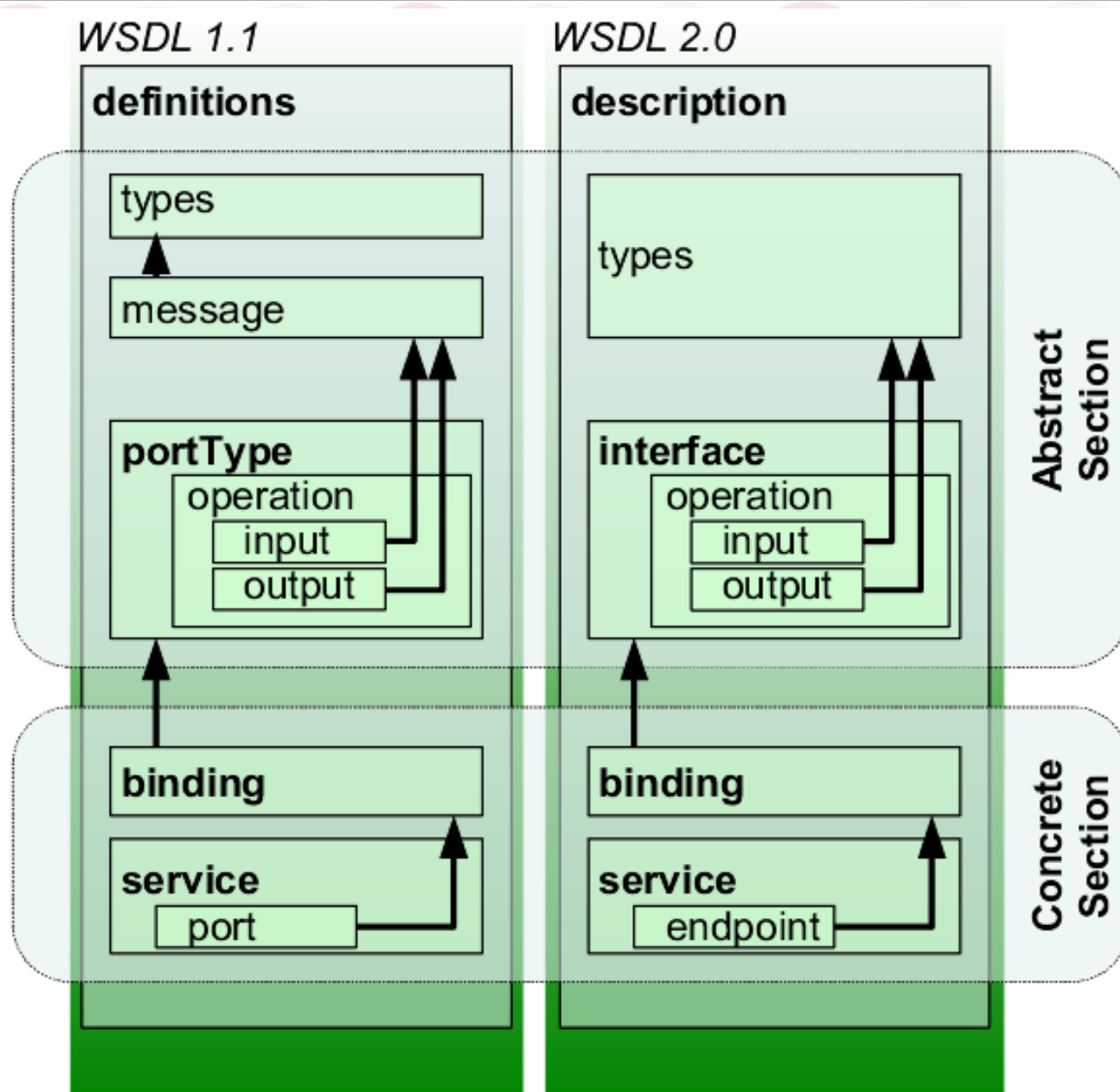
```
  <port name="HelloPort" binding="tns:HelloPortBinding">
```

```
    <soap:address location="http://localhost:8080/helloservice-  
      basicauth/HelloService"/>
```

```
  </port>
```

```
</service>
```

```
</definitions>
```



## Differences between WSDL 1.1 and WSDL 2.0



# Web Application Description Language (WADL)

- XML-based file format providing machine-readable description of HTTP-based web application resources – typically RESTful web services
- WADL is a W3C Member Submission
  - Multiple resources
  - Inter-connections between resources
  - HTTP methods that can be applied accessing each resource
  - Expected inputs, outputs and their data-type formats
  - XML Schema data-type formats for representing the RESTful resources

# Universal Description, Discovery, and Integration (UDDI)

- SOAP-based protocol defining the communication between Web Service Clients and UDDI Registry
- UDDI 2 provides abilities three query patterns:
  - Browse pattern
  - Drill-down pattern
  - Invocation pattern
- UDDI is best complemented by semantic mappings and metadata descriptions (using RDF or OWL) stored as tModels in UDDI Registry
- Web Services Inspection Language (WSIL) is a distributed alternative of UDDI focusing on web services, and not business entities descriptions

# SOA Manifesto (October 2009)

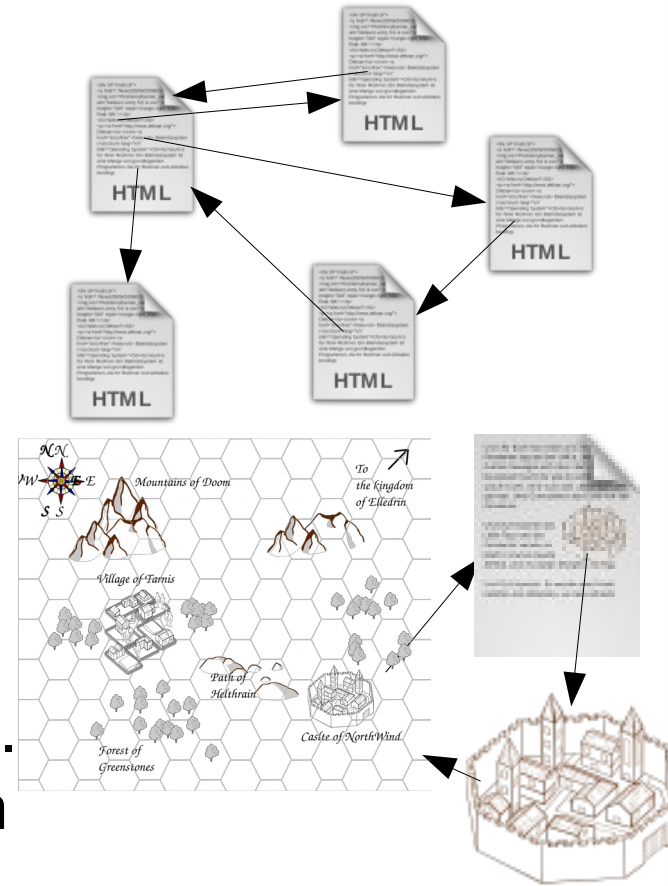
- Business value over technical strategy.
- Strategic goals over project-specific benefits.
- Intrinsic interoperability over custom integration.
- Shared services over specific-purpose implementations.
- Flexibility over optimization.
- Evolutionary refinement over pursuit of initial perfection.

# SOA Perspectives

- SOA & Web 2.0 Mashups
- Service-Oriented Business Applications (SOBA)
- Event-driven SOA (SOA 2.0) –  
[http://en.wikipedia.org/wiki/Event-driven\\_SOA](http://en.wikipedia.org/wiki/Event-driven_SOA)
- Internet of Services
- Digital Nervous System –  
[http://en.wikipedia.org/wiki/Digital\\_Nervous\\_System](http://en.wikipedia.org/wiki/Digital_Nervous_System)


# Hypertext & Hypermedia

- ❖ **Hypertext** is structured text that uses logical links (hyperlinks) between nodes containing text
- ❖ **HTTP** is the protocol to exchange or transfer hypertext
- ❖ **Hypermedia** - extension of the term hypertext, is a nonlinear medium of information which includes multimedia (text, graphics, audio, video, etc.) and hyperlinks of different media types (e.g. image or animation/video fragment can be linked to a detailed description).





# Multipurpose Internet Mail Extensions (MIME)

- ❖ Different types of media are represented using different text/binary encoding formats – for example:
  - Text -> plain, html, xml ...
  - Image (Graphics) -> gif, png, jpeg, svg ...
- ❖ Multipurpose Internet Mail Extensions (MIME) allows the client to recognize how to handle/present the particular multimedia asset/node:  
Ex.: **Content-Type: text/plain**  

- ❖ More examples for standard MIME types:  
[https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics\\_of\\_HTTP/MIME\\_types](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types)
- ❖ Vendor specific media (MIME) types:  
application/vnd.\*+json/xml

# HTTP Request Structure

**GET** /context/Servlet **HTTP/1.1**

**Host:** *Client\_Host\_Name*

*Header2: Header2\_Data*

...

*HeaderN: HeaderN\_Data*

*<Празен ред>*

**POST** /context/Servlet  
**HTTP/1.1**

**Host:** *Client\_Host\_Name*

*Header2: Header2\_Data*

...

*HeaderN: HeaderN\_Data*

*<Празен ред>*

*POST\_Data*

# HTTP Response Structure

**HTTP/1.1 200 OK**

**Content-Type:**  
**application/json**

*Header2: Header2\_Data*

...

*HeaderN: HeaderN\_Data*

*<Празен ред>*

```
[{ "id":1,  
  "name":"Novelties in Java EE 7 ...",  
  "description":"The presentation is ...",  
  "created":"2014-05-10T12:37:59",  
  "modified":"2014-05-10T13:50:02",  
},  
{ "id":2,  
  "name":"Mobile Apps with HTML5 ...",  
  "description":"Building Mobile ...",  
  "created":"2014-05-10T12:40:01",  
  "modified":"2014-05-10T12:40:01",  
}]
```

# Architectural Properties

According to **Dr. Roy Fielding** [Architectural Styles and the Design of Network-based Software Architectures, 2000]:

- Performance
- Scalability
- Reliability
- Simplicity
- Extensibility
- Dynamic evolvability
- Customizability
- Configurability
- Visibility

❖ All of them should be present in a desired Web Architecture and REST architectural style tries to preserve them by consistently applying several **architectural constraints**

# REST Architecture

According to **Roy Fielding** [Architectural Styles and the Design of Network-based Software Architectures, 2000]:

- Client-Server
- Stateless
- Uniform Interface:
  - Identification of resources
  - Manipulation of resources through representations
  - Self-descriptive messages
  - Hypermedia as the engine of application state (HATEOAS)
- Layered System
- Code on Demand (optional)



# Representational State Transfer(REST)

- ❖ REpresentational State Transfer (REST) is an architecture for accessing distributed hypermedia web-services
- ❖ The resources are identified by URIs and are accessed and manipulated using an HTTP interface base methods (GET, POST, PUT, DELETE, OPTIONS, HEAD, PATCH)
- ❖ Information is exchanged using representations of these resources
- ❖ Lightweight alternative to SOAP+WSDL -> HTTP + Any representation format (e.g. JavaScript™ Object Notation – JSON)

# Representational State Transfer(REST)

- ❖ Identification of resources – URIs
- ❖ Representation of resources – e.g. HTML, XML, JSON, etc.
- ❖ Manipulation of resources through these representations
- ❖ Self-descriptive messages - Internet media type (**MIME type**) provides enough information to describe how to process the message. Responses also explicitly indicate their **cacheability**.
- ❖ Hypermedia as the engine of application state (aka **HATEOAS**)
- ❖ Application contracts are expressed as **media types** and [semantic] link relations (**rel** attribute - RFC5988, "Web Linking")

# Hypermedia As The Engine Of Application State (HATEOAS) – New Link Header (RFC 5988) Example

Content-Length →1656

Content-Type →application/json

Link →<http://localhost:8080/polling/resources/polls/629>;  
**rel="prev"**; type="application/json"; title="Previous poll",  
<http://localhost:8080/polling/resources/polls/632>;  
**rel="next"**; type="application/json"; title="Next poll",  
<http://localhost:8080/polling/resources/polls>;  
**rel="collection"**; type="application/json"; title="Polls  
collection", <http://localhost:8080/polling/resources/polls>;  
**rel="collection up"**; type="application/json"; title="Self  
link", <http://localhost:8080/polling/resources/polls/630>;  
**rel="self"**

# Example: URLs + HTTP Methods

Uniform Resource Locator (URL)	GET	PUT	POST	DELETE
Collection, such as <a href="http://api.example.com/comments/">http://api.example.com/comments/</a>	List the URIs and perhaps other details of the collection's members.	Replace the entire collection with another collection.	Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.	Delete the entire collection.
Element, such as <a href="http://api.example.com/comments/11">http://api.example.com/comments/11</a>	Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	Replace the addressed member of the collection, or if it does not exist, create it.	Not generally used. Treat the addressed member as a collection in its own right and create a new entry in it.	Delete the addressed member of the collection.

Source: [https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer)

# Advantages of REST

- ❖ Scalability of component interactions – through layering the client server-communication and enabling load-balancing, shared caching, security policy enforcement;
- ❖ Generality of interfaces – allowing simplicity, reliability, security and improved visibility by intermediaries, easy configuration, robustness, and greater efficiency by fully utilizing the capabilities of HTTP protocol;
- ❖ Independent development and evolution of components, dynamic evolvability of services, without breaking existing clients.
- ❖ Fault tolerant, Recoverable, Secure, Loosely coupled

# Asynchronous JavaScript & XML - AJAX

- Ajax – A New Approach to Web Applications, J. Garrett  
February, 2005  
<http://www.adaptivepath.com/publications/essays/archives/000385.php>
- Presentation based on standards HTML 5 / XHTML, CSS
- Dynamic visualisation and interaction using Document Object Model (DOM)
- Exchange and manipulation of data using XML and XSLT or JavaScript Object Notation (JSON)
- Asynchronous data fetch using **XMLHttpRequest**
- And JavaScript who wraps everything above in one application



# AJAX and Traditional Web Applications

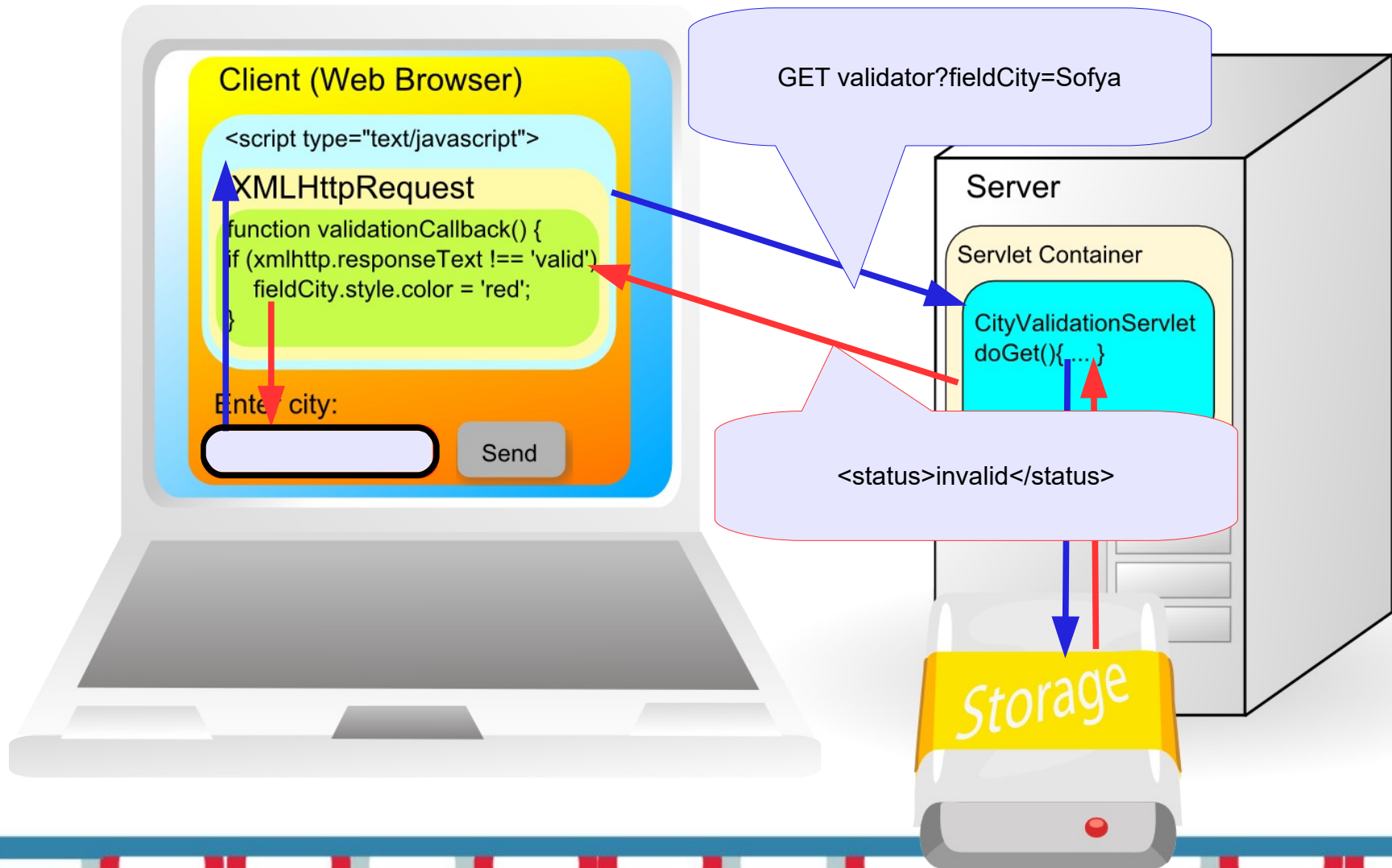
Main difference:

- Ajax apps are based on processing of **events** and **data**
- Traditional web applications are based on presenting pages and hyperlink transitions between them

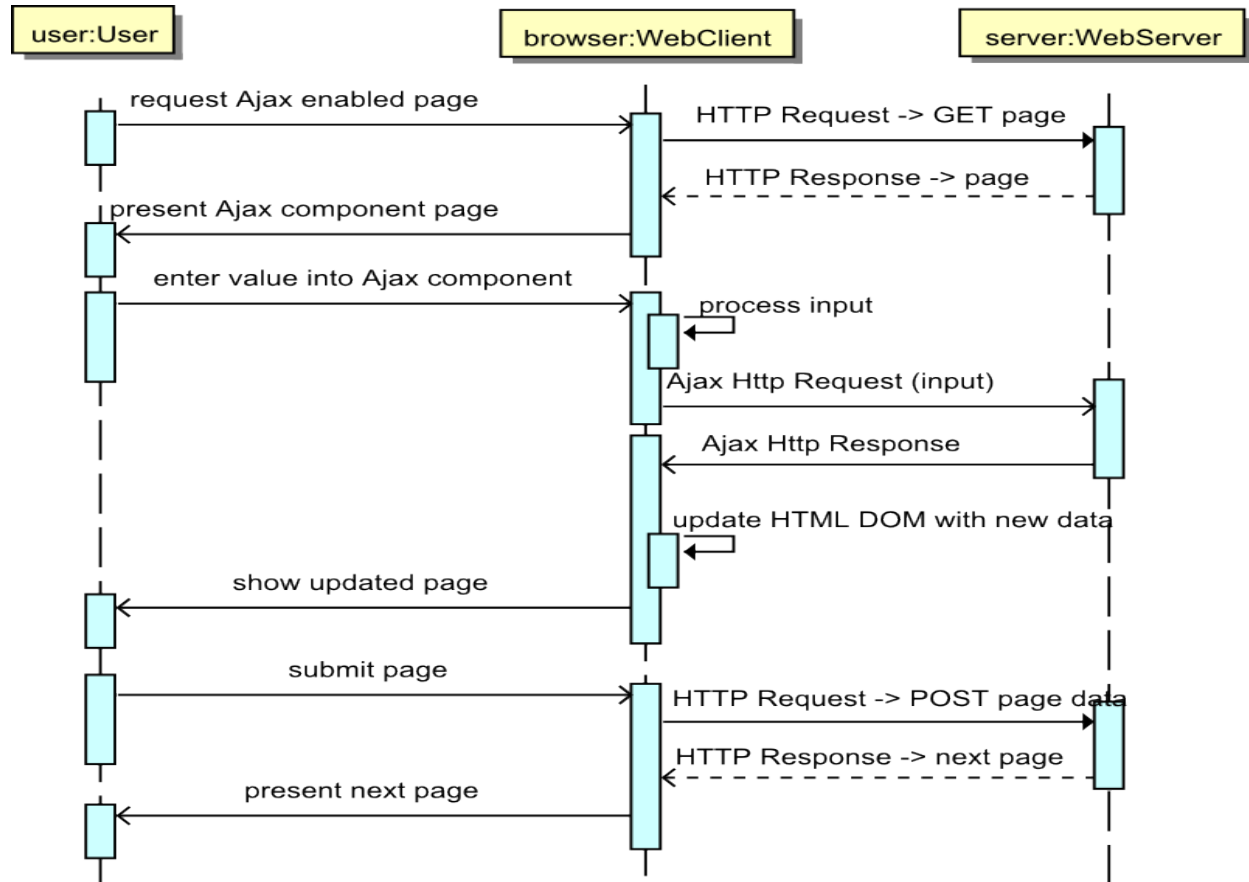
# Problems connected with AJAX (1)

- Sandboxing
- Scripting switched off
- Speed of client processing
- Time for script download
- Losing integrity
- Search engine indexing
- Accessibility
- More complex development
- More complex profiling – 2 cycles
- Cross Domain AJAX

# AJAX Interactions



# AJAX Interactions Flowchart




# AJAX Examples

- Google Suggest
- Gmail
- Google Maps
- Google Docs
- Google Calender
- Product Search on Amazon – A9
- Blogger
- Yahoo! News
- и много други

# Basic Structure of Synchronous AJAX Request

```
var method = "GET";  
var url = "resources/ajax_info.html";  
  
if (window.XMLHttpRequest) { // IE7+, Firefox, Safari, Chrome,  
    Opera,  
        xmlhttp=new XMLHttpRequest();  
    } else { // IE5, IE6  
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");  
    }  
}  
  
xmlhttp.open(method, url, false);  
xmlhttp.send();  
document.getElementById("results").innerHTML =  
    xmlhttp.responseText;
```

isAsynchronous = false





# AJAX Request with XML Processing and Authentication

```
if (window.XMLHttpRequest) { // IE7+, Firefox, Safari, Chrome, Opera,
    xmlhttp=new XMLHttpRequest();
} else { // IE5, IE6
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.open("GET", "protected/product_catalog.xml", false,
    "trayan", "mypass");
xmlhttp.send();
if (xmlhttp.status == 200 &&
    xmlhttp.getResponseHeader("Content-Type") ==
    "text/xml") {
    var xmlDoc = xmlhttp.responseXML;
    showBookCatalog(xmlDoc); // Do something with xml document
}
}
```

# AJAX Request with XML Processing (2)

```
function showBookCatalog(xmlDoc){
    txt("<table><tr><th>Title</th><th>Artist</th></tr>");
    var x=xmlDoc.getElementsByTagName("TITLE");
    var y=xmlDoc.getElementsByTagName("AUTHOR");
    for (i=0;i<x.length;i++) {
        txt=txt + "<tr><td>"
            + x[i].firstChild.nodeValue
            + "</td><td>" + y[i].firstChild.nodeValue
            + "</td></tr>";
    }
    txt += "</table>"
    document.getElementById("book_results").innerHTML=txt;
}
```

# Basic Structure of **Asynchronous** AJAX Request

```
if (window.XMLHttpRequest) { // IE7+, Firefox, Safari, Chrome,
    Opera,
    xmlhttp=new XMLHttpRequest();
} else { // IE5, IE6
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.onreadystatechange = function() { ← Callback function
    if (xmlhttp.readyState==4 && xmlhttp.status==200){
        callback(xmlhttp);
    }
}
xmlhttp.open(method, url, true); ← isAsynchronous = true
xmlhttp.setRequestHeader("Content-type","application/x-www-form-
    urlencoded");
xmlhttp.send(paramStr);
```

# XMLHttpRequest.readyState

Код	Значение
1	след като XMLHttpRequest.open() е извикан успешно
2	заглавните части на отговора на HTTP заявката (HTTP response headers) са успешно получени
3	начало на зреждане на съдържанието на HTTP отговора (HTTP response content)
4	съдържанието на HTTP отговора е заредено успешно от браузъра

# HTTP Request Headers

- В **HTTP 1.0** всички заглавни части са опционални
- В **HTTP 1.1** са опционални всички заглавни части без **Host**
- Необходимо е винаги да се проверява дали съответната заглавна част е различна от **null**

# HTTP Requests Status Codes - RFC2616

- **Accept**
- **Accept-Charset**
- **Accept-Encoding**
- **Accept-Language**
- **Accept-Language**
- **Authorization**
- **Connection**
- **Content-Length**
- **Cookie**
- **Host**
- **If-Modified-Since**
- **If-Unmodified-Since**
- **Referer**
- **User-Agent**



# HTTP Request Structure

**GET** /context/Servlet HTTP/1.1

**Host:** Client\_Host\_Name

Header2: Header2\_Data

...

HeaderN: HeaderN\_Data

<Празен ред>

**POST** /context/Servlet HTTP/1.1

**Host:** Client\_Host\_Name

Header2: Header2\_Data

...

HeaderN: HeaderN\_Data

<Празен ред>

POST\_Data

# HTTP Response Structure

**HTTP/1.1 200 OK**

**Content-Type:**  
**application/json**

*Header2: Header2\_Data*

...

*HeaderN: HeaderN\_Data*

*<Празен ред>*

```
[ { "id":1,  
  "name":"Novelties in Java EE 7 ...",  
  "description":"The presentation  
is ...",  
  "created":"2014-05-10T12:37:59",  
  "modified":"2014-05-10T13:50:02",  
},  
{ "id":2,  
  "name":"Mobile Apps with  
HTML5 ...",  
  "description":"Building Mobile ...",  
  "created":"2014-05-10T12:40:01",  
  "modified":"2014-05-10T12:40:01",  
}]
```

# Response Status Codes

- **100 Continue**
- **101 Switching Protocols**
- **200 OK**
- **201 Created**
- **202 Accepted**
- **203 Non-Authoritative Information**
- **204 No Content**
- **205 Reset Content**
- **301 Moved Permanently**
- **302 Found**
- **303 See Other**
- **304 Not Modified**
- **307 Temporary Redirect**
- **400 Bad Request**
- **401 Unauthorized**
- **403 Forbidden**
- **404 Not Found**

# Response Status Codes

- **405 Method Not Allowed**
- **415 Unsupported Media Type**
- **417 Expectation Failed**
- **500 Internal Server Error**
- **501 Not Implemented**
- **503 Service Unavailable**
- **505 HTTP Version Not Supported**

# HTTP Response Headers

- **Allow**
- **Cache-Control**
- **Pragma**
- **Connection**
- **Content-Disposition**
- **Content-Encoding**
- **Content-Language**
- **Content-Length**
- **Content-Type**
- **Expires**
- **Last-Modified**
- **Location**
- **Refresh**
- **Retry-After**
- **Set-Cookie**
- **WWW-Authenticate**

# Richardson's Web Maturity Model

According to **Leonard Richardson** [Talk at QCon, 2008 - <http://www.crummy.com/writing/speaking/2008-QCon/act3.html>]:

- ❖ **Level 0 – POX:** Single URI (XML-RPC, SOAP)
- ❖ **Level 1 – Resources:** Many URIs, Single Verb (URI Tunneling)
- ❖ **Level 2 – HTTP Verbs:** Many URIs, Many Verbs (CRUD – e.g Amazon S3)
- ❖ **Level 3 – Hypermedia Links Control the Application State = HATEOAS (Hypertext As The Engine Of Application State) === **truely** RESTful Services**



# Fetch API

[ [https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) ]

- The **Fetch API** provides an interface for fetching resources like XMLHttpRequest, but more powerful and flexible feature set.
- **Promise<Response> WorkerOrGlobalScope.fetch(input[, init])**
  - **input** - resource that you wish to fetch – url string or Request
  - **init** - custom settings that you want to apply to the request: **method**: (e.g., GET, POST), **headers**, **body**(Blob, BufferSource, FormData, URLSearchParams, or USVString), **mode**: (cors, no-cors, or same-origin), **credentials**(omit, same-origin, or include. to automatically send cookies this option must be provided), **cache**: (default, no-store, reload, no-cache, force-cache, or only-if-cached), **redirect** (follow, error or manual), **referrer** (default is client), **referrerPolicy**: (no-referrer, no-referrer-when-downgrade, origin, origin-when-cross-origin, unsafe-url), **integrity** (subresource integrity value of request)

# Cross-Origin Resource Sharing(CORS)

- ❖ Позволява осъществяване на заявки за ресурси към домейни различни от този за извикващия скрипт, като едновременно предоставя възможност на сървъра да прецени към кои скриптове (от кои домейни – Origin) да връща ресурса и какъв тип заявки да разрешава (GET, POST)
- ❖ За да се осъществи това, когато заявката е с HTTP метод различен от GET се прави предварителна (preflight) OPTIONS заявка в отговор на която сървърът връща кои методи са достъпни за съответния Origin и съответния ресурс

# CORS HTTP Headers - Simple

## ❖ HTTP GET request

GET /crossDomainResource/ HTTP/1.1

Referer: <http://sample.com/crossDomainMashup/>

Origin: <http://sample.com>

## ❖ HTTP GET response

[Access-Control-Allow-Origin: http://sample.com](#)

[Content-Type: application/xml](#)

# CORS HTTP HEADERS – POST ...

## ❖ HTTP OPTIONS preflight request

OPTIONS /crossDomainPOSTResource/ HTTP/1.1

Origin: http://sample.com

Access-Control-Request-Method: POST

Access-Control-Request-Headers: MYHEADER

## ❖ HTTP response

HTTP/1.1 200 OK

Access-Control-Allow-Origin: http://sample.com

Access-Control-Allow-Methods: POST, GET, OPTIONS

Access-Control-Allow-Headers: MYHEADER

Access-Control-Max-Age: 864000

# Resources

- Crockford, D., JavaScript: The Good Parts. O'Reilly, 2008.
- Douglas Crockford: JavaScript: The Good Parts video at YouTube – [http://www.youtube.com/watch?v=\\_DKkVvOt6dk](http://www.youtube.com/watch?v=_DKkVvOt6dk)
- Douglas Crockford: JavaScript: The Good Parts presentation at <http://crockford.com/onjs/2.pptx>
- Koss, M., Object Oriented Programming in JavaScript – <http://mckoss.com/jscript/object.htm>
- Osmani, A., Essential JavaScript Design Patterns for Beginners <http://addyosmani.com/resources/essentialjsdesignpatterns/book/>
- Fielding's REST blog – <http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>

# Thank you for your attention!



**Trayan Iliev**

**CEO of IPT – Intellectual Products  
& Technologies**

<http://iproduct.org/>

<http://robolearn.org/>

<https://github.com/iproduct>

<https://twitter.com/trayaniliev>

<https://www.facebook.com/IPT.EACAD>

<https://plus.google.com/+IproductOrg>