<epam> Excellence in Software Engineering

Test automation for WSs

Web Services (WSs) and Service-oriented architecture (SOA)

Introduction

Anastasiya Babovich
E-mail: Anastsiya_Babovich@epam.com
Skype: anastasiya_babovich

- Senior Software Test Automation Engeneer
- More than 2 year in EPAM
- Expertise:
  - UI Test Automation based on Java
  - WS Test Automation using tools and custom frameworks
  - BDD approaches

Some rules

Listen attentively

The rule of hand

Turn off your phones

Be punctual (song or poem)

Training objectives and schedule

Type of lesson	Theme of lesson	Date	Shedule
Lecture №1	What is WSs and SOA	07.04.2015	10.00 – 14.00
Lecture №2	REST and WADL	09.04.2015	10.00 – 12.00
Lecture №3	SOAP and WSDL	09.04.2015	12.00 – 14.00
Lecture №4	Soap UI	11.04.2015	10.00 – 12.00
Code review №1	Code review for “Home task №1”	11.04.2015	12.00 – 13.00
Code review №2	Code review for “Home task №2”	11.04.2015	13.00 – 14.00
Code review №3	Code review for “Home task №3”	11.04.2015	14.00 – 15.00
Lecture №5	Services creation	14.04.2015	10.00 – 12.00
Code review №4	Code review for “Hometask №4”	14.04.2015	12.00 – 14.00
Lecture №6	Java-based approach	16.04.2015	10.00 – 12.00
Code review №5	Code review for “Hometask №5”	16.04.2015	12.00 – 14.00
Lecture №7	Popular libraries and frameworks	23.04.2015	10.00 – 12.00
Code review №6	Code review for “Hometask №6”	23.04.2015	12.00 – 14.00
Code review №7	Code review for “Hometask №7”	27.04.2015	10.00 – 12.00

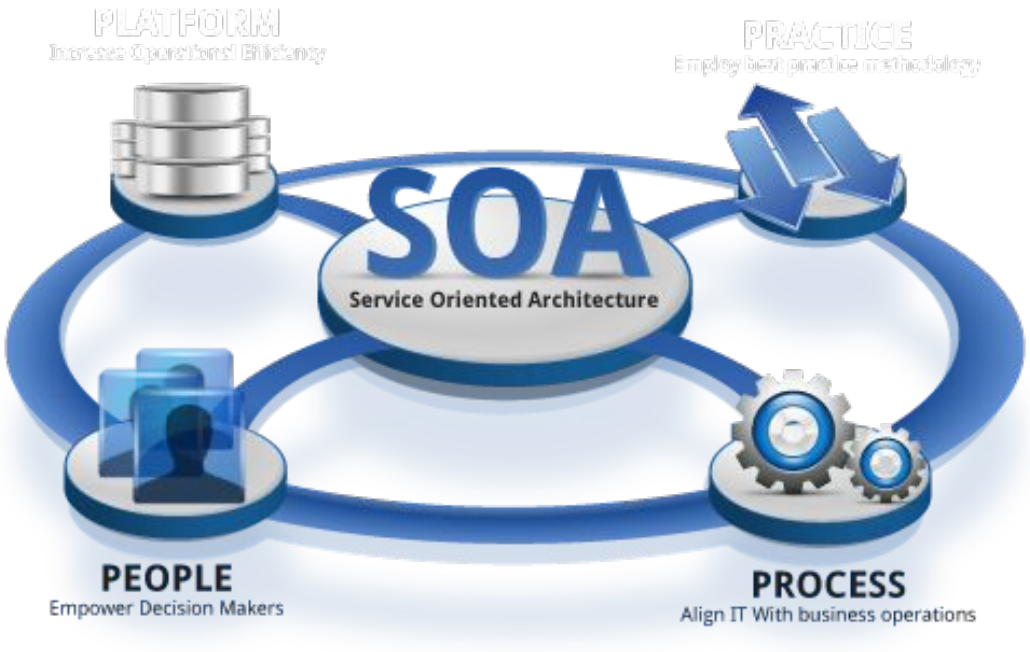
SOA

Service-Oriented Architecture

What is SOA?

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.



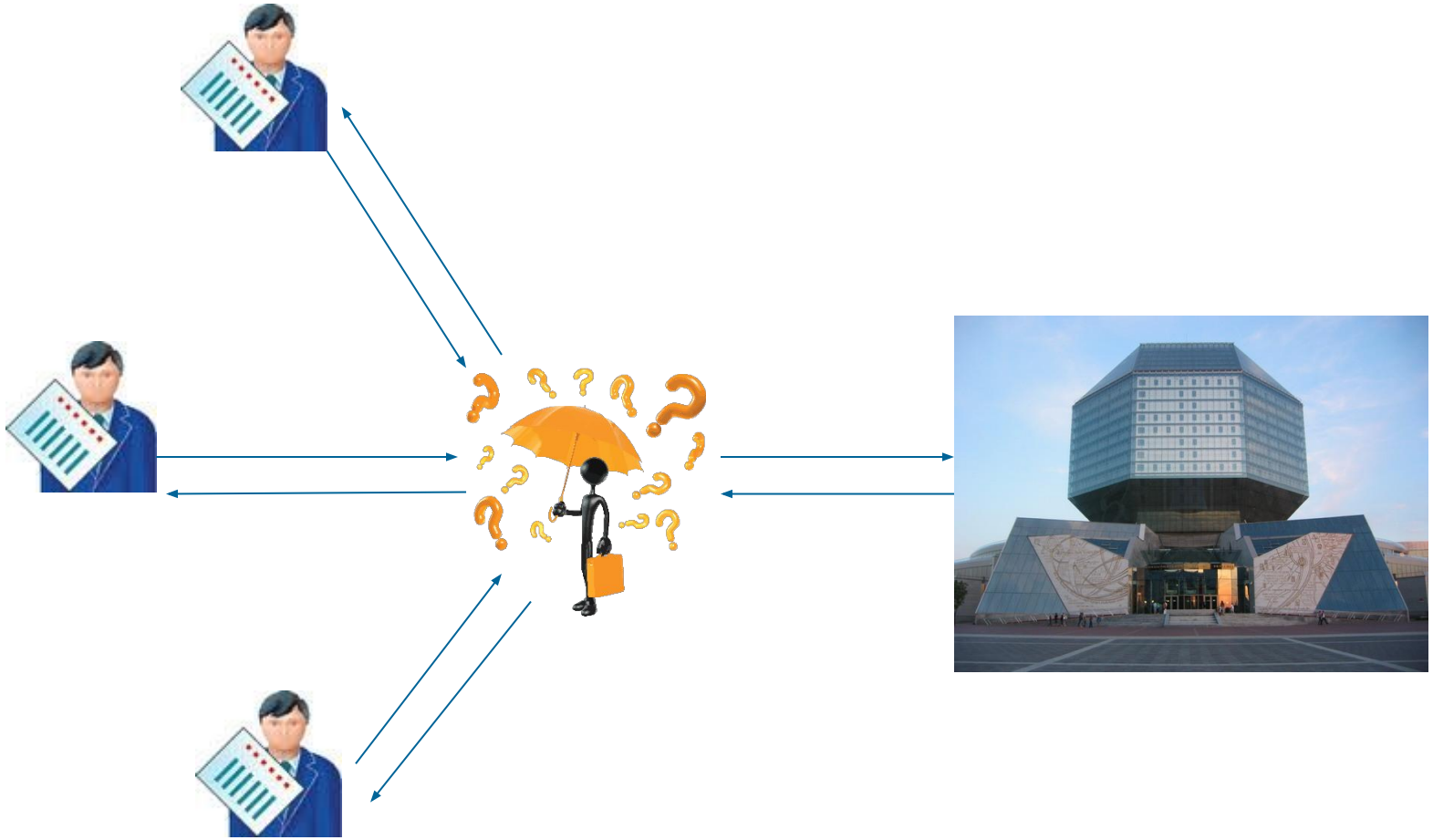
What is SOA?

SOA also generally provides a way for consumers of services, such as web-based applications, to be aware of available SOA-based services.

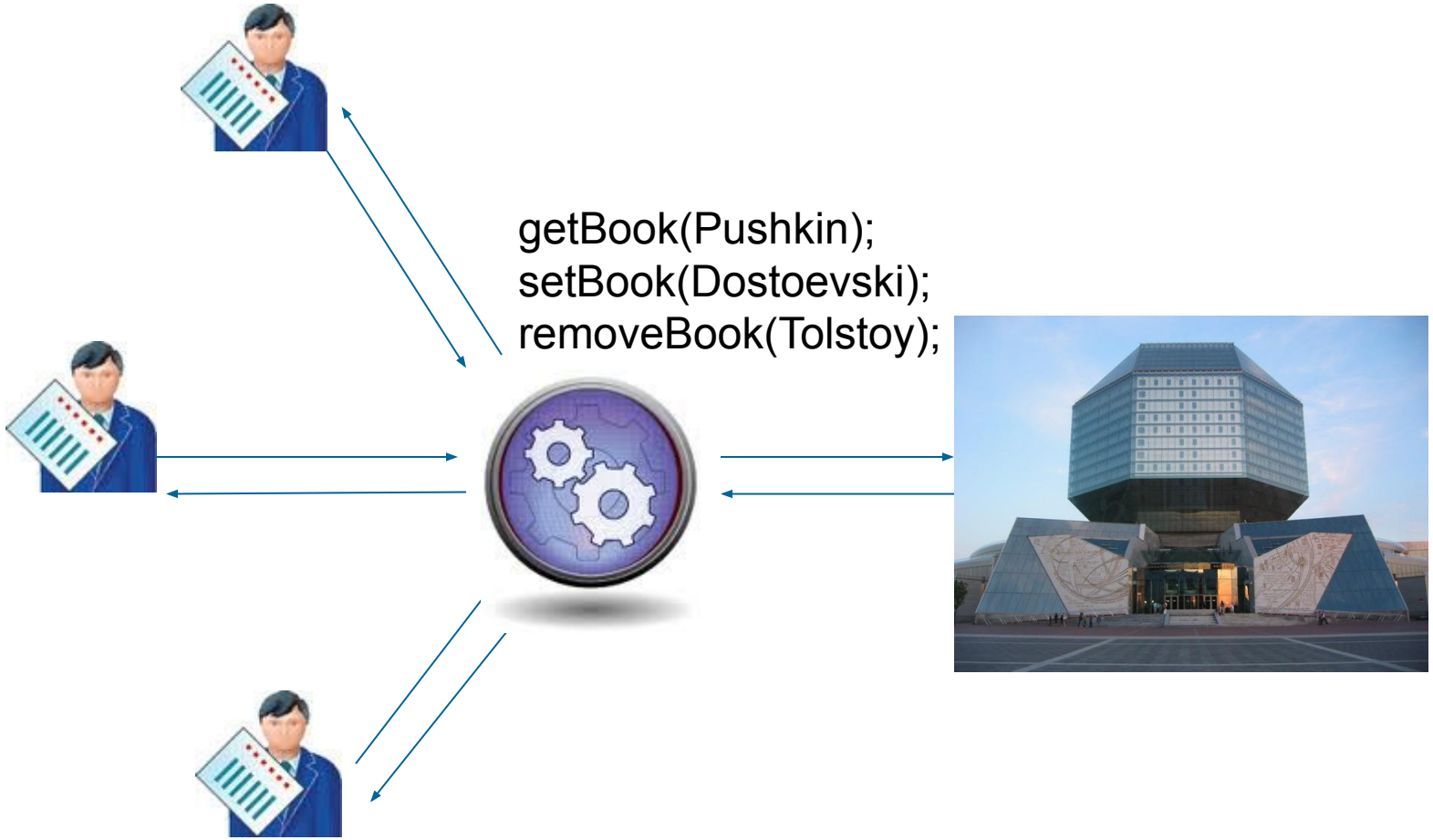
SOA defines how to integrate widely disparate applications for a Web-based environment and uses multiple implementation platforms.

Rather than defining an API, SOA defines the interface in terms of protocols and functionality. An *endpoint* is the entry point for such a SOA implementation.

Example of SOA From Life



Example of SOA From Life



SOA Goals

- Reducing costs when developing applications, due to streamlining the development process;
- Improved manageability of systems produced;
- Increased scalability posed systems;
- Independence of the platform, tools, development languages;
- Increased code reuse.

SOA Principles

Architecture is not tied to any particular technology.

Independence of the organization system on your computing platform (platforms).

Independence of the organization system used by programming languages.

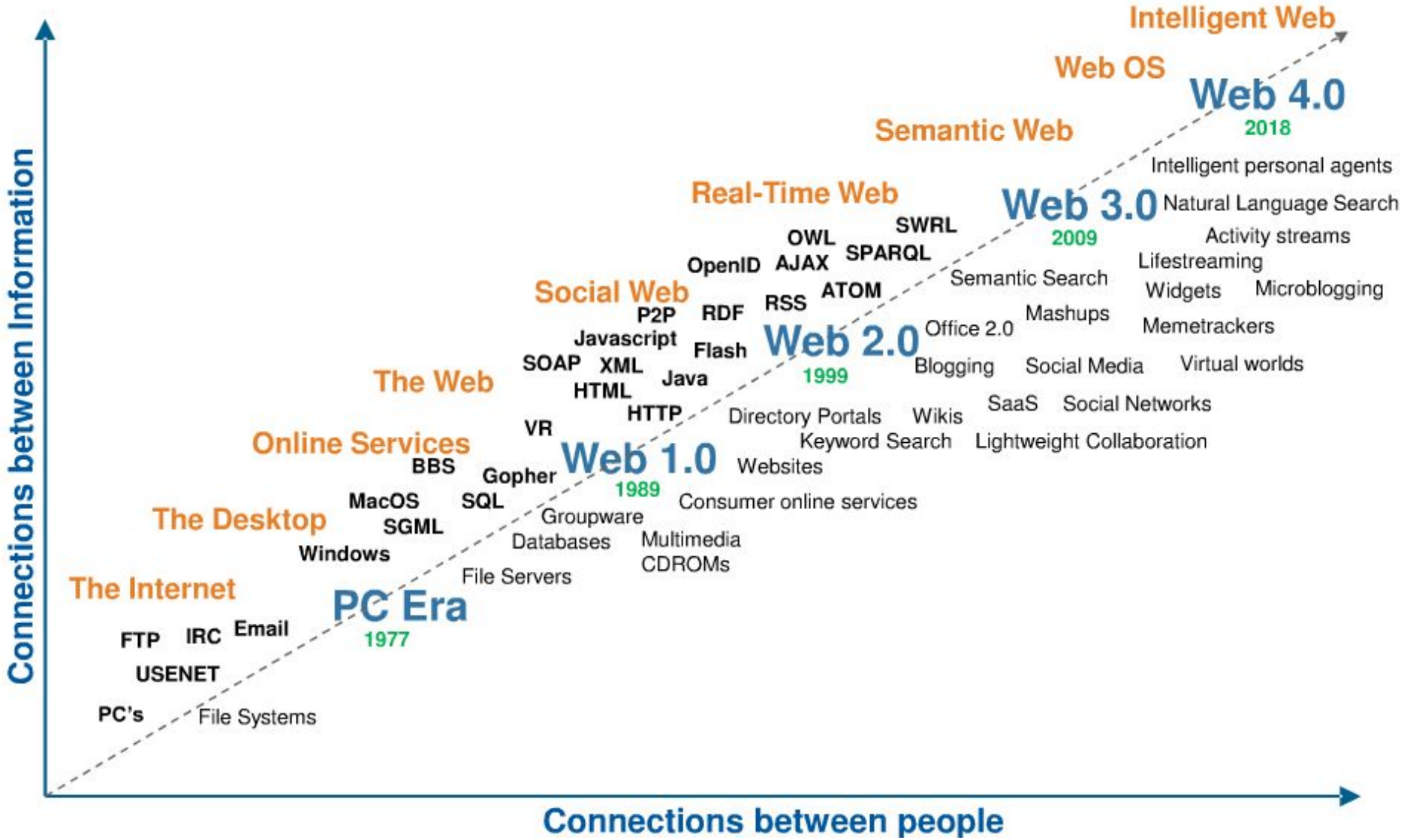
Use services that are independent of specific applications, with a uniform interface to access them.

Organization of services as a weakly-coupled components for building systems.

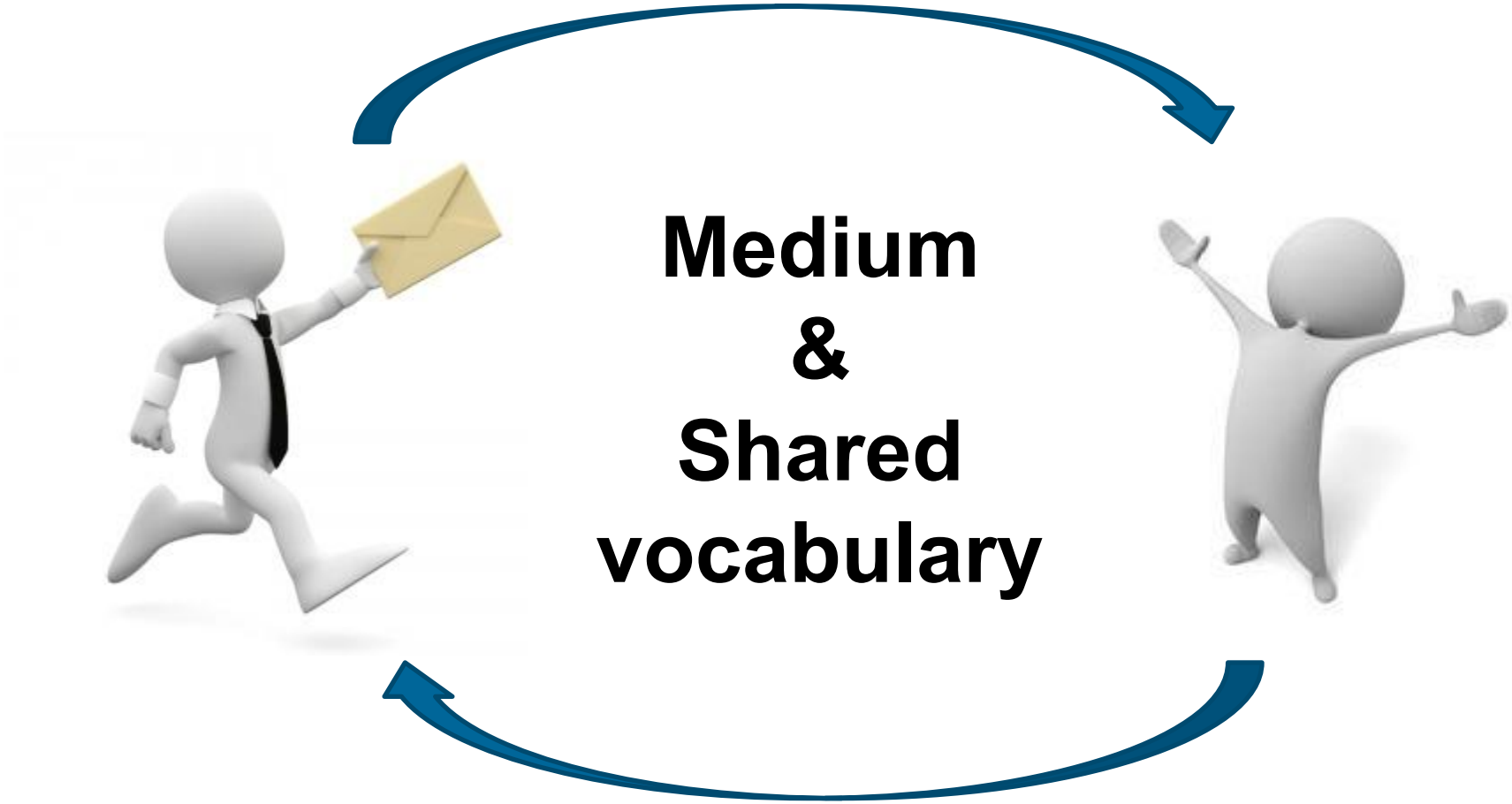
WS

Why do we need Web Services?

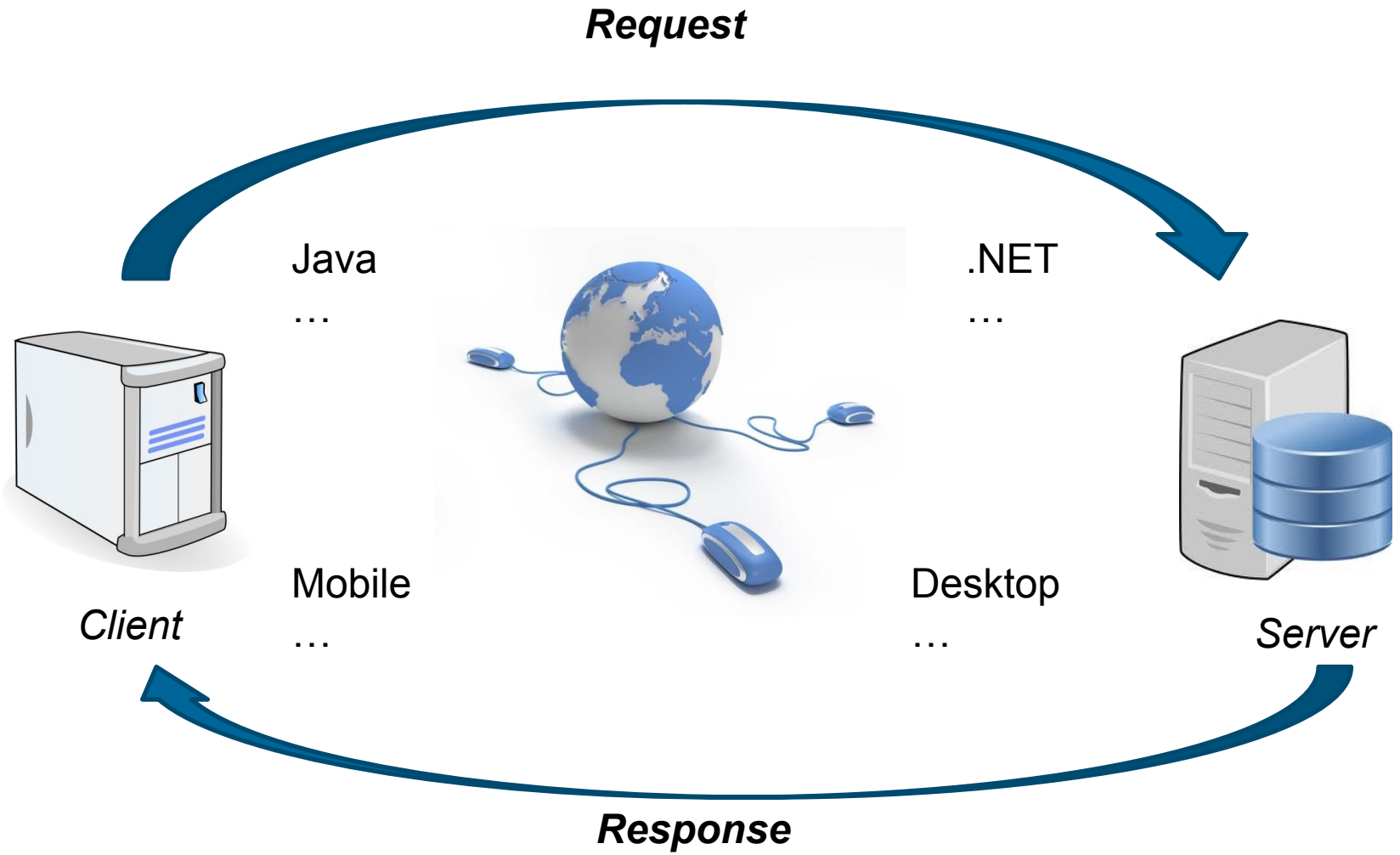
The Intelligence is in the Connections



How do people communication?



How does software communicate?



Web Service - Definition

W3C Definition

"Web Service is a software application identified by a *URI*, whose interfaces and bindings are capable of being *defined*, *described*, and *discovered* by XML artifacts and which supports direct *interactions* with other software applications using XML-based messages via internet-based protocols".

Wikipedia

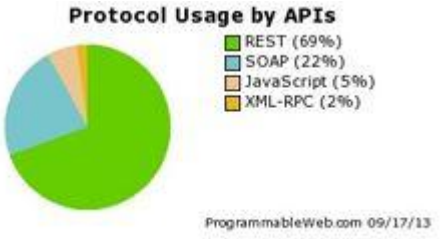
A **Web service** is a method of communication between two electronic devices over a network. It is a software function provided at a network address over the Web with the service *always on* as in the concept of utility computing.

Vangie Beal

The term *Web services* describes a standardized way of integrating Web-based applications using the XML, SOAP, WSDL and UDDI open standarts over an Internet protocol backbone

Web Service – what are?

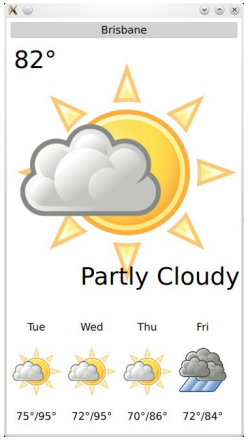
- Web services are application components
- Communicate using open protocols
- Can be used by other applications
- Self-contained and self-describing



Web Service – type of usage

Reusable application-components.

- There are things applications need very often. So why make these over and over again?
- Web services can offer application-components like: currency conversion, weather reports, or even language translation as services.

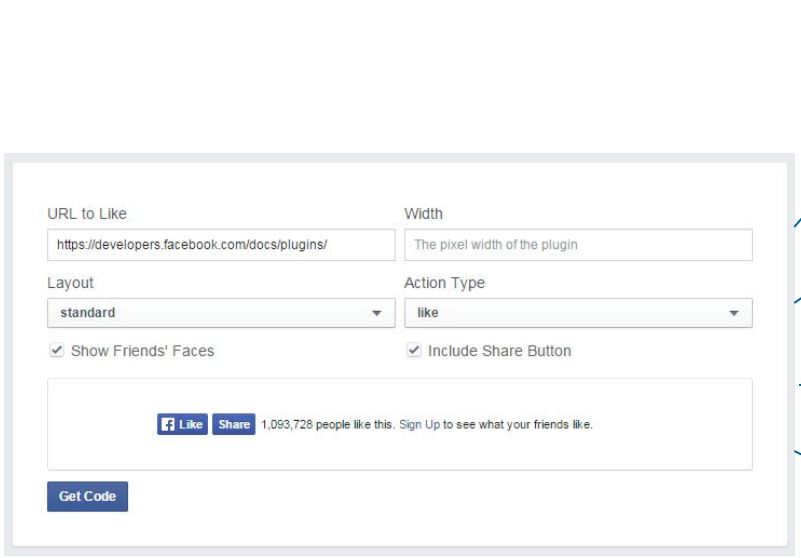


Connect existing software.

- Web services can help to solve the interoperability problem by giving different applications a way to link their data.
- With Web services you can exchange data between different applications and different platforms.

Reusable application-components

Facebook like



Standard

Box count

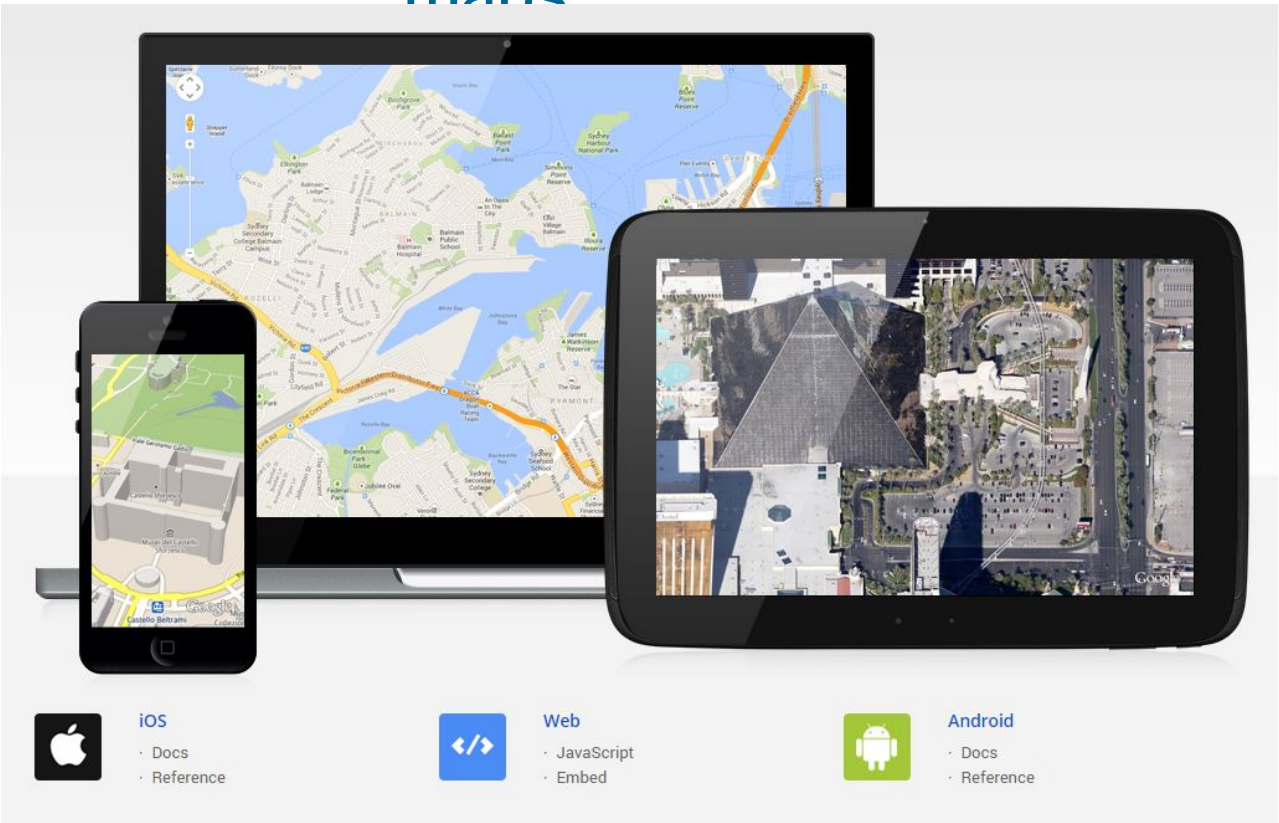
Button count

Button

Source: https://developers.facebook.com/docs/plugins/like-button/

Connect existing software

Google maps



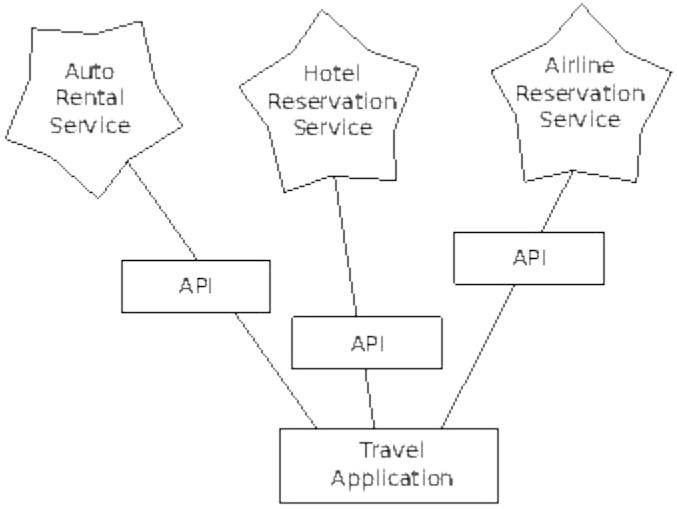
Source: https://developers.google.com/maps/

Real project example

Epam - Expedia

Real project example

Epam - Expedia



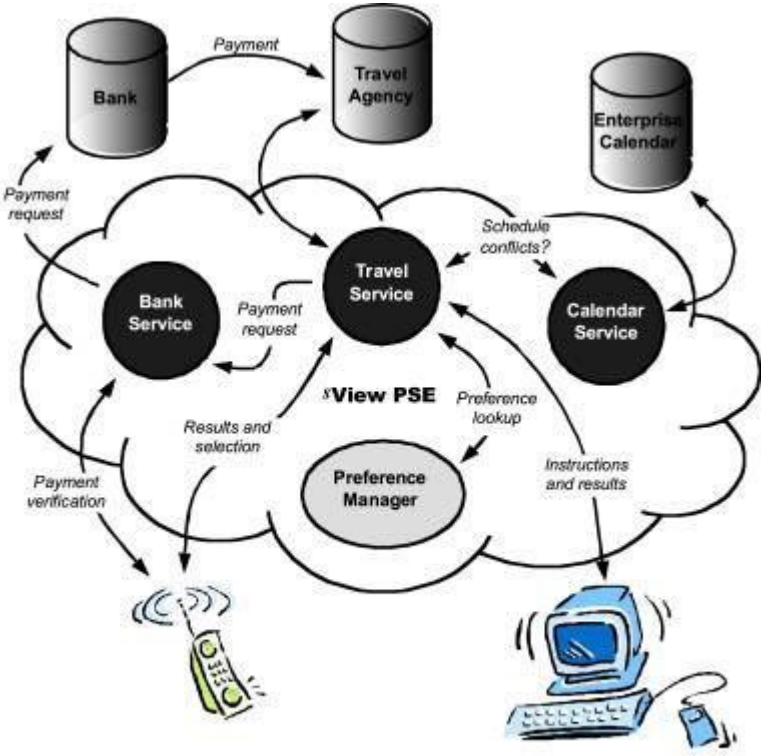
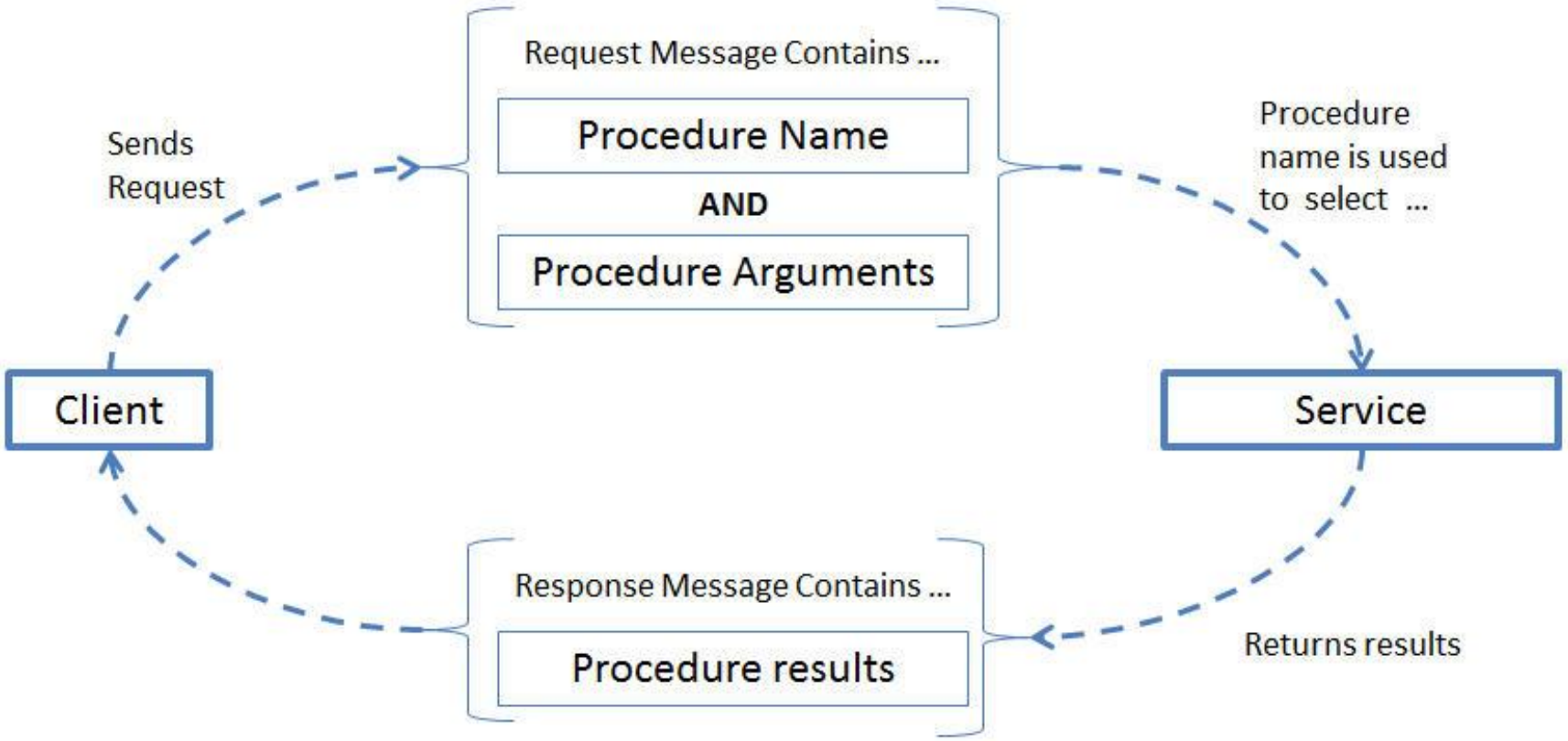


Figure 1: A schematic overview of the described usage scenario.

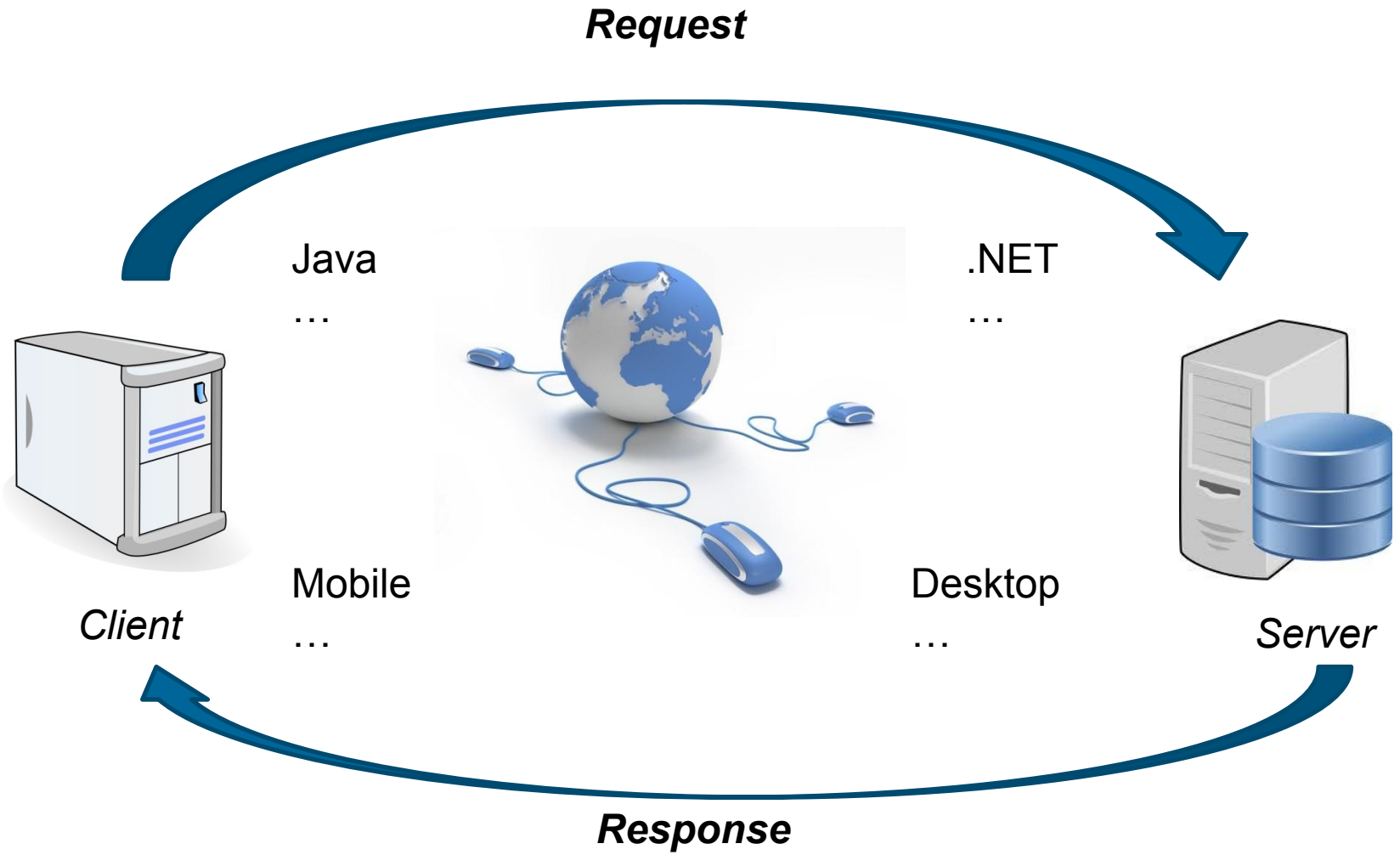
RPC – remote procedure call



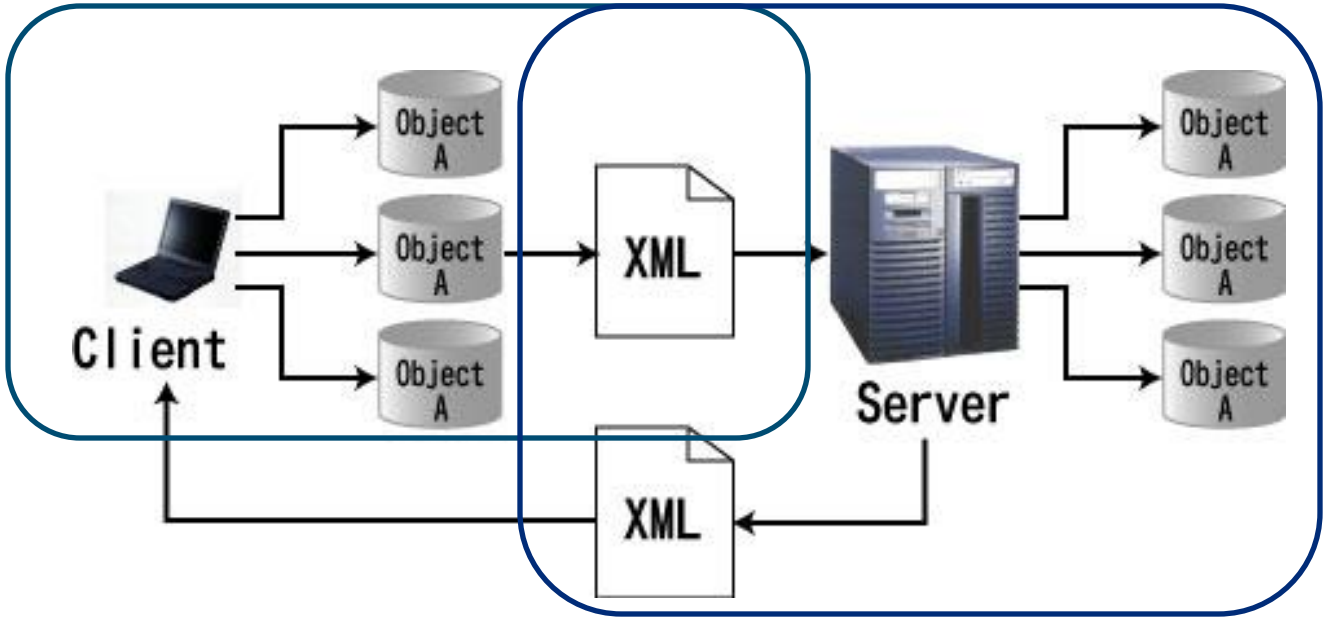
RPC – types

Java RMI	• Java's Java Remote Method Invocation () API provides similar functionality to standard Unix RPC methods.
XML - RPC	• RPC protocol that uses XML to encode its calls and HTTP as a transport mechanism
JSON - RPC	• RPC protocol that uses JSON-encoded messages
SOAP	• Successor of XML-RPC and also uses XML to encode its HTTP-based calls

How does software communicate?



RPC – XML – example



Web services & Web applications

Widely Well-known words

Web Application

- Usually a collection of dynamic web pages
- Usually restricted to the intranet
- Can be implemented as desktop application
- Information accessible using front end user interfaces
- Accessed by authorised users only

Web Site

- Collection of static and dynamic web pages
- Available on the internet, or an organization's intranet
- Cannot be implemented as desktop application
- Information accessible using front end user interfaces
- Accessed by anybody

Widely Well-known words

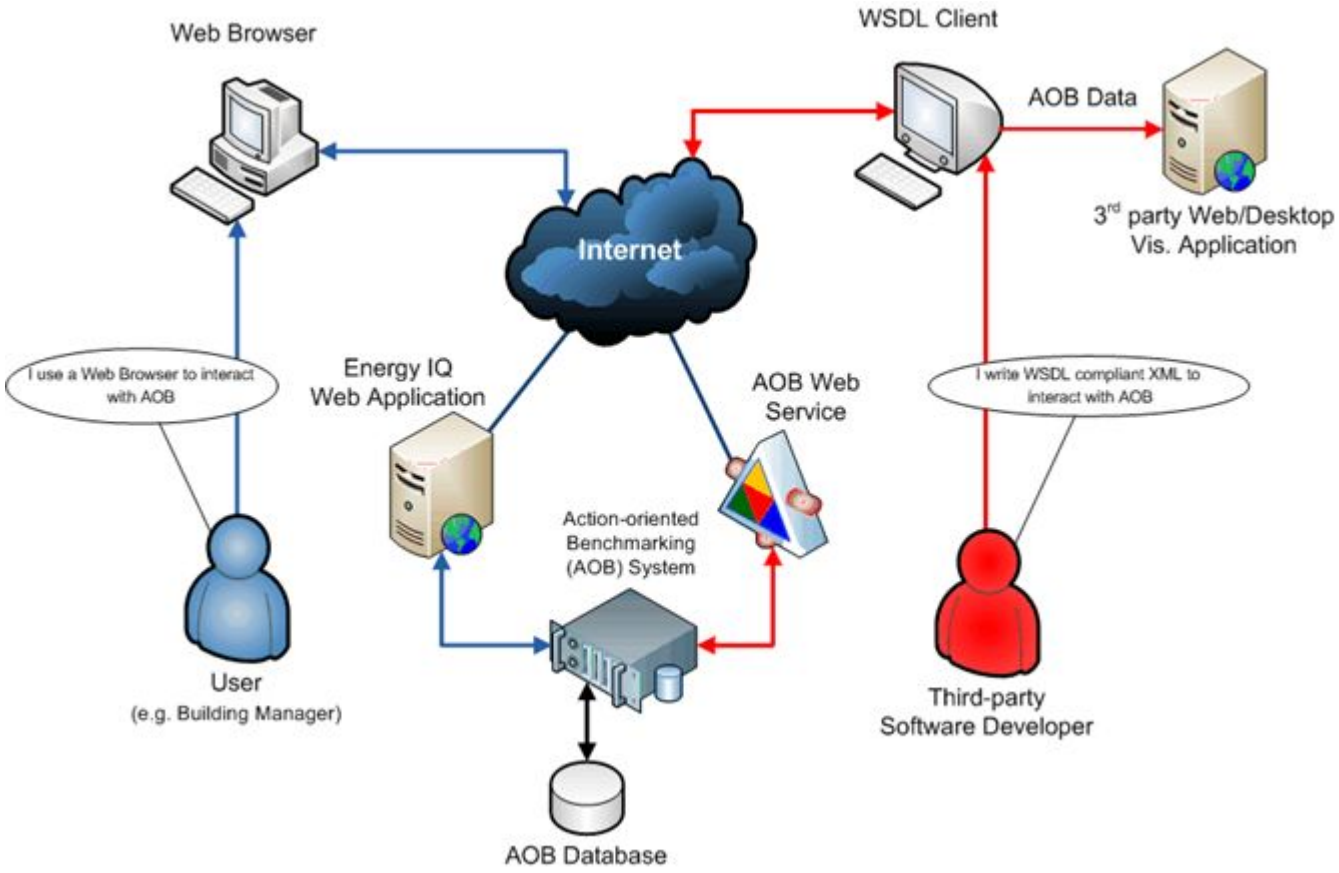
Web Server

- Software designed to serve web pages/web sites/web services. Examples are IIS, Apache, etc.

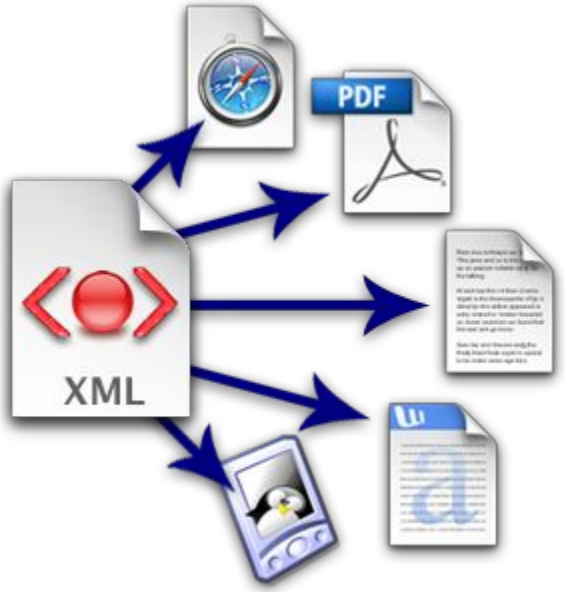
Web Service

- Application run by a web server, performing tasks and returning structured data to a calling program, rather than html for a browser.
- Only “provides” information; does not “present” information
- Publicly available and standardized for use by all programmers

Web applications and Web services



Extensible Markup Language (XML)



XML

- Based on Standard Generalized Markup Language (SGML)
- Language similar to HTML
- Created for data description
- XML tags are not defined by default in XML, they should be defined by ourselves
- DTD or XML scheme are used for description of rules

XML

- XML data can be stored as in separate file as well as inside of HTML, that will be responsible just for the format, but not for the data
- XML can be used for communication between two incompatible systems
- XML is used for as for data storage in file system as well as for storage and operation
- Make the information available for internet users

XML

- Give the life for derived from the XML languages like WAP and WML
- For communication between client and server in Web applications (Ajax)

Well-formed XML

XML document has just one root element

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName>Peter</givenName>
    <familyName>Kress</familyName>
</person>
<!-- Below is invalid element -->
<person>
    <givenName>John</givenName>
    <familyName>Doe</familyName>
</person>
<person/>
```

Well-formed XML

All elements must have closing tags

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <familyName>Kress</familyName>
    <not_closed_element>
</person>
```

Well-formed XML

Names of elements are case-sensitive, so the opening and closing tags must be in the same case.

```
<message>This is correct</message>

<Message>This is incorrect</message>
```

Well-formed XML

Elements can not overlap

```
<!-- valid -->
<b>This is bold text.</b> <i><b>This is
bold italic text.</b> This is italic
text.</i>

<!-- invalid -->
<b>This is bold text. <i>This is bold
italic text.</b> This is italic
text.</i>
```

Well-formed XML

All attribute values must be enclosed in quotes

```
<!-- valid -->
<person name="John" surname='Doe'/>
<!-- invalid -->
<person name=John/>
```

<,>, & Can not be used in text blocks

```
<!-- valid -->
<text>I &amp; my dog</text>
<!-- invalid -->
<text>I & my dog</text>
```

Well-formed XML

Unlike HTML, XML does not cut blanks and blank lines

For example, in HTML

```
<h4> Hello                    my name is Tove</h4>
```

We will get

```
Hello my name is Tove
```

XML and HTML. Semantics

- The values of the elements defined in the HTML rigidly

- Semantics and structure of elements in XML can be different
- There are several approaches to describe the structure of XML (Schema and DTD)

- XSLT can be used to effectively transform the XML document in any format: HTML, Plain text, XML, JavaScript, and others.

Pros of XML format

- Creation of own named structures for storage of information
- Task of analysis (parsing) XML is well-formalized and there are many implementations
- XML uses Unicode, that is simplifies internationalization
- Checking the document structure and data types is a standard operation
- XML - is a text format, easy to read and debug
- Tools for work with XML are available on all platforms
- XML allows you to use the infrastructure, created for HTML, including HTTP and some browsers

Cons of XML format

- XML documents are usually less concise than comparable binary formats
- Transferring of XML creates more traffic or more CPU overhead when using compression
- XML parsing can be slower and more demanding of memory than parsing optimized binary documents

Component parts of XML document

- ✔ Prologue
- ✔ Elements
- ✔ Processing instructions
- ✔ Comments
- ✔ Attributes
- ✔ Pointers
- ✔ Text blocks
- ✔ CDATA blocks
- ✔ Namespaces

Prologue

- Prologue - this is part of the XML document from the beginning to the opening tag of the root element
- Prologue includes information relating to the entire document, such as the encoding of the document structure
- Prolog can contains comments

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl"
                 href="show_book.xsl"?>
<!DOCTYPE catalog SYSTEM "catalog.dtd">
<!--catalog last updated 2010-11-01-->
<person> ... </person>
```

Prologue

XML declaration

- XML declaration is usually the first line in the XML document

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

- XML declaration may contains the following information:

- Version number:

```
version="1.0"
```

- An indication of the character encoding in which the document is written:

```
encoding="UTF-8"
```

- Parameter «standalone» which indicates whether the prohibited links to external documents:

```
standalone="yes"
```

Processing instructions

```
<?xml-stylesheet type="text/xsl"
                 href="show_book.xsl"?>

<!-- legal but not effective-->
<?style – oh, try /style.css ?>
```

The DOCTYPE declaration

```
<!DOCTYPE rootElement SYSTEM "URIreference"[
    declarations
]>
```

Elements

```
<book id="bk109">
    <author>Kress, Peter</author>
    <title>Paradox Lost</title>
    <genre>Science Fiction</genre>
    <price>6.95</price>
</book>
```

Tags

- Tags define the boundaries of the element
- Opening tags indicate the start of an element:

```
<elementName att1Name="att1Value"
             att2Name="att2Value" ...>
```

- End tags mark the end of an element. They haven't attributes:

```
</elementName>
```

- Empty tags are used to create elements without text content. They may include attributes:

```
<elementName att1Name="att1Value"
             att2Name="att2Value" ... />
```

- It is believed that the element includes an opening and closing tags and everything in between

Comments

```
<!--
 Here some notes about document can be placed
<!-- Invalid comment -->
-->
<person <!-- Invalid comment -->>
<!--
 Just another comment
-->
</person>
<!--
 Comment in the end of the document
-->
```

Text blocks

Instead of characters <,>, & the& lt; & gt; and & amp should be used;

```
<node_with_text>
This is text block
<nested_element/>
This is another text block. Symbols <, >
and & can’t be used directly.
</node_with_text>
```

Pointers to characters and entities

- Pointers are used when it is impossible or undesirable to include a character or string "directly“
- Pointer to start with an ampersand & and end with a semicolon (;)
- Pointers to characters provide an opportunity to include in the document Unicode characters using the number
- & # value;

  - The syntax for the decimal indicator
- & # xvalue;

  - The syntax for hexadecimal pointers.

Pointers to characters

- Some of the most frequently used pointers to characters.

Pointer	Symbol
&lt;	<
&gt;	>
&amp;	&
&apos;	'
&quot;	"

CDATA Blocks

- CDATA sections give information for parser that there are no markup characters within them;

```
<![CDATA[An in-depth look at creating
applications with XML, using <, >,]]>
```

- CDATA can not contains another CDATA blocks;

- Characters inside CDATA should be from class of allowed XML documents

<sender>John Smith</sender>

<![CDATA[<sender>John Smith</sender>]]>

Attributes

- `<color RGB="true">#ff08ff</color>`
- `<color RGB="false">white</color>`

- `<font color="white"`
- `name="Arial">Black</font>`

Usage of Namespaces

- XML namespace – is the collection of the names identifiable by reference URI [RFC2396], that is used in XML documents to indicate the element types and attribute naming.

- XML namespace differs from the "namespaces", that are commonly used in computer science, in that it is an XML option has an internal structure, and from a mathematical point of view, is not a set.

The scope of namespace

- Namespace refers to an element where it was declared and to all child elements

```
<?xml version="1.0"?>
<!--
Все элементы здесь явно соотнесены с пространством имен HTML
-->
<html:html xmlns:html='http://www.w3.org/TR/REC-html40'>
  <html:head>
    <html:title>Frobnostication</html:title>
  </html:head>
  <html:body>
    <html:p>
      Moved to<html:a href='http://frob.com'>here.</html:a>
    </html:p>
  </html:body>
</html:html>
```

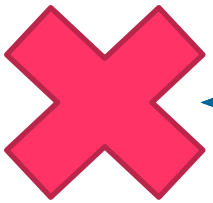
XSD

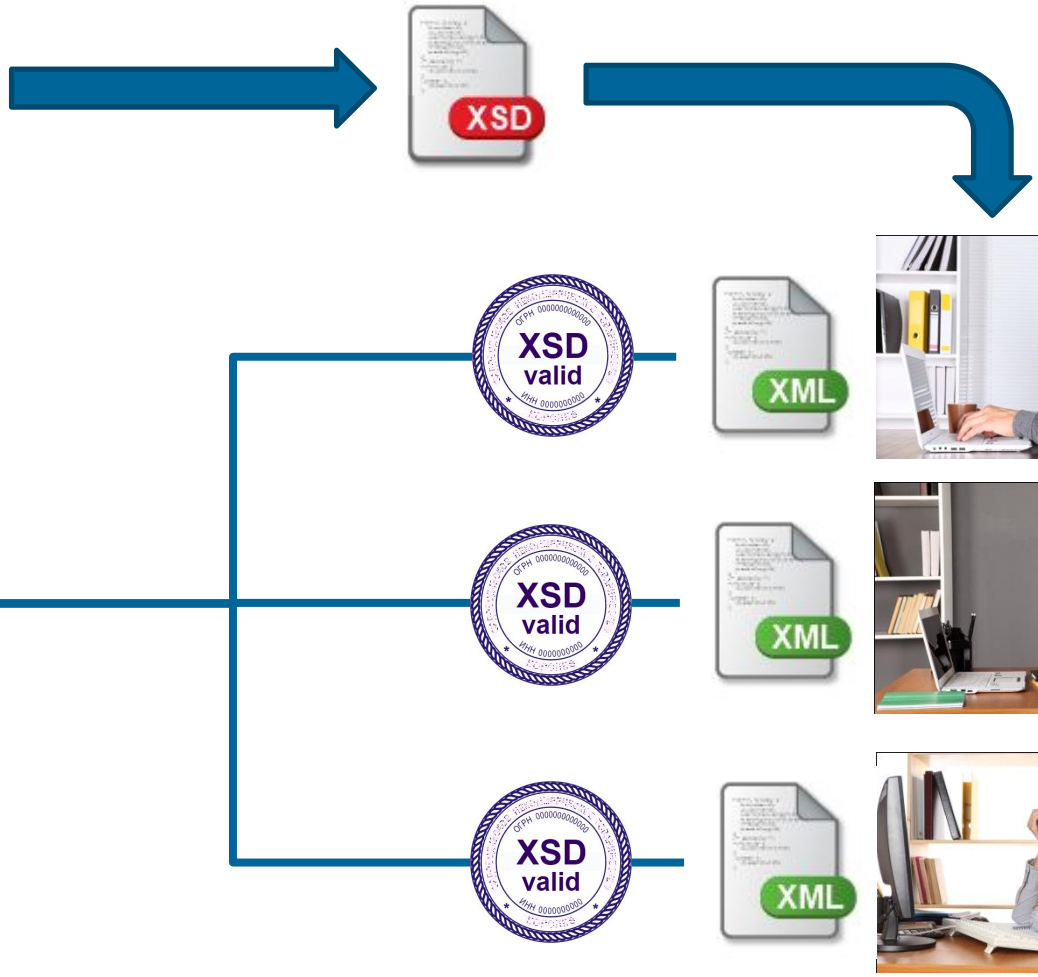
What is XML schema (XSD)?

- XML Schema definition language (XSD) – language for XML documents and data
- 2 May 2001, World Wide Web Consortium (W3C) published version 1.0 standard XSD.
- XML Schema describes elements of XML document
- Describes attributes in XML
- Describes child nodes, following and size
- Describes types of data for elements and attributes

Why we need schemes?

Server-side programmer





Client-side programmers

Creation of XML Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.epam.com"
  xmlns="http://www.epam.com"
  elementFormDefault="qualified">
  ...
</xs:schema>
```

- Element **<schema>** - root element of every XML schema
- **xmlns:xs="http://www.w3.org/2001/XMLSchema"** - Namespace declaration of the XML schema with the prefix xs
- **targetNamespace="http://www.epam.com"** – namespace for this scheme is applied
- **xmlns="http://www.epam.com"** – the default namespace (no prefix)
- **elementFormDefault="qualified"** - all elements must be namespace qualified

Simple elements

- Elements are declared using the element `<element>`.
- A simple element can only contain text.
- A simple element can not contain attributes.
- Declares a simple type (basic type or the new type with the extension or restriction of the base type using the element c simpleType).

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

<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Attributes

- Attributes are declared using the element <attribute>
- Attribute is always declared simple type

```
<xs:attribute name="lang" type="xs:string"/>

<lastname lang="EN">Smith</lastname>
```

- For an attribute, you can specify a default value or a fixed value

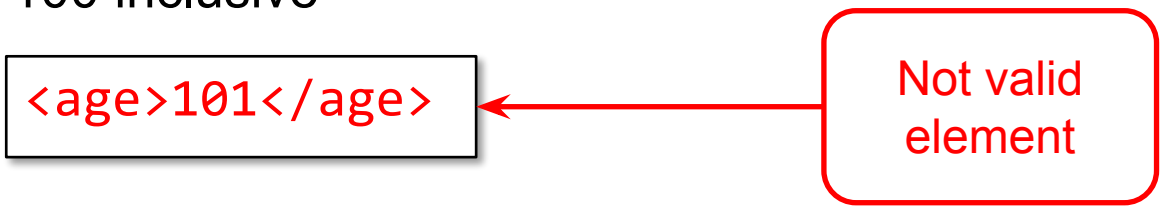
```
<xs:attribute name="lang" type="xs:string" default="EN"/>

<xs:attribute name="lang" type="xs:string" fixed="EN"/>
```

Simple Type

- SimpleType element defines a simple type elements or attributes, imposing restrictions or extensions on the basic types
- Can be named or anonymous within an element (attribute)

```
<xs:element name="age" type="ageType"/>

<xs:simpleType name="ageType">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="100"/>
  </xs:restriction>
</xs:simpleType>
```

- Describe the elements of "age" which can have numeric values from 0 to 100 inclusive



Complex Type

- Composite type describes the element that contains other elements and / or attributes
- The composite type is described by an element complexType
- Can be named or anonymous inside the cell

```
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Element simpleContent

- For expansion / composite type restrictions that may contain only the contents of the text element is used simpleContent
- Used to add attributes

```
<xsd:element name="shoesize">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:integer">
        <xsd:attribute name="country" type="xsd:string"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

Element complexContent

- ComplexContent element serves to expand or limit the types of compound, previously announced

```
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Types

- Types of elements / attributes can be
  - Local and anonymous (in the body of the element element)
  - Global and named (directly in the element schema)

```
<xs:simpleType name="newType">
    <xs:restriction base="xs:integer">
        <xs:minInclusive value="0"/>
        <xs:maxInclusive value="100"/>
    </xs:restriction>
</xs:simpleType>
<xs:complexType name="compType">
    <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="age" type="newType"/>
<xs:element name="note" type="compType"/>
```

Using of elements and attributes

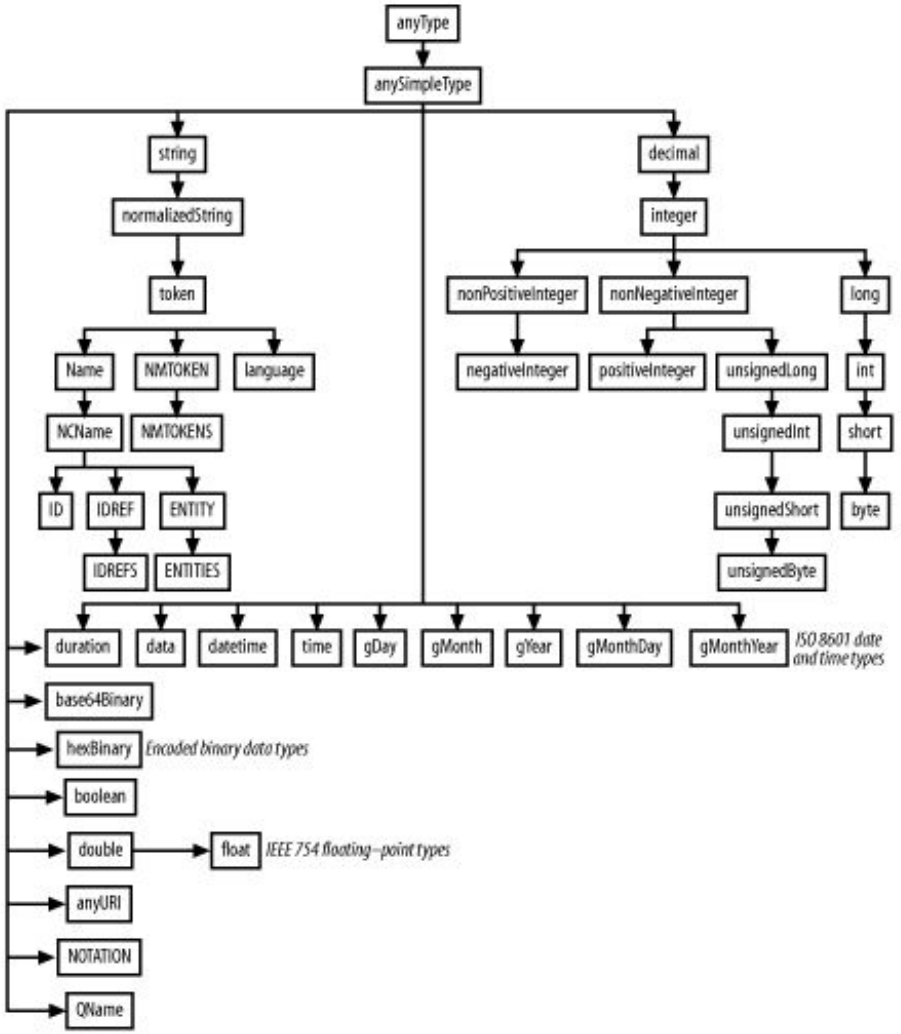
- On the named element and attribute declarations can be referenced using the ref attribute

```
<xs:attribute name="blue"/>
<xs:complexType name="eyeColorType">
    <xs:attribute ref="blue"/>
    <xs:attribute name="light"/>
</xs:complexType>

<xs:element name="eyeColor" type="eyeColorType"/>
<xs:element name="Catalog">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="eyeColor"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Base types

XML schema contains 44 base types. General:

- string
- decimal
- integer
- boolean
- date
- time



Restrictions

Restrictions used for control of possible values of elements and XML attributes.

Element	Description
enumeration	One of possible values from list
fractionDigits	Size of marks after dot (>=0)
length	Size of symbols (>=0)
maxExclusive	Max digital value exclusive
maxInclusive	Max digital value include
maxLength	Max number of symbols (>=0)
minExclusive	Min digital value exclude
minInclusive	Min digital value include
minLength	Min number if symbols
pattern	Regular exspressions
totalDigits	Number of digital

Restrictions for values

```
<xs:element name="age">
    <xs:simpleType>
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="100"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>

<xs:element name="car">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="Audi"/>
            <xs:enumeration value="Golf"/>
            <xs:enumeration value="BMW"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Digital value
$0 \leq age \leq 100$

List restriction

Template restriction

```
<xs:element name="letter">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:pattern value="[a-z]"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Restriction in one lower case letter

```
<xs:element name="initials">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:pattern value="[A-Z][A-Z][A-Z]"/>
        </xs:restriction>
   </xs:simpleType>
</xs:element>
```

Restriction in three upper case letters

Restriction for length

```
<xs:element name="password">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:length value="8"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>

<xs:element name="password">
    <xs:simpleType>

        <xs:restriction  base="xs:string">
            <xs:minLength value="5"/>
            <xs:maxLength value="8"/>
        </xs:restriction>

      </xs:simpleType>
</xs:element>
```

Value contains 8 symbols

Max and min size of symbols

White spaces restriction

- Restriction `whiteSpace` can have 3 values:
  - `preserve` – leave all white spaces as they are
  - `replace` – replace all such symbols with one
  - `collapse` – delete all white spaces before and after

```
<xs:element name="address">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:whiteSpace value="preserve"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Indicators

XML Schema has 7 element – indicators, which describes behavior of elements and attributes in XML

Indicators following:

- `all`
- `choice`
- `sequence`

Indicators input:

- `maxOccurs`
- `minOccurs`

Indicators group:

- `group`
- `attributeGroup`

Following indicators

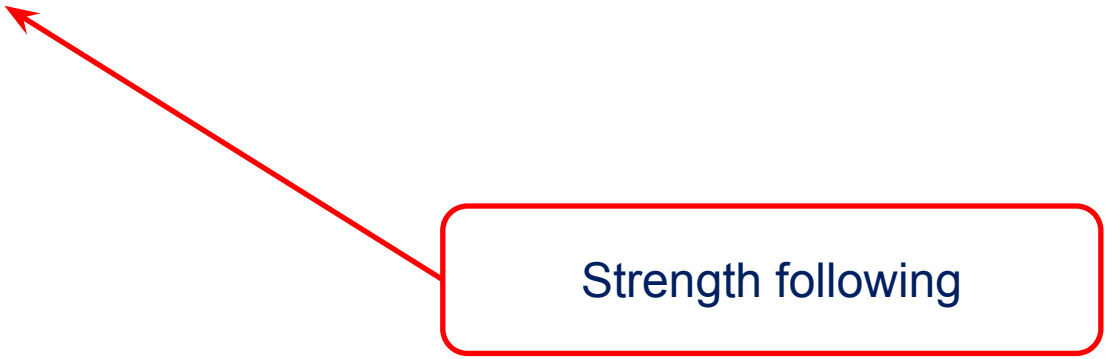
```
<xs:element name="person">
    <xs:complexType>
        <xs:all>
            <xs:element name="firstname" type="xs:string"/>
            <xs:element name="lastname" type="xs:string"/>
        </xs:all>
    </xs:complexType>
</xs:element>

<xs:element name="person">
    <xs:complexType>
        <xs:choice>
            <xs:element name="employee" type="employee"/>
            <xs:element name="member" type="member"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
```

Every following

Only one from elements

Following indicators

```
<xs:element name="person">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="firstname" type="xs:string"/>
            <xs:element name="lastname" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```



Input indicators

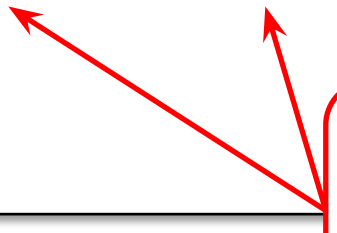
- For size of input of element
- Default - one.
  - minOccurs=1
  - maxOccurs=1

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
                  maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Indicators input

- Indicator `maxOccurs` can have min value – 10,`minOccurs` value 0
- For every count - `maxOccurs="unbounded"`

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
                  minOccurs="0" maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```



<child_name> can be missed, or follow after <full_name> max 10 time

Indicator group

- Binding of set of elements and attributes

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence>
</xs:group>
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:group ref="persongroup"/>
    <xs:element name="country" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="person" type="personinfo"/>
```

Element unique

- Attribute or element value should be unique

```
<unique id=ID name=NCName
  {any attributes with non-schema Namespace}...>

  Content: (annotation?, selector, field+)
</unique>
```

Unique – Example

```
<?xml version="1.0"?>
<cities ... >
  <city name="Milan"
    country="Italy"/>
  <city name="Paris"
    country="France"/>
  <city name="Munich"
    country="Germany"/>
  <city name="Lyon"
    country="France"/>
  <city name="Venice"
    country="Italy"/>
  <city name="Venice"
    country="Italy"/>
</cities>
```

```
<xs:element name="cities">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="e:city"
        maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:unique name="cityName">
    <xs:selector xpath="e:city"/>
    <xs:field xpath="@name"/>
  </xs:unique>
</xs:element>
```

Not according to the rule

Schema documentation

- Can us simple XML comments

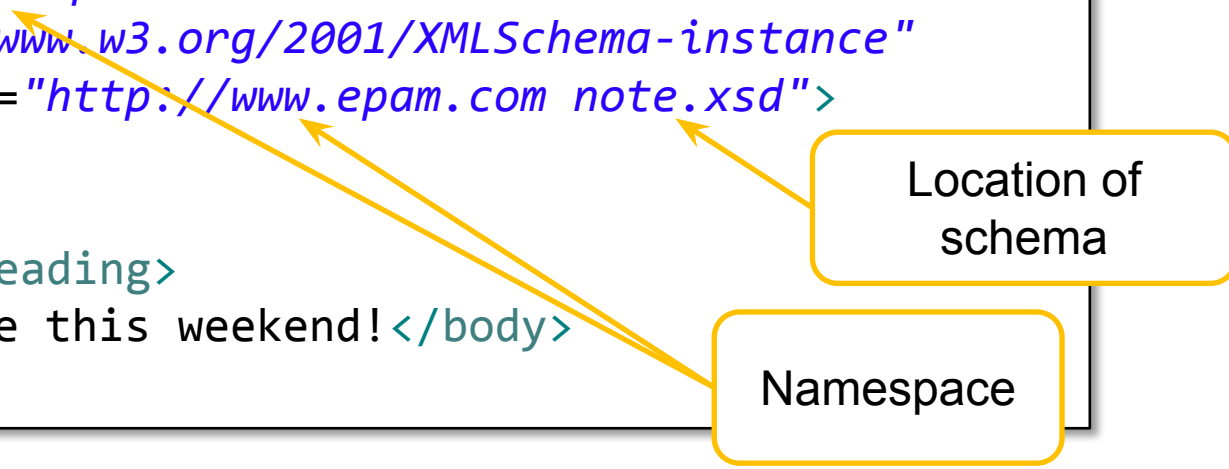
```
<!-- This is a comment -->
```

- Annotations can appear everywhere in schema
  - <annotation> - parent for appinfo> and <documentation>
  - <appinfo> - info for external apps
  - <documentation> - comments for developers

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:appinfo>Schema for processing submitted applications</xs:appinfo>
    <xs:documentation>Submitted for Human Resources</xs:documentation>
  </xs:annotation>
  <xs:element name="relocationStatus">
    <xs:complexType>
      <xs:annotation>
        <xs:appinfo>No further processing is required.</xs:appinfo>
        <xs:documentation>This element is for determining relocation
status to determine help determine costs.</xs:documentation>
      </xs:annotation>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Describe ref to XSD in XML

```
<?xml version="1.0"?>
<note xmlns="http://www.epam.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.epam.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```



- **xmlns="http://www.epam.com"** - default namespace

- **xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**
- **xsi:schemaLocation="http://www.epam.com note.xsd"** - show which schema is correspond to current XML and where is present schema for validation of current XML

Describing of ref to XSD in XML

- xsi:noNamespaceSchemaLocation is used, when namespace is not used

```
<book xsi:noNamespaceSchemaLocation="http://www.epam.com/epam.xsd">
  <title>Presenting XML</title>
  <author>Richard Light</author>
</book>
```

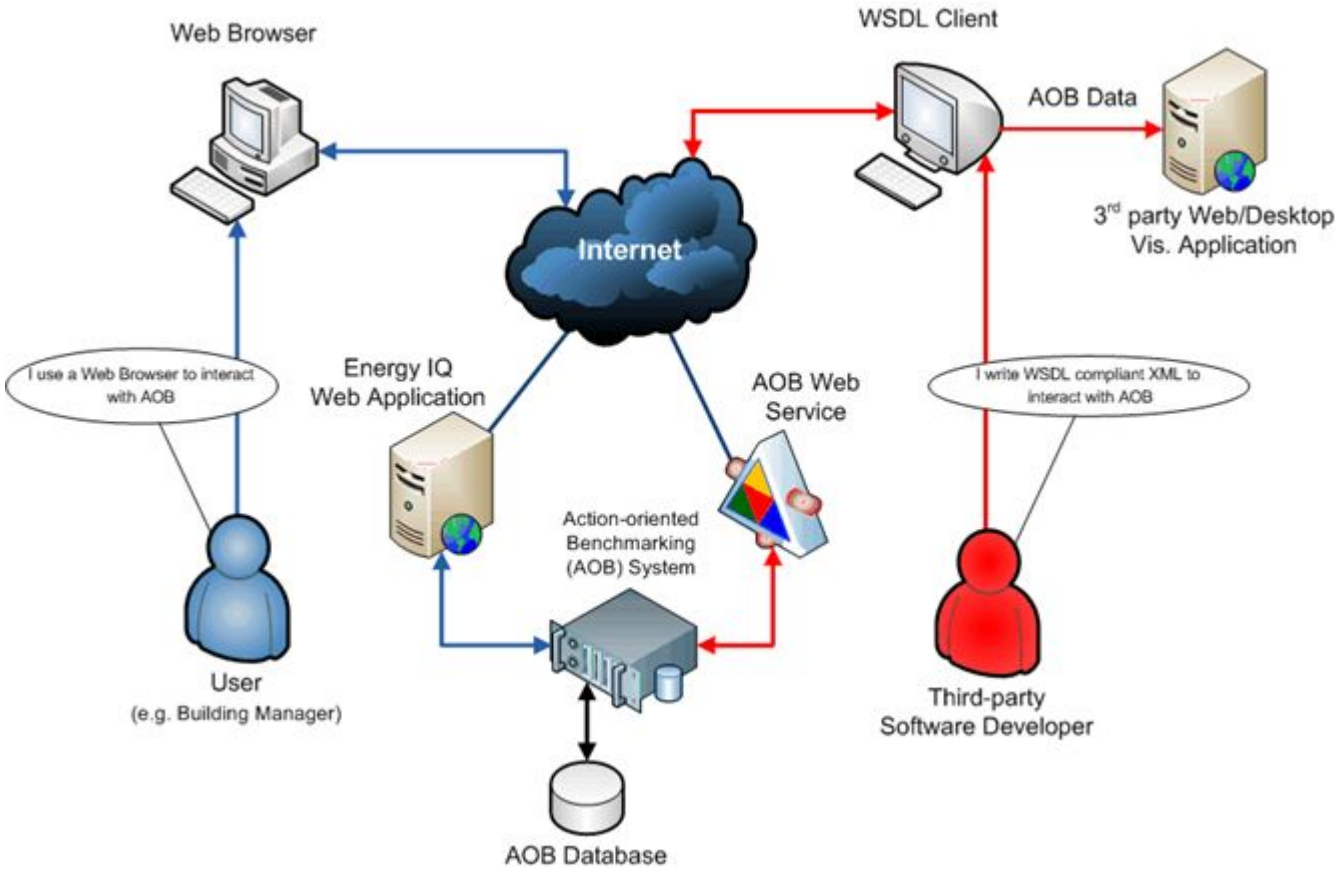
- Можно использовать URL файловой системы

```
xsi:noNamespaceSchemaLocation="file:///d:/xml/schemas/epam.xsd"
```

- xsi:schemaLocation is used, when prefix of spaces is described and used

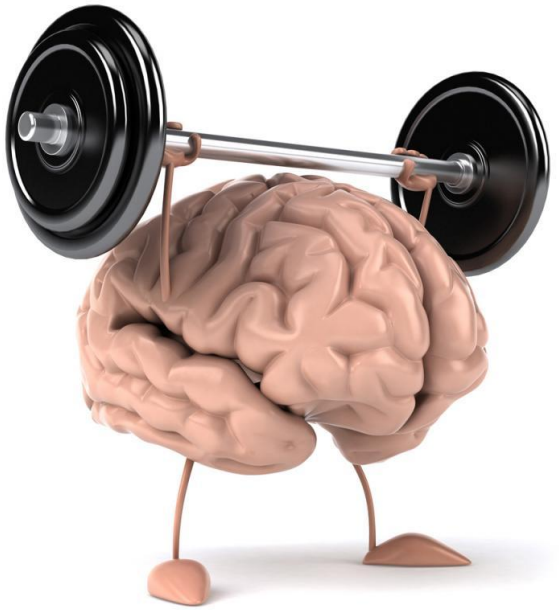
```
<data:catalog xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:data="http://www.epam.com"
  xsi:schemaLocation="http://www.epam.com
    http://www.epam.com/epam.xsd">
  ...
</data:catalog>
```

What was learnt for today?



What was learnt for today?

- Web application / Web site / Web Server
- Web service
- XML and how to create XML
- XSD and how to create XSD



Thank you for your attention!

Anastasiya Babovich
Anastasiya_Babovich@epam.com