XML in Programming 2, Web services

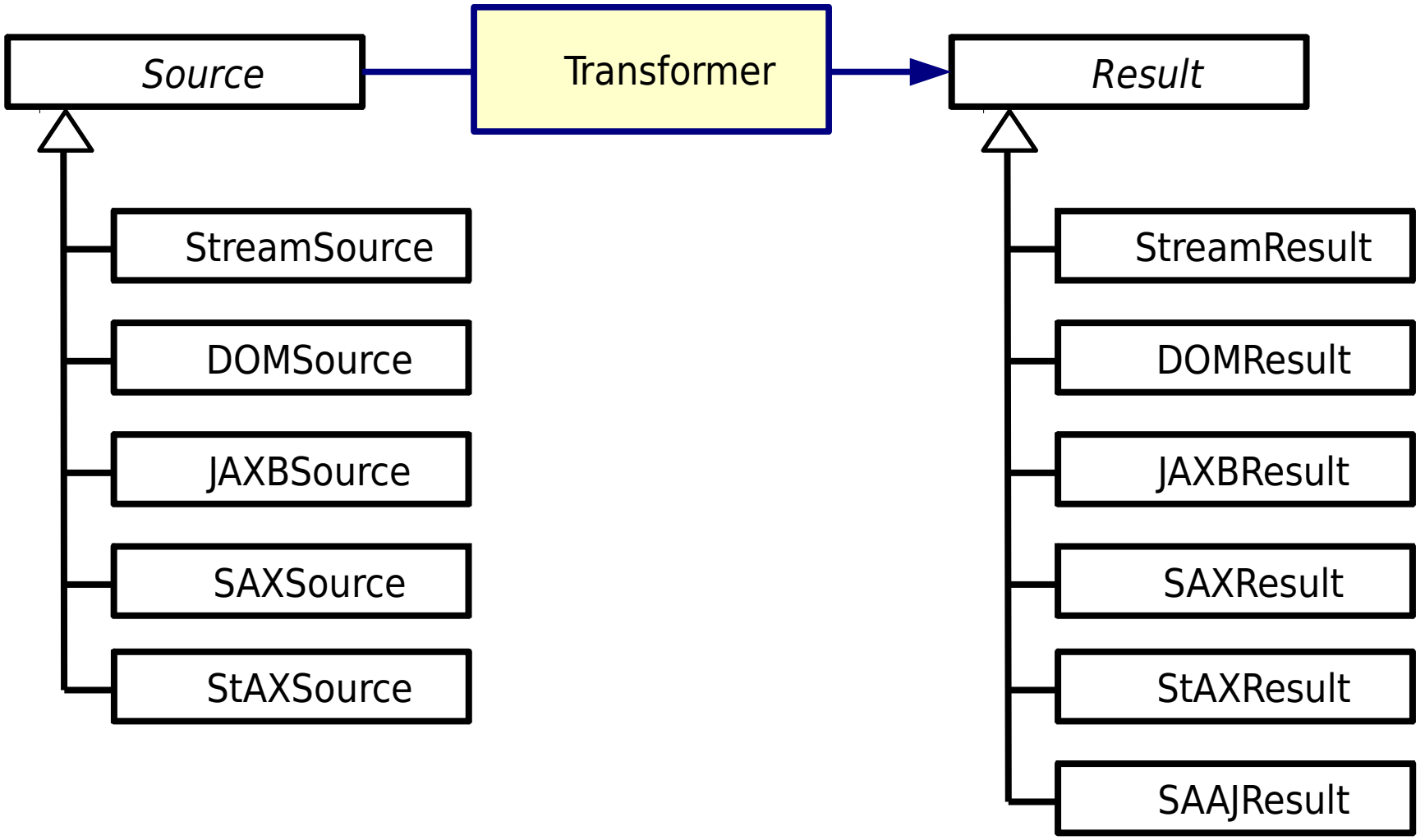
Patryk Czarnik

XML and Applications 2013/2014
Lecture 5 – 4.11.2013

Features of JAXP

- 3 models of XML documents in Java: DOM, SAX, StAX
  - Formally JAXB is a separate specification
- Reading and writing documents
- Transformations of XML documents (Transformers)
  - applying XSLT in our programs
  - translating internal form of representation
- XPath support
- Validation
  - against DTD (only during parsing)
  - against XML Schema (during parsing or using Validators)
  - against XML Schema 1.1, Relax NG, or other alternative standards – when implementation supports

Transformer: source and result



Applications of Transformers

- Simple:
  - invoking XSLT transformations from Java
  - changing internal representation of XML in our program
- Tricky:
  - parsing and writing documents, e.g. serialisation of a DOM tree
  - serialisation of modified (or generated) sequences of SAX events
  - (together with SAX filters) enabling “on-the-fly” processing of large XML documents

Editing XML documents

- More natural when whole document present in memory
  - DOM – generic API
  - JAXB – deep embedding of XML in application model
- Harder, but possible, using node-by-node processing
  - required when processing big documents within small amount of memory
  - suggested for big (“long and flat”) documents and simple local operations – then we can save substantial resources
  - StAX – possible using “writers”
    - IMO XMLEventWriter more convenient than XMLStreamWriter
  - SAX
    - no direct support for editing/writing
    - available indirect solution: SAX filters and Transformer

Editing XML documents – examples

Validation

- Against DTD
  - setValidating(true) before parsing
- Against XML Schema (or other schema formats, if implementation supports)
  - setSchema(schema) before parsing
  - using Validator
- Validator API
  - validate(Source) – only checking of correctness
  - validate(Source, Result) – augmented document returned
    - not possible to use as Transformer – source and result must be of the same kind
    - (my private observation) – not always working as expected

Handling errors

- Most JAXP components (specifically SAX and DOM parsers, Validators)
  - may throw SAXException
  - signal errors through ErrorHandler events
- Interface ErrorHandler
  - 3 methods (and severity levels): warning, error, fatalError
  - registering with setErrorHandler allows to override default error handling
- Required to manually handle validation errors

Validation – examples

XPath support in Java

- DOM XPath module implementation
  - org.w3c.dom.xpath
  - officially not a part of Java SE, but available in practice (by inclusion of Xerces in Oracle Java SE runtime)
- JAXP XPath API
  - javax.xml.xpath
  - most efficient when applied for documents in memory (DOM trees)
  - our examples show this solution
- Note: using XPath may significantly reduce developer's work, but the application may be less efficient (than if we used SAX, for example)

Electronic data interchange (EDI) – motivation

- How to interchange data between companies / institutions (B2B)?
  - paper
  - electronic data interchange
- How to establish EDI protocol?
  - customer receives (or buys) a tool from provider
  - smaller partner complies to bigger parter
  - ad-hoc created conversion tools
  - **standard**
- Standard deployment levels
  - software developed according to standard from beginning
  - interface added to legacy system

Pre-XML solutions

- ANSI Accredited Standards Committee X12 sub-group
  - USA national standard
  - used mainly in America
- EDIFACT
  - international standard (UN/CEFACT and ISO)
  - used mainly in Europe and Asia

EDIFACT characteristic

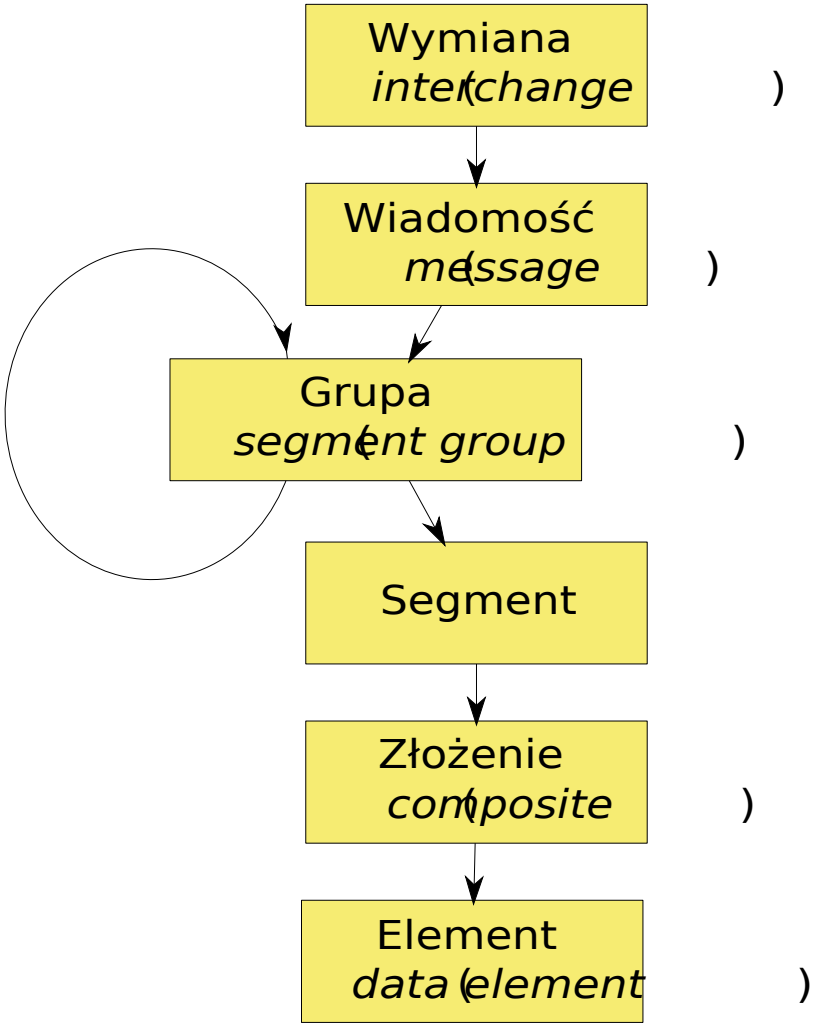
- Format
  - text
  - hardly readable
  - tree structure
- Predefined dictionaries
- 193 message types
- 279 segments
- 186 elements
- (counted for version 08a, 2008)

EDIFACT

- EDIFACT message example

```
UNB+IATB:1+6XPPC+LHPPC+940101:0950+1'
UNH+1+PAORES:93:1:IA'
MSG+1:45'
IFT+3+XYZCOMPANY AVAILABILITY'
ERC+A7V:1:AMD'
IFT+3+NO MORE FLIGHTS'
ODI'
TVL+240493:1000::1220+FRA+JFK+DL+400+C'
PDI++C:3+Y::3+F::1'
APD+74C:0:::6++++++6X'
TVL+240493:1740::2030+JFK+MIA+DL+081+C'
PDI++C:4'
APD+EM2:0:1630::6+++++++DA'
UNT+13+1'
UNZ+1+1'
```

EDIFACT structure




```
MEA+WT+AAD+KGM:690+X5
```

+ KGM:690 +

: 690

XML EDI

Idea: use XML as data format for EDI

- Traditional EDI
  - Documents unreadable without specification
  - Compact messages
  - Centralised standard maintenance
  - Changes in format requires software change
  - Specialised tools needed
- XML EDI
  - “Self-descriptioning” documents format
  - Verbose messages
  - “Pluggable”, flexible standards
  - Well written software ready to extensions of format
  - XML-format layer handled by general XML libraries

XML EDI flexibility

- Format flexibility
  - Structures: choosing, repeating, nesting, optionality
  - Format extensions and mixing via namespaces
- Applications
  - Data interchange between partners' systems
  - Web interface (with little help from XSLT)
- Web Services integration

XML EDI standardisation

- Framework level
  - general rules for all kinds of data
  - data of the same kind should be represented in the same way
    (not to define the same twice)
  - example: Electronic Business XML (**ebXML**).
- Industry standards (examples)
  - banking
  - trade and logistic
  - Automotive Industry Action Group – motor industry (mainly American)
  - Health Level Seven – health care
  - Open Travel Alliance – (people) transport and tourist services

XML for application integration

- Goal – data interchange between applications
  - applications/modules/components with different internal formats
  - XML as interface
- Usage:
  - client/server communication
  - nodes of distributed systems
  - components integration
  - remote configuration and monitoring of applications

Local and global applications

- “Local” integration
  - within single project or related projects of a single owner
  - communication between components
  - possibly in distributed architecture
  - ad-hoc solutions for given problems
  - possibility of using standard
- “Global” integration
  - services available in Internet for any party
  - different parts cooperation
  - standardisation required
  - motivation to use Web Services

Web Services

- Idea: a website for programs (instead of people)
- General definition
  - communication based on high-level protocols
  - structural messages
  - services described
  - searching services
- Concrete definition: “Classical” Web-Services
  - HTTP or other protocols
  - SOAP
  - WSDL
  - UDDI

Web Services – typical applications

- Providing data (for free or paid)
  - timetables
  - weather
  - stock and currency notes
- Services
  - searching
  - software updates
- Business operation between partners
  - booking tickets or hotel rooms
  - ordering (and tracing order status)
  - electronic data interchange
- e-Administration

Web Services standardisation

- SOAP (initially Simple Object Access Protocol:
  - beginnings: 1998
  - v1.1: W3C Note, 2001 (still in use)
  - v1.2: W3C Recommendation, June 2003 (also used)
- Web Services Description Language:
  - W3C Note, 2001 (most applications use this version!)
  - v2.0: W3C Recommendation, June 2007
- Universal Description Discovery and Integration:
  - OASIS project

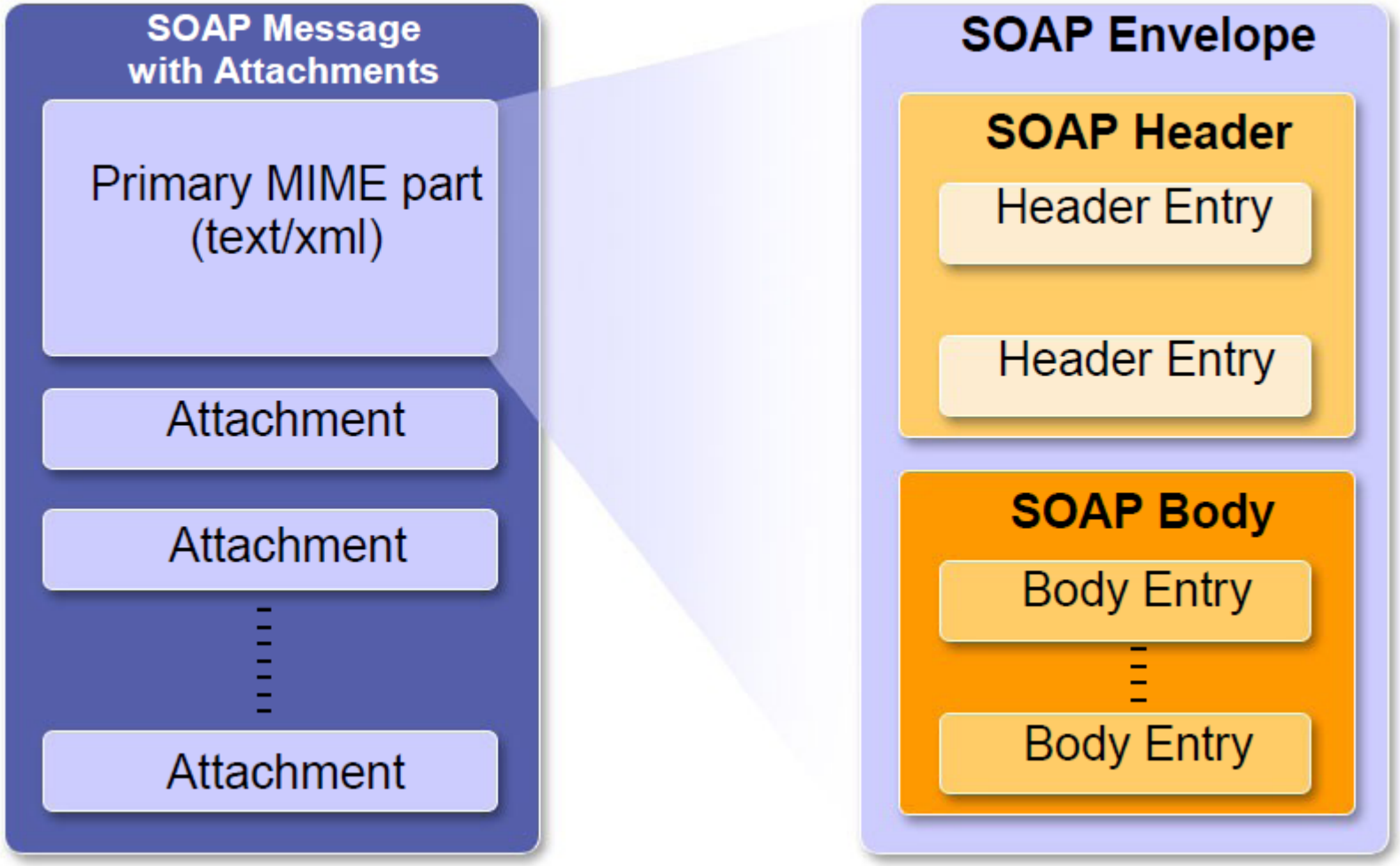
Web Services standardisation (2)

- Web Services Interoperability – levels of WS compliance:
  - WS-I Basic Profile, Simple Soap Binding Profile, ...
- WS-* standards: various standards, usually not W3C:
  - WS-Eventing, WS-Addressing, WS-Routing, WS-Security
- Business Process Execution Language (OASIS) – WS semantics description, programming using WS as building blocks

SOAP – communication protocol

- Built on top of existing transport protocol (HTTP or other)
- Message format
  - main message part – XML
  - envelope and some special elements defined in standard
  - implementation-dependent content
  - additional attachments in any format (even binary)
- Differences to RPC, CORBA, DCOM etc.:
  - data represented in extensible, structural format (XML)
  - data types independent of platform (XML Schema)
  - lower efficiency

SOAP message – general form



SOAP 1.2 message

```
<soap:Envelope
      xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
      soap:encodingStyle="http://www.w3.org/2001/12/literal">

   <soap:Header>
      <t:Trans xmlns:t="http://www.w3schools.com/transaction/"
         soap:mustUnderstand="1">234</m:Trans>
   </soap:Header>

   <soap:Body>
      <m:GetPrice xmlns:m="http://www.w3schools.com/prices">
         <m:Item>Apples</m:Item>
         <m:Currency>PLN</m:Currency>
      </m:GetPrice>
   </soap:Body>
</soap:Envelope>
```

SOAP 1.2 – normal response

```
<soap:Envelope
      xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
      soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
   <soap:Body>
      <m:GetPriceResponse xmlns:m="http://www.w3schools.com/prices">
         <m:Price>1.90</m:Price>
         <m:Currency>PLN</m:Currency>
      </m:GetPriceResponse>
   </soap:Body>
</soap:Envelope>
```

SOAP 1.2 – fault response

```
<soap:Envelope xmlns:usos="urn:USOS"
      xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
      soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
   <soap:Body>
      <soap:Fault>
         <soap:faultcode>soap:Receiver</soap:faultcode>
         <soap:faultstring>Data missing</soap:faultstring>
         <soap:faultdetail>
            <usos:exception>Found no student identified
               with <usos:ind>123</usos:ind>
            </usos:exception>
         </soap:faultdetail>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

SOAP – more info

- Request and response have the same structure.
  - In fact, we can think of SOAP as a document transport protocol, not necessarily in client-server architecture.
- Header part optional, Body part required.
- Restrictions on XML part:
  - no DTD (and external entity references),
  - no processing instructions.
- Although SOAP allows many body elements (elements within soap:Body), WS-I BP requires exactly one.
  - To make applications portable we should follow this restriction.

WSDL – service description

- XML document describing a service
- Interface (“visit card”) of a service (or set of services)
- Specifies (from abstract to concrete things)
  - XML types and elements (using XML Schema)
  - types of messages
  - port types – available operations, their input and output
  - details of binding abstract operations to a concrete protocol (SOAP in case of “classical” services)
  - ports – concrete instances of services, with their URL
- Splitting definitions into several files and using external schema definitions available

WSDL 1.1 structure

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name='HelloWorldService'
   targetNamespace='http://example.com/hello'
   xmlns='http://schemas.xmlsoap.org/wsdl/'
   xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
   xmlns:tns='http://example.com/hello'
   xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
 <types>
.............
 </types>
 <message name='HelloWorld_sayHello'>
.............
 </message>
 <message name='HelloWorld_sayHelloResponse'>
.............
 </message>
 <portType name='HelloWorld'>
.............
 </portType>
 <binding name='HelloWorldBinding' type='tns:HelloWorld'>
.............
 </binding>
 <service name='HelloWorldService'>
  <port binding='tns:HelloWorldBinding' name='HelloWorldMyPort'>
.............
  </port>
 </service>
</definitions>
```

Elementy i typy XMLSchema

Komunikaty – parametry wejściowe i rezultat

Interfejs określający udostępniane operacje

Wiązanie z konkretnym protokołem

Definicja usługi sieciowej

WSDL and SOAP interaction

- Basically – specified through binding element in WSDL
  - not so simple, because of many possibilities
- RPC style
  - SOAP XML structure derived basing on operation name and message parts
- Document style
  - theoretically designed to allow sending arbitrary XML documents
  - in practice also used for RPC realisation, but the author of WSDL has to define an appropriate document structure
    - (some tools may be helpful, e.g. bottom-up service generation in Java JAX-WS)
- Message use: literal or encoded.
  - We should use literal in modern applications.

Service registration and discovery

- Idea
  - service provider registers service
  - user searches for service and finds it in registry
- Universal Description Discovery and Integration (UDDI)
  - available as service (SOAP)
  - business category-based directory (“yellow pages”)
  - searching basing on service name, description (“white pages”)
  - registration and updates for service providers

UDDI – issues

- Main issue – who can register?
  - anybody – chaos and low reliability
  - accepted partners – institution responsible for access policy needed, no such (widely accepted) institution exists
- Reality
  - UDDI rarely used
  - if ever – for “local” SOA-based solutions (intranets)

Service Oriented Architecture

- Idea
  - services built basing on other services
  - even addition defined as a Web Service :)
  - software split into components and layers with WS interfaces between them
  - precise specification required (interesting research field...)
- Critique
  - modular, flexible, and scalable solutions
- by the cost of (sometimes) irrational inefficiency and complexity
- **Use reasonably!**

Are Web Services good or bad?

- Web Service recommended when
  - Many partners or public service (standardisation)
  - Heterogeneous architecture
  - Text and structural data already present in problem domain
  - Interoperability and flexibility more important than efficiency
- Web Service?... not necessarily
  - Internal, homogeneous solution.
  - Binary and flat data
  - Efficiency more important than interoperability and flexibility