

Web services




Patryk Czarnik

XML and Applications 2016/2017
Lecture 6 – 7.04.2017

Motivation for web services

- Electronic data interchange
- Distributed applications
 - even as simple as client / server
- Interoperability and flexibility
 - need for (high-level and device-independent) standards
- Service Oriented Architecture

Evolution of internet applications

- human  human
 - email
 - web sites written manually
- application  human
 - web applications (e.g. an internet shop)
- application  application
 - “electronic data interchange”
 - low-level technologies and ad-hoc solutions
 - pre-XML standards (e.g. EDIFACT)
 - “web services”
 - REST, AJAX, etc.

Electronic data interchange (EDI) - motivation

- How to exchange 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 partner
 - 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

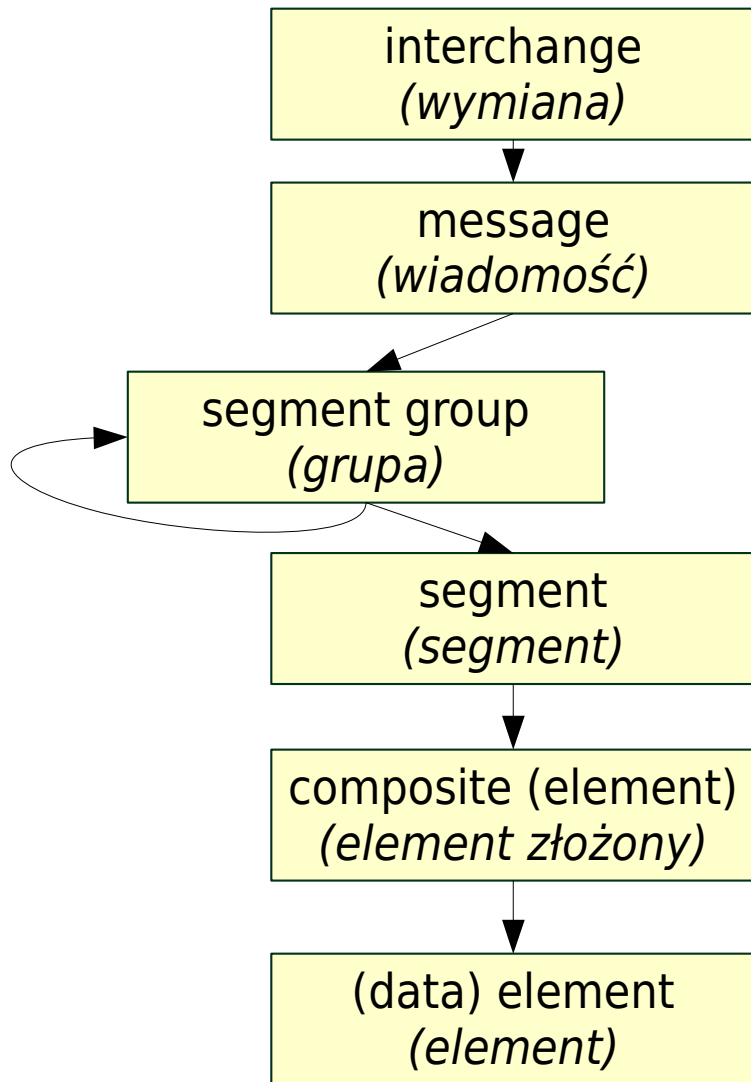
(counted for version 08a, 2008)

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



TVL+240493:1000:::1220+FRA+JFK+DL+400+C '
PDI++C:3+Y:::3+F:::1 '
APD+74C:0:::6+++++6X'

TVL+240493:1000:::1220+FRA+JFK+DL+400+C'

+240493:1000:::1220+

:1000:

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

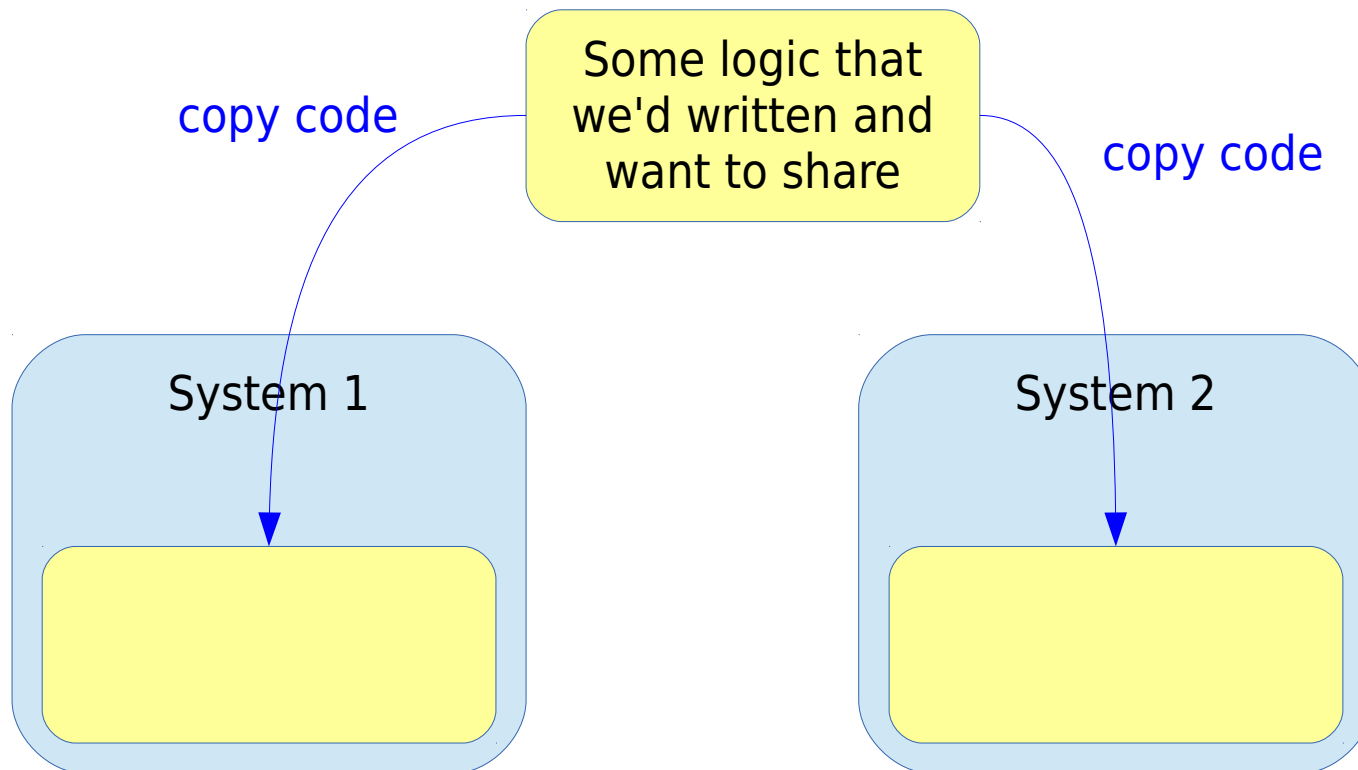
- Format flexibility
 - Structures: choosing, repeating, nesting, optionality
 - Format extensions and mixing via namespaces
- Applications
 - Data interchange between partners' systems
 - Web interface (with little help from XSLT)
- Web Services integration

Service Oriented Architecture (SOA)

- Build software modules as **services**
 - available for other services and programs
 - using other services when necessary
- Share working services, not code
- It is a general characteristic, it does not require to use any particular type of services or standards.
 - However, standardisation helps to prepare *interoperable* services which are independent of hardware and software architecture.

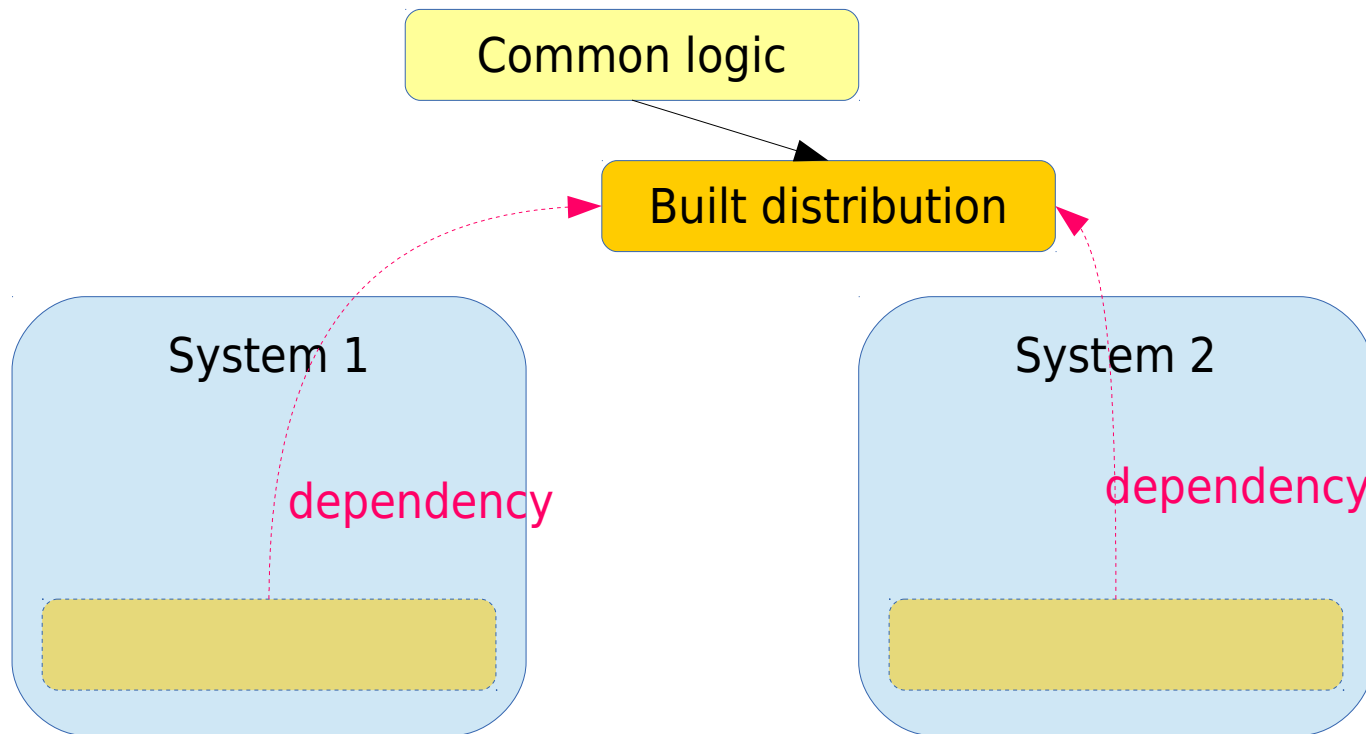
Bad approach to share systems logic

- Write the same thing twice
- Copy & paste source code
 - Different systems include the same source code; at best – synchronised to a common repository



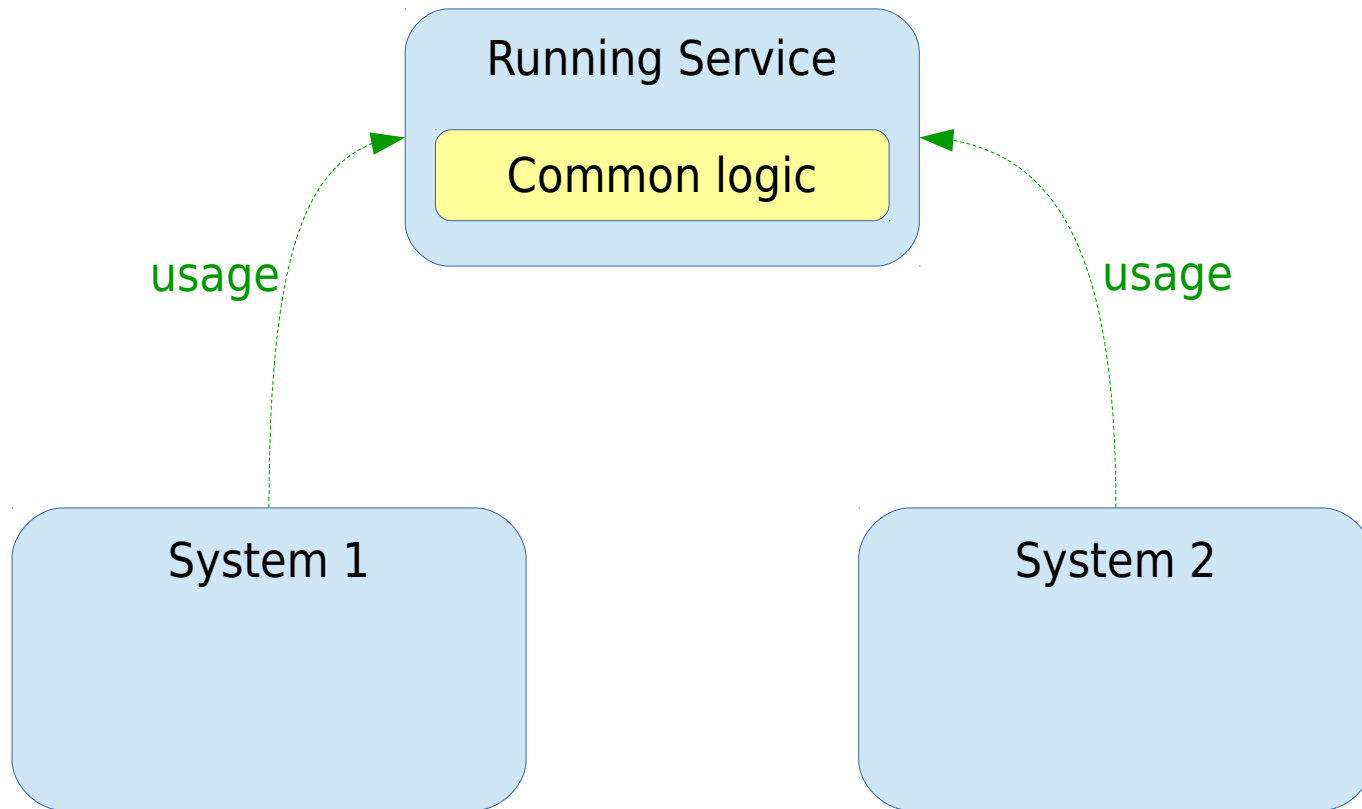
Traditional *component* approach

- Build software components that can be used **within** different software systems via their public API.
 - Usually realised as dependencies (e.g. Maven for Java) and sharing of compiled software libraries



Service approach

- Expose the common logic as a running service and make other systems use it.
 - Only one running instance of the logic, no copying.



Benefits and costs of SOA

- Systems are lighter (at least in theory) as they do not duplicate logic.
- A (well designed) service available for different client platforms (*interoperability*).
- One place of (that "logic") management:
 - bug fixes and enhancements,
 - security, access control.

But...

- In practice, standard-based web services adds a notable cost:
 - weight of libraries required by systems
 - runtime cost of data conversion

Web Services

Basic concept: *a website for programs* (rather than people)

- General definition
 - communication based on high-level protocols
 - structural messages
 - services described and available for clients via well defined "API"

	(server-side) web application	web service
used by	human with web browser	client applications
communication	HTTP	HTTP, occasionally other protocols (e.g. JMS)
data	HTML, multimedia, styles	"pure data": XML, JSON, occasionally plain text, ...

Common approaches to web services

- Non-standard custom solutions – always possible, but contrary to standardisation postulate
- Outdated standardised solutions:
 - XML-RPC, XML over HTTP, ...
- "Classic" web services
 - SOAP (XML) and WSDL
 - document interchange or remote procedure calls
 - enterprise and administration applications, B2B EDI
- REST-ful services
 - HTTP, different data formats (but most often JSON)
 - remote data access
 - internet services, "web APIs", support for JavaScript web applications and mobile apps, ...

Web Services standardisation

"Classic" web services

- 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:
 - registration and searching of services
 - rarely used, but always enumerated in such lists :)

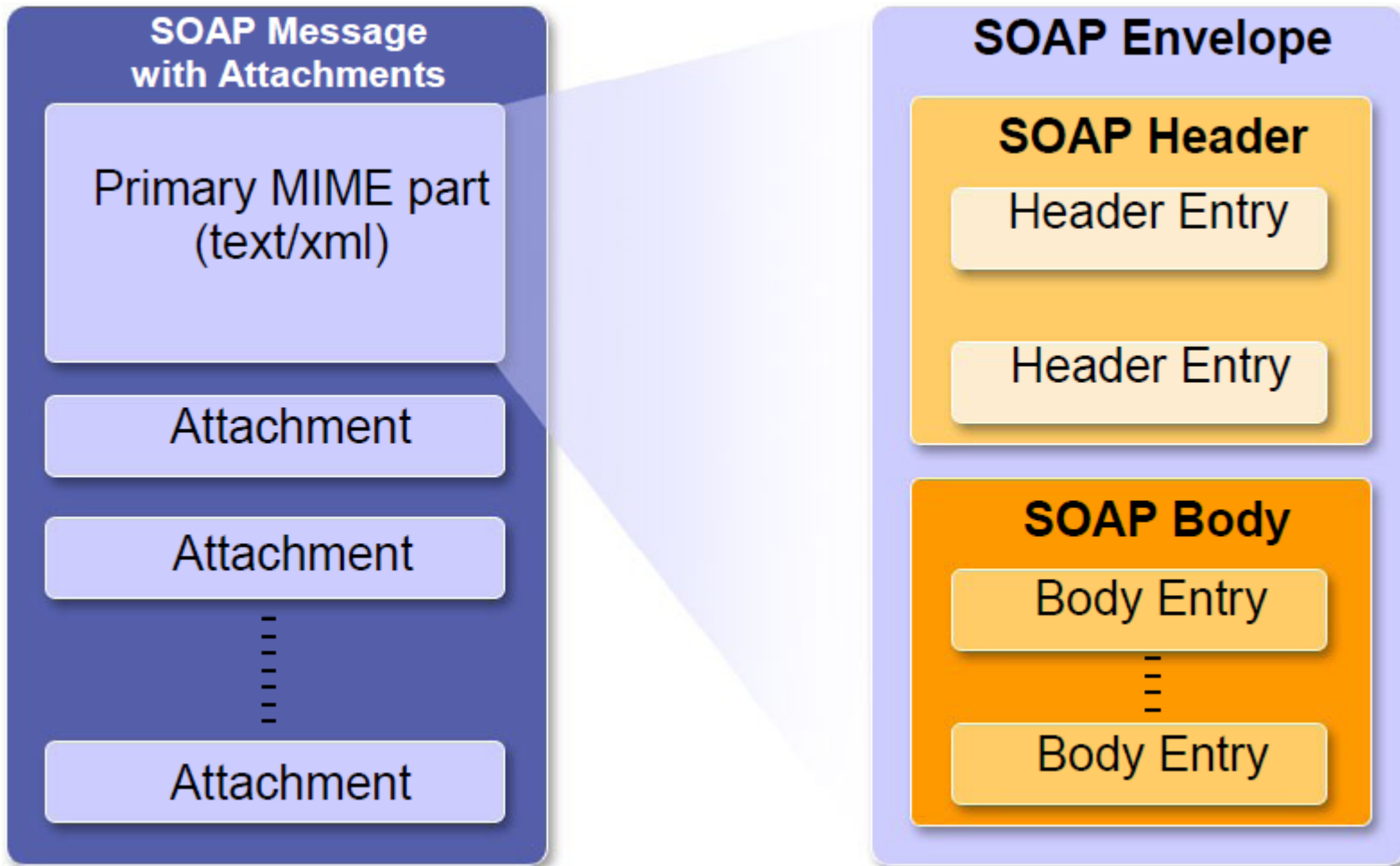
Web Services standardisation (2)

- Web Services Interoperability
 - restricts general WSDL and SOAP standards to create a common base for "proper" services
 - 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

SOAP – communication protocol

- Built on top of existing transport protocol (HTTP or other)
- Message format
 - XML message with optional binary attachments
 - headers (optional XML elements) and body content
 - envelope and some special elements defined in standard
 - implementation-dependent content
- 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>
```

XML Schema element and type defs.

Message defs.

The interface (set of operations)

Binding with a particular protocol, usually SOAP

Web Service instance (concrete URL given)

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 content enclosed in XML documents
 - in practice - also used for RPC realisation, but the author of WSDL has to define the 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 registries
 - service providers register their services
 - clients search for services and find them in registries
- 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 – an institution responsible for access policy needed, no such (widely accepted) institution exists
- Reality
 - UDDI rarely used
 - if ever – for “local” SOA-based solutions (intranets)

Web Services advantages and problems

- Advantages:
 - Standardised, platform-independent technology
 - Interoperability
 - Existing tools and libraries
- Main drawbacks:
 - Inefficiency
 - size of messages → transfer, memory usage
 - data representation translated many times on the road from client to server (and vice versa) → processor usage / time
 - Complex standards, especially when using something more than raw WSDL+SOAP

Are Web Services good or bad?

- SOA and Web Services give an opportunity to build
 - modular, flexible, and scalable solutions
 - (sometimes) by the cost of irrational inefficiency and complexity
- 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

Web services in Java

Basically – web services and web service clients can be built from scratch in any technology

- but it would be the same mistake as reading XML documents char by char.
- Low-level technologies:
 - HTTP servlets and HTTP clients supported by XML processing APIs (DOM, SAX, StAX, JAXB, Transformers, ...)
 - SOAP with Attachments API for Java (**SAAJ**)
 - extension of DOM directly supporting SOAP
- High level approach (with low level hooks available):
 - Java API for XML Web Services (**JAX-WS**)

Web services in Java

- WS support (XML APIs, SAAJ, JAX-WS) present in Java SE
 - JAX-WS and some of XML APIs since version 6.0
- Client side:
 - Possible to develop and run WS client in Java SE without any additional libraries!
- Server side:
 - Developing and compiling WS server (without any vendor-specific extensions) available in Java SE
 - Running a service requires an application server and a WS implementation
 - “Big” app servers (Glassfish, JBoss, WebLogic...) have preinstalled WS implementations
 - Lightweight servers (e.g. Tomcat) can be used by applications equipped with appropriate libraries and configuration

SAAJ

- Package `javax.xml.soap`
- Main class – `SOAPMessage`
- Tree-like representation of SOAP messages
 - extension of DOM
 - easy access to existing and building fresh SOAP messages
 - support for HTTP headers, binary attachments, ...
- Easy sending of requests from client side
 - see example `Client_Weather_SAAJ`
- Possible implementation of server side as a servlet
 - see example `Server_SAAJ`

JAX-WS – introduction

- Annotation-driven
- Uses JAXB to translate Java objects to/from XML
- Central point: Service Endpoint Interface (SEI)
 - Java interface representing a WS port type
 - `kalkulator.Kalkulator` and `pakiet.Service` in our examples
- Translation between web services world (WSDL) and Java
 - **top-down**: from WSDL generate Java
 - server side – service interface and implementation skeleton
 - client side – proxy class enabling easy remote invocations
 - both sides – auxiliary classes, usually JAXB counterparts of XML elements appearing in messages
 - **bottom-up**: from Java code generate WSDL (and treat the Java code as a WS implementation)
 - usually done automatically during application deployment

Advantages and risks of using JAX-WS

- High level view on web service
 - details of communication and SOAP/XML not (necessarily) visible to a programmer
 - proxy object on client side enables to transparently invoke methods on server-side just like on local objects
- Automatic generation/interpretation of WSDL
 - conformance to WSDL controlled by system
- Bottom-up scenario – easy introduction of WS interface to already existing systems
 - or for programmers not familiar with WSDL/XML details
- Risk of
 - accidental service interface (WSDL) (automatically generated, not elaborated enough)
 - inefficiency

JAX-WS – main elements

- Class level annotations:
 - `@WebService`, `@SOAPBinding`
- Method-level annotations:
 - `@WebMethod`, `@OneWay`, `@SOAPBinding`,
`@RequestWrapper`, `@ResponseWrapper`
- Parameter-level annotations:
 - `@WebParam`
 - `@WebResult` (syntactically a method annotation, applies to what the method returns)
- Support for specific technologies
 - `@MTOM` – automatically created binary attachments
 - `@Addressing` – adds WS-Addressing headers

JAX-WS – low level hooks

- Providers – low level server side
 - Useful when
 - high efficiency required (e.g. streaming processing)
 - XML technology used in implementation
- Dispatch – low level client side
- One way methods
- Asynchronous client calls
- Handlers and handler chains
 - additional processing of messages between client and server logic
 - one place to perform common logic: logging, authentication, session binding

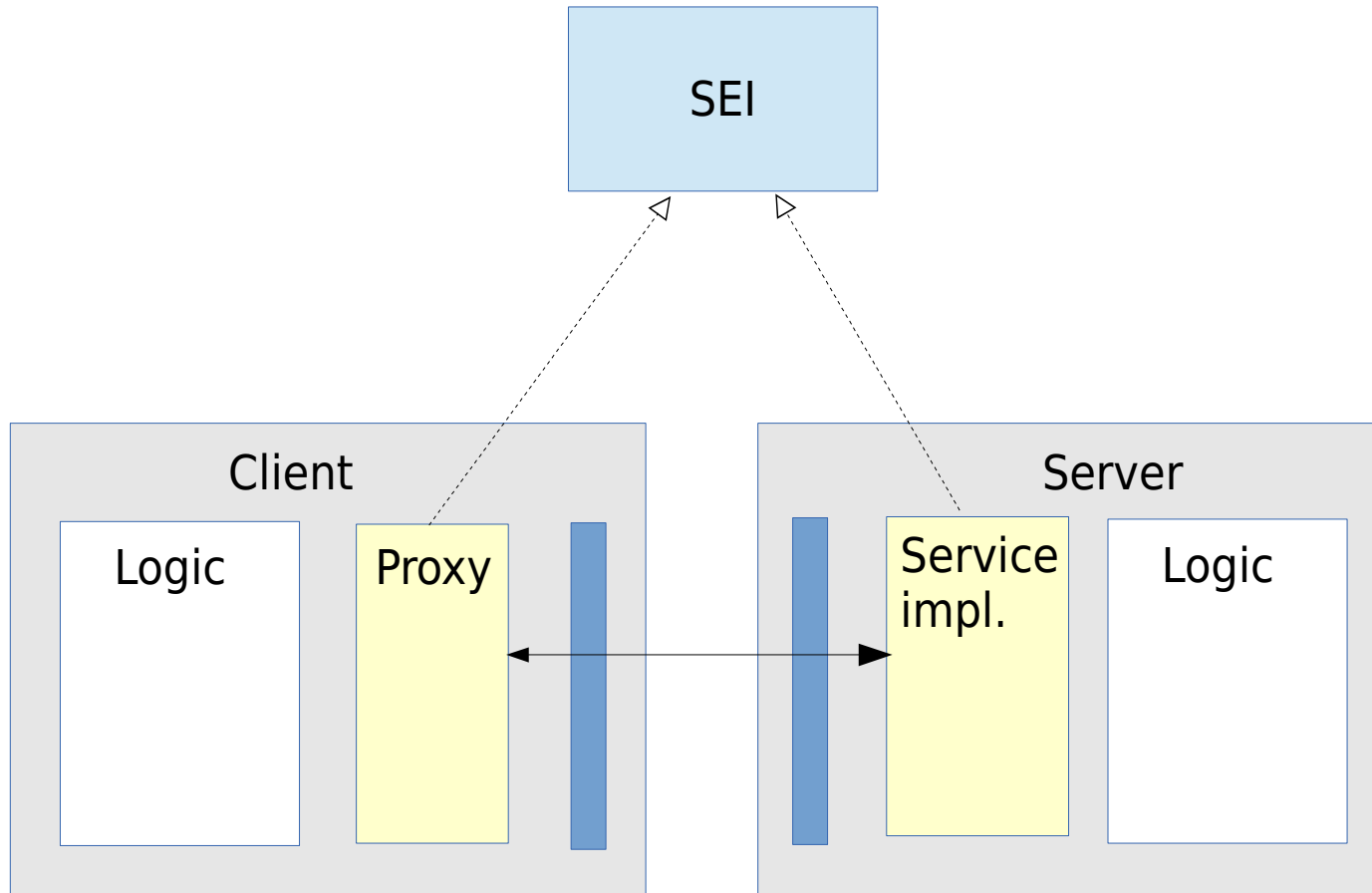
JAX-WS examples

Details to note:

- top-down (**Kalkulator**):
 - (different) form of WSDL in RPC and Document styles
 - 3 ways WSDL can be translated to Java (and SOAP) (RPC, document-wrapped, document-bare)
 - **@WebService** annotation in implementation class
- bottom-up (**Hello**)
 - how annotations affect SOAP messages (and WSDL)
 - how Java objects are represented in SOAP messages (JAXB)
- high level proxy clients (**Client_Weather_JAXWS**)

JAX-WS architecture

When both sides written in Java...



High level Java clients available
also for non-Java servers!