# XML, JSON & NoSQL Databases

Gilles Degols
based on the initial work of Ken Hasselmann

# Course organization

- XML, JSON
  - Theory and exercises
- NoSQL Databases
  - Theory
  - Developing a small python application iteratively through exercises
- Not everything will be written in the slides
  - If you do not come to the class, you will miss some information needed for the evaluations
  - Slides can be updated at any time, as well as the project/exercises deliverables (communicated orally)
  - https://github.com/gilles-degols/ecam-nosql
- Deliverables: must be in English

# Evaluation - Exercises

- XML, JSON (15%)
  - Submit exercises the next day of each related course (23:59)
- NoSQL Databases (35%)
  - Submit exercises the next day of each related course (23:59)
- Submit: g3d@ecam.be with "Exercises: {XML/JSON/NoSQL}" + .zip
  - Late submission: 0/20 to the related evaluation

# Evaluation - Project

- Project (50%)
  - 3-people teams unless exception
  - Design, implementation & setup of the database in a docker-compose.yml + application
  - Presentation (.pdf) and code (app + database setup) must be sent ***the day before*** the evaluation at 23:59 the latest
  - Last course: 20 minutes presentation + 20 minutes Q/A
    - Time allocation is free to change if deemed necessary by the lecturer
  - Everyone will listen to every presentation
  - Different notes can be given depending on the contribution & comprehension of each student
- Submit: g3d@ecam.be with "Project - Team {i}" + .zip
  - Late submission: 0/20 to the related evaluation

# Evaluation - Project

| Database & Implementation justification | Feature implementation | Rating mark |
|---|---|---|
| Yes | Yes (full scope) | [14; 20] |
| Yes | No (full scope not done) | [0; 14[ |
| No | Yes | 0 |
| No | No | [0; 14[ |

Sending the code is part of "feature implementation"
(no code, no feature)

# About the lecturer

- Software Engineer / Big Data → Data Engineer
- Teaching Assistant at Université Libre de Bruxelles
- Companies I worked for
  - Université Libre de Bruxelles
  - Macq
  - ADB Safegate
  - Evonik
  - Proximus
  - Engie GEMS
- Course content
  - Directly related to day-to-day work

# Intro to XML

Why?
How to use it ?
How to validate it ?

# XML: Why ?

- Extensible Markup Language
- Markup language
- File format

- Goals
  - Simplicity
  - Generality
  - Usability
- To communicate data in a structured format (!= HTML)

# XML: Why ?

- SGML: Standard Generalized Markup Language
  - Released in 1986
  - Enable sharing of machine-readable documents, for several decades
- HTML is a variant of SGML
  - Pre-defined tags
  - Presentation layer
- XML is a variant of SGML
  - Data layer

# XML: Why ?

- Define your logging (log4j)

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration debug="true" xmlns:log4j='http://jakarta.apache.org/log4j/'>
   <appender name="console" class="org.apache.log4j.ConsoleAppender">
      <layout class="org.apache.log4j.PatternLayout">
           <param name="ConversionPattern" value="%d{yyyy-MM-dd HH:mm:ss}" />
      </layout>
   </appender>
   <root>
      <level value="DEBUG" />
      <appender-ref ref="console" />
   </root>
</log4j:configuration>
```

# XML: Why ?

- Define your build settings (maven)

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1</version>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>1.2.17</version>
  </dependency>
</project>
```

- And a multitude of other use cases across a lot of applications, languages, build tools, …,
- Also extensively used to transfer data

# XML: Why ?

- Send & Receive data from an API (SOAP)
  - Envelope: identifies the XML document as SOAP message
  - Header: header information (authentication, …)
  - Body: call & response
  - Fault: errors & status

# XML: Why ?

- SOAP
  - 
```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

  <soap:Header>
  ...
  </soap:Header>

  <soap:Body>
  ...
    <soap:Fault>
    ...
    </soap:Fault>
  </soap:Body>

</soap:Envelope>
```

# XML: Why ?

- SVG – Scalable Vector Graphics
  - ```
    <svg height="100" width="100" xmlns="http://www.w3.org/2000/svg">
       <circle r="45" cx="50" cy="50" stroke="green" stroke-width="3"
         fill="red" opacity="0.5" />
    </svg>
    ```


    ```

# XML: How to use it ?

- Standard XML syntax, v1.0 (5th edition)
  - Released in 1998
  - Public format: https://w3.org/TR/xml
- Most languages have an XML library
- Structure definition (and validation)
  - DTD
  - XML Schema (XSD)

# XML: Some properties

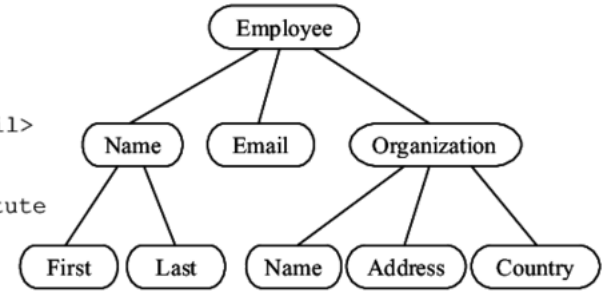
- An XML document is *well-formed* if it follows the syntax rules of XML
- An XML document is *valid* if it is *well-formed* and follows the structured defined in a Document Type Definition (DTD) or XML Schema (XSD)
- An XML document does not contain any information on how it should be rendered

# XML: Declaration

- `<?xml version="1.0" encoding="UTF-8" ?>`
- Basic information for the XML parser:
  - XML version
  - Character encoding (optional) - most of the time, UTF-8
- But, how would you read the encoding of the first line without knowing the encoding ?

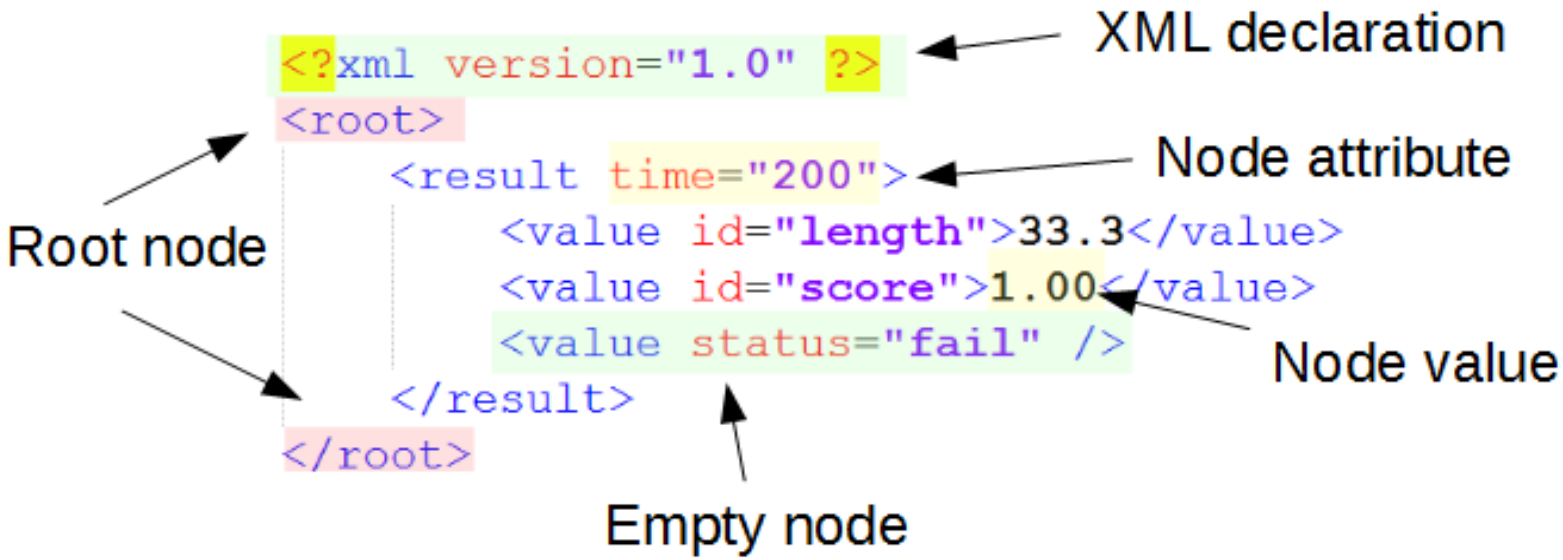
# XML: Structure

```
<Employee>
    <Name>
        <First>Lassi</First>
        <Last>Lehto</Last>
    </Name>
    <Email>Lassi.Lehto@fgi.fi</Email>
    <Organization>
        <Name>
            Finnish Geodetic Institute
        </Name>
        <Address>
            PO Box 15,
            FIN-02431 Masala
        </Address>
        <Country CountryCode="358">Finland</Country>
    </Organization>
</Employee>
```




- Root: `Employee`
- Nodes: `Name, Email, …`
- Attributes: `CountryCode=358`

# XML: Structure



# XML: Structure

- All elements start with a start tag and end with an end tag
- The name of the element is formed using
  - Alphanumeric characters a-zA-Z0-9
  - Underscore, dash, dot
  - Colons (:) are possible but they define a namespace
  - No space
  - Does not start with a number
  - Does not start with "xml"

# XML: Namespace

- Within an XML Schema, you might want to re-use some tags
- Namespaces
  - `log4j:configuration`
  - `soap:body`
- You must define them
  - For html code: `xmlns="http://www.w3.org/TR/html4/"`
  - `xmlns:log4j="http://jakarta.apache.org/log4j/ "`
  - `xmlns:soap="http://www.w3.org/2003/05/soap-envelope"`
- Default namespace
  - Avoid always putting the namespace as prefix

# XML: Elements

- Start and end tag must correspond
- No crossings: `<intro>…<title>…</intro>…</title>`
- Case sensitive: `<Title>` and `<title>` are different tags
- Only one root element
  - At the top of the document
  - Cannot appear again elsewhere in the tree
- XML comments: `<!-- comment -->`

# XML: Elements

- Elements can be:
  - Non empty: start with opening tag and end with closing tag, can contain text and other elements
    - `<title>The lord of the rings<title>`
  - Empty: do not contain text nor other elements
    - `<title></title>` or `<title />`
- Elements can have attributes:
  - `<title type="fantasy">The lord of the rings<title>`
  - Attributes should be defined between quotes (') or double quotes (")

# XML: Elements

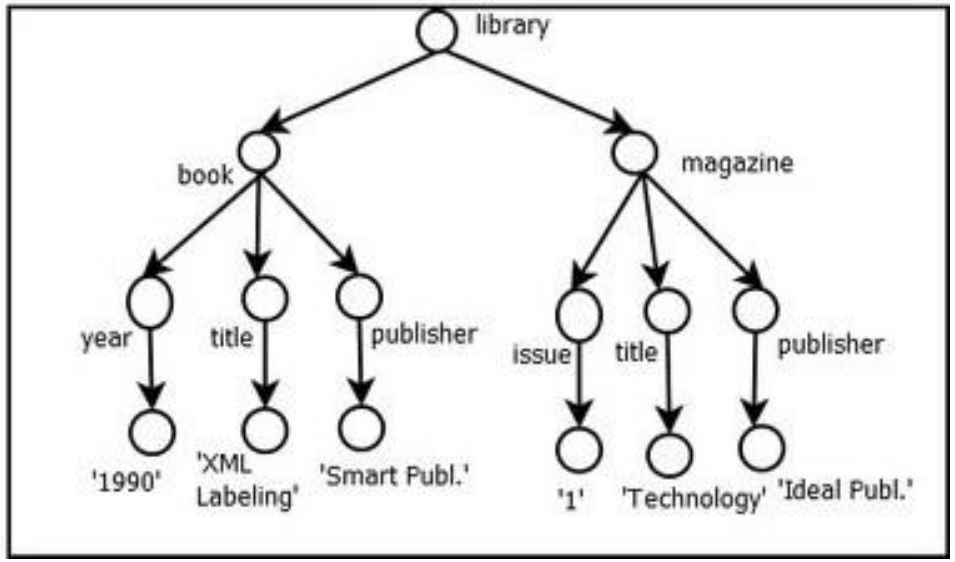
```
<parent>
  <sibling1> ... </sibling1>
  <sibling2> ... </sibling2>
  <self>
    <child1>
      ... <desc1></desc1> ... <desc2></desc2> ...
    </child1>
    <child2> ... </child2>
    <child3>
      ... <desc3><desc4> ... </desc4></desc3> ...
    </child3>
  </self>
  <sibling3> ... </sibling3>
</parent>
```

# XML: Related features

- XLink
  - Link to other xml documents, like "href" in html
- XPath
  - `/bookstore/book[1]/title`
  - `/bookstore/book[price>35]/price`
- XQuery
  - The SQL for your XMLs
  - ```
    for $x in doc("books.xml")/bookstore/book
    where $x/price>30
    return $x/title
    ```
- …
    ```

# XML - Exercise 1

- Transform the following XML tree into a valid XML file (by hand)



# XML - Exercise 2

- Transform the following recipe you received from a friend into an XML file (by hand)
  - The XML is going to be used by a website to show all ingredients (wherever they appear), total execution time, necessary tooling …
  - Make sure the generated xml is consistent and easy to process by a software
- Recipe for Japanese Curry
- Ingredients
  - Beef, chopped: 450g
  - Onions, minced: 350g
  - Carrot, chopped: 100g
  - Potato, chopped: 150g
  - Water: 500ml
  - Golden Curry Sauce Mix: 92g
- Directions
  - Stir-fry meat and vegetables with oil in a large skillet on medium heat for approx. 5 min.
  - Add water and bring to boil. Reduce heat, cover and simmer until ingredients are tender, approx. 15min.
  - Turn the heat off, break S&B Golden Curry Sauce Mix into pieces and add them to the skillet. Stir until sauce mixes are completely melted. Simmer approx. 5 min., stirring constantly.
  - Serve hot over rice or noodles.

# DTD: What is it?

- Defines structural constraints in XML
- The Document Type Definition (DTD) defines the elements and their rules
- A document - with a related DTD - is valid if:
  - It is well-formed
  - It references a DTD
  - It complies with the DTD

# External DTD

- The DTD can be included directly in the document, or in an external file
- External DTD
  - `<!DOCTYPE root_element SYSTEM|PUBLIC [name] DTD_uri>`
  - `<!DOCTYPE people_list SYSTEM "example.dtd">`
    `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"`
    `"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`
- The DOCTYPE allows to declare the type of the document, the identifier for the root element is needed
  - SYSTEM : is local to computer, PUBLIC: can be retrieved from a catalog

# Internal DTD

- The DTD can be directly included in the document file
  - <!DOCTYPE people_list [
    ...
    ]>

# DTD: Example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE people_list [
  <!ELEMENT people_list (person*)>
  <!ELEMENT person (name, birthdate?, gender?,
socialsecuritynumber?)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT birthdate (#PCDATA)>
  <!ELEMENT gender (#PCDATA)>
  <!ELEMENT socialsecuritynumber (#PCDATA)>
]>
<people_list>
  <person>
    <name>Fred Bloggs</name>
    <birthdate>2008-11-27</birthdate>
    <gender>Male</gender>
  </person>
</people_list>
```

# DTD: Issues

- A DTD can be used to create a denial-of-service attack by defining nested entities expanding exponentially, or by sending the XML parser to an external resource that never returns
- Many frameworks & software (Microsoft Office) will not open files containing DTD declarations
- Other issues
  - It does not use an XML syntax
  - No typing of content
  - No regex matching
- → Replaced by XML Schema

# XML Schema: Overview

- Describe the structure of an XML document
- XML Document

```
<?xml version="1.0"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

# XML Schema: Overview

- DTD Rules

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

# XML Schema: Overview

- XSD

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="https://www.w3schools.com"
xmlns="https://www.w3schools.com"
elementFormDefault="qualified">

<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>

</xs:schema>
```

# XML Schema - Benefits

- Introduce data types
- Use XML
  - Same language
  - Same parser
  - Same editor
- Extensible
  - Re-use a Schema in other Schemas
  - Create your own data type
  - Use multiple schemas in the same document

# XML Schema: Another example

- XML

```
<?xml version="1.0"?>
<Book>
    <Title>XML Schema Essentials</Title>
    <Authors>
        <Author>R. Allen Wyke</Author>
        <Author>Andrew Watt</Author>
    </Authors>
    <Publisher>John Wiley</Publisher>
</Book>
```

# XML Schema: Another example

- XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:annotation>
        <xsd:documentation>
            This is a sample XML Schema for Chapter 1 of XML Schema
            Essentials.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="Book">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Title"/>
                <xsd:element ref="Authors"/>
                <xsd:element ref="Publisher"/>
            </xsd:sequence>
            <xsd:attribute name="pubCountry" type="xsd:string"/>
        </xsd:complexType>
    </xsd:element>
```

…

# XML Schema: Another example

```
…
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Authors">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Author" minOccurs="1"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Author" type="xsd:string"/>
    <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

# XML Schema - A few keywords

- Tags
  - `element`
  - `complexType`
  - `sequence`
  - `attribute`
- Attributes
  - `type`
  - `name`
  - `maxOccurs`
  - `minOccurs`
  - `ref`

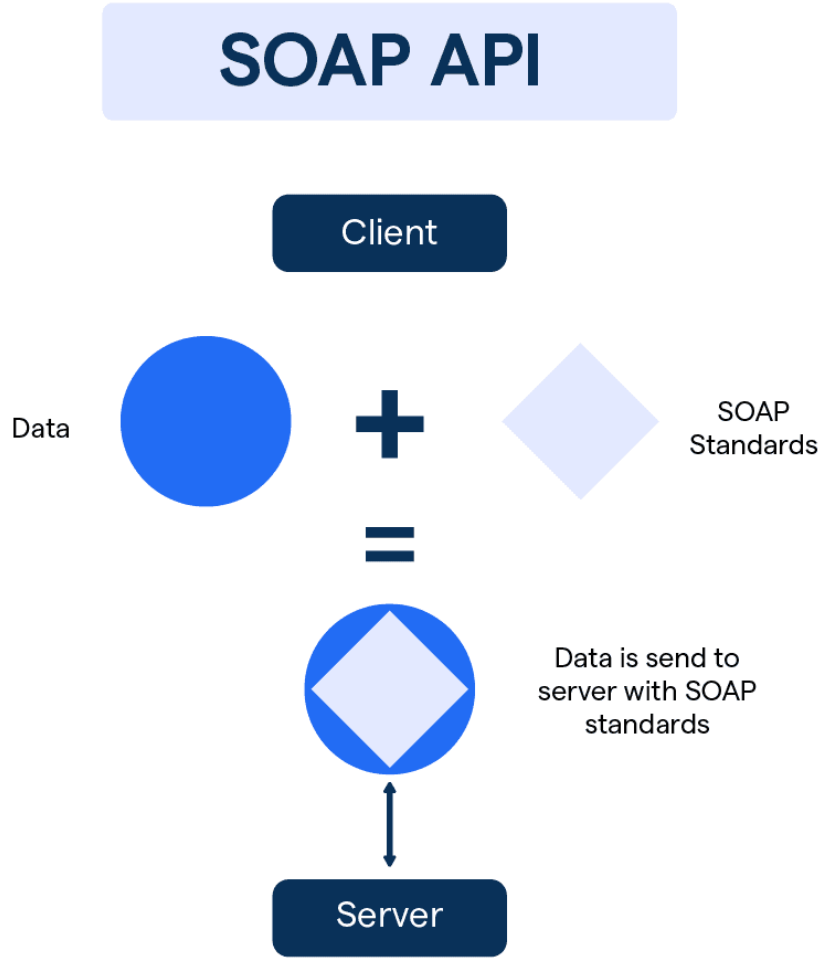
# XML - Exercise 3

- Create a XSD to validate the following XML
- ```
  <?xml version="1.0" encoding="UTF-8"?>
  <Race date="2020-12-15" name="Holiday Meet">
      <Course>
          <CourseName>The track</CourseName>
          <Address>Track road 123</Address>
      </Course>
      <Horses>
          <Horse name="Bonfire">
              <Value>5000</Value>
              <Birthdate>1998-05-01</Birthdate>
              <Gender>M</Gender>
          </Horse>
          <Horse name="Dobby">
              <Value>1000</Value>
              <Birthdate>2001-04-05</Birthdate>
              <Gender>F</Gender>
          </Horse>
      </Horses>
  </Race>
  ```
  ```

# XML - Exercise 4

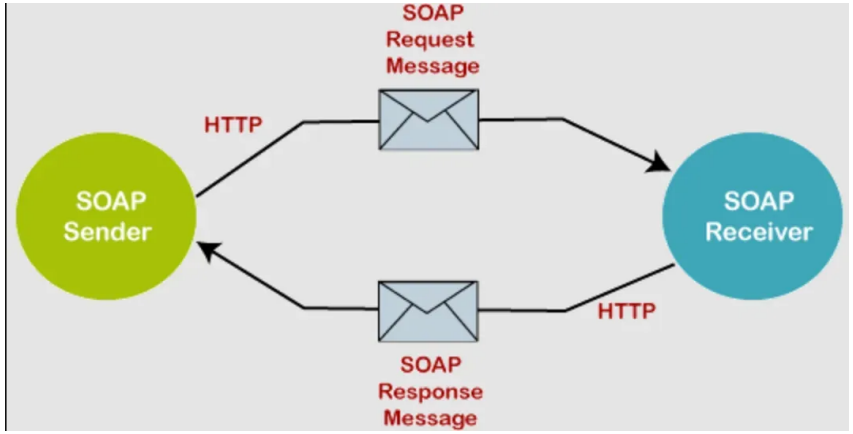
- Use Python and the library lxml to load the xml of any given file, and list the content of any given xpath
- Example:
  - python ex4.py myfile.xml /Race/Horses/Horse
  - `b'<Horse name="Bonfire">\n\t\t\t<Value>5000</Value>\n\t\t\t<Birthdate>1998-05-01</Birthdate>\n\t\t\t<Gender>M</Gender>\n\t\t</Horse>\n\t\t\n'`
  - `b'<Horse name="Dobby">\n\t\t\t<Value>1000</Value>\n\t\t\t<Birthdate>2001-04-05</Birthdate>\n\t\t\t<Gender>F</Gender>\n\t\t</Horse>\n\t\n'`

# XML & HTTP: SOAP



# SOAP

- Simple Object Access Protocol
- Enveloppe
  - Root element with XML namespaces
- Header
  - Optional
  - Authentication tokens, encryption details, custom headers, …
- Body
  - Payload itself
- Fault
  - Error codes, error messages



# SOAP - Request

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="https://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetUser>
      <m:UserId>123</m:UserId>
    </m:GetUser>
  </soap:Body>
</soap:Envelope>
```

# SOAP - Response

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="https://www.w3.org/2003/05/soap-envelope/"
               soap:encodingStyle="https://www.w3.org/2003/05/soap-
encoding">
    <soap:Body>
        <m:GetUserResponse>
            <m:Username>Tony Stark</m:Username>
        </m:GetUserResponse>
    </soap:Body>
</soap:Envelope>
```

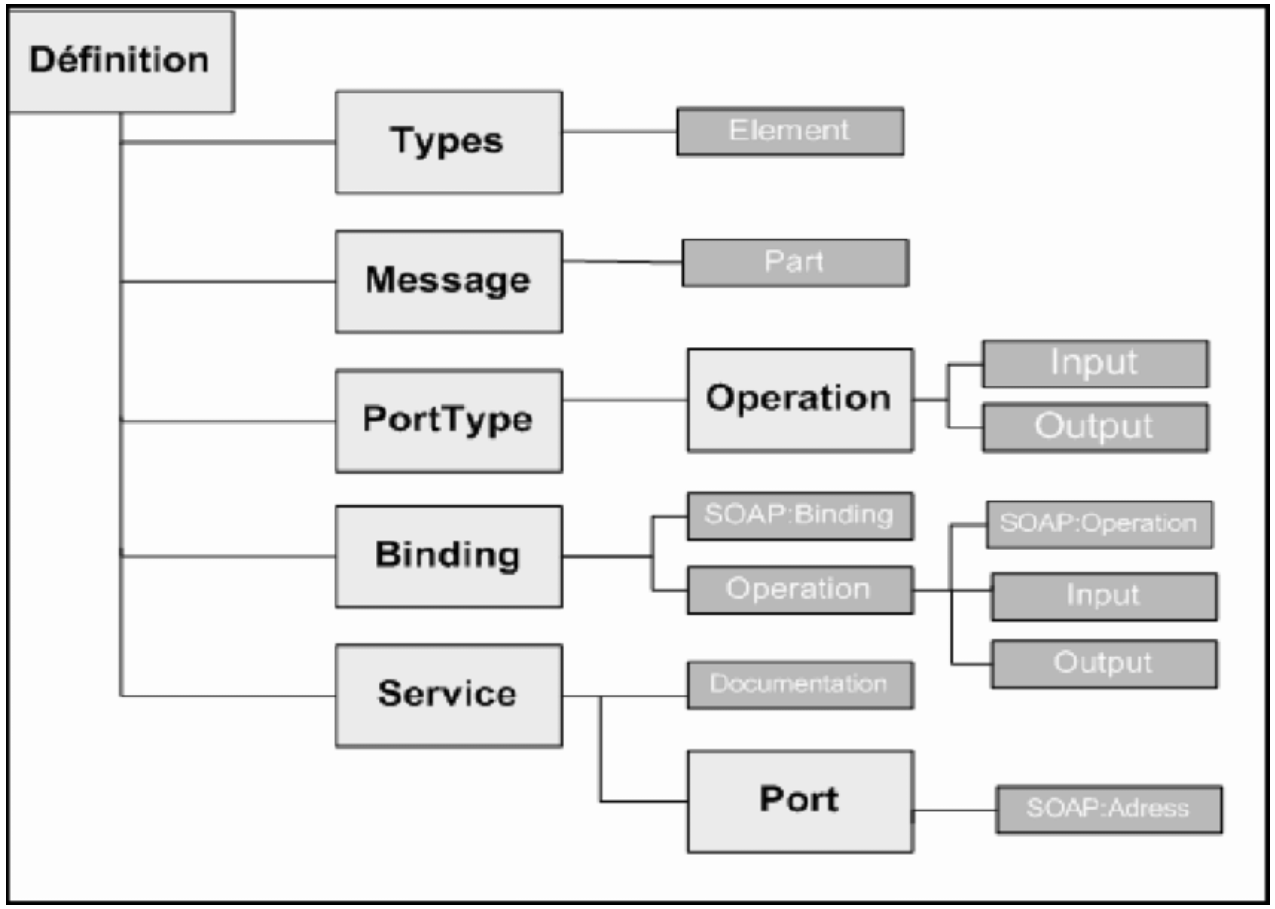
# SOAP API

- XSD
  - Describe the structure of the data types being exchanged
  - Describe the fields and restrictions on fields (max length, …)
- WSDL - Web Services Description Language
  - Describe the API and its operations
  - List of methods, parameters and returned values
  - Abstract definitions of endpoints and messages separated from the network deployment or data format bindings

# SOAP API - WSDL

- Definitions
  - `targetNamespace`
  - `xmlns`: default namespace of the WSDL document
  - `xmlns:tns` current namespace
- Types
  - Contains various xsd
- Message
- Operation
- portType
- Binding
- Port
- service

# SOAP API - WSDL



# SOAP API

- Demo

# XML - Exercise 5

- Use Python to create a SOAP API providing the various features
  - In memory "database"
  - (Shop) objects
    - Attributes: name, remaining quantity, price
    - List, create, update & delete
  - Orders
    - Attributes: object_id, customer_id, quantity
    - List, create, update & delete
  - Apply some basic validations
    - quantity >= 0, name must be of length [4;100], birth date is a real date ...
- Provide a python script to test each endpoint

# XML - Credits & references

- Ken Hasselmann
  - Introduction au XML: https://brunomartin.be/cours/xml.pdf
  - Working with XML trees: https://docs.fab-image.com/studio/programming_tips/UsingXml.html
  - XML documentation: https://www.w3.org/XML/
- Official XML Schema tutorial from w3schools
  - https://www.w3schools.com/xml/schema_intro.asp
- XML Schema Essentials
  - https://nuleren.be/ebooks/xml-schema-essentials.pdf
- Japanese curry recipe
  - https://www.sbfoods-worldwide.com/recipes/010.html