INTRODUCTION TO XML

Mario G.C.A. Cimino

Department of Information Engineering

XML representation

✓ So far we have experimented transformations between UML models and Imperative (platform dependent) programming code (Java, but also C#, Objective C, PHP, …). Let us experiment transformations between UML models and XML code.

✓ XML (eXtensible Markup Language) is a W3C-endorsed standard format, used to mark up data with human-readable tags: data is surrounded by **attributes** and **elements**:

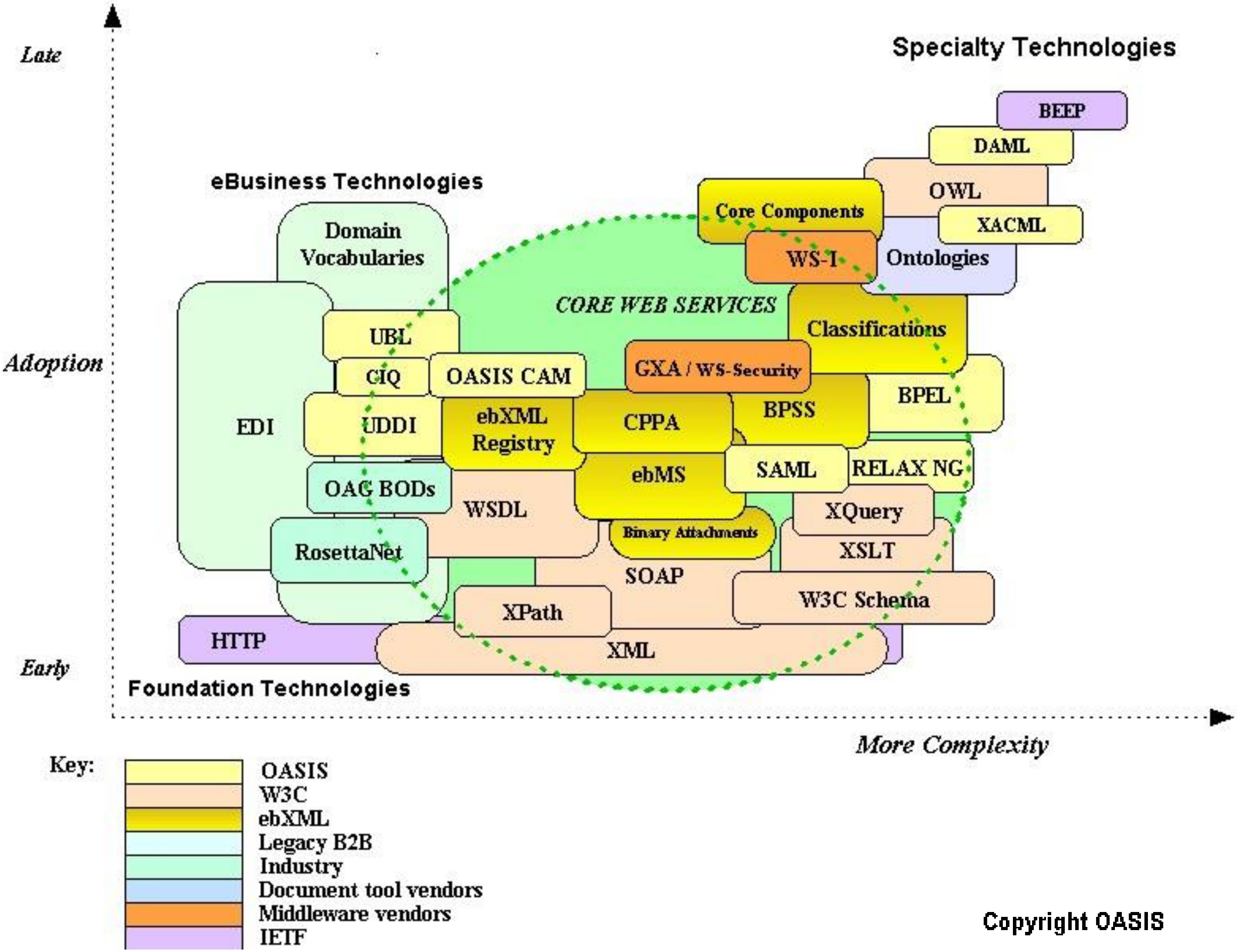
```
<birthdate format="gg/mm/aaaa">13/01/1944</birthdate>
```

✓ XML files can be easily processed with Object Oriented languages, using the Object Document Model (DOM) API standard (as HTML files with Javascript).

✓ XML is actually a **metamarkup** language: it does not have a fixed set of tags (as HTML does), but allows to define elements and attributes (**XML grammars**) for any application domain: formatted text (XHTML), vector image (SVG), sheet music (MusicXML), chemistry (CML), Math (MathML), …

✓ Some XML grammars have been standardized (by W3C, OASIS, …)

(http://en.wikipedia.org/wiki/List_of_XML_markup_languages)


Late
Adoption
Early
More Complexity
Specialty Technologies
eBusiness Technologies
Foundation Technologies
CORE WEB SERVICES
BEEP
DAML
OWL
XACML
Core Components
WS-I
Ontologies
Domain Vocabularies
Classifications
EDI
UBL
CIQ
OASIS CAM
GXA / WS-Security
BPSS
BPEL
UDDI
ebXML Registry
CPPA
ebMS
SAML
RELAX NG
OAG BODs
WSDL
XQuery
Binary Attachments
XSLT
RosettaNet
SOAP
W3C Schema
XPath
HTTP
XML
Key:
OASIS
W3C
ebXML
Legacy B2B
Industry
Document tool vendors
Middleware vendors
IETF

- ✓ The markup in an XML grammar describes the structure of the document: which elements and attributes are allowed, and their relationship (semantics). The markup permitted in a particular XML grammar can be documented in a **Schema**. Instance documents that match the schema are said to be **valid** (similarly to HTML validation using DTD strict, transitional, frameset).

- ✓ Nothing prevents you from representing XML programs, i.e., creating an XML grammar as a programming language. For instance: **XSLT** (Extensible Stylesheet Language Transformations) is a Turing-Complete XML language for transforming XML documents into other XML documents; **ANT** is an XML task manager used for compilation management (similar to Make); BPEL is a statically typed Turing-Complete scripting language used to **orchestrate** a distributed system of services.

- ✓ A **service** is a software system designed to support interoperable and distributed XML machine-to-machine interaction. **Orchestration** refers to the **central control** (by the conductor) of the behavior of a distributed system, in analogy with an orchestra consisting of many players centrally controlled). **Choreography** refers to a distributed system (in analogy with a dancing team) which operates according to rules (the choreography) but without centralized control.

- ✓ XML-based (e.g. BPEL) and platform-dependent (e.g. Java) languages are functionally similar, but their use impacts differently on the life-cycle of a software applications.

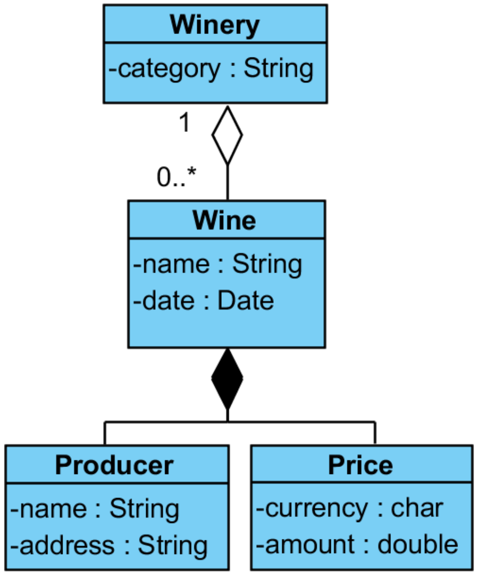
FROM UML CLASS DIAGRAM TO XML SCHEMA DEFINITION (XSD)

Mario G.C.A. Cimino

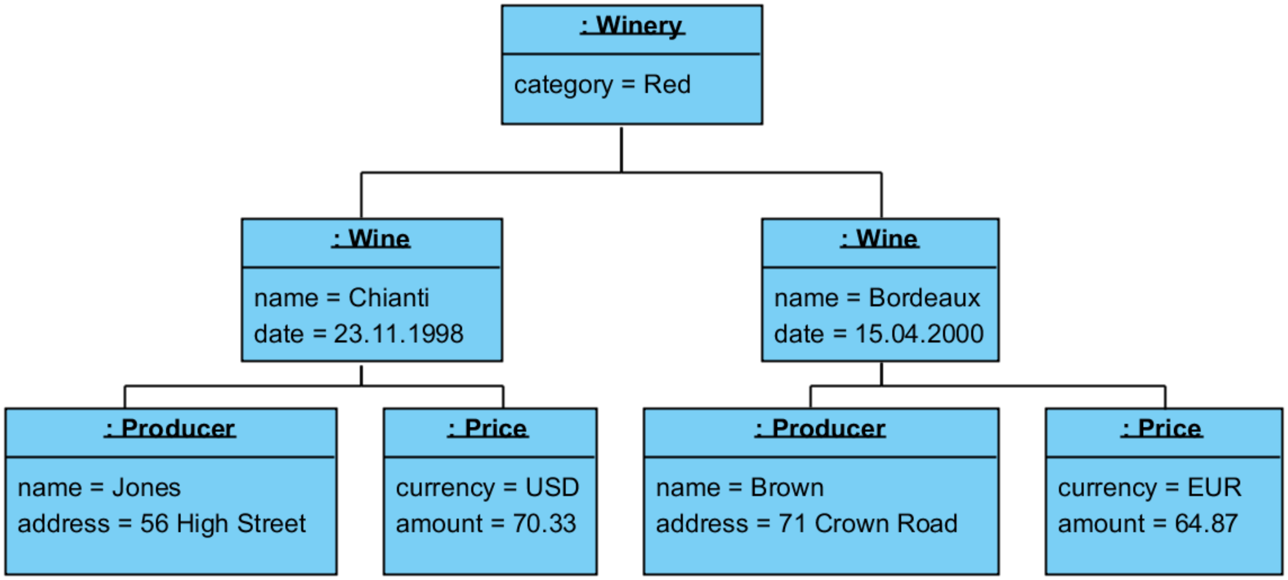
Department of Information Engineering

XML data representation

- ✓ Sample UML data model (to implement on Visual Paradigm).



Class diagram



Object (instance) diagram

- ✓ Object diagram: right click on an object → Select classifier → Select classifiers → Classifiers → Add → select Class → OK → Slots → (select) → Define Slot → Edit values → Add → text → (enter a value) → OK → OK.
- ✓ Tools → Code Engineering → Instant Generator → XML Schema → Class Diagram → Preview → click on the xsd file.
- ✓ Tools → Code Engineering → Instant reverse → XML → Path: winery.xml → OK → select all classes → OK.

- ✓ Different XML grammars can be generated by a class model, and vice-versa. **However, the XSD generator available in Visual Paradigm for UML is not easily manageable for the purposes of the project. For this reason students will adopt a specific UML-XML mapping, without automatic conversion functions.**

- ✓ Students can use any basic XML editor, such as Notepad++ (http://www.iet.unipi.it/m.cimino/sse/res/npp.zip)

- ✓ A bad *XML instance document*

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- a bad sample -->
<Winery>
  <category>Red</category>
  <Wine>
    <name>Chianti</name>
    <date day = "23" month = "11" year = "1998"/>
    <Producer format = "name, address">Jones, 56 High Street</Producer>
    <Price amount = "70.33 USD"/>
  </Wine>
  <Wine>
    <name>Bordeaux</name>
    <date day = "15" month = "04" year = "2000"/>
    <Producer format = "name, address">Brown, 71 Crown Road</Producer>
    <Price amount = "64.87 EUR"/>
  </Wine>
</Winery>
```

<? Processing instruction?>

Element (DOM)

Attribute (DOM)

- ✓ A good *XML instance document (see the XML design Rule-of-thumb)*

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- winery.xml, a good sample for the project -->
<Winery category = "Red"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation='Winery.xsd'>
  <Wine>
    <name>Chianti</name>
    <date>1998-11-23</date>
    <Producer>
     <name>Jones</name>
     <address>56 High Street</address>
    </Producer>
    <Price currency = "USD" >70.33</Price>
  </Wine>
  <Wine>
    <name>Bordeaux</name>
    <date>2000-04-15</date>
    <Producer>
     <name>Brown</name>
     <address>71 Crown Road</address>
    </Producer>
    <Price currency = "EUR" >64.87</Price>
  </Wine>
</Winery>
```

- ✓ How to define an XML grammar? The W3C developed an XML grammar for defining XML grammars: XML Schema Definition (XSD).

The XML design Rules-Of-Thumb:

should I use an element or an attribute?

R1) Elements are used to encapsulate structured data or self-contained data;

R2) Attributes are used to provide accompanying information about an element;

R3) Use attributes if your information requires data of a simple type and:

  a) It is a default or almost-fixed value;
  b) It is (structural or descriptive) metadata for an existing element;
  c) If the size of your XML file is an issue;

R4) Use elements if the ordering/option is significant (attributes can occur in any order).

R5) Elements can occur more than once, attributes cannot.

Exercises

1) Look at this text, structure it as an XML document and then define an XML schema.

Note. To Tove from Jani. Reminder: Do not forget me this weekend

note.xml

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

note.xsd

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

2) Swap the two elements *<to>* and *<from>* in the xml, and validate. Replace *sequence* with *all* in xsd, and validate again.

3) Look at this text, structure it as an XML document and then define an XML schema.

In my family there are three persons, with the following full names (child names): Hege Refsnes ("Cecilie"), Tove Refsnes ("Hege", "Stale" or "Borge"), and Stale Refsnes. No more than three child names are allowed.

family.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="family.xsd">
<person>
  <full_name>Hege Refsnes</full_name>
  <child_name>Cecilie</child_name>
</person>
<person>
  <full_name>Tove Refsnes</full_name>
  <child_name>Hege</child_name>
  <child_name>Stale</child_name>
  <child_name>Borge</child_name>
</person>
<person>
  <full_name>Stale Refsnes</full_name>
</person>
</persons>
```

family.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="persons">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="person" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="full_name" type="xs:string"/>
            <xs:element name="child_name" type="xs:string"
            minOccurs="0" maxOccurs="3"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

4) Add a new child name to Tove. Add information about name and surname.

family.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persons ... format="name surname">…
```

family2.xsd

```
<?xml version="1.0" encoding="UTF-8"?>...
   <xs:attribute name="format" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
</xs:schema>
```

XML/XSD Basics

- ✓ **<xs:schema xmlns:xs=…**: to define an XML namespace, a tag prefix to avoid element name conflicts when trying to mix XML documents from different XML applications (like packages in programming languages). URIs are often used as a namespace, to have a globally unique prefix based on the DNS. The <xs:schema> element is the root element of every XML Schema.
- ✓ **URI** (Universal Resource Identifier): is a string of characters used to identify a web resource. It does not necessarily point to an actual document or page, as the URL does providing the network "location".
- ✓ **<xs:element name=…**: to define the name of an element. The number of allowed occurrences of the element in the parent element is 1 by default.
- ✓ **<xs:attribute name=…**: to define the name of an attribute. Attributes are optional by default. To specify that the attribute is required, use the "**use**" attribute.
- ✓ **type="xs:string","xs:date","xs:time","xs:decimal", "xs:integer", "xs:boolean"**: to declare the type of a simple content (simple type).
- ✓ **<xs:complexType><xs:sequence>,<xs:all>,<xs:choice>,…"**: to define a complex element made of an ordered/unordered, complete/alternative sequence of sub-elements. A simple element contains simple type, a complex elements contains other elements and/or attributes.

  (http://www.w3.org/XML/Schema.html, **http://www.w3schools.com/schema**)

✓ XML Schema definition document

```
<?xml version=”1.0” encoding="UTF-8"?>
<!-- winery.xsd, a good sample for the project -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
 <xs:element name="Winery">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="Wine" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="category" type="xs:string" use="required"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="Wine">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="date"            type="xs:date"/>
    <xs:element ref ="Producer"/>
    <xs:element ref ="Price"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="Producer">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="name"       type="xs:string"/>
    <xs:element name="address"    type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="Price">
  <xs:complexType>
   <xs:simpleContent>
    <xs:extension base="xs:decimal">
     <xs:attribute name="currency" type="xs:string" use="required"/>
    </xs:extension>
   </xs:simpleContent>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

URI. A Universal Resource Identifier (URI) is a string of characters used to identify a web resource. A Uniform Resource Locator (URL), is a URI providing the network "location".

XML validation

- ✓ http://www.xmlvalidation.com/
- ✓ Validate an XML instance document against an XML Schema document.