Modelling XML Applications

Patryk Czarnik

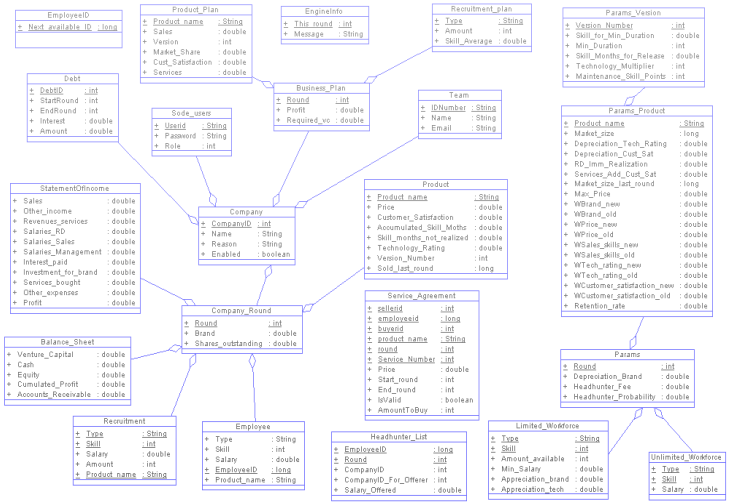
XML and Applications 2013/2014
Lecture 2 – 14.10.2013

XML application (recall)

- **XML application** (*zastosowanie XML*)
  - A concrete language with XML syntax
- Typically defined as:
  - Fixed set of acceptable tag names (elements and attributes, sometimes also entities and notations)
  - Structure enforced on markup, e.g.: “`<person>` may contain one or more `<first-name>` and must contain exactly one `<surname>`”
  - Semantics of particular markups (at least informally)

Modelling new XML application

- Analysis & design
  - analysis of existing documents, new requirements, etc.
  - identifying nouns, their role and dependencies
  - data types, constraints, limits
- Writing down
  - structure definition – “schema”
  - semantics description – usually in natural language; in schema (comments, annotations) or a separate document



Standards for defining structure of XML documents

- **DTD**
  - part of XML standard (1998, 2004)
  - origins from SGML (1974)
- **XML Schema** – W3C Recommendation(s)
  - version 1.0 – 2001
  - version 1.1 – 2012
- **Relax NG**
  - OASIS Committee Specification – 2001
  - ISO/IEC 19757-2 – 2003
- **Schematron**
  - alternative standard and alternative approach
  - several version since 1999
  - impact on XML Schema 1.1

Benefits of formal definition

- Tangible asset resulting from analysis & design
  - Formal, unambiguous definition of language
  - Reference for humans (document authors and readers, programmers and tool engineers)
- Ability to **validate** documents using tools or libraries
  - Programs may assume correctness of content of validated documents (less conditions to check!)
- Content assist in editors
  - autocomplete during typing, stub document generation

Two levels of document correctness (recall)

- Document is **well-formed** (*poprawny składniowo*) if:
  - conforms to XML grammar,
  - and satisfies additional *well-formedness constraints* defined in XML recommendation.
  - Then it is accessible by XML processors (parsers).
- Document is **valid** (*poprawny strukturalnie, "waliduje się"*) if additionally:
  - is consistent with specified document structure definition; from context: DTD, XML Schema, or other;
  - in strict sense (DTD): satisfies *validity constraints* given in the recommendation.
  - Then it is an instance of a logical structure and makes sense in a particular context.

Element content – simple case

Example content

```
<student>
   <first-name>Monika</first-name>
   <surname>Domżałowicz</surname>
   <birth-date>1990-03-13</birth-date>
</student>
```

DTD definition

```
<!ELEMENT student (first-name, surname, birth-date)>
<!ELEMENT first-name (#PCDATA)>
<!ELEMENT surname (#PCDATA)>
<!ELEMENT birth-date (#PCDATA)>
```

XML Schema definition

```
<xs:element name="student">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="first-name" type="xs:string"/>
         <xs:element name="surname" type="xs:string"/>
         <xs:element name="birth-date" type="xs:date"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Details in examples!

Disclaimer

Taking our experience and students' opinions into account we will try not to *copy standard specifications onto slides* but rather to show by examples:

- some typical usage,
- different paths to do a thing – so you can choose your approach depending on needs,
- chosen cases of advanced usage and rarely used features – it is impossible to show all of them during a short lecture,
- some good and bad practices.

It also means, in particular, that slides are not a complete source of knowledge required to pass the exam.

Details in examples!

Examples presented during the lecture:

- Structure of DTD, structure of XML Schema definition
- Typical element definition
- Controlling number of occurrences
- Sequence and choice
- Building complex models
- Any order (`xs:all`)– schema only
- Defining attributes in schema and DTD

Details in examples! (ctd)

Examples on lab classes (see lab scenario)

- Mixed content
  - DTD approach – (#PCDATA| a | b)*
  - Mixed content with controlled subelements – schema only
- Avoiding code duplication and different ways of writing definitions in schemas
  - Local definitions vs global definitions
  - Anonymous types vs named (global) types
  - Named groups
  - Extending complex types

Common design decisions
Natural language

- Which natural language to use?
  - It would be a nonsense not to use English in solutions that have a potential to be used worldwide.
  - But XML supports internationalisation. You may choose to honour you native language if the solution is dedicated for a particular country.
    - Law acts processed in Polish Parliament use Polish tags,
    - as well as XML-version forms for tax declarations.

```
<Podmiot1 poz="P_1A" rola="Podatnik">
    <etd:OsobaFizyczna>
        <etd:PESEL>00000000000</etd:PESEL>
        <etd:ImiePierwsze>a</etd:ImiePierwsze>
        <etd:Nazwisko>a</etd:Nazwisko>
        <etd:DataUrodzenia>1900-01-01</etd:DataUrodzenia>
    </etd:OsobaFizyczna>
```

Common design decisions
Element or attribute?

- Where should a field be written:
  - in an attribute

```
<teacher tel="5544458">
   <first-name>Patryk</first-name>
   <surname>Czarnik</surname>
</teacher>
```

  - or in an element?

```
<teacher>
   <first-name>Patryk</first-name>
   <surname>Czarnik</surname>
   <tel>5544458</tel>
</teacher>
```

- I would write this particular one in an element.

```
<tel type="office">55<internal>44458</internal></tel>
```

Common design decisions
Element or attribute?

- Advantages of attributes:
  - more compact syntax
  - (only in DTD) some features available only for attributes
- Technical restrictions of attributes:
  - only text, without marked up structure
  - multiple attributes with the same same forbidden
- General hints
  - Semantic hint: Use elements for **data**, attributes for **metadata** (whatever it means in you case ;)).
  - Presentational hint: If you had to print your document on paper, which parts of text would you print literally (they are elements) and which parts would only have some impact (or no impact) on the way things are presented (should become attributes)?

Common design decisions
Names

- How descriptive (and long) should a name be?
- To use multipart names, or assume that the context is known?

```
<program>
   <item>Introduction to XML</item>
   <item>XML Schema</item>
</program>
```

```
<lecture-program>
   <lecture-program-item>Introduction to XML</lecture-program-item>
   <lecture-program-item>XML Schema</lecture-program-item>
</program>
```

  - Unique element names simplify some kinds of document processing
    - CSS, SAX, or DOM (in some cases) – in large scope
    - XPath/XSLT/XQuery – not so important

Common design decisions
Wrappers

```
<group>
   <number>1</number>
   <students>
     <student>Jan Kowalski</student>
     <student>Anna Nowakowska</student>
     ...
   </students>
</group>
```

```
<group>
   <number>1</number>
   <student>Jan Kowalski</student>
   <student>Anna Nowakowska</student>
   ...
</group>
```

Modularisation options

- Combining multiple files
  - DTD – external parameter entities
  - Schema – include, import, redefine
- Reusing fragments of model definition
  - DTD – parameter entities
  - Schema – groups and attribute groups
    (in practice equivalent to the above)
  - Schema – types, type derivation (no such feature in DTD)
- Global and local definitions
  - In DTD all elements global, all attributes local
  - In schema both can be global or local, depending on case

See examples for details!

Import or include?

- `xs:import`
  - Imports foreign definitions to refer to
- `xs:redefine`
  - Includes external definitions, but a local definition overrides external one if they share the same name
- `xs:include`
  - Basic command, almost like textual insertion
  - Imported module must have the same target namespace or no target namespace

A multi-module, namespace-aware project with overused `xs:include` leads to duplication of logic in the software that processes documents (or enforces meta-programming tricks to avoid it). */based on personal experience/*

Schema and namespaces

- DTD is namespace-ignorant
- XML Schema conceptually and technically bound with XML namespaces
  - Basic approach: one schema (file) = one namespace
    - Splitting one ns into several files technically possible
  - Referring to components from other namespaces available
- Important attributes
  - `targetNamespace` – if given, all global definitions within a schema go into that namespace
  - `elementFormDefault`, `attributeFormDefault` – should local elements or attributes have qualified names?
    - default for both: `unqualified`
    - typical approach: elements qualified, attributes unqualified
    - setting may be changed for individual definitions

Using namespaces in XML Schema

Different technical approaches to handle namespaces in XML Schema

- XML Schema ns. bound to `xs:` or `xsd:`, no target namespace
- XML Schema ns. bound to `xs:` or `xsd:`, target namespace as default namespace
  - Convenient as long as we don't use keys and keyrefs
- Target namespace bound to a prefix (`tns:` by convention)
- Then we can declare XML Schema as default namespace and avoid using `xs:` or `xsd:`