Modelling XML Applications

Patryk Czarnik

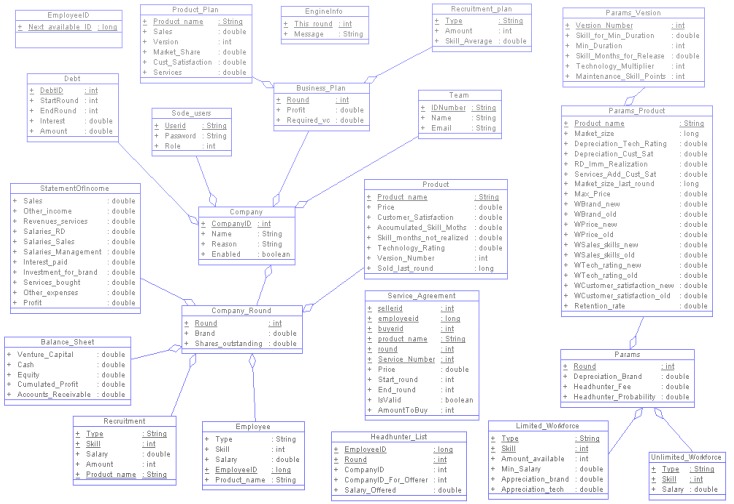
XML and Applications 2016/2017
Lecture 2 – 10.03.2016

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 the 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>
```

Document Type Definition (DTD)

- Defines structure of a class of XML documents (“XML application”).
- Optional and not very popular in new applications.
  - Replaced by XML Schema and alternative standards.
  - It is worth to know it, though. Important for many technologies created 10-30 years ago and still in use.
- Contains declarations of:
  - elements (“element types” to be precise)
  - attributes (“attribute lists”...)
  - entities – described last week
  - notations – extremely rarely used, we'll skip them

Example DTD (fragments)

```
<!ELEMENT teacher (first-name+, last-name)>
<!ATTLIST teacher degree (MSc | PhD | Prof) #REQUIRED
                  guest (yes | no) "no">

<!ELEMENT student (first-name+, last-name, birth-date, idetification)>

<!ELEMENT identification (PESEL | (passport-nr, country)>

<!ELEMENT first-name (#PCDATA)>
...
```

```
<teacher degree="MSc">
    <first-name>Patryk</first-name>
    <last-name>Czarnik</last-name>
</teacher>
```

```
<student>
    <first-name>Henry</first-name>
    <first-name>Walton</first-name>
    <first-name>Junior</first-name>
    <last-name>Jones</last-name>
    <birth-date>1905-05-05</birth-date>
    <identification>
        <passport-nr>1234567890</passport-nr>
        <country>USA</country>
    </identification>
</student>
```

Element declaration in DTD

- Element name
- Element type; one of:
  - EMPTY
  - ANY
  - (*content specification*)
- Content specification is built of
  - element names
  - #PCDATA token*

  joint together using basic regular expression operators.

  - *) #PCDATA is allowed only under special conditions

Symbols in DTD element specifications

- Parenthesis ( )
- Occurrence indicators (postfix operators)
  - ? – zero or one
  - * – zero or more
  - + – one or more
  - no symbol – exactly one
- Combination (infix associative operators)
  - , – sequence (all in the given order)
  - | – choice (one of the given)

XML Schema

- Replacement for DTD in new applications of XML
- Separate W3C standard
  - v 1.0 in 2001 – 3 recommendations
  - v 1.1 in 2012 – 2 recommendations
- “XML Schema definition” (*.xsd) is itself XML document
- Similar capabilities for tree-level structure specification
- Much more capabilities than in DTD for
  - text-level content (“simple types”/ “datatypes”)
  - modularisation of the definition
    (type inference, imports, namespace support)
  - identity constraints (keys and references)
    - in v 1.1 also more advanced constraints
- Much more verbose than DTD

Types in XML Schema

Concept of **type** – one of basic distinctions wrt DTD

- Elements and attributes have specified types
- Type specify allowable content of an element / attribute
  - for elements – also their attributes
  - type spec. does not include identity constraints
- Type is independent of element (or attribute) name
  - many elements may have the same type
  - elements with the same name may have different types “in different places”

Types – categorisation

Types can be categorised with respect to:

- complexity
  - **complex types** define tree-level structure: subelements and attributes; they can be applied to elements only
  - **simple types** define text-level content; they can be applied to elements and attributes
- scope
  - **named types** are defined in global scope and can be used many times
  - **anonymous types** are defined in the place of use
- origin
  - **predefined** / **built-in** – provided by XML Schema
  - **user-defined**

Element declaration

```
<xs:element name="student" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="first-name" type="xs:string" maxOccurs="3" />
      <xs:element name="last-name" type="xs:string" />
      <xs:element name="birth-date" type="xs:date" />
      <xs:element name="identification">
      <xs:complexType>
        <xs:choice>
          <xs:element name="PESEL" type="xs:string"/>
            <xs:sequence>
              <xs:element name="passport-nr" type="xs:string"/>
              <xs:element name="country" type="xs:string"/>
            </xs:sequence>
          </xs:choice>
        </xs:complexType>
      </xs:element>
   </xs:sequence> ...
</xs:complexType></xs:element>
```

```
<!ELEMENT student (first-name+, last-name, birth-date, idetification)>
<!ELEMENT identification (PESEL | (passport-nr, country)>
<!ELEMENT first-name (#PCDATA)>
...
```

Element (or attribute) definition

- Scope of definition
  - **global** – defined at schema level
    - can be referenced from anywhere using `<element ref="...">`
    - element name must be unique within schema (/*namespace*)
    - always qualified names, if schema has a targetNamespace
  - **local** – defined within a complex type or group definition
    - cannot be referenced from other places
    - many elements of same name allowed in schema
    - qualified or unqualified names, determined by `form` and `elementFormDefault` attributes

More 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.

Basic things to look in the examples

“students” - several ways to write a schema for the same document

- Structure of DTD, structure of XML Schema definition
- Typical element definition
- Controlling number of occurrences
- Sequence and choice
- Building complex models (nested groups)
- Defining attributes in schema and DTD

More possibilities
see lab classes

- 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
- Mixed content
  - DTD approach – (#PCDATA| a | b)*
  - Mixed content with controlled subelements – schema only
- Any order (`xs:all`) – schema only

Model groups

- Element content defined with model groups:
  - sequence – all in the given order
  - choice – one of the given choices
  - all – all given elements in any order
- sequence and choice – may be nested, multiplied, etc.
- all – restricted
  - may not be mixed with sequence and choice
  - may not be nested
  - can contain only elements with different names and occurrence number <= 1

Namespaces – motivation

- Same names of tags may denote different things.
- Problematic especially when combining document fragments from different sources into one document.

```
<article code="A1250">
  <title>Assignment in Pascal and C</title>
  <author>
    <fname>Jan</fname> <surname>Mądralski</surname>
    <address>...
      <code>01-234<code>
    </address>
  </author>
  <body>
    <paragraph>
      Assignment is written as <code>x = 5</code> in C
      and <code>x := 5</code> in Pascal.
    </paragraph>
  </body>
</article>
```

XML namespaces – realisation

- **Namespace name** (*identyfikator przestrzeni nazw*)
  – globally unique identifier
  - Universal Resource Identifier (URI) in XML v1.0
  - Internationalized Resource Identifier (IRI) in XML v1.1
- **Namespace prefix** (*prefiks przestrzeni nazw*)
  – local, for convenient reference
  - Local for document or fragment
  - Processors should not depend on prefixes!
- Names resolved and interpreted as pairs:
  (namespace name, local name)
- To make things more complex:
  - scope and overrding
  - default namespace

Usage of namespaces and prefixes

```
<art:article code="A1250"
  xmlns:art="http://xml.mimuw.edu.pl/ns/article"
  xmlns:t="http://xml.mimuw.edu.pl/ns/text-document"
  xmlns:ad="urn:addresses">
  <art:title>Assignment in Pascal and C</art:title>
  <art:author>
    <fname>Jan</fname> <surname>Mądralski</surname>
    <ad:address>...
      <ad:code>01-234</ad:code>
    </ad:address>
  </art:author>
  <art:body>
    <t:paragraph>
      Assignment is written as <t:code>x = 5</t:code> in C
      and <t:code>x := 5</t:code> in Pascal.
    </t:paragraph>
  </art:body>
</art:article>
```

Namespaces – overriding and scopes

```
<pre:article code="A1250" xmlns:pre="http://xml.mimuw.edu.pl/ns/article">
  <pre:title>Assignment in Pascal and C</pre:title>
  <pre:author>
    <fname>Jan</fname> <surname>Mądralski</surname>
    <pre:address xmlns:pre="urn:addresses">...
      <pre:code>01-234</pre:code>
    </pre:address>
  </pre:author>
  <pre:body>
    <pre:paragraph xmlns:pre="http://xml.mimuw.edu.pl/ns/text-document">
      Assignment is written as <pre:code>x = 5</pre:code> in C
      and <pre:code>x := 5</pre:code> in Pascal.
    </pre:paragraph>
  </pre:body>
</pre:article>
```

Default namespace

- Applies to element names which do not have a prefix.
- Does not apply to attributes.

```
<article code="A1250" xmlns="http://xml.mimuw.edu.pl/ns/article">
  <title>Assignment in Pascal and C</title>
  <author>
    <fname>Jan</fname> <surname>Mądralski</surname>
    <address xmlns="urn:addresses">...
      <code>01-234</code>
    </address>
  </author>
  <body>
    <paragraph xmlns="http://xml.mimuw.edu.pl/ns/text-document">
      Assignment is written as <code>x = 5</code> in C
      and <code>x := 5</code> in Pascal.
    </paragraph>
  </body>
</article>
```

Namespaces – supplement

- **Qualified name** – name with non-empty ns.URI
- **Unqualified name** – name with null (not assigned) ns.
  - elements without prefixes when no default namespace
  - attributes without prefixes – always
- Namespace name
  - Only identifier, even if in form of an address!
  - Should be in form of URI / IRI; some processors do not check it, though
  - Pay attention to every character (uppercase/lowercase, etc.) – most processors simply compare strings
- XML namespaces may be used not only for element and attribute names – e.g. type names in XML Schema

Namespace awareness

- A document may be well-formed as XML while erroneous from the point of view of namespaces.
  - For some applications (usually old ones...) such document might be proper and usable.
- Modern parsers can be configured to process namespaces or not.

  The mentioned document would be
  - parsed successfully by a parser which is not namespace-aware,
  - revoked by a namespace-aware parser.

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

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:`
- See examples ns1.xsd – ns4.xsd