Università degli Studi di Bari

Laboratorio di Acquisizione della Conoscenza e Apprendimento delle Macchine

<? PRACTISING XML ?>

Luigi Iannone

&

Daniele Capursi

Overview

- **Building DTDs**
- **DTDs limits**
- **XML Schemas**
- **Importing pre- existing Schemas**
- **Namespaces**
- **Java & XML**

DTDs in short...

Describe the XML document structure:

- Allowed elements
- Allowed attributes for each element
- Allowed values for attributes
- Proper nestings of elements
- Entities definitions (optional)

... DTDs in short

Two ways of defining them:

- In an external file referencing it from the XML Document
- Embedding the DTD definition inside the document itself (bad solution, does not allow reuse of same DTD for more than one document)

DTD inside the XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Book [
        <!ELEMENT Author (#PCDATA)>
        <!ELEMENT Title (#PCDATA)>
        <!ELEMENT Book (Author+, Title, Editor)>
        <!ELEMENT Editor (#PCDATA)>
]>
<Book>
        <Author>Stephen King</Author>
        <Title>IT</Title>
        <Editor>Sperling &amp;amp; Kupfer</Editor>
</Book>
```

From Internet Explorer

D:\Documents and Settings\Luigi\Documenti\Gestione della Conscenza

File Modifica Visualizza Preferiti Strumenti ?

Indietro Cerca Preferiti Cronologia

Indirizzo D:\Documents and Settings\Luigi\Documenti\Gestione della Conscenza

```
  <?xml version="1.0" encoding="UTF-8" ?>
  <!DOCTYPE Book (View Source for full doctype...)>
- <Book>
    <Author>Stephen King</Author>
    <Title>IT</Title>
    <Editor>Sperling & Kupfer</Editor>
  </Book>
```

DTD defined externally

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Family SYSTEM "Family.dtd">
<Family>
        <Parent name="Leonardo"/>
        <Parent name="Maria"/>
        <Child name="Luigi"/>
</Family>
```

Let us build a DTD

Objective:

We want to write XML documents that describe a family

First idea

A family is composed by parents and children;

each one of them has a name:

Family.dtd

Problems with this DTD

We could have some weird families, e.g.:

WeirdFamily.xml

Fine Tuning our DTD

A family is compound by no more than two parents and zero or more children:

FamilyWithFixedNumberOfParents.dtd

A family

A valid family could be:

MyFamily.xml

But also:

WeirdFamilyWithTwoMaleParents.xml

Fine Tuning again

We want that a family has:

- **One husband**
- **One wife**
- **0 or more children**

So we need…

- **Two different elements in order to distinguish husband and wife**
- **The child element (as before)**

FamilyWithHusbandAndWife.dtd

What's wrong with that?

- **Repeated ATTLIST for each attribute that is owned by different elements (in our case "name")**
- **Cannot invert Husband and Wife saving document validity against this DTD (need a more complicated one enumerating all combinations FamilyWithHusbandAndWifeInAnyOrder.dtd)**

What do we need?

- **Some support for abstract datatypes**
- **Some support for writing concisely structures more complicated than simple sequences**
- **... We want it in XML itself**

What do we need?

XML Schema

Describing the Family by means of XML Schema

- **Let us create a complex type that abstracts all the common attribute for family members**
- **Let us create elements for the different members of the family**
- **Let us put them all together in the structure**

(Family.xsd)

Still problems

- **Parents' sequence order is still important - we could use the "Choice" Model but it allows still weird families**
  WeirdFamilyWithSchema.xml
- **We could use the all Model and it works fine**
  MyFamilyWithRevisedAgainSchemaWithInve
  .xml

Short review

- **We created an abstract datatype for common features of elements (impossible with DTD)**
- **We worked out the problem of describing data structures that are more complex than a sequence (very complicated with DTD)**
- **We made it in XML**

More nice features of XML Schema

- **Import**
- **Built- in Namespace support**

Import allows

- **Sharing Datatypes, Elements and Attributes across several Schemas**
- **Build our Schemas incrementally starting from one (or more) core Schema(s) and building new ones on it (them)**

Namespaces usefulness

Namespaces allow us to distinguish in an XML document between two items with the same name, but defined twice or more within different Schemas

Example

Objective:

We want to build up some XML Documents that represents data for exchanging information about matches in the very famous Italian game called Fantacalcio

Options

- **Build our XML Schema from scratch**
- **Reusing pre- existing Schemas**

What do we have?

- **A previously released schema by FIGC about players and teams**
- **Our application that will analyze Fantacalcio players teams accepting them in a fomat that looks like <Formazione><Giocatore></Giocatore><Giocatore>...</Formazione>**

FICG Schema embeds:

- **Two simple types:**
  - **Ruolo**
  - **NomeSquadraSerieA**
- **One complex type: GiocatoreDiSerieA**
- **Two Elements:**
  - **Formazione**
  - **Giocatore (with type GiocatoreDiSerieA)**

XML Schema by FIGC

Our Schema will

- **Import the former one by FICG and use the Giocatore element**
- **Use the Formazione Element to indicate Fantacalcio players' team.**

Steps

- **Import the FIGC Schema in our DummyFantacalcio Schema**
- **Create our Formazione Element**

Problems

- **We should have a conflict between the Formazione Element from FIGC Schema and the one from DummyFantacalcio**

Namespace definition works it out!

What did we get?

- **We could reuse an “officially” pre- released Schema without any change to it, so gaining:**
  - **High reuse**
  - **Standard compliance for our application**
- **We could keep the accepted format for our application (if already existing) - this means less code to write or change**

Java & XML

Java support for XML includes:

- **JAX-P (Java API for XML Processing)**
- **JAX-M (Java API for XML Messaging)**
- **JAX-RPC (Java API for XML RPC)**
- **JAX-B (Java API for XML Binding)**

JAX-P what is it?

The Java API for XML Processing (JAXP) is for processing XML data using applications written in the Java programming language.

JAX-P

Document Manipulation by

- Document Object Model (DOM)
- Simple Application programming interface for eXtensible mark- up language (SAX)

Factory Design Pattern

- **Definition**
  Provide an interface for creating families of related or dependent objects without specifying their concrete classes[1].

[1]From
http://www.dofactory.com/patterns/PatternAbstract.aspx

JAX-P Factories

The JAXP API package contains two vendor-neutral factory classes:

- SAXParserFactory
- DocumentBuilderFactory

Ability to plug in an XML implementation offered

The implementation depending on javax.xml.parsers.SAXParserFactory and javax.xml.parsers.DocumentBuilderFactory system properties. The default values point to Sun's implementation.

JAX-P Packages

The SAX and DOM APIs are defined by XML-DEV group and by the W3C,respectively. The libraries that define those APIs are:

- **javax.xml.parsers:** common interface for different vendors' SAX and DOM parsers.
- **org.w3c.dom** classes for all of the components of a DOM.
- **org.xml.sax** basic SAX APIs.
- **javax.xml.transform** XSLT APIs.

The "Simple API" for XML (SAX)

- Event-driven, serial-access mechanism
- Element-by-element processing.
- Reads and writes XML to a data repository or the Web (URL).
- For server-side and high-performance apps.

Drawbacks:

- Hard event model to manage even simple situations

The DOM API

- Generally an easier API to use.
- Tree structure of objects.
- Ideal for interactive applications because the entire object model is present in memory.

Drawbacks

- In memory model threatens overall performance

SAX Architecture

Java WS Tutorial

DOM architecture

Java WS Tutorial

Transform

Java WS Tutorial

Loading DOM sample

Code For Loading DOM from XML f

Further Readings

- **Java Web Services Tutorial:**
  http://java.sun.com/webservices/download