XML and Web Application Programming

Overview

- Schema languages for XML
- XML in programming languages
- Web application frameworks

Part I

- Schema languages for XML
  - what is the essence of DTD / XML Schema / RELAX NG?
  - expressiveness of schema languages
  - formal models, variations of regular tree grammars
  - foundation for Part II

Part II

- XML in programming languages
  - how can we integrate XML processing in programming languages?
  - XML schemas as types in programming languages
  - research projects:
    - XDuce
    - XACT

Part III

- Web application frameworks
  - why is Web programming so complicated?
  - Java Servlets and JSP
  - JWIC
  - Google Web Toolkit

Objectives

- These topics illustrate a few areas in (XML∪WWW) ∩ Programming Languages
- At different stages:
  - XML and schemas: now well-understood, essential to other areas of XML/WWW
  - XML programming: many proposals, (still) no "silver bullet"
  - Web programming: new frameworks appear frequently, needs consolidation
- Lots of remaining research opportunities!

Prerequisites

- I assume that you have a basic knowledge of
  - Regular languages
  - Java (and ML)
  - WWW and XML/HTML

Part I

- W3C XML Schema is one spec where your eyes will still twitch and your head still go buzzZZZ even when you read it for the tenth time. Except that you will no longer be surprised by surprises.
  - Michael Kay, editor of W3C's XSLT 2.0

Part II

- "Wow! Thanks for releasing another Java Web application development framework. The 31,918 frameworks currently available fact offer little choice."
  - But now I feel better... I know there are some wild and crazy guys, and I, for one, am going to nominate [insert framework name] for the Web Application Development Frameworks Award. I'm happy I found the time to appreciate the richness and quality of the end-user experience. It seems like you have earned it..."
  - ontheserverside.com as a comment to a tool announcement

Part III

- There are some wild and crazy guys, and I, for one, am going to nominate [insert framework name] for the Web Application Development Frameworks Award. I'm happy I found the time to appreciate the richness and quality of the end-user experience. It seems like you have earned it..."
Overview

- Motivation
- DTD
- XML Schema
- RELAX NG

- focus on language constructs that are essential to expressiveness

Example: XML for business cards

- An XML-based language is a set of XML documents (with some semantics)

Schemas for XML

Validation of XML documents:
- A schema describes the syntax of an XML-based language
- A schema language (DTD / XML Schema / RELAX NG / ...) is a formal notation for specifying schemas
  - like BNF for programming language syntax
- A schema processor checks validity of a document, relative to a schema

This presentation is based on...


XML in programming languages

- Many recent programming languages and APIs are designed for processing XML data
  - Type checking?
  - Type inference??
  - We need formal models of schemas and schema languages!

DTD – Document Type Definition

- Introduced with XML 1.0
- Focuses on elements and attributes
- Little control of attribute values and chardata
- No support for namespaces
- Widely used (still!)

Element declarations

```
<ELEMENT element-name content-model />
```

where content-model is generally a restricted regular expression over element names and #PCDATA

Observations:
- the content model of a particular element only depends on its name!
- content models must be deterministic
- limited control over chardata
Attribute declarations

Each attribute definition consists of:
- an attribute name
- an attribute type (CDATA, enum, ID, IDREF, ...)
- a default declaration (#REQUIRED, #IMPLIED, ...)

Observations:
- the requirements for a particular attribute only depends on its name and the name of the element!
- limited control over attribute values

Example

```xml
<!ATTLIST element-name attribute-definitions

<TYPE card (name, title, email?, phone?, logo?)>
  <ELEMENT name (#PCDATA)>
  <ELEMENT title (#PCDATA)>
  <ELEMENT email (#PCDATA)>
  <ELEMENT phone (#PCDATA)>
  <ELEMENT logo EMPTY>
  <ATTLIST logo uri CDATA #REQUIRED>
</TYPE>
```

Example (1/2)

```xml
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:b="http://businesscard.org"
targetNamespace="http://businesscard.org">
  <element name="card" type="b:card_type"/>
  <element name="name" type="string"/>
  <element name="title" type="string"/>
  <element name="email" type="string"/>
  <element name="phone" type="string"/>
  <element name="logo" type="b:logo_type"/>
  <attribute name="uri" type="anyURI"/>
</schema>
```

Example (2/2)

```xml
<complexType name="card_type">
  <sequence>
    <element ref="b:name"/>
    <element ref="b:title"/>
    <element ref="b:email" minOccurs="0"/>
    <element ref="b:phone" minOccurs="0"/>
  </sequence>
</complexType>
<complexType name="logo_type">
  <attribute ref="b:uri" use="required"/>
</complexType>
</schema>
```

General observations

- DTD can only express "local" properties on XML trees!
- An XML document can be validated by checking the element nodes in any order

Overview

- Motivation
- DTD
- XML Schema
- RELAX NG

XML Schema

- W3C's design goal: make "a better DTD"
- Successes:
  - Namespace support
  - Modularization
  - Data types
  - Type derivation mechanism
- Critique:
  - lots... it's an easy prey

Types and declarations

- Simple type definition:
  defines a family of Unicode text strings (i.e. no markup)
- Complex type definition:
  defines a content and attribute model
- Element declaration:
  associates an element name with a simple or complex type
- Attribute declaration:
  associates an attribute name with a simple type

Complicated structural features

- Type derivation by restriction or extension
- Substitution groups
- Overloading with local declarations
- ...
  - many of these features are not essential to the formal expressiveness
Overloading with local declarations

```xml
<element name="cardlist">
    <complexType>
        <sequence>
            <element name="title" type="string"/>
            <element ref="x:card" .../>
        </sequence>
    </complexType>
</element>
<element name="card">
    <complexType>
        <sequence>
            <element name="title" type="x:Title"/>
        </sequence>
    </complexType>
</element>
```

The Element Declarations Consistent rule

If a content model contains two or more element declarations with the same element name then they must have the same type definition.

Simplifies implementation, central for formal modeling of schemas...

Unique interpretations

- For some applications, an important side-effect of validation is that each tree node is assigned a schema type (i.e. an interpretation)
- Aka. the "post-schema-validation infoset"
- The EDC rule in XML Schema ensures unique interpretations!

What’s being used in practice?

A study of 819 schemas shows:
- simple types and restrictions are heavily used!
- complex type extensions used in only 20%
- substitution groups used in only 6%
- only 27% pass the Schema Quality Check
  - of those, the element structure could in 85% be expressed using DTD (i.e. no overloading)
  - in most of the remaining 15%, types only depend on the parent context!
- Any good explanations?

General observations

- The type associated with an element may (only) depend on the node itself and its ancestors
- The crucial mechanisms are:
  - overloading with local declarations
  - the EDC rule

Overview

- Motivation
- DTD
- XML Schema
- RELAX NG

RELAX NG

- OASIS + ISO competitor to XML Schema
- Designed for simplicity and expressiveness, solid mathematical foundation

Processing model

- For a valid instance document, the root element must match a designated pattern
- A pattern may match elements, attributes, or character data
- Element patterns can contain sub-patterns that describe contents and attributes

Patterns

- `<element name="..."> ... </element>
- `<attribute name="..."> ... </attribute>
- `<text/>
- `<group> ... </group>` (concatenation)
- `<choice> ... </choice>` (union)
- `<zeroOrMore>` ... </zeroOrMore> (Kleene star)
- `<oneOrMore>` ... </oneOrMore>
- `<optional>` ... </optional>
- `<empty/>
- `<...`
Example

```
<element name="card">  
  <element name="name">John Doe</element>  
  <element name="title">CEO</element>  
  <element name="email">john.doe@example.com</element>  
  <optional>
    <element name="phone">(123) 456-7890</element>
  </optional>
</element>
```

Grammars

- **Pattern definitions and references** allow description of recursive structures

```
<grammar ...
  <start>...
  </start>
  <define name="...">...
  </define>
</grammar>
```

Grammar simplification

- The spec explains a simplification process
  - §4.19:
    - every `element` is the child of a `define`
    - the child of every `define` is an `element`
    (i.e., we may assume a 1-1 correspondence between named pattern definitions and element patterns)
    - ...
    (we need this later...)

No surprising restrictions!

```
<element name="X">  
  <choice>  
    <group>  
      <element name="X">  
        <optional><ref name="P"></ref></optional>  
        <optional>  
          <attribute name="A"><text/></attribute>  
          <attribute name="B"><text/></attribute>  
        </optional>  
      </element>  
      <element name="X"></element>  
    </group>  
  </choice>  
</element>
```

General observations

- RELAX NG schemas can express many "non-local" properties on XML trees!
- Validation is more difficult but can be performed in linear time (in the size of the XML tree)!
- Unique-interpretations property fails!
- We shall later examine the formal model underlying the design of RELAX NG...

Formal models of XML schemas: regular tree grammars and subclasses

```
Σ  δ  ε  \rightarrow
```

Schemas as formal languages

- Let \( S \) be a schema (written in DTD, XML Schema, or RELAX NG)
- Define \( \mathcal{L}(S) \) as the set of XML documents that are valid relative to \( S \):
  \[ \mathcal{L}(S) = \{ X | X \text{ is valid relative to } S \} \]

Decision problems

[Membership] The problem "given a schema \( S \) and an XML document \( X \), is \( X \in \mathcal{L}(S) ? \)" is (fortunately) decidable

How about the following?

- [Equality] "given two schemas \( S_1 \) and \( S_2 \), is \( \mathcal{L}(S_1) = \mathcal{L}(S_2) ? \)"
- [Subset] "given two schemas \( S_1 \) and \( S_2 \), is \( \mathcal{L}(S_1) \subseteq \mathcal{L}(S_2) ? \)"
- [Emptiness] "given a schema \( S \), is \( \mathcal{L}(S) = \emptyset ? \)"

Translating schemas


dk.brics.schematools: [http://www.brics.dk/schematools/](http://www.brics.dk/schematools/)
Closure properties

- Is DTD (or XML Schema or RELAX NG) closed under union?
  ... intersection?
  ... complement?

All these decision problems and closure properties are relevant when considering schemas as types in programming languages!

Overview

all sets of XML trees

regular tree grammars
  ➔ RELAX NG

single-type tree grammars
  ➔ XML Schema

local tree grammars
  ➔ DTD

[Murata et al., TOIT 2005]

Regular languages

- You know about regular languages on strings... (finite automata, regular grammars, etc.)

- We shall now work with regular languages on trees

Which kinds of trees?

- Labeled, ordered, unranked trees
  - each node has a label (i.e. the element name; we ignore attributes and character data...)
  - each node has an ordered sequence of child nodes
  - the child sequences may have different lengths

Context-free grammars (over strings)

- A context-free grammar is a 4-tuple $G = (N, T, S, P)$ where
  - $N$ is a finite set of nonterminals
  - $T$ is a finite set of terminals
  - $S$ is a set of start symbols, where $S \subseteq N$
  - $P$ is a finite set of production rules of the form $X \rightarrow \alpha$ where $X \in N$ and $\alpha \in (T \cup N)^*$

- The language of $G$, denoted $L(G)$, is the set of strings over $T$ that can be derived from a start symbol (with the usual definition of "derived"...)

Regular tree grammars

- A regular tree grammar is a 4-tuple $G = (N, T, S, P)$ where
  - $N$ is a finite set of nonterminals
  - $T$ is a finite set of terminals
  - $S$ is a set of start symbols, where $S \subseteq N$
  - $P$ is a finite set of production rules of the form $X \rightarrow a[r]$ where $X \in N$, $a \in T$, and $r$ is a regular expression over $N$

- The language of $G$, denoted $L(G)$, is the set of trees over $T$ that can be derived from a start symbol (with the obvious definition of "derived"...)

Example

Let $G = (N, T, S, P)$ be defined by

- $N = \{Doc, Para1, Para2, PCDATA\}$
- $T = \{doc, para, pCDATA\}$
- $S = \{Doc\}$
- $P = \{Doc \rightarrow doc[Para1,Para2*], Para1 \rightarrow para[ε], Para2 \rightarrow para[PCDATA], PCDATA \rightarrow pCDATA[ε]\}$

- A tree that can be derived from $G$:

  (The corresponding XML document:
   `<doc><para/><para>text chunk</para></doc>`)

- What is $L(G)$?

Regular tree grammars and finite-state automata

- Everything you know about regular languages and finite automata on strings generalizes elegantly to trees!
  - grammars vs. automata
  - automata operations
  - closure properties
  - decision procedures
  - ...
  - and (hopefully) also your intuition about expressibility and limitations!

Regular tree grammars vs. RELAX NG

- From (simplified) RELAX NG schema to regular tree grammar*:
  - named pattern $⇒$ nonterminal
  - element name $⇒$ terminal

* Ignoring name classes, interleaves, attributes, and chardata...
Local tree grammars

A local tree grammar is a regular tree grammar $G = (N, T, S, P)$ where $P$ contains no two productions on the form

$X \rightarrow \alpha[r]$  
$Y \rightarrow \alpha[r']$

where $X \neq Y$, $\alpha \in T$, and $r$ and $r'$ are regular expressions over $N$.

In other words, the terminal (=element name) uniquely determines the regular expression (=content model).

Local tree grammars vs. DTD

In DTD, the element name uniquely determines the content model!

From DTD schema to local tree grammar:

```
<!ELEMENT foo (bar|baz+)>
<!ELEMENT bar ...>
<!ELEMENT baz ...>
```

```
Foo \rightarrow foo[Bar|Baz+]
Bar \rightarrow bar[...]
Baz \rightarrow baz[...]
```

We omit the (many) details.....

Regular tree grammars vs. XML Schema

[Murata et al., TOIT 2005] explains how XML Schema can be modeled with regular tree grammars:

```
[complexType]
complexType \Rightarrow nonterminal
elementDeclaration \Rightarrow terminal
```

We omit the (many) details.....

Single-type tree grammars

Let $G = (N, T, S, P)$ be a regular tree grammar

Two different nonterminals $X,Y \in N$ compete if $P$ contains two productions on the form

$X \rightarrow \alpha[r]$  
$Y \rightarrow \alpha[r']$

for some $\alpha \in T$.

$G$ is a single-type tree grammar if

- for each production rule in $P$ no two nonterminals on the right-hand-side compete, and
- start symbols in $S$ do not compete

$G$ local $\Rightarrow$ $G$ single-type

Example of violation

```
<element name="E">
  <complexType>
    <choice>
      <element name="F" type="X"/>
      <element name="F" type="Y"/>
    </choice>
  </complexType>
</element>
```

violates EDC!

```
F1 \rightarrow F[X]  
F2 \rightarrow F[Y]
```

$F_1$ and $F_2$ compete!
A design bug...  

- XML Schema can essentially be modeled as single-type tree grammars, but...
  
- the any constructs in XML Schema makes it possible to express certain non-single-type tree grammars!

- See example in [Murata et al., TOIT 2005]

Closure properties

<table>
<thead>
<tr>
<th></th>
<th>∩</th>
<th>∪</th>
<th>\</th>
</tr>
</thead>
<tbody>
<tr>
<td>regular (RELAX NG)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>single-type (XML Schema)</td>
<td>✓</td>
<td>▷</td>
<td>▷</td>
</tr>
<tr>
<td>local (DTD)</td>
<td>✓</td>
<td>▷</td>
<td>▷</td>
</tr>
</tbody>
</table>

[Murata et al., TOIT 2005]

Summary

- foundation for treating **XML schemas as types** in XML programming languages