9. Documents and Document Models

INFO 202 - 26 September 2007

Bob Glushko
Plan for INFO Lecture #9

What is a Document?

Document Types

Using XML to Encode Models of Document Types

"Business rules" Governing Documents and Document Types

The Document Type Spectrum

XML Vocabularies

SylViA, Microformats, Tim Bray, and the Modeling Tradeoff
What Is A Document?

A *Document* is a purposeful and self-contained collection of information.

This definition focuses on the information content, not on the physical container or medium, format, or technology in which the information is conveyed.

This broader definition includes information extracted from databases or applications, spreadsheets, printed or web forms, as well as traditional printed documents.
Document Types

Any definition of "document" allows for a notion of different types of documents.

This idea can be very intuitive and very informal, or we can be more precise and define a *model of a document type* as the rules or constraints that distinguish one type from another.

This expression of the model is *conceptual* and is independent of the syntax and technology in which document instances are ultimately implemented.

But most of the time the model is ultimately implemented as a *physical* view in some specific syntax like XML.
Models of Document Types

A model of a document type captures the distinctions between documents that make a difference.

Similar types of content occur in many document models and there is often overlap in information and structural patterns.

Models of document types can be very specific ("purchase order for industrial chemicals when buyer and seller are in different countries") or very abstract ("fill-in-the-blank legal form for contract").

The constraints on structure, data, and behavior of the information in the conceptual and physical models of a document type can collectively be called the Business rules.
How XML Implements Models of Document Types

XML gives the idea of document type a more physical, formal foundation

XML has syntactic mechanisms that capture the conceptual distinctions between document types in terms of:

- *Elements* and *attributes* used to encode their content
- Rules that govern how elements and attributes are organized
- Possible values for elements and attributes

These are the *vocabulary* and the *grammar* of the language defined by the document type
Comparing the Syntax of HTML and XML

```html
<HTML>
<BODY>
<H1>Purchase Order (#1234)</H1>
<HR>
<H2>Buyer Information</H2>
<P>Smith and Company (Buyer # AB24567)
...
</BODY>
</HTML>

<PurchaseOrder OrderNo="1234">
<Buyer BuyerNo="AB24567">
{Name>Smith and Company{Name>
<Address1>123 High Street<Address1>
<Address2>Suite 100<Address2>
<City>Anytown</City>
<State>California</State>
<ZipCode>12345</ZipCode>
...
```
HTML's Design

Simple structures: headings, lists, links (HTML 1.0 had only 13 elements in 1991)

Format-oriented, meant to be understood "by eye" and not by computers

Browsers are "hard wired" to render HTML in fixed manner as web pages and forgive markup errors

Around 1994 after the development of the first graphical browsers many companies and entrepreneurs wanted to use the Internet and Web as a platform for business

But HTML wasn't designed for this purpose
XML's Design

HTML's idea of using tags to mark up pieces of text according to how they should appear on the page for people to read them is very easy to understand.

XML's purpose is radically different – it is a general syntax for encoding models and instances of "things and processes".

The big difference is that the HTML document has a fixed set of element types ("tags") that it might contain, while an XML document instance is potentially unlimited in what tags it can contain.

So XML is a *metalanguage* that can be used to define new languages or *applications*; HTML is an example of one.
Markup is "Information IQ"

Content/structure-based text objects: XML, SGML, databases

Formatted electronic text: HTML, EDI, word processing files

Unstructured electronic text: ASCII

Printed text

Easier to translate to

more "processability" / reusability
Smart Markup Creates Explicit Text Objects

How much you can do with information depends on the extent and explicitness of its markup

Markup transforms a flat stream of text into a set of objects or elements that can be manipulated by other applications

HTML is rich or smart compared with ASCII

And all of these syntaxes are poor or dumb compared with a content-oriented data model, such as those possible with XML, SGML, or a database
My Information Can Beat Up Your Information [1]

Two separate and important ideas:

- Use a syntax capable of encoding information in a rich and easily processed manner
- Use it effectively

It is almost always better to make information smart so that it can be processed by simple and generic tools than to bury the intelligence into a custom processing application tuned for a particular ad hoc syntax.

Try to capture your data when it is smartest, when it is created, and keep it in that smart data model as long as you can.
My Information Can Beat Up Your Information [2]

- Smart data is more portable and reusable by multiple applications
- Smart data can always be turned into "dumb" data, but not vice versa
- Programs that work with dumb data are always more brittle because they have to do more processing to detect and recover from data errors
Creating Document Type Models

Encoding a conceptual information model in XML means choosing elements or attributes as the containers for information, adding information about data types, applying naming rules, creating structures to organize repeated content components, and so on.

If you've done a careful document analysis and design, the encoding stage is relatively simple and straight-forward and can even be automated in some cases.
Elements

Elements are the building blocks in XML documents.

They define the hierarchy or logical "containers" by enclosing content with both a begin and end tag; the hierarchy provides context for understanding the child elements.
Attributes

Elements may also have one or more attributes (a name-value pair) associated only with their start tag and the values must always be quoted with matching ' or "

- `<PurchaseOrder number="12">purchase order content</PurchaseOrder>`
- The order of attributes is not significant

Elements with attributes but no content are said to be "empty" and have a different tag syntax

- `<Portrait image="bob.gif"/>`
Elements {and,or,vs} Attributes

<Book>
  <Title>Moby Dick</Title>
  <Author>Melville</Author>
  <PublicationYear>1851</PublicationYear>
  <Category>Fiction</Category>
</Book>

<Book title="Moby Dick" author="Melville" publicationYear="1851" category="Fiction"/>

<Book title="Moby Dick" author="Melville" category="Fiction" publicationYear="1851"/>

<Book title="Moby Dick" author="Melville" publicationYear="1851" fiction="True"/>
XML Schemas

The formal description of a document model in XML is called its *schema*

XML schemas (lower case "s" for now) attempt to encode the conceptual model in terms of the syntactic constructs of elements and attributes

- What elements are allowed (the *vocabulary*)
- Where the are allowed – sequence, choice, occurrence and co-occurrence (the *grammar*)
- What values they can take (*datatypes*) – string, integer, decimal, etc.
Why We Need Schemas

If you can represent these rules that define a document type in a form that is "computable" or "processable" then:

- It can guide the creation of valid document instances in editors like XML Spy or Oxygen, or when information is exported from a database or other application
- It can be a model for application programming in the development of Web forms or other GUIs or can be a template for objects in other programming environments
- It can communicate the model to others who need to create or receive document instances
XML Schemas and Schema Languages

XML has several schema languages that differ in how completely they can encode a document type's conceptual model.

The most common of these are Document Type Definitions (DTDs) and XML Schemas (XSD) -- examples to follow.

No schema language is perfect; there is always some compromise between:

- Expressiveness – the range of models that can be described
- Functionality – the set of features used to define a model
- Usability – the ease with which a model can be defined
- Reusability – how readily a model or parts of models can be included in another one
- ...and a range of other "ilities"
DTD for Simple Event Calendar

```xml
<!ELEMENT Calendar (Organization, TimePeriod, Events)>
<!ELEMENT Organization (#PCDATA)>
<!ELEMENT TimePeriod (#PCDATA)>
<!ELEMENT Events (Event+)>  

<!ELEMENT Event (Title, Description?, Speaker?, DateTime, Location)>  
<!ATTLIST Event   
    type (Lecture | Workshop) #IMPLIED>  

<!ELEMENT Title (#PCDATA)>  
<!ELEMENT Description (#PCDATA | Keyword)*>  

<!ELEMENT Keyword (#PCDATA)>  
<!ELEMENT Speaker (Name, Affiliation)>  
<!ELEMENT Name (#PCDATA)>  
<!ELEMENT Affiliation (#PCDATA)>  
<!ELEMENT DateTime (#PCDATA)>  
<!ELEMENT Location (#PCDATA)>  
```
XML Schema for Simple Event Calendar

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="Calendar">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Organization" type="xs:string"/>
        <xs:element name="TimePeriod" type="xs:string"/>
        <xs:element name="Events">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Event" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Title" type="xs:string"/>
                    <xs:element name="Description" minOccurs="0">
                      <xs:complexType mixed="true">
                        <xs:choice minOccurs="0" maxOccurs="unbounded">
                          <xs:element name="Keyword" type="xs:string"/>
                        </xs:choice>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="Speaker" minOccurs="0"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
"Business Rule" is an up and coming buzzword in information systems that is a lot broader than its name might suggest.

A Business Rule is a formal or refined statement of a requirement that is expressed in a technology-independent fashion.

By capturing requirements as Business Rules we can envision implementing or enforcing them externally from the "generic" aspects of applications rather than scattering them into application code.
My Taxonomy of Rules [1]

Rules that Apply to Conceptual Models

- Semantic
- Structural
- Usage
My Taxonomy of Rules [2]

Rules that (Can Also) Apply to Physical Models
  - Syntactic
  - Processing
  - Presentational

Rules that Apply to Instances
  - Content
Semantic Rules

Establish properties: "The Order Number is an identifier that is unique to the Buyer"

Express generalizations: "An Auto is a type of Vehicle"

Expose dependencies: "The Price of an Item can depend on the Buyer"
Structural Rules

Define co-occurrence or aggregation relationships between components

"Each Illustration must have a Caption"

"Every Order must have an Order Number, a Buyer and an Issue Date"
Usage Rules

Define policies or privileges governing access to information

Often based on organizational roles or responsibilities

"An Employee Salary can be viewed by a Manager but can only be changed by someone in the Human Resources Department"
Syntactic Rules

Concern the form in which models are encoded

 Might specify a particular XML vocabulary

 "All Purchase Orders must be encoded according to ANSI X12 850"

 "All Purchase Orders must conform to the Universal Business Language"
Processing Rules

Define actions to be taken whenever a given condition occurs or set of information is encountered

Also called *Procedural* or *Behavioral* rules

"The Seller will respond to an Order with an Order Response"

"If the Item is Out of Stock, create a Back Order Request"

"If the Customer Account Balance is greater than $1000, do not allow any more purchases"
Presentation Rules

Govern the appearance or rendering of an information component

"The Item Description must be adjacent to the Product Image"

"The Title should be centered"
Content Rules

Establish constraints about the values of information components or dependencies among them

"The Total Value of an Order cannot exceed $100,000"

"The Currency Code should be expressed using ISO 4217 codes"

"The Shipping Address must be the same as the Billing Address"

"The Start Date must be earlier than the End Date"

"The Customer's Account Balance must be greater than 0"
Combined Rule Types

"If the Purchase Order Amount exceeds $10000, the Party must supply a Duns Number"

"If the Customer's Account Balance exceeds $1000, display it in Red"

"If the Customer's Account Balance exceeds $10000, send a Past Due Reminder by email, fax, and voice mail"
Patterns of Business Rules

Document types differ in the mixture of business rules from these seven categories.

Put another way, the mixture of rules determines the kind of document type it is:

- **Transactional** documents have many semantic, usage, processing, and content rules.
- **Narrative** documents tend to have more structural and presentational rules.

The mixture of these business rules varies in systematic ways across the "spectrum" of document types.
The Document Type Spectrum
XML Vocabularies

When a model of a document type is (AT BEST PARTIALLY) encoded in XML it is often called an XML "vocabulary" or XML "application"

But this conflicts with the more common usage of that word to mean "software application" so I try to always use vocabulary

The best thing about XML is the ease with which anyone can create a new vocabulary

The worst thing about XML is the same as the best thing: the ease with which anyone can create a new vocabulary
Sylvia -- The Syllabus Viewer

Started as project in Spring 2004 XML programming course, became Final Project in 2005 (Carolyn Cracraft and Lisa de Larios-Heiman)

The problem - different course syllabi with incompatible data models

No automated reuse of information to create composite syllabus that every student needs to manage work load and assignments

Each syllabus created by different process, in different formats, and with different model for topics, readings, assignments, and so on

Typical of problems that occur in every large organization with timesheets, expense reimbursement, registration, etc. and also representative of B2B interactions with incompatible catalogs, orders, etc.
Syllabus Project Goals

Create an overall conceptual model that could describe the syllabi for I-School courses

Eliminate (or encourage the migration from) incompatible syllabus models and applications

Enable the creation of new types of syllabus documents that assemble syllabi from different courses
Microformats represent a very different approach to modeling and encoding "document types"

Instead of following rigorous (and slow) design methods to create standard reusable information models they are more of a grass-roots effort to create XHTML "chunks" embeddable in web pages

Microformats currently exist for personal contact information, events, and a few other small chunks of structured data
The Modeling Debate

Some problems and some domains are inherently complex and a careful, rigorous modeling approach is required to identity and encode the semantic constraints and other "business rules"

- This "heavyweight" position argues that there are no modeling shortcuts
- Schemas for complex domains that are developed without a stage of careful conceptual design are rarely very useful because they are too closely tied to the particular instances used

But some people argue that modeling "involves a substantial amount of work that is often political, tedious, and unpleasant" that should be avoided whenever possible

- Some domains and use cases might be simple enough ("microformats") that less "heavyweight" modeling approaches could suffice

A rebuttal argument from the "heavyweight" side might point to the Dublin Core
Readings for INFO Lecture #10

"Introduction to Relational Databases" Ian Gilfillan

"Database Normalization" Ian Gilfillan