Data Modeling in Semantic Web

Basics

Erdoğan Doğdu

Adapted from Dieter Fensel and Federico Facca (University of Innsbruck) notes

Data modeling

- Web of Document (no structured data)
- XML (semi-structured data)
- Web of Data (structured data)
  - Structured data → Schema
  - Schema → Data model and language

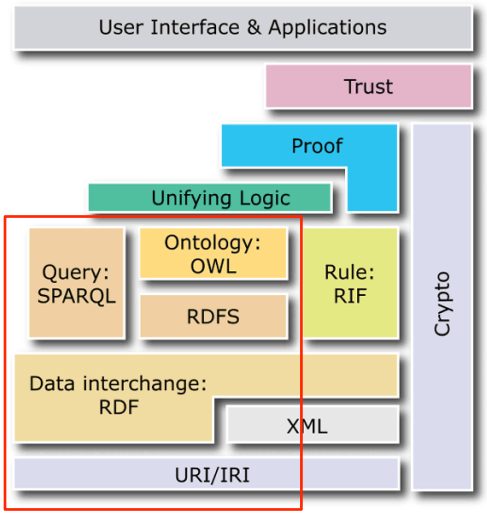
Semantic web requirements

- Layered and extensible
- Support for
  - Web standards (URI, HTTP, …)
  - Complex data
  - Querying
  - …

Semantic Web data model

- Based on ontologies and ontology languages
  - RDF: Resource Description Language
  - RDFS: RDF Schema
  - OWL: Web Ontology Language
- All 3 are W3C standards
  - RDF Primer: http://www.w3.org/TR/rdf-primer/
  - RDF Semantics: http://www.w3.org/TR/rdf-mt/
  - OWL2: http://www.w3.org/TR/owl2-overview/

Standards



Standards

- Underlying standards first
  - **UNICODE**: Character encoding
  - **URI**
  - **XML** and **XML Schema**

    - Following notes are from Dieter Fensel and Federico Facca's Semantic Web lecture notes at Innsbruck University
      - http://www.sti-innsbruck.at/teaching/curriculum/semantic-web

Outline

- Rest of the notes are from *Dieter Fensel and Federico Facca (University of Innsbruck) notes*
- Includes
  - UNICODE
  - URI
  - XML
  - Namespaces
  - XML Schema

More than a-z, A-Z

UNICODE

Character Sets

- ASCII – 7 bit, 128 characters (a-z, A-Z, 0-9, punctuation)
- Extension code pages – 128 chars (ß, Ä, ñ, ø, Š, etc.)
  - Different systems, many different code pages
  - ISO Latin 1, CP1252 – Western languages (197 = Å)
  - ISO Latin 2, CP1250 – East Europe (197 = Ĺ)
- Code page is an interpretation, not a property of text
- Thus if we do not interpret correctly the code page, the result visualized will not be the expected one

UNICODE: a unambiguous code

- We need a solution that can be unambiguously interpreted, i.e. where to a code correspond a single character and viceversa
- That's why UNICODE was created!

$	Å	Ĺ	Æ	ή
U+0024	U+00C5	U+0139	U+00C6	U+03AE

ญ	Ж	♥	⅝	ك
U+0643	U+215D	U+2665	U+0416	U+0E0D

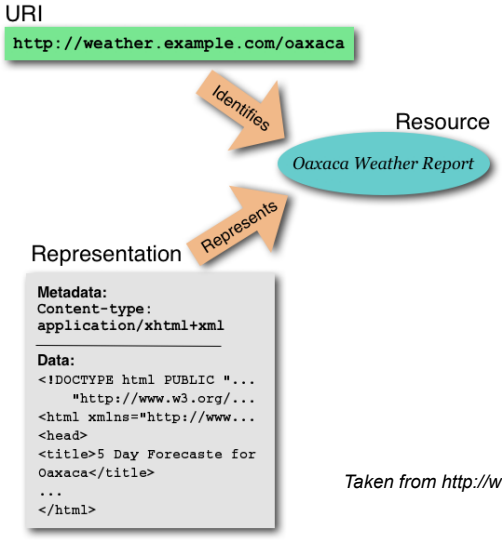
UNICODE

- ISO standard
  - About 100'000 characters, space for 1'000'000
  - Unique code points from U-0000 through U-FFFF to U-10FFFF
  - Well-defined process for adding characters
- When dealing with any text, simply use UNICODE
  - Character code charts: http://www.unicode.org/charts/
- See also:
  - http://www.tbray.org/talks/rubyconf2006.pdf
  - http://tbray.org/ongoing/When/200x/2003/04/06/Unicode

How to identify things on the Web

URI: UNIFORM RESOURCE IDENTIFIERS

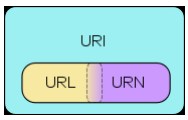
Identifier, Resource, Representation



Taken from http://www.w3.org/TR/webarch/

URI, URN, URL

- A Uniform Resource Identifier (**URI**) is a string of characters used to **identify a name or a resource on the Internet**



- A URI **can be a URL or a URN**
- A Uniform Resource Name (**URN**) defines an item's identity
  - the URN *urn:isbn:0-395-36341-1* is a URI that specifies the identifier system, i.e. International Standard Book Number (ISBN), as well as the unique reference within that system and allows one to talk about a book, but doesn't suggest where and how to obtain an actual copy of it
- A Uniform Resource Locator (**URL**) provides a method for finding it
  - the URL *http://www.sti-innsbruck.at/* identifies a resource (STI's home page) and implies that a representation of that resource (such as the home page's current HTML code, as encoded characters) is obtainable via HTTP from a network host named www.sti-innsbruck.at

URI Syntax

- Examples
  - http://www.ietf.org/rfc/rfc3986.txt
  - mailto:John.Doe@example.com
  - news:comp.infosystems.www.servers.unix
  - telnet://melvyl.ucop.edu/
- URI Syntax **scheme: [//authority] [/path] [?query] [#fragid]**
  - The scheme distinguishes different kinds of URIs
  - Authority normally identifies a server
  - Path normally identifies a directory and a file
  - Query adds extra parameters
  - Fragment ID identifies a *secondary resource*

URI Syntax cont'd

- Reserved characters (like /:?#@$&+*)
- Many allowed characters
- Rest percent-encoded from UTF-8
  - http://google.com/search?q=technikerstra%C3%9Fe
- IRI – Internationalized Resource Identifier
  - Allows whole UNICODE
  - Specifies transformation into URI – mostly UTF-8 encoding

URI Schemes

- Schemes partition the URI space into subspaces
- Schemes can add or clarify properties of resources
  - Ownership (how authorities are formed)
  - Persistence (how stable the URIs should be)
  - Protocol (default access protocol)

Scheme	Description	RFC
file	Host-specific file names	[1738]
ftp	File Transfer Protocol	[1738]
http	Hypertext Transfer Protocol	[2616]
https	Hypertext Transfer Protocol Secure	[2818]
im	Instant Messaging	[3860]
imap	internet message access protocol	[5092]
ipp	Internet Printing Protocol	[3510]
iris	Internet Registry Information Service	[3981]
ldap	Lightweight Directory Access Protocol	[4516]
mailto	Electronic mail address	[2368]
mid	message identifier	[2392]

From http://www.iana.org/assignments/uri-schemes.html

How to exchange structured data on the Web

XML: EXTENSIBLE MARKUP LANGUAGE

eXtensible Markup Language

- Language for creating languages
  - "Meta-language"
  - XHTML is a language: HTML expressed in XML
- W3C Recommendation (standard)
  - XML is, for the information industry, what the container is for international shipping
  - For structured and semistructured data
- Main plus: wide support, interoperability
  - Platform-independent
- Applying new tools to old data

Structure of XML Documents

- Elements, attributes, content
- One root element in document
- Characters, child elements in content

XML Element

- Syntax **<name>contents</name>**
  - **<name>** is called the opening tag
  - **</name>** is called the closing tag
- Examples

  <gender>Female</gender>

  <story>Once upon a time there was…. </story>
- Element names case-sensitive

Attributes to XML Elements

- Name/value pairs, part of element contents
- Syntax

<name attribute_name="attribute_value">contents</name>

- Values surrounded by single or double quotes
- Example

  <temperature unit="F">64</temperature>

  <swearword language='fr'>con</swearword>

Empty Elements

- Empty element: **<name></name>**
- This can be shortened: **<name/>**
- Empty elements may have attributes

- Example

```
<grade value='A'/>
```

Comments

- May occur anywhere in element contents or outside the root element
- Start with **<!--**
- End with **-->**
- May not contain a double hyphen
- Comments cannot be nested

- Example:

```
<element>content
  <!-- a comment, will be ignored in processing -->
</element>
<!-- comment outside the root element -->
```

Nesting Elements

- Elements may contain other (*child*) elements
  - The containing element is the *parent element*
- Elements must be properly nested
- Example with improper nesting:

  bold bold-italic italic?

- The above is not XML (not well-formed)

Special Characters in XML

- < and > are obviously reserved in content
  - Written as < and >
- Same for ' and " in attribute values
  - Written as ' and "
- Now & is also reserved
  - Written as &
- Any character: ß or ß → ß
  - Decimal or hexa-decimal unicode code point
- Elements and attributes whose name starts with “xml” are also special

Uses of XML

- Document mark-up – XHTML
  - HTML is a language, so it can be expressed in XML
- Exchanged data
  - Scalable vector graphics – SVG
  - E-commerce – ebXML
  - Messaging in general – SOAP
  - And many more standards
- Internal data
  - Databases
  - Configuration files
- Etc.

Why XML?

- For semistructured data:
  - Loose but constrained structure
  - Unspecified content length
- For structured data:
  - Table(s) or similar rows
  - Well-defined structure, data types
  - Good interoperability
  - But: requirements for quick access, processing

XML Parsers

- Document Object Model (DOM) builder
  - Creates an object model of XML document
  - In-memory representation, random access
  - DOM complex, simpler JDOM etc.
- Simple API for XML parsing (SAX)
  - Views XML as stream of events
  - el_start("date"), attribute("day", "10"), el_end("date")
  - Calls your handler
  - DOM builder can use SAX
- Pull parsers
  - You call source of XML events

How to distinguish categories of resources

NAMESPACES

The Problem

- Documents use different vocabularies
  - Example 1: CD music collection
  - Example 2: online order transaction
- Merging multiple documents together
  - Name collisions can occur
    - Example 1: albums have a **<name>**
    - Example 2: customers have a **<name>**
  - How do you differentiate between the two?

The Solution: Namespaces!

- What is a namespace?
  - A syntactic way to differentiate similar names in an XML document
- Binding namespaces
  - Uses Uniform Resource Identifier (URI)
    - e.g. “http://example.com/NS”
  - Can bind to a named or “default” prefix

Namespace Binding Syntax

- Use "xmlns" attribute
  - Named prefix
    - **<a:foo xmlns:a='http://example.com/NS'/>**
  - Default prefix
    - **<foo xmlns='http://example.com/NS'/>**
- Element and attribute names are "qualified"
  - URI, local part (or "local name") pair
    - e.g. { "http://example.com/NS" , "foo" }

Example Document I

- Namespace binding

```
<?xml version='1.0' encoding='UTF-8'?>
 <order>
    <item code='BK123'>
          <name>Care and Feeding of Wombats</name>
          <desc xmlns:html='http://www.w3.org/1999/xhtml'>
                The <html:b>best</html:b> book ever written!
          </desc>
    </item>
</order>
```

Example Document II

- Namespace scope

```
<?xml version='1.0' encoding='UTF-8'?>
 <order>
    <item code='BK123'>
          <name>Care and Feeding of Wombats</name>
          <desc xmlns:html='http://www.w3.org/1999/xhtml'>
                The <html:b>best</html:b> book ever written!
          </desc>
    </item>
</order>
```

Example Document III

- Bound elements

```
<?xml version='1.0' encoding='UTF-8'?>
 <order>
    <item code='BK123'>
          <name>Care and Feeding of Wombats</name>
          <desc xmlns:html='http://www.w3.org/1999/xhtml'>
                The <html:b>best</html:b> book ever written!
          </desc>
    </item>
</order>
```

How to define XML document structures

XML SCHEMA

What is it?

- A grammar definition language
  - Like DTDs but better
    - Uses XML syntax
  - Defined by W3C
- Primary features
  - Datatypes
    - e.g. integer, float, date, etc…
  - More powerful content models
    - e.g. namespace-aware, type derivation, etc…

XML Schema Types

- Simple types
  - Basic datatypes
  - Can be used for attributes and element text
  - Extendable
- Complex types
  - Defines structure of elements
  - Extendable
- Types can be named or “anonymous”

Simple Types

- DTD datatypes
  - Strings, ID/IDREF, NMTOKEN, etc…
- Numbers
  - Integer, long, float, double, etc…
- Other
  - Binary (base64, hex)
  - QName, URI, date/time
  - etc…

Deriving Simple Types

- Apply facets
  - Specify enumerated values
  - Add restrictions to data
  - Restrict lexical space
    - Allowed length, pattern, etc…
  - Restrict value space
    - Minimum/maximum values, etc…
- Extend by list or union

A Simple Type Example

- Integer with value (1234, 5678]

```
<xsd:simpleType name='MyInteger'>
    <xsd:restriction base='xsd:integer'>
        <xsd:minExclusive value='1234'/>
        <xsd:maxInclusive value='5678'/>
    </xsd:restriction>
</xsd:simpleType>
```

A Simple Type Example II

- Validating integer with value (1234, 5678]

```
<data xsi:type='MyInteger'></data>                  INVALID
<data xsi:type='MyInteger'>Andy</data>              INVALID
<data xsi:type='MyInteger'>-32</data>               INVALID
<data xsi:type='MyInteger'>1233</data>              INVALID
<data xsi:type='MyInteger'>1234</data>              INVALID
<data xsi:type='MyInteger'>1235</data>
<data xsi:type='MyInteger'>5678</data>
<data xsi:type='MyInteger'>5679</data>              INVALID
```

Complex Types

- Element content models
  - Simple
  - Mixed
    - Unlike DTDs, elements in mixed content can be ordered
  - Sequences and choices
    - Can contain nested sequences and choices
  - All
    - All elements required but order is not important

A Complex Type Example I

- Mixed content that allows <b>, <i>, and <u>

```
<xsd:complexType name='RichText' mixed='true'>
  <xsd:choice minOccurs='0' maxOccurs='unbounded'>
    <xsd:element name='b' type='RichText'/>
    <xsd:element name='i' type='RichText'/>
    <xsd:element name='u' type='RichText'/>
  </xsd:choice>
</xsd:complexType>
```

A Complex Type Example II

- Validation of RichText

```
<content xsi:type='RichText'></content>
<content xsi:type='RichText'>Andy</content>
<content xsi:type='RichText'>XML is <i>awesome</i>.</
  content>
<content xsi:type='RichText'><B>bold</B></content>
  INVALID
<content xsi:type='RichText'><foo/></content> INVALID
```

How to distinguish categories of resources

NAMESPACES

The Problem

- Documents use different vocabularies
  - Example 1: CD music collection
  - Example 2: online order transaction
- Merging multiple documents together
  - Name collisions can occur
    - Example 1: albums have a **<name>**
    - Example 2: customers have a **<name>**
  - How do you differentiate between the two?

The Solution: Namespaces!

- What is a namespace?
  - A syntactic way to differentiate similar names in an XML document
- Binding namespaces
  - Uses Uniform Resource Identifier (URI)
    - e.g. “http://example.com/NS”
  - Can bind to a named or “default” prefix

Namespace Binding Syntax

- Use “xmlns” attribute
  - Named prefix
    - **<a:foo xmlns:a=‘http://example.com/NS’/>**
  - Default prefix
    - **<foo xmlns=‘http://example.com/NS’/>**
- Element and attribute names are “qualified”
  - URI, local part (or “local name”) pair
    - e.g. { “http://example.com/NS” , “foo” }

Example Document I

- Namespace binding

```
<?xml version='1.0' encoding='UTF-8'?>
 <order>
    <item code='BK123'>
          <name>Care and Feeding of Wombats</name>
          <desc xmlns:html='http://www.w3.org/1999/xhtml'>
                The <html:b>best</html:b> book ever written!
          </desc>
    </item>
</order>
```

51

Example Document II

- Namespace scope

```
<?xml version='1.0' encoding='UTF-8'?>
 <order>
    <item code='BK123'>
          <name>Care and Feeding of Wombats</name>
          <desc xmlns:html='http://www.w3.org/1999/xhtml'>
                The <html:b>best</html:b> book ever written!
          </desc>
    </item>
</order>
```

52

Example Document III

- Bound elements

```
<?xml version='1.0' encoding='UTF-8'?>
 <order>
    <item code='BK123'>
          <name>Care and Feeding of Wombats</name>
          <desc xmlns:html='http://www.w3.org/1999/xhtml'>
                The <html:b>best</html:b> book ever written!
          </desc>
    </item>
</order>
```