Introduction to the Semantic Web and RDF

ZPUG DC
December 2, 2004

A.M. Kuchling
www.amk.ca
amk @ amk.ca

The Semantic Web has been a W3C project since around 1999.

The existing Web of HTML documents is good for humans:

- Needs only a single program to access it: the browser.
- Programs have a harder time with it; it's too messy.
- Screen-scrapers can extract information from HTML, but they're hard to write.

The Semantic Web will augment the existing human-readable Web with structured data that's easy for software to process.

An Amazon page

SEARCH | BROWSE SUBJECTS | BESTSELLERS | MAGAZINES | CORPORATE ACCOUNTS | E-BOOKS & DOCS | BARGAIN BOOKS | USED BOOKS

The Best American Crime Writing: 2004 Edition : The Year's Best True Crime Reporting
by OTTO PENZLER, THOMAS H. COOK

List Price: ~~$14.00~~
Price: **$11.20**
You Save: $2.80 (20%)
Availability: Usually ships within 24 hours from Amazon.com
Edition: Paperback

Editorial Reviews

From Publishers Weekly
Penzler and Cook's annual compendium of crime journalism showcases 20 essays ...

From Booklist
The third installment in this excellent annual series of nonfiction crime writing ... *Alan Moores*

Product Details:

- **Paperback:** 544 pages
- **Publisher:** Vintage (August 10, 2004)
- **ISBN:** 0375713026
- **Product Dimensions:** 8.0 x 5.2 x 0.9 inches
- **Shipping Weight:** 13.4 ounces. (View shipping rates and policies)
- **Average Customer Review:** based on 1 review. (Write a review)
- **Amazon.com Sales Rank in Books:** #63,332
- **In-Print Editions:** Hardcover | All Editions

Layers of the Semantic Web

The Semantic Web is split into three layers:

Web Ontology Language (OWL) Relationships between vocabularies	• "Persons" in vocabulary A are the same thing as "Users" in vocabulary B. • Resource X and resource Y are referring to the same thing.
RDF Schema: Vocabulary definitions	• There is a class called "Person". • Resource X is an instance of "Person".
Resource Description Framework (RDF) Assertions of facts	Resource X is named "Drew".

Overview of RDF

RDF is a specification that defines a model for representing the world, and a syntax for serializing and exchanging the model.

Facts are 3-tuples of (subject, property, object).

Subject* has a *property* of *object
Resource X has a *name* of *"Drew"*
ISBN 1234567890 has an *author* of *resource X*
Resource X has a *type* of *Person*

Noteworthy RDF vocabularies

Dublin Core

- Namespace: `http://purl.org/dc/elements/1.1/`
- Properties: title, creator, publisher, subject, identifier

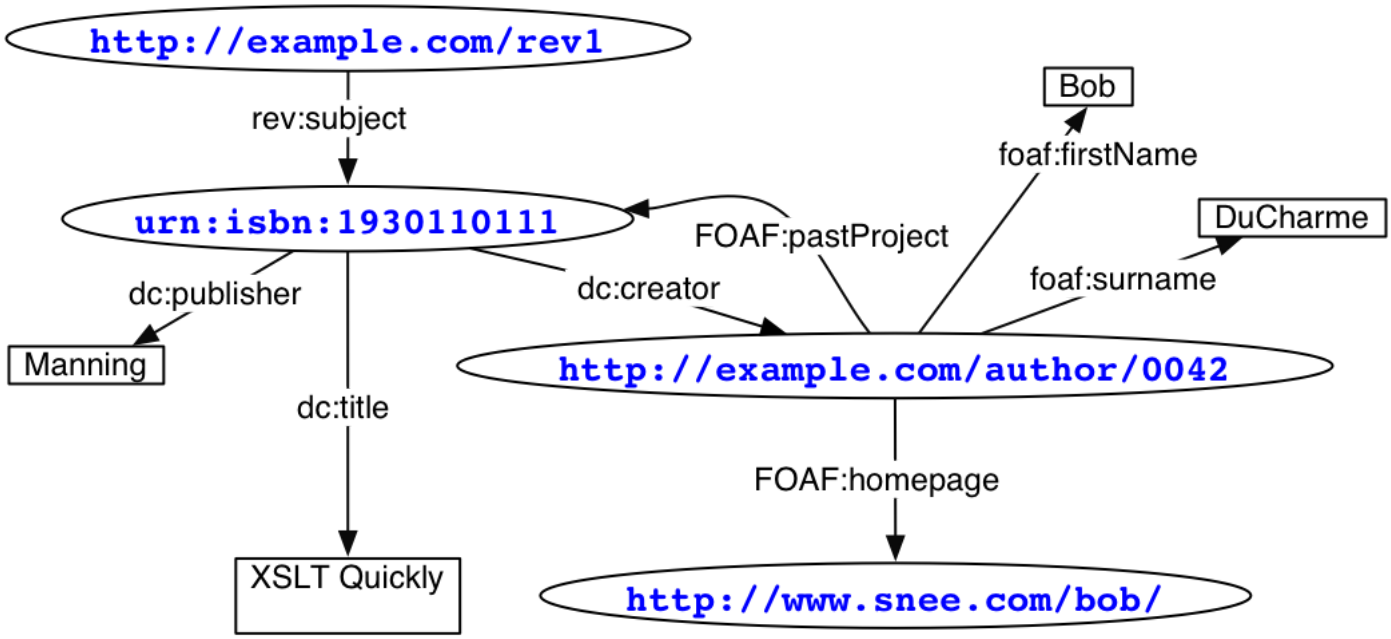
FOAF (Friend-of-a-friend)

- Describes people
- Classes: Person
- Properties: name, interest, mbox, schoolHomepage, workplaceHomepage
- Namespace: `http://xmlns.com/foaf/0.1/`

DOAP (Description of a Project)

- Describes open source projects
- Classes: Project, Repository
- Properties: name, homepage, mailing-list, license, maintainer
- Namespace: `http://usefulinc.com/ns/doap#`

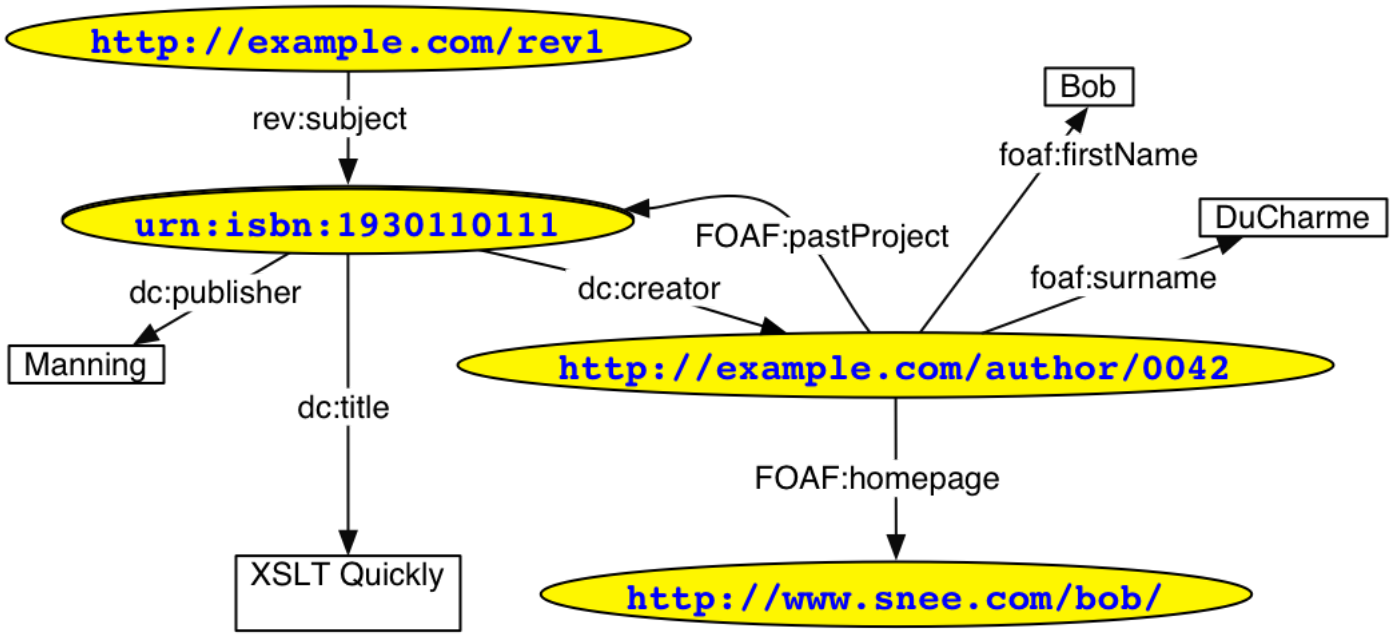
An Example RDF Graph



RDF Graph: Resources

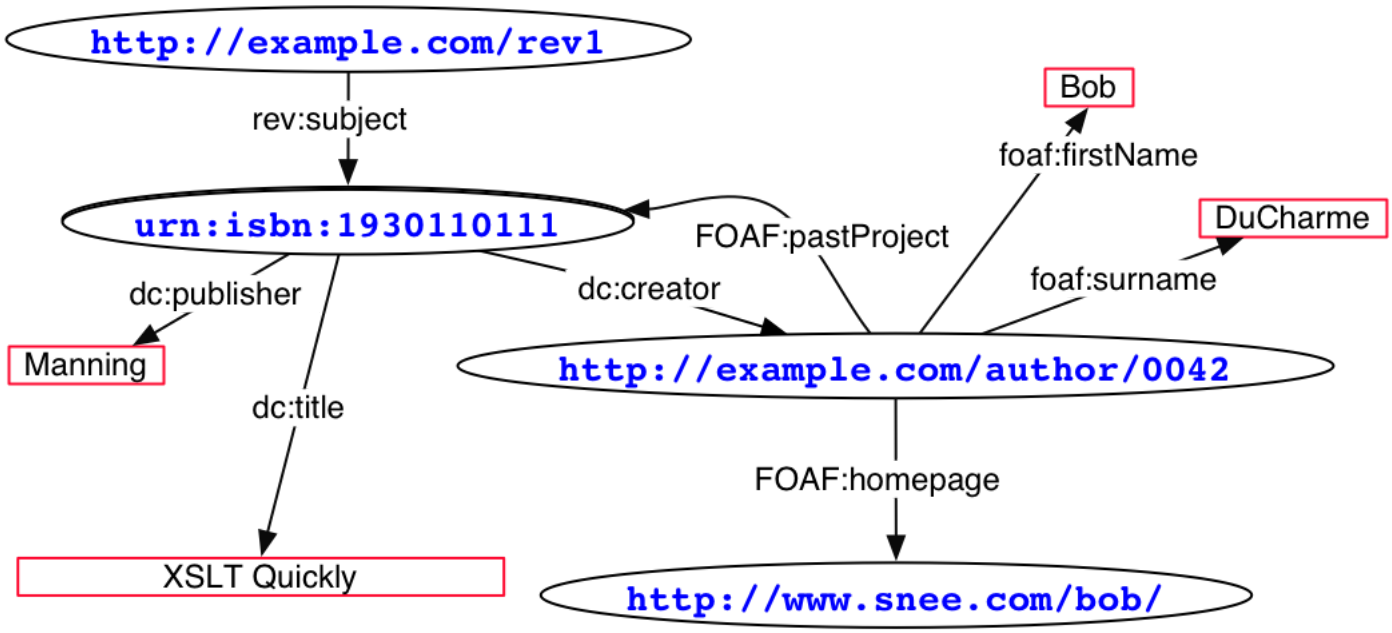
Resources are identified by URIs

- e.g. `http://example.com/person/0042`, `urn:isbn:1930110111`



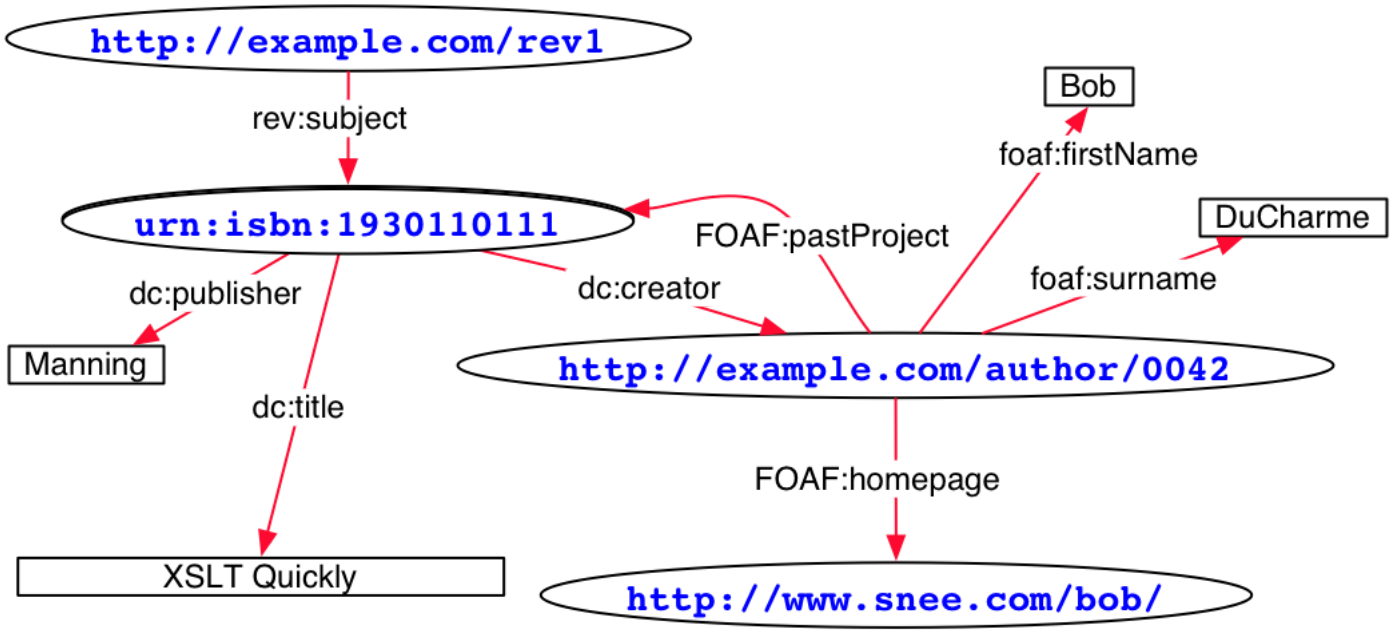
RDF Graph: Literals

- A primitive string value.
- The interpretation of the string is up to your application.



RDF Graph: Properties

- An attribute or aspect of a resource.
- A property value can be a literal or another resource.
- Multiple values are allowed; no value at all is also legal.



RDF Graph: Property URIs

How are properties identified? They could be just names or serial numbers, but that wouldn't be very scalable.

Instead, properties have URIs just like resources.

- Pick a base URI for your RDF vocabulary:
  `http://amk.ca/xml/review/1.0#`
- The base URI is assigned to a prefix, such as "rev".
- Properties are then referenced as 'rev:subject', 'rev:topic', etc.
- The RDF parser will concatenate the base URI (from the prefix) and the name:
  - rev:subject → `http://amk.ca/xml/review/1.0#subject`

RDF statements and triples

Graphs are usually represented as a bunch of (subject,property,object) 3-tuples.

Subject	Property	Object
`http://example.com/rev1`	`rev:subject` → `http://amk.ca/xml/review/1.0#subject`	`urn:isbn:1930110111`
`urn:isbn:1930110111`	`dc:title` → `http://purl.org/dc/elements/1.1/title`	"XSLT Quickly"
`urn:isbn:1930110111`	`dc:creator` → `http://purl.org/dc/elements/1.1/creator`	`http://example.com/author/0042`
`http://example.com/author/0042`	`FOAF:surname` → `http://xmlns.com/foaf/0.1/surname`	DuCharme
`http://example.com/author/0042`	`FOAF:homepage` → `http://xmlns.com/foaf/0.1/homepage`	`http://www.snee.com/bob/`
`http://example.com/author/0042`	`FOAF:pastProject` → `http://xmlns.com/foaf/0.1/pastProject`	`urn:isbn:1930110111`

RDF syntaxes: RDF/XML

RDF Core defines an XML-based serialization for RDF.

```
<rdf:RDF
    xmlns:FOAF="http://xmlns.com/foaf/0.1/"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rev="http://amk.ca/xml/review/1.0#">

    <!-- Implies rdf:type property is rev:Review -->
    <rev:Review rdf:about="http://example.com/rev1">
        <rev:subject rdf:resource="urn:isbn:1930110111"/>
    </rev:Review>

    <rdf:Description rdf:about="http://example.com/author/0042">
        <FOAF:firstName>Bob</FOAF:firstName>
        <FOAF:homepage rdf:resource="http://www.snee.com/bob/"/>
        <FOAF:pastProject rdf:resource="urn:isbn:1930110111"/>
        <FOAF:surname>DuCharme</FOAF:surname>
    </rdf:Description>
</rdf:RDF>
```

RDF syntaxes: Notation-3 (or N3)

An informal syntax that's easier to read and easier to scribble.

```
@prefix rev: <http://amk.ca/xml/review/1.0#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix FOAF: <http://xmlns.com/foaf/0.1/> .

<http://example.com/author/0042>
    FOAF:firstName "Bob";
    FOAF:surname "DuCharme";
    FOAF:homepage <http://www.snee.com/bob/>;
    FOAF:pastProject <urn:isbn:1930110111> .

<http://example.com/rev1> rev:subject [
   = <urn:isbn:1930110111>;
   dc:title "XSLT Quickly";
   dc:creator <http://example.com/author/0042>;
   dc:publisher "Manning" ] .
```

RDF's Sins and Virtues

Virtues:

- RDF exists and is strictly specified.
- Conceptually the graph model is pretty simple.
- RDF has a number of implementations.
- RDF is decentralized:
    - Anyone can create a vocabulary.
    - Anyone can publish data about other resources.

Sins:

- RDF/XML is rather verbose, and tedious to read/write by hand.
- Programming interfaces require you to know about triples, URIs, and all these low-level details.

Available RDF Software

The most basic form of RDF software is simply an RDF parser. Parsers are available for most of the languages you might need:

- Python (rdflib, 4RDF, pyrple, cwm)
- Perl (RDF::Core)
- C (Redland, libwww)
    - Thanks to SWIG, Redland also has interfaces for most of the other scripting languages.
- Java (Jena, Sesame)
- PHP, Ruby, Prolog all have RDF parsers.

Example code: Initializing an RDF database

Here's a Python example using rdflib 2.0.4 (www.rdflib.net).

```
#
# Initial setup -- create a TripleStore to hold RDF data
#

from rdflib.TripleStore import TripleStore
store = TripleStore()
```

You can add the contents of several URLs, parsing the data as RDF/XML:

```
store.load('http://www.amk.ca/amk.rdf')
store.load('http://www.python.org/pypi/?project=Twisted?format=doap')
store.load(...)
```

You can output the contents of a store:

```
print store.serialize(format='xml')
```

Example code: Modifying the database

You can add triples to a store:

```
from rdflib.URIRef import URIRef
from rdflib.Literal import Literal
from rdflib.Namespace import Namespace

REVIEW_NS = Namespace('http://amk.ca/xml/review/1.0#')
REVIEW_SUBJECT = REVIEW_NS['subject']
# Equivalent to:
##REVIEW_SUBJECT = URIRef('http://amk.ca/xml/review/1.0#subject')

book_uri = URIRef('urn:isbn:0609602330')

t = (URIRef('http://www.amk.ca/books/h/Isaacs_Storm.html'),
     REVIEW_SUBJECT, book_uri)
store.add(t)
```

You can also remove a triple:

```
store.remove(t)
```

Example code: Querying the database

The most general query method is triples(), which takes a (subject, property, object) 3-tuple, returning an iterator over the matching triples.

- Each element of the tuple can contain a URIRef or Literal instance, or None to match anything.

For example, to list all things which have a dc:title property:

```
>>> DC_TITLE = DC_NS['title']
>>> for s,p,o in store.triples((None, DC_TITLE, None)):
...     print s,p,o
...
urn:isbn:0609602330 http://purl.org/dc/elements/1.1/title \
   Isaac's Storm
urn:isbn:1930110111 http://purl.org/dc/elements/1.1/title \
   XSLT Quickly
>>>
```

Someday, there will be a query language (SPARQL example):

```
SELECT ?title
WHERE (<urn:isbn:1930110111> dc:title ?title)
```

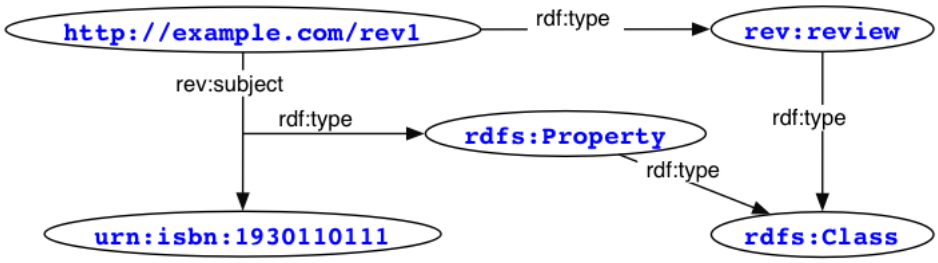
RDF Schema

Lets us define vocabularies (sets of classes and/or properties).

Example vocabulary:

- Some resources (web pages, articles, books) are reviews of other resources (books, music, articles, web pages).
  - Implies a `Review` class.
- A review has one or more subjects that are the items being reviewed.
  - Implies a `subject` property.

RDF Schema: Using rdf:type



RDF Schema: Describing a resource

First, define a prefix for the schema's namespace URI:

```
@prefix rev: http://amk.ca/xml/review/1.0#
```

To declare that a particular resource is a rev:Review, assert that the resource's rdf:type property is the class:

```
# Declare a resource
<http://example.com/review1> rdf:type rev:Review .
```

Describe what this resource is reviewing; what's the subject?

```
# Supply subject
<http://example.com/review1> rev:subject
      <http://www.music.com/album/6542>.
<http://example.com/review1> rev:subject <urn:isbn:1930110111>.
```

RDF Schema: Classes

So how do we define the class?

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rev: <http://amk.ca/xml/review/1.0#> .

# Declare a Review class

rev:Review  # class URI: http://amk.ca/xml/review/1.0#Review
   rdf:type rdfs:Class ;
   rdf:ID "Review" ;
   rdfs:comment """Reviews are resources that express an opinion
about some other resource.""" ;
.

# Declare a subclass of Review.
rev:ComparativeReview
   rdf:type rdfs:Class ;
   rdfs:subClassOf rev:Review .
   rdfs:comment """Comparative reviews examine multiple resources,
comparing their relative merits and usually offering an opinion
about which one is the best.""" ;
```

rdfs:Property

You can also specify properties in a vocabulary. The following fragment defines the rev:subject property:

```
rev:subject
    rdf:type rdf:Property;
    rdfs:label "Subject property" ;
    # Resources which can have this property
    rdfs:domain rev:Review ;
    # Values this property can take
    rdfs:range rdfs:Resource ;
    rdfs:comment "Value is the resource being reviewed." ;
    .

rev:title
    rdf:type rdf:Property;
    # This property only takes literal values
    rdfs:range rdfs:Literal;
    .
```

OWL: Connecting vocabularies

With RDF Schema, we know:

- Which classes exist,
- What their properties are.

We don't know:

- When are two classes the same?
- Can properties have multiple values?
  - e.g. Can a book have more than one author?
  - ... no authors?
  - ... more than one publisher?
- If X *prop* Y and Y *prop* Z are both true, which of the following are true?
  - X *property* Z (transitivity)
  - Y *property* X (symmetry)
  - Z *property* X

OWL: Web Ontology Language

OWL is a W3C language for defining this sort of relationship. Possible relationships:

- For classes:
  - equivalentClass
  - disjointWith
- For properties:
  - equivalentProperty
  - inverseOf
  - TransitiveProperty
  - SymmetricProperty
  - minCardinality
  - maxCardinality
- For resources:
  - sameAs
  - differentFrom

OWL: Defining a class

Here's an OWL declaration of a class representing persons:

```
@prefix gen: <http://genealogy.example.com/schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

gen:Person
    rdf:type owl:Class;
    rdf:ID "person" ;
    rdfs:comment "Resource representing a person." ;
    owl:equivalentClass foaf:Person;
    .
```

OWL: Property declaration

Define a property:

```
gen:ancestor
    # Declare as a transitive property:
    # X -> Y, Y -> Z implies X -> Z
    rdf:type owl:TransitiveProperty;
    rdfs:domain gen:Person;
    rdfs:range gen:Person;

    # Declare as inverse of some other property
    owl:inverseOf gen:descendant;
    .
```

Web ontology: What's the point?

OWL adds the ability to indicate when two classes or properties are identical.

- This lets bodies of data in different vocabularies be linked together.
- Anyone can do this by publishing the right snippet of RDF.

OWL declarations provide additional information to let rule-checking and theorem-proving systems work with RDF data.

- e.g. the 'ancestor' property is transitive.
- If (X ancestor Y) and (Y ancestor Z) are true, a system could infer that (X ancestor Z) is also true.

What should you care about?

So how much of this stuff do you need to learn about and use?

- RDF
  - Yes, definitely!
  - It's the base of the pyramid, and is widely implemented.
- RDF Schema
  - Useful for documentation.
  - Will likely be important in future.
  - But... you *can* get useful work done without it.
- OWL
  - Few libraries support OWL
  - Implementations are largely research projects.
  - For many applications theorem-proving will be irrelevant.

Why hasn't RDF caught on yet?

- HTML was much easier to learn; RDF is more abstract.
- There's no obvious killer application.
- RDF software APIs require you to know too much.
- Introductory tutorials are few.

Starting Small

But we don't need to aim for the stars. Simple things can be done without much effort, and can still be useful:

- Do you have some data that can be published as RDF?
- Can you invent a vocabulary for a new application domain?
- Write Semantic Web-enabled applications that store data as RDF, or provide export/import to RDF.
- Can you link two different data sets to do something interesting?

There are signs of life: FOAF has caught on, DOAP is rising, and many small projects are using RDF internally.

- So the Semantic Web *is* coming... slowly.

Demos

- Welkin -- a generic RDF browser.
- LiveJournal has FOAF files; try `http://<whatever>.livejournal.com/data/foaf`.
  - example link
- lorebot -- an IRC bot
- My family pictures -- password required
  - Schema in N3

Questions, comments?

These slides: www.amk.ca/talks/2004-12-02

For further information:

- The W3C's RDF Primer is good: www.w3.org/TR/rdf-primer/.
- The Jena project's tutorial: jena.sourceforge.net/tutorial/RDF_API.
- A lengthy presentation: www.w3.org/Consortium/Offices/Presentations/RDFTutorial/
- The O'Reilly book, *Practical RDF*, is decent but not great.

What Python library to use?

- rdflib (www.rdflib.net) is the most commonly used; complicated, though.
- Redland (www.librdf.org) is a widely-used C library with Python bindings.
- A number of small libraries exist (rdfxml.py, pyrple); Google for "python rdf".
- Java libraries (Jena, Kowari) can be used from Jython.

PyCon reminder

PyCon will be March 23-25 at GWU's Cafritz Center.

Deadline for proposals: Dec. 31st.

Call for papers:
`http://www.python.org/pycon/2005/cfp.html`

Proposal submissions: `http://submit.pycon.org`