Describing Web Resources in RDF

CSE 595 – Semantic Web

Instructor: Dr. Paul Fodor

Stony Brook University

http://www3.cs.stonybrook.edu/~pfodor/courses/cse595.html

Lecture Outline

- Current Web
- RDF: Data Model
- RDF Syntaxes (Turtle, RDF/XML: XML-based Syntax of RDF, RDFa)
- RDFS: Adding Semantics
- The Language of RDF Schema
- RDF and RDF Schema in RDF Schema
- Axiomatic Semantics for RDF and RDFS
- Direct Semantics based on Inference Rules

Current Web

- The success of the WWW has shown the power of having standard mechanisms to exchange and communicate information.
- HTML is the standard language in which web pages are written.
  - It allows anyone to publish a document and have confidence that this document will be rendered correctly by any web browser.

Current Web

- There are three components that HTML and any exchange language has:
  - a syntax (tells us how to write data down),
  - a data model (tells us the structure or organization of the data), and
  - a semantics (tells us how to interpret that data).
- The syntax, data model, and semantics are all defined within the HTML standard.

Current Web

- HTML example:

```
<html>
  <head>
    <title>Apartments for Rent</title>
  </head>
  <body>
    <ol>
      <li> Studio apartment on Florida Ave.
      <li> 3 bedroom Apartment on Baron Way
    </ol>
  </body>
</html>
```

Current Web

- The syntax of HTML is text with tags (e.g. **`<title>`**) written using angle brackets.
- The data model of HTML, known as the Document Object Model (DOM), defines the organization of these elements defined by tags into a hierarchical tree structure.
  - For example, **`<head>`** should come before **`<body>`** and **`<li>`** elements should appear within **`<ol>`** elements

Current Web

- The semantics of HTML tell us how the browser should interpret the web page
  - The browser should render the content of the web page's body within the browser window and elements should be displayed as an ordered list.
- Drawback: HTML is designed to communicate information about the structure of documents for human consumption.
  - For the Semantic Web, we need something richer
    - We need a data model that can be used by multiple applications, not just for describing documents for people but for describing application-specific information

Current Web

- The data model needs to be domain independent so that applications ranging from real estate to social networks can leverage it.
  - In addition to a flexible data model, we also need a mechanism to assign semantics to the information represented using this data model.
    - It should allow users to describe how an application should interpret “friend” in a social network description and “city” in a geographical description.
- XML is a good candidate, but it is just syntax.

Drawbacks of XML

- XML is a universal metalanguage for defining markup
- It provides a uniform framework for interchange of data and metadata between applications
- However, XML does not provide any means of talking about the semantics (meaning) of data
- E.g., there is no intended meaning associated with the nesting of tags
  - It is up to each application to interpret the nesting

Nesting of Tags in XML

- David Billington is a lecturer of Discrete Maths

```
<course name="Discrete Maths">
  <lecturer>David Billington</lecturer>
</course>

<lecturer name="David Billington">
  <teaches>Discrete Maths</teaches>
</lecturer>
```

- Opposite nesting, but same information!

Basic Ideas of RDF

- RDF (Resource Description Framework) provides a flexible domain independent data model.
- Basic building block: entity-attribute-value triple
  - It is called a ***statement***
  - Sentence about **`Billington`** is such a statement
- RDF has been given a syntax in XML
  - This syntax inherits the benefits of XML
  - Other syntactic representations of RDF possible

Basic Ideas of RDF

- Because RDF is not particular to any domain or use, it is necessary for users to define the terminology they use within these statements.
  - RDF Schema (RDFS) allows users to precisely define how their *vocabulary* (i.e. their terminology) should be interpreted.
- Combined, these technologies define the components of a standard language for exchanging arbitrary data between machines:
  - RDF – data model
  - RDFS – semantics
  - Turtle / RDF-XML – syntax / RDFa / JSON-LD

RDF: Data Model

- The fundamental concepts of RDF are:
  - resources
  - properties
  - statements

Resources

- We can think of a resource as an object, a "thing" we want to talk about
  - E.g. authors, books, publishers, places, people, hotels
- Every resource has a URI, a Universal Resource Identifier
- A URI can be
  - a URL (Web address) or
  - some other kind of unique identifier

Resources

- URI schemes have been defined not only for web locations but also for telephone numbers, ISBN numbers, and geographic locations.
- URIs provide a mechanism to unambiguously identify the “thing” we want to talk about.
  - The *homonym problem* is about how to identify unambiguously a "thing"
    - For example, if referring to a swimming pool, we can use a URI assigned to swimming pools and not have it be confused with billiards (pool) or a group of people.

Resources

- Advantages of using URIs:
  - A global, worldwide, unique naming scheme
  - Reduces the homonym problem of distributed data representation

Properties

- Properties are a special kind of resources
- They describe <u>relations between resources</u>
  - E.g. “written by”, “age”, “title”, etc.
- Properties are also identified by URIs
  - We can also dereference property URLs to find their descriptions.

Statements

- Statements assert the properties of resources
- A statement is an **entity-attribute-value** triple
  - It consists of a **resource**, a **property**, and a **value**
- Values can be resources or literals
  - Literals are atomic values (for example, numbers, strings, dates)
- We often use the word ***subject*** to refer to the entity in a statement and ***object*** to refer to its value.

Three Views of a Statement

- A triple
- A piece of a graph
- A piece of XML code (RDF/XML) or some other formal syntax (Turtle, JSON-LD, RDFa)

- Thus an RDF document can be viewed as:
  - A set of triples
  - A graph (semantic net)
  - An XML document

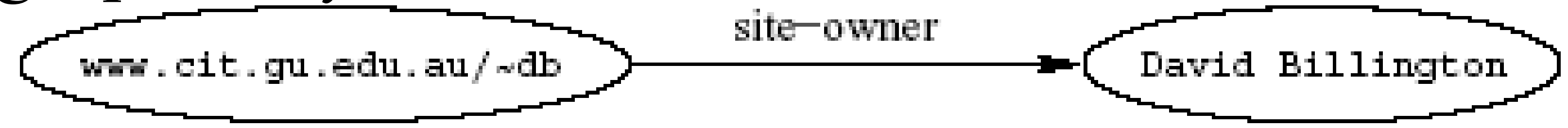
Statements as Triples

`(http://www.cit.gu.edu.au/~db,`
`http://www.mydomain.org/site-owner,`
`#David Billington)`

- The triple **`(x,P,y)`** can be considered as a logical formula **`P(x,y)`**
  - Binary predicate **`P`** relates object **`x`** to object **`y`**
  - RDF offers only binary predicates (properties)
- Notice how we used URLs to identify the things we are referring to in our statement.

Graphs

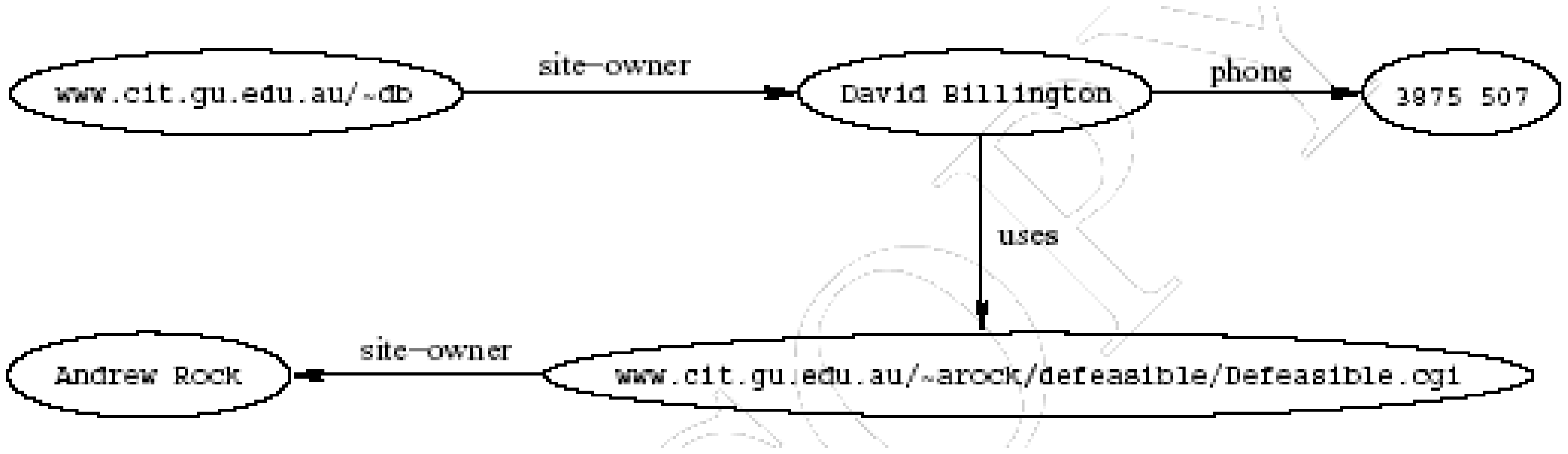
- We can also write this same statement down graphically



  - A directed graph with labeled nodes and arcs
    - from the resource (the subject of the statement)
    - to the value (the object of the statement)
- Known in AI as a *semantic net*
- The value of a statement may be a resource
  - It may be linked to other resources

A Set of Triples as a Semantic Net

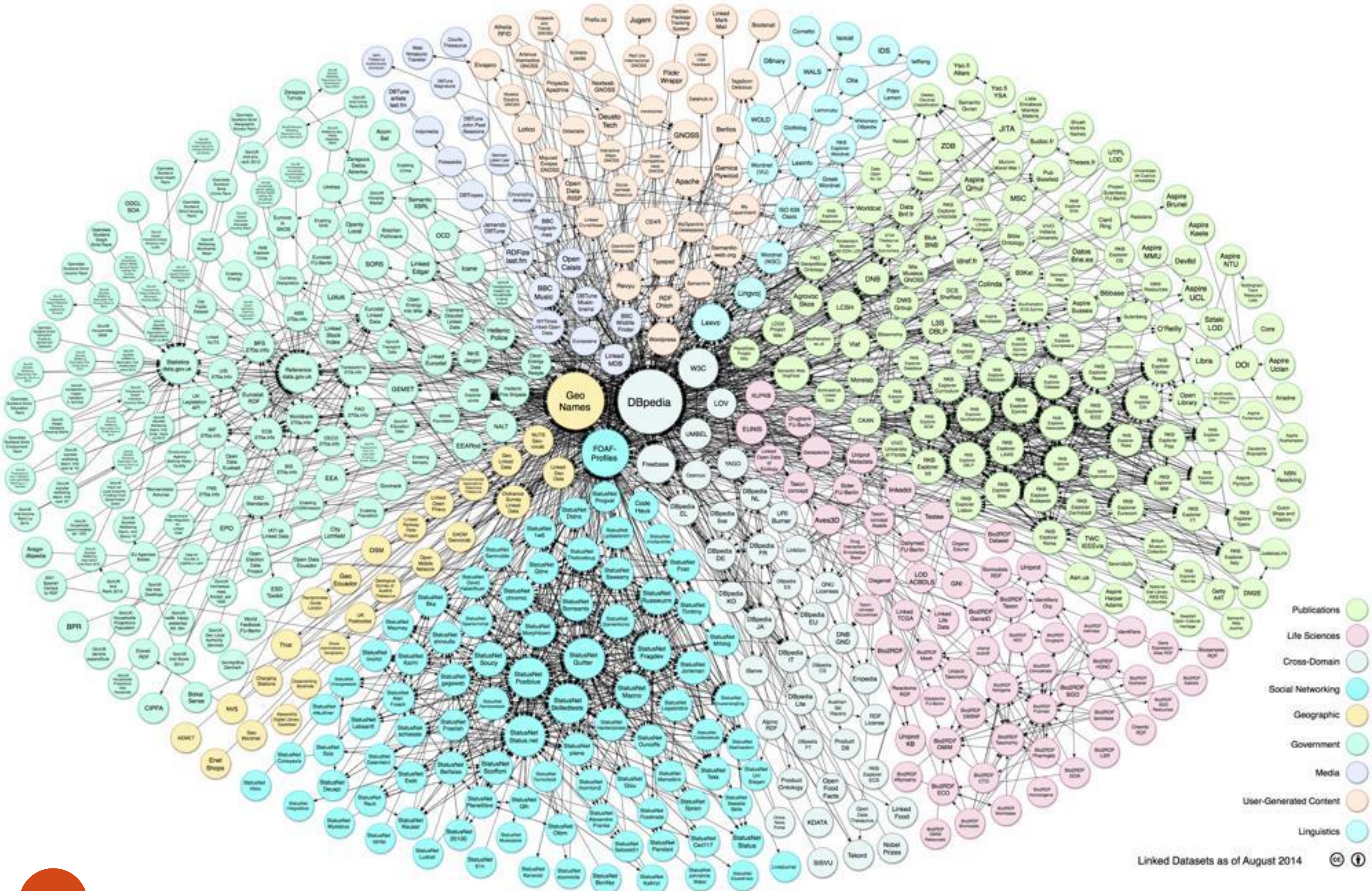
- The object of a statement can be the subject of another statement



- this graph can be created in a *distributed* fashion by multiple different participants just by using the same URLs.
- this allows us to create a *Web of Data*

A Set of Triples as a Semantic Net

- Global ontologies allow for knowledge to be reused
  - for example, if we find RDF on the web describing a person, we can reuse that information just by using that URL.
- There is a set of best practices, called the ***Linked Data principles***, that encourage us to reuse and make available information to help create this global graph.


Geo Names
DBpedia
FOAF-Profiles
W3C
Freebase
Lexvo
LOV
UMBEL
YAGO
Reference data.gov.uk
Statistics data.gov.uk
StatusNet Status.net
GNOSS
Publications
Life Sciences
Cross-Domain
Social Networking
Geographic
Government
Media
User-Generated Content
Linguistics
Linked Datasets as of August 2014

Linked Data principles

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF).
4. Include links to other URIs so that they can discover more things.

- Example triple from DBPedia:

```
<http://www.semanticwebprimer.org/ontology/apartments.ttl#
             BaronWayBuilding>
      <http://dbpedia.org/ontology/location>
      <http://dbpedia.org/resource/Amsterdam>.
```

You can follow these URLs to find out more information about the referred to concepts.

Reification

- In RDF it is possible to make statements about statements
  - ***Grigoris believes that David Billington is the creator of*** http://www.cit.gu.edu.au/~db
- Such statements can be used to describe belief or trust in other statements
- The solution is to assign a **unique identifier** to each statement
  - It can be used to refer to the statement

Reification

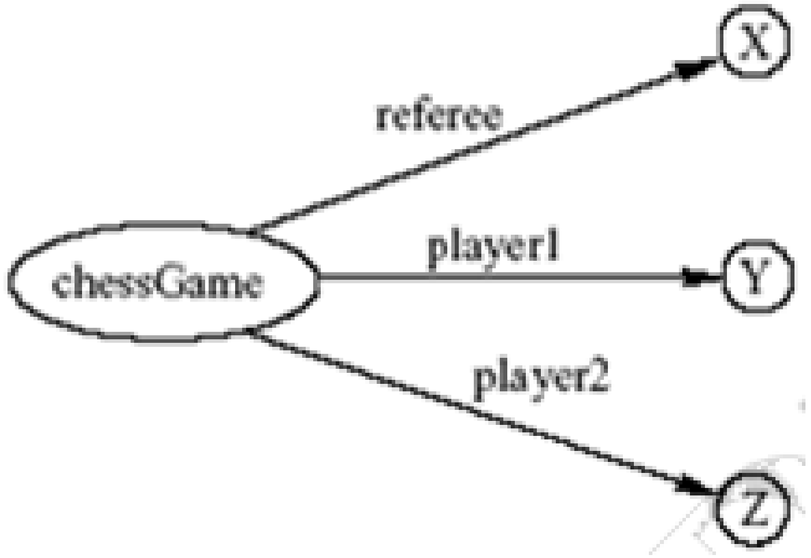
- There are only triples in RDF; therefore we cannot add an identifier directly to a triple (then it would be a quadruple)

A Critical View of RDF: Binary Predicates

- RDF uses only binary properties
  - This is a restriction because often we use predicates with more than 2 arguments
  - But binary predicates can simulate these
- Example: **`referee(X,Y,Z)`**
  - **`X`** is the **`referee`** in a chess game between players **`Y`** and **`Z`**

A Critical View of RDF: Binary Predicates

- We introduce:
  - a new auxiliary resource **`chessGame1`**
  - the binary predicates **`referee`**, **`player1`**, and **`player2`**
- We can represent **`referee(X,Y,Z)`** as:



A Critical View of RDF: Properties

- Properties are special kinds of resources
  - Properties can be used as the object in an **entity-attribute-value** triple (statement)
  - They are defined independent of resources
- This possibility offers flexibility
- But it is unusual for modelling languages and OO programming languages
- It can be confusing for modellers

A Critical View of RDF: Reification

- The reification mechanism is quite powerful
- It appears misplaced in a simple language like RDF
- Making statements about statements introduces a level of complexity that is not necessary for a basic layer of the Semantic Web
- Instead, it would have appeared more natural to include it in more powerful layers, which provide richer representational capabilities

A Critical View of RDF

- RDF has its idiosyncrasies and is not an optimal modeling language but
  - It is already a de facto standard
  - It has sufficient expressive power
    - At least as for more layers to build on top
- Using RDF offers the benefit that information maps unambiguously to a model

Reification

- Introduce an auxiliary object (e.g. **`belief1`**)
  - relate it to each of the 3 parts of the original statement through the properties subject, predicate and object
- In the preceding example
  - subject of **`belief1`** is **`David Billington`**
  - predicate of **`belief1`** is **`creator`**
  - object of **`belief1`** is http://www.cit.gu.edu.au/~db

Reification

- Because of the overhead of reification, in newer versions of the RDF standard, the notion of ***named graphs*** was introduced: an explicit identifier (again a URL) is given to a statement or set of statements.
  - This identifier can then be referred to in normal triples.
  - This is a more straightforward mechanism for identifying statements as well as graphs

RDF Syntaxes

- We have already seen one syntax for RDF, namely, a graphical syntax.
  - Graphs are a powerful tool for human understanding BUT
  - The Semantic Web vision requires machine-accessible and machine-processable representations
  - And the graph syntax is neither machine interpretable nor standardized.
- We will introduce standard machine interpretable syntaxes for RDF: Turtle, RDF/XML and JSON-LD

Turtle

- Terse RDF Triple Language (Turtle) is a text-based syntax for RDF.
- The file extension used for Turtle text files is ".ttl".
- Example:

```
<http://www.semanticwebprimer.org/ontology/apartments.ttl#BaronWayBuilding>
  <http://dbpedia.org/ontology/location>
  <http://dbpedia.org/resource/Amsterdam>.
```

  - URLs are enclosed in angle brackets.
  - The subject, property, and object of a statement appear in order, followed by a period.

Turtle

- We can write a whole RDF graph just using this approach:

```
<http://www.semanticwebprimer.org/ontology/apartments.ttl#>
  <http://www.semanticwebprimer.org/ontology/apartments.ttl#
      isPartOf>
  <http://www.semanticwebprimer.org/ontology/apartments.ttl#
      BaronWayBuilding>.

<http://www.semanticwebprimer.org/ontology/ apartments.ttl#
      BaronWayBuilding>
  <http://dbpedia.org/ontology/location>
  <http://dbpedia.org/resource/Amsterdam>.
```

Literals

- In Turtle, we write literals by simply enclosing the value in quotes and appending it with the data type of the value
  - Data type tells us whether we should interpret a value as string, a date, integer or some other type
    - Data types are again expressed as URLs
    - It is recommend practice to use the data types defined by XML Schema.
    - If no data type is specified after a literal, it is assumed to be a string.

Data Types

- In practice, the most widely used data typing scheme will be the one by XML Schema
  - But the use of any externally defined data typing scheme is allowed in RDF documents
- XML Schema predefines a large range of data types
  - E.g. Booleans, integers, floating-point numbers, times, dates, etc.
- ^^-notation indicates the type of a literal

Literals

- Common data types and how they look in Turtle:

```
string – "Baron Way"
integers – "1"^^<http://www.w3.org/2001/XMLSchema#integer>
decimals – "1.23" <http://www.w3.org/2001/XMLSchema#decimal>
dates – "2020-08-30"^^<http://www.w3.org/2001/XMLSchema#date>
time – "11:24:00"^^<http://www.w3.org/2001/XMLSchema#time>
date with a time – "2020-08-30T11:24:00"^^
      <http://www.w3.org/2001/XMLSchema#dateTime>
```

- Turtle:

```
<http://www.semanticwebprimer.org/ontology/apartments.ttl#
      BaronWayApartment>
  <http://www.semanticwebprimer.org/ontology/apartments.ttl#
      hasNumberOfBedrooms>
  "3"^^<http://www.w3.org/2001/XMLSchema#integer>.
```

Abbreviations

- Multiple resources are defined at the URL: **`http://www.semanticwebprimer.org/ontology/apartments.ttl`**
- This URL defines what is termed the *namespace* of those resources
- Turtle introduces the **`@prefix`** syntax to define short stand-ins for particular namespaces

```
@prefix swp: <http://www.semanticwebprimer.org/
        ontology/apartments.ttl#>.
```

  - **`swp`** is termed a ***qualified name***

Abbreviations

- Turtle:

```
@prefix swp: <http://www.semanticwebprimer.org/ontology/
     apartments.ttl#>.
@prefix dbpedia: <http://dbpedia.org/resource/>.
@prefix dbpedia-owl: <http://dbpedia.org/ontology/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

swp:BaronWayApartment swp:hasNumberOfBedrooms "3"^^<xsd:integer>.
swp:BaronWayApartment swp:isPartOf swp:BaronWayBuilding.
swp:BaronWayBuilding dbpedia-owl:location dbpedia:Amsterdam.
```

  - angle brackets are dropped from around resources that are referred to using a qualified name
  - we can mix and match regular URLs with these qualified names.

Abbreviations

- Turtle also allows us to not repeat particular subjects when they are used repeatedly:
  - **`swp:BaronWayApartment`** is used as the subject of two triples
    - This can be written more compactly by using a semicolon at the end of a statement

```
swp:BaronWayApartment
  swp:hasNumberOfBedrooms "3"^^<xsd:integer>;
  swp:isPartOf swp:BaronWayBuilding.
```

Abbreviations

- Turtle also allows us to abbreviate common data types
  - For example, numbers can be written without quotes
  - If they contain a decimal (e.g. 14.3), they are interpreted as decimals
  - If they do not contain a decimal (e.g. 1), they are interpreted as integers:

```
swp:BaronWayApartment swp:hasNumberOfBedrooms 3.
```

Named Graphs

- Trig is an extension to Turtle that allows named graphs (quadruples instead of triples)
  - Put brackets around the set of statements we want and assigning that set of statements a URL

Named Graphs

- Example: Baron Way Apartment were created by a person, Frank, identified by the URL http://www.cs.vu.nl/~frankh

```
@prefix swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
@prefix dbpedia: <http://dbpedia.org/resource/>.
@prefix dbpedia-owl: <http://dbpedia.org/ontology/>.
@prefix dc: <http://purl.org/dc/terms/>.

{
  <http://www.semanticwebprimer.org/ontology/apartments.ttl#> dc:creator
  <http://www.cs.vu.nl/frankh>
}

<http://www.semanticwebprimer.org/ontology/apartments.ttl#>
{
  swp:BaronWayApartment swp:hasNumberOfBedrooms 3;
      swp:isPartOf swp:BaronWayBuilding.
  swp:BaronWayBuilding dbpedia-owl:location
            dbpedia:Amsterdam, dbpedia:Netherlands.
}
```

Statements in XML Syntax

- There is an RDF representation based on XML:
  - RDF/XML is an encoding of RDF in the XML language
  - RDF/XML allows RDF to be used with existing XML processing tools
  - But XML is not a part of the RDF data model
    - i.e. the serialization in XML is irrelevant for RDF

Statements in XML Syntax

- All RDF/XML should be enclosed in an element **`rdf:RDF`**.
- Subjects are denoted by the **`rdf:about`** within an **`rdf:Description`** element (enclosed in brackets).
- Predicates and objects related to that subject are enclosed in the **`rdf:Description`** element.
- Namespaces can be used through the XML namespaces (**`xmlns:`**) construct.

RDF Statements in XML

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:mydomain="http://www.mydomain.org/my-rdf-ns">

  <rdf:Description
      rdf:about="http://www.cit.gu.edu.au/~db">
    <mydomain:site-owner rdf:resource="#David Billington"/>
  </rdf:Description>
</rdf:RDF>
```

Statements in RDF/XML

- The **`rdf:Description`** element makes a statement about the resource http://www.cit.gu.edu.au/~db
- Within the description
  - the property is used as a tag (**`<mydomain:site-owner>`**)
  - the content is the value of the property

```xml
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:dbpedia-owl="http://dbpedia.org/ontology/"
    xmlns:dbpedia="http://dbpedia.org/resource/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:swp="http://www.semanticwebprimer.org/ontology/apartments.ttl#">
  <rdf:Description
rdf:about="http://www.semanticwebprimer.org/ontology/apartments.ttl#BaronWayApartment">
    <swp:hasNumberOfBedrooms rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">
      3
    </swp:hasNumberOfBedrooms>
  </rdf:Description>
  <rdf:Description
rdf:about="http://www.semanticwebprimer.org/ontology/apartments.ttl#BaronWayApartment">
    <swp:isPartOf
rdf:resource="http://www.semanticwebprimer.org/ontology/apartments.ttl#BaronWayBuilding"/>
  </rdf:Description>
  <rdf:Description
rdf:about="http://www.semanticwebprimer.org/ontology/apartments.ttl#BaronWayBuilding">
    <dbpedia-owl:location rdf:resource="http://dbpedia.org/resource/Amsterdam"/>
  </rdf:Description>
  <rdf:Description
rdf:about="http://www.semanticwebprimer.org/ontology/apartments.ttl#BaronWayBuilding">
    <dbpedia-owl:location rdf:resource="http://dbpedia.org/resource/Netherlands"/>
  </rdf:Description>
</rdf:RDF>
```

RDFa

- One use case of RDF is to describe or mark up the content of HTML web pages
  - The RDFa syntax was introduced to help with that use case.
  - RDFa embeds RDF within the attributes of HTML tags.
- Example of old HTML:

```
<html>
<body>
<H1> Baron Way Apartment for Sale</H1>
The Baron Way Apartment has three bedrooms and is
located in the family
friendly Baron Way Building. The Apartment is located in
the north of Amsterdam.
</body>
</html>
```

  - This page does not contain any machine readable description

RDFa

- We can mark up the page using RDFa as follows:

```
<html xmlns:dbpedia="http://dbpedia.org/resource/"
  xmlns:dbpediaowl="http://dbpedia.org/ontology/"
  xmlns:swp="http://www.semanticwebprimer.org/ontology/
    apartments.ttl#"
  xmlns:geo="http://www.geonames.org/ontology#">
<body>
<H1> Baron Way Apartment for Sale</H1>
<div about="[swp:BaronWayFlat]">
  The Baron Way Flat has
  <span property="swp:hasNumberOfBedrooms">3</span>
  bedrooms and is located in the family friendly
  <span rel="swp:isPartOf"
    resource="[swp:BaronWayBuilding]">
  Baron Way Building</span>
```

RDFa

```
  <div about="[swp:BaronWayBuilding]">
    The building is located in the north of Amsterdam.
    <span rel="dbpediaowl:location"
      resource="[dbpedia:Amsterdam]"></span>
    <span rel="dbpediaowl:location"
      resource="[dbpedia:Netherlands]"></span>
  </div>
</div>
</body>
</html>
```

- Since the RDF is encoded in tags such as spans, paragraphs, and links, the RDF will not be rendered by browsers when displaying the HTML page

RDFa

- Similar to RDF/XML, namespaces are encoded using the **`xmlns`** declaration.
- Subjects are identified by the **`about`** attribute
- Properties are identified by either a **`rel`** or **`property`** attribute.
  - **`rel`** attributes are used when the object of the statement is a resource
  - **`property`** attribute is used when the object of a statement is a literal.
- Properties are associated with subjects through the use of the hierarchal structure of HTML.

XML-based Syntax of RDF

- Advanced XML-based Syntax of RDF
  - similar to Turtle, RDF/XML allows shortcuts and other techniques to write RDF in more concise ways

Example of University Courses

```
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
      xmlns:uni="http://www.mydomain.org/uni-ns">

      <rdf:Description rdf:ID="949318">
            <uni:name>David Billington</uni:name>
            <uni:title>Associate Professor</uni:title>
            <uni:age rdf:datatype="&xsd:integer">27<uni:age>
      </rdf:Description>

      <rdf:Description rdf:about="CIT1111">
            <uni:courseName>Discrete Maths</uni:courseName>
            <uni:isTaughtBy>David Billington</uni:isTaughtBy>
      </rdf:Description>
```

Example of University Courses

```
        <rdf:Description rdf:about="CIT2112">
              <uni:courseName>Programming III</uni:courseName>
              <uni:isTaughtBy>Michael Maher</uni:isTaughtBy>
        </rdf:Description>

  </rdf:RDF>
```

`rdf:about` versus `rdf:ID`

- An element **`rdf:Description`** has
  - An **`rdf:ID`** attribute indicating that the resource is defined, or
  - An **`rdf:about`** attribute indicating that the resource has been "defined" elsewhere
- Formally, there is no such thing as "*defining*" an object in one place and referring to it elsewhere
  - Sometimes is useful (for human readability) to have a defining location, while other locations state "additional" properties

Property Elements

- Content of **`rdf:Description`** elements

```
<rdf:Description rdf:about="CIT3116">
    <uni:courseName>Knowledge Representation</uni:courseName>
    <uni:isTaughtBy>Grigoris Antoniou</uni:isTaughtBy>
</rdf:Description>
```

- **`uni:courseName`** and **`uni:isTaughtBy`** define two property-value pairs for CIT3116 (two RDF statements)
  - read conjunctively

Data Types

- The attribute **`rdf:datatype="&xsd:integer"`** is used to indicate the data type of the value of the age property

```
<rdf:Description rdf:about="949318">
      <uni:name>David Billington</uni:name>
      <uni:title>Associate Professor</uni:title>
      <uni:age rdf:datatype="&xsd:integer">27<uni:age>
</rdf:Description>
```

Data Types

- The age property has been defined to have "**`&xsd:integer`**" as its range
  - It is still required to indicate the type of the value of this property each time it is used
    - This is to ensure that an RDF processor can assign the correct type of the property value even if it has not "seen" the corresponding RDF Schema definition before
      - This scenario is quite likely to occur in the unrestricted WWW

The `rdf:resource` Attribute

- The relationships between courses and lecturers (in the example) were not formally defined but existed implicitly through the use of the same name
- The use of the same name may just be a coincidence for a machine
- We can denote that two entities are the same using the **`rdf:resource`** attribute

The `rdf:resource` Attribute

```
<rdf:Description rdf:ID="949318">
   <uni:name>David Billington</uni:name>
   <uni:title>Associate Professor</uni:title>
</rdf:Description>

<rdf:Description rdf:about="CIT1111">
   <uni:courseName>Discrete Mathematics</uni:courseName>
   <uni:isTaughtBy rdf:resource="949318"/>
</rdf:Description>
```

Referencing Externally Defined Resources

- E.g., to refer the externally defined resource **`CIT1111`**:

  `http://www.mydomain.org/uni-ns#CIT1111`

as the value of **`rdf:about`**

- **`www.mydomain.org/uni-ns`** is the URI where the definition of **`CIT1111`** is found
- A description with an ID defines a fragment URI, which can be used to reference the defined description

Nested Descriptions: Example

```
<rdf:Description rdf:about="CIT1111">
      <uni:courseName>Discrete Maths</uni:courseName>
      <uni:isTaughtBy>
            <rdf:Description rdf:ID="949318">
              <uni:name>David Billington</uni:name>
              <uni:title>Associate Professor</uni:title>
            </rdf:Description>
      </uni:isTaughtBy>
</rdf:Description>
```

Nested Descriptions

- Descriptions may be defined within other descriptions
- Other courses, such as **`CIT3112`**, can still refer to the new resource with ID **`949318`**
- Although a description may be defined within another description, its scope is global

Introducing some Structure to RDF Documents using the **rdf:type** Element

```
<rdf:Description rdf:ID="CIT1111">
        <rdf:type
          rdf:resource="http://www.mydomain.org/uni-ns#course"/>
        <uni:courseName>Discrete Maths</uni:courseName>
        <uni:isTaughtBy rdf:resource="#949318"/>
</rdf:Description>

<rdf:Description rdf:ID="949318">
        <rdf:type
         rdf:resource="http://www.mydomain.org/uni-ns#lecturer"/>
        <uni:name>David Billington</uni:name>
        <uni:title>Associate Professor</uni:title>
</rdf:Description>
```

Abbreviated Syntax

- Simplification rules:
  - Childless property elements within description elements may be replaced by XML attributes
  - For description elements with a typing element we can use the name specified in the **`rdf:type`** element instead of **`rdf:Description`**
- These rules create syntactic variations of the same RDF statement
  - They are equivalent according to the RDF data model, although they have different XML syntax

Abbreviated Syntax: Example

```
<rdf:Description rdf:ID="CIT1111">
  <rdf:type
    rdf:resource="http://www.mydomain.org/uni-ns#course"/>
  <uni:courseName>Discrete Maths</uni:courseName>
  <uni:isTaughtBy rdf:resource="#949318"/>
</rdf:Description>
```

Application of First Simplification Rule

```
<rdf:Description
    rdf:ID="CIT1111"
    uni:courseName="Discrete Maths">
  <rdf:type
    rdf:resource="http://www.mydomain.org/uni-ns#course"/>
  <uni:isTaughtBy rdf:resource="#949318"/>
</rdf:Description>
```

Application of 2nd Simplification Rule

```
<uni:course
    rdf:ID="CIT1111"
    uni:courseName="Discrete Maths">
  <uni:isTaughtBy rdf:resource="#949318"/>
</uni:course>
```

Container Elements

- Collect a number of resources or attributes about which we want to make statements as a whole
  - E.g., we may wish to talk about the courses given by a particular lecturer
- The content of container elements are named **`rdf:_1`**, **`rdf:_2`**, etc.
  - Alternatively **`rdf:li`**

Types of Container Elements

- `rdf:Bag` is an unordered container, allowing multiple occurrences
  - E.g. members of the faculty board, documents in a folder
- `rdf:Seq` is an ordered container, which may contain multiple occurrences
  - E.g. modules of a course, items on an agenda, an alphabetized list of staff members (order is imposed)
- `rdf:Alt` is a set of alternatives
  - E.g. the document home and mirrors, translations of a document in various languages
- `rdfs:Container` is a superclass of all container classes, including the three preceding ones.

Example for a Bag

```
<uni:lecturer
    rdf:ID="949352"
    uni:name="Grigoris Antoniou"
    uni:title="Professor">
  <uni:coursesTaught>
     <rdf:Bag>
          <rdf:_1 rdf:resource="#CIT1112"/>
          <rdf:_2 rdf:resource="#CIT3116"/>
     </rdf:Bag>
  </uni:coursesTaught>
</uni:lecturer>
```

Example for Alternative

```
<uni:course
    rdf:ID="CIT1111"
    uni:courseName="Discrete Mathematics">
  <uni:lecturer>
     <rdf:Alt>
          <rdf:li rdf:resource="#949352"/>
          <rdf:li rdf:resource="#949318"/>
     </rdf:Alt>
  </uni:lecturer>
</uni:course>
```

RDF Collections

- A limitation of these containers is that there is no way to close them
  - "these are all the members of the container"
- RDF provides support for describing groups containing only the specified members, in the form of RDF collections
  - list structure in the RDF graph
    - constructed using a predefined collection vocabulary: **`rdf:List`**, **`rdf:first`**, **`rdf:rest`** and **`rdf:nil`**

RDF Collections

- Shorthand syntax:
  - "**`Collection`**" value for the **`rdf:parseType`** attribute:

```
<rdf:Description rdf:about="#CIT2112">
     <uni:isTaughtBy rdf:parseType="Collection">
          <rdf:Description rdf:about="#949111"/>
          <rdf:Description rdf:about="#949352"/>
          <rdf:Description rdf:about="#949318"/>
     </uni:isTaughtBy>
</rdf:Description>
```

Reification Example

```
<rdf:Description rdf:about="#949352">
      <uni:name>Grigoris Antoniou</uni:name>
</rdf:Description>
```

reifies as

```
<rdf:Statement rdf:ID="StatementAbout949352">
  <rdf:subject rdf:resource="#949352"/>
  <rdf:predicate
    rdf:resource="http://www.mydomain.org/uni-ns#name"/>
  <rdf:object>Grigoris Antoniou</rdf:object>
</rdf:Statement>
```

Reification

- `rdf:subject`, `rdf:predicate` and `rdf:object` allow us to access the parts of a statement
- The ID of the statement can be used to refer to it, as can be done for any description
- We write an `rdf:Description` if we don't want to talk about a statement further
- We write an `rdf:Statement` if we wish to refer to a statement

Basic Ideas of RDF Schema

- RDF is a universal language that lets users describe resources in their own vocabularies
  - RDF does not assume, nor does it define semantics of any particular application domain
- RDF Schema (RDFS):
  - The user can add a particular domain in RDF Schema using:
    - Classes and Properties
    - Class Hierarchies and Inheritance
    - Property Hierarchies

Classes and their Instances

- We must distinguish between
  - Concrete "*things*" (***individual*** objects) in the domain: **`Discrete Maths`**, **`David Billington`**, etc.
  - Sets of individuals sharing properties called ***classes***: lecturers, students, courses etc.
- Individual objects that belong to a class are referred to as instances of that class
- The relationship between instances and classes in RDF is through **`rdf:type`**

Why Classes are Useful?

- Impose restrictions on what can be stated in an RDF document using the schema
  - As in programming languages
    - E.g. prevent **`A+1`**, where **`A`** is an array
      - the arguments of + must be numbers
  - Disallow nonsense from being stated

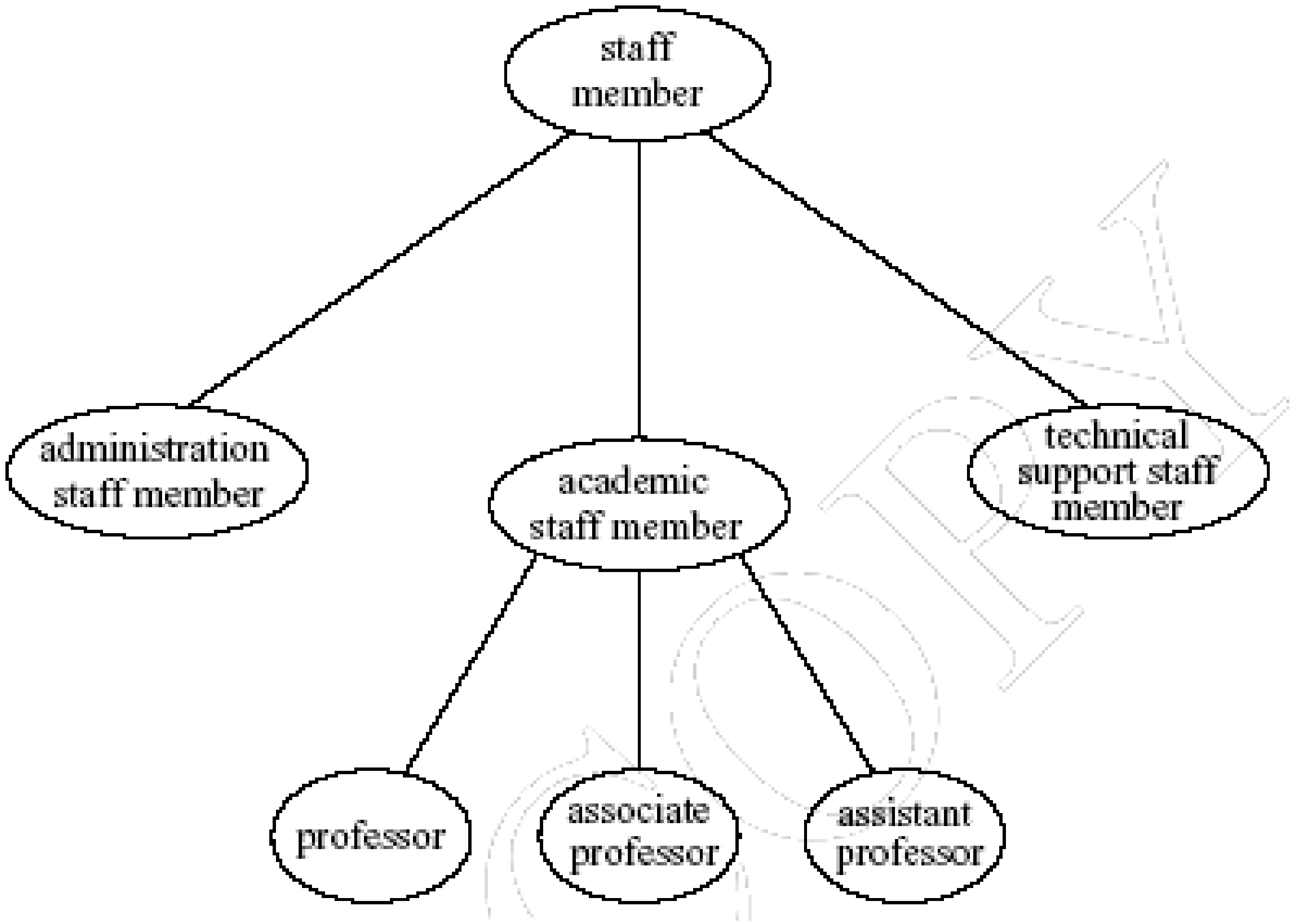
Nonsensical Statements disallowed through the Use of Classes

- **`Discrete Maths`** is taught by **`Concrete Maths`**
  - We want courses to be taught by lecturers only
  - Restriction on values of the property “is taught by” (***range*** restriction)
- Room **`MZH5760`** is taught by **`David Billington`**
  - Only courses can be taught
  - This imposes a restriction on the objects to which the property can be applied (***domain*** restriction)

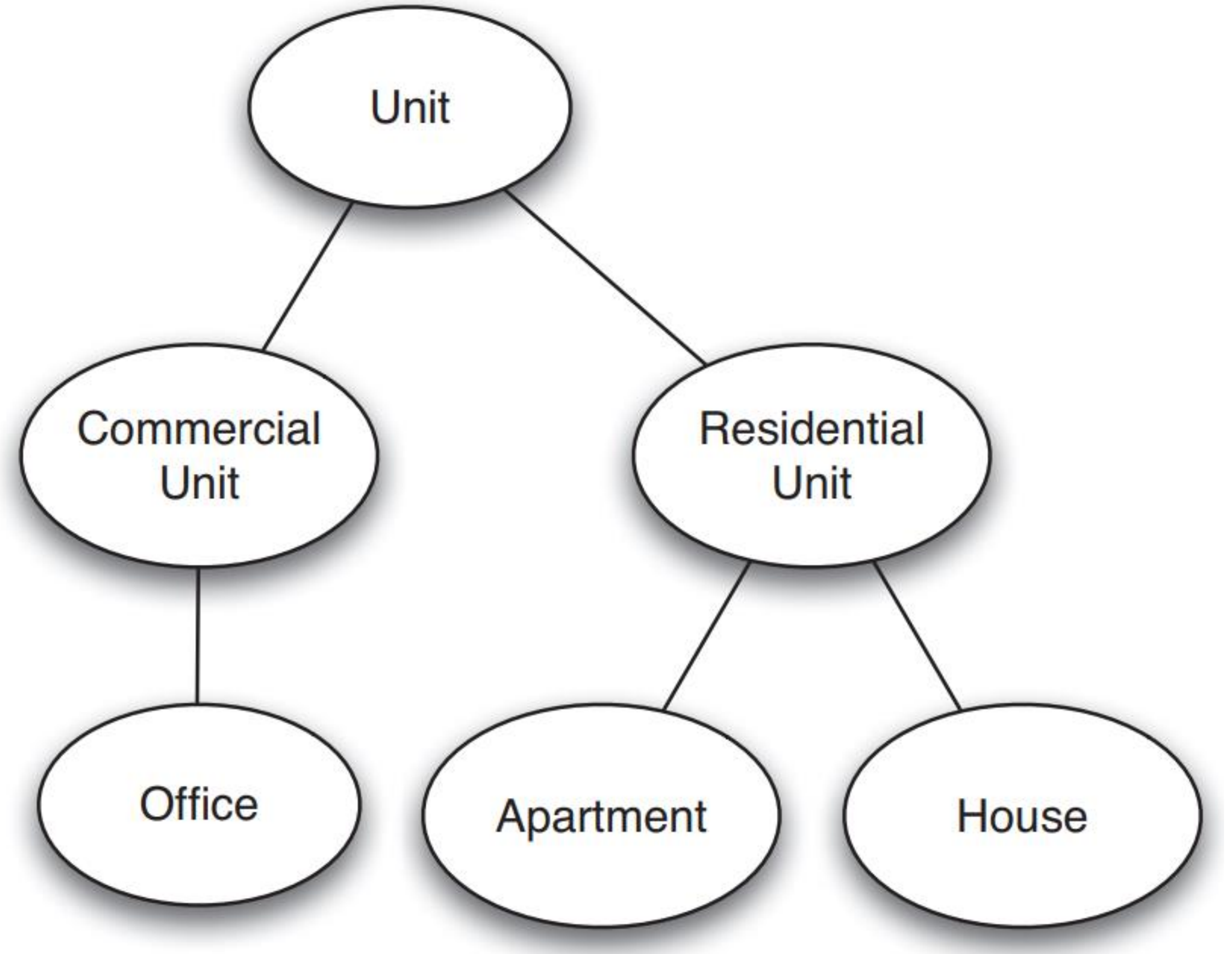
Class Hierarchies

- Classes can be organized in hierarchies
  - **`A`** is a *subclass* of **`B`** if every instance of **`A`** is also an instance of **`B`**
  - Then **`B`** is a *superclass* of **`A`**
- There is no requirement in RDF Schema that the classes together form a strict hierarchy
  - A subclass graph need not be a tree
  - A class may have multiple superclasses
    - If a class A is a subclass of both B1 and B2, this simply means that every instance of A is both an instance of B1 and an instance of B2.

Class Hierarchy Example



Class Hierarchy Example



Inheritance in Class Hierarchies

- ***Range restriction***: Courses must be taught by academic staff members only
  - **`Michael Maher`** is a professor
  - He inherits the ability to teach from the class of academic staff members
- This is done in RDF Schema by fixing the semantics of "is a subclass of"
  - It is not up to an application (RDF processing software) to interpret "is a" subclass of

Object-oriented

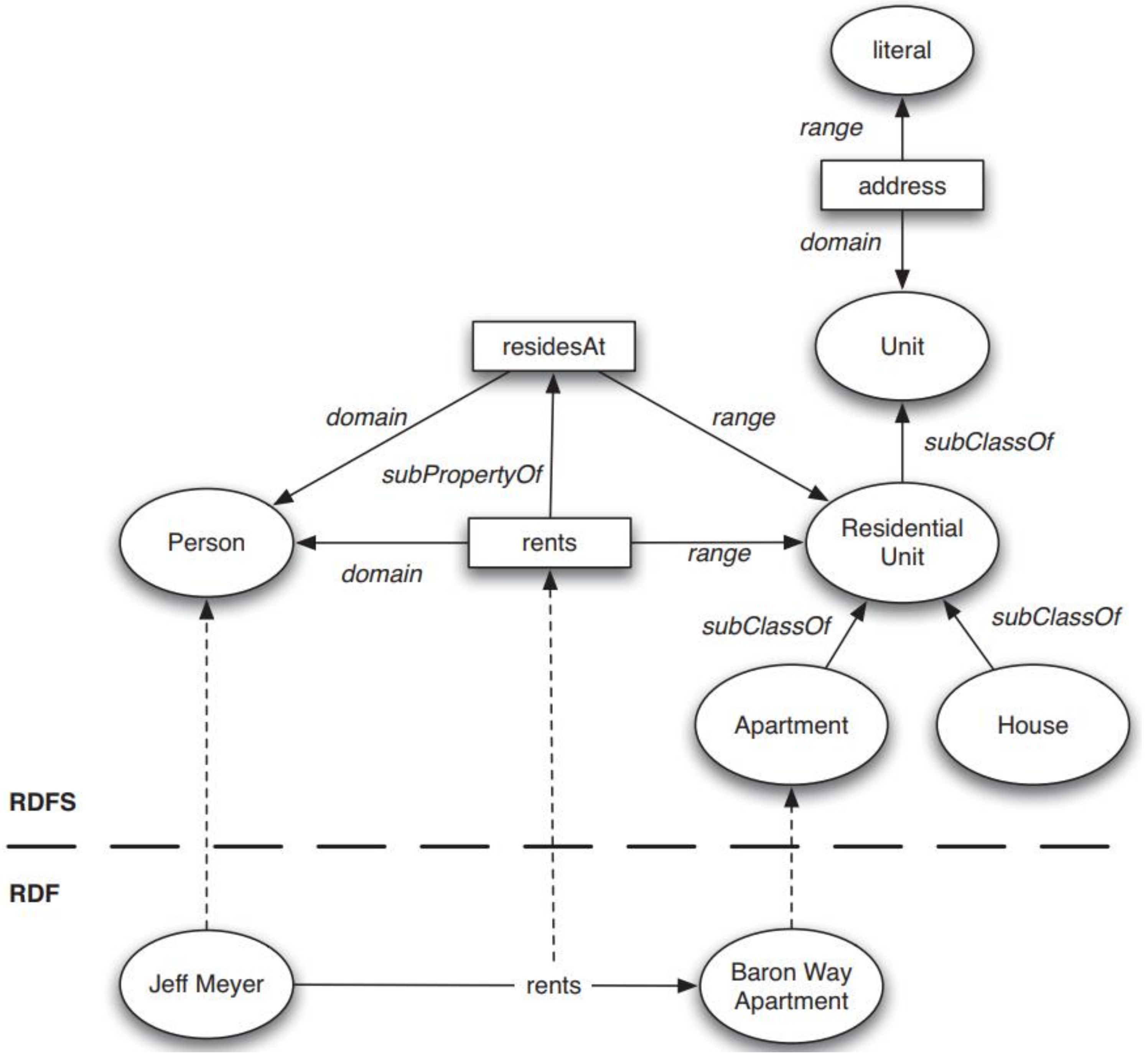
- There are differences between RDFS and OO:
  - In object-oriented programming, a class defines the properties that apply to it.
    - To add new properties to a class means to modify the class.
  - In RDFS, properties are not encapsulated as attributes in class definitions.
    - It is possible to define new properties that apply to an existing class without changing that class.
    - This is a powerful mechanism with far-reaching consequences: we may use classes defined by others and adapt them to our requirements through new properties

Property Hierarchies

- Hierarchical relationships for properties
  - E.g., "is taught by" is a subproperty of "involves"
  - If a course **`C`** is taught by an academic staff member **`A`**, then **`C`** also involves **`A`**
- The converse is not necessarily true
  - E.g., **`A`** may be the teacher of the course **`C`**, or
  - a tutor who marks student homework but does not teach **`C`**
- **`P`** is a subproperty of **`Q`**, if **`Q(x,y)`** is true whenever **`P(x,y)`** is true

RDF Layer vs RDF Schema Layer

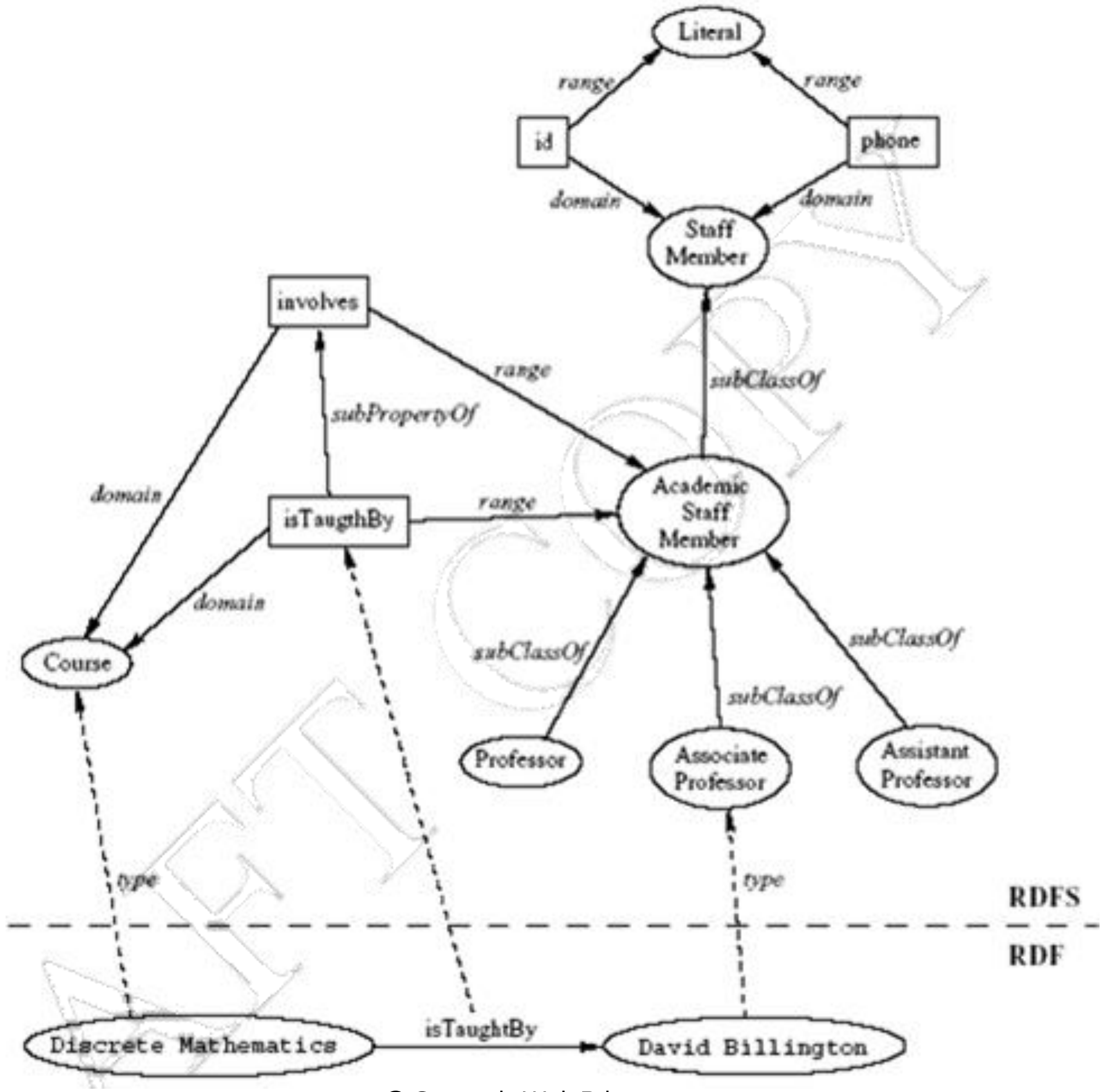
- Consider the RDF statement:
  - *Jeff Meyer rents the Baron Way Apartment.*
    - The schema for this statement may contain classes such as person, apartments, houses, units, and properties such as rents, resides at, or address.
- The schema is itself written in a formal language, RDF Schema, that can express its ingredients:
  - **`subClassOf`**,
  - **`Class`** (bubbles above the dashed line),
  - **`Property`** (blocks),
  - **`subPropertyOf`**,
  - **`Resource`** (bubbles below the dashed line are instances)


literal
range
address
domain
Unit
residesAt
domain
range
subClassOf
subPropertyOf
Person
rents
range
Residential Unit
domain
subClassOf
subClassOf
Apartment
House
RDFS
RDF
Jeff Meyer
rents
Baron Way Apartment

RDF Layer vs RDF Schema Layer

- Another example:
  - ***Discrete Mathematics*** *is taught by* ***David Billington***

RDF Layer vs RDF Schema Layer



@ Semantic Web Primer

RDF Schema: The Language

- RDF Schema provides modeling primitives
  - One decision that must be made is what formal language to use.
  - The modeling primitives of RDF Schema are defined using resources and properties in RDF itself!
    - a labeled graph that can be encoded in RDF.

RDF Schema: The Language

- If we wish to say that the class "apartment" is a subclass of "residential unit"
  - Define the required resources for **`apartment`**, **`residential_unit`**, and **`subClassOf`**
  - define **`subClassOf`** to be a property;
  - write the triple (**`apartment subClassOf residential_unit`**)
- All these steps are within the capabilities of RDF.
  - So, an RDFS document is just an RDF document, and we use one of the standard syntaxes for RDF

Core Classes

- The core classes are:
  - **`rdfs:Resource`**, the class of all resources
  - **`rdfs:Class`**, the class of all classes
  - **`rdfs:Literal`**, the class of all literals (strings)
  - **`rdf:Property`**, the class of all properties
  - **`rdf:Statement`**, the class of all reified statements

Core Properties for Defining Relationships

- The core properties for defining relationships are:
  - **`rdf:type`**, which relates a resource to its class
  - **`rdfs:subClassOf`**, which relates a class to one of its superclasses
  - **`rdfs:subPropertyOf`**, which relates a property to one of its superproperties
- **`rdfs:subClassOf`** and **`rdfs:subPropertyOf`** are transitive
- **`rdfs:Class`** is a subclass of **`rdfs:Resource`** (every class is a resource)
- **`rdfs:Resource`** is an instance of **`rdfs:Class`** (**`rdfs:Resource`** is the class of all resources, so it is a class)

Core Properties for Restricting Properties

- The core properties for restricting properties are:
  - **`rdfs:domain`**, which specifies the domain of a property **`P`** and states that any resource that has a given property is an instance of the domain classes.
  - **`rdfs:range`**, which specifies the range of a property **`P`** and states that the values of a property are instances of the range classes.

Utility Properties

- A resource may be defined and described in many places on the web. The following properties allow us to define links to those addresses:
  - **`rdfs:seeAlso`** relates a resource to another resource that explains it.
  - **`rdfs:isDefinedBy`** is a subproperty of **`rdfs:seeAlso`** and relates a resource to the place where its definition, typically an RDF schema, is found.

Utility Properties

- Properties that allow us to provide more information intended for human readers:
  - **`rdfs:comment`**, comments, typically longer text, can be associated with a resource.
  - **`rdfs:label`**, a human-friendly label (name) is associated with a resource.
    - Among other purposes, it may serve as the name of a node in a graphic representation of the RDF document.

Example: Housing

```
@prefix swp:<http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

swp:Person rdf:type rdfs:Class.
swp:Person rdfs:comment "The class of people".

swp:Unit rdf:type rdfs:Class.
swp:Unit rdfs:comment "A self-contained section of accommodations
in a larger building or group of buildings.".

swp:ResidentialUnit rdf:type rdfs:Class.
swp:ResidentialUnit rdfs:subClassOf swp:Unit.
swp:ResidentialUnit rdfs:comment "The class of all units or places where
people live.".

swp:Apartment rdf:type rdfs:Class.
swp:Apartment rdfs:subClassOf swp:ResidentialUnit.
swp:Apartment rdfs:comments "The class of apartments"
```

Example: Housing

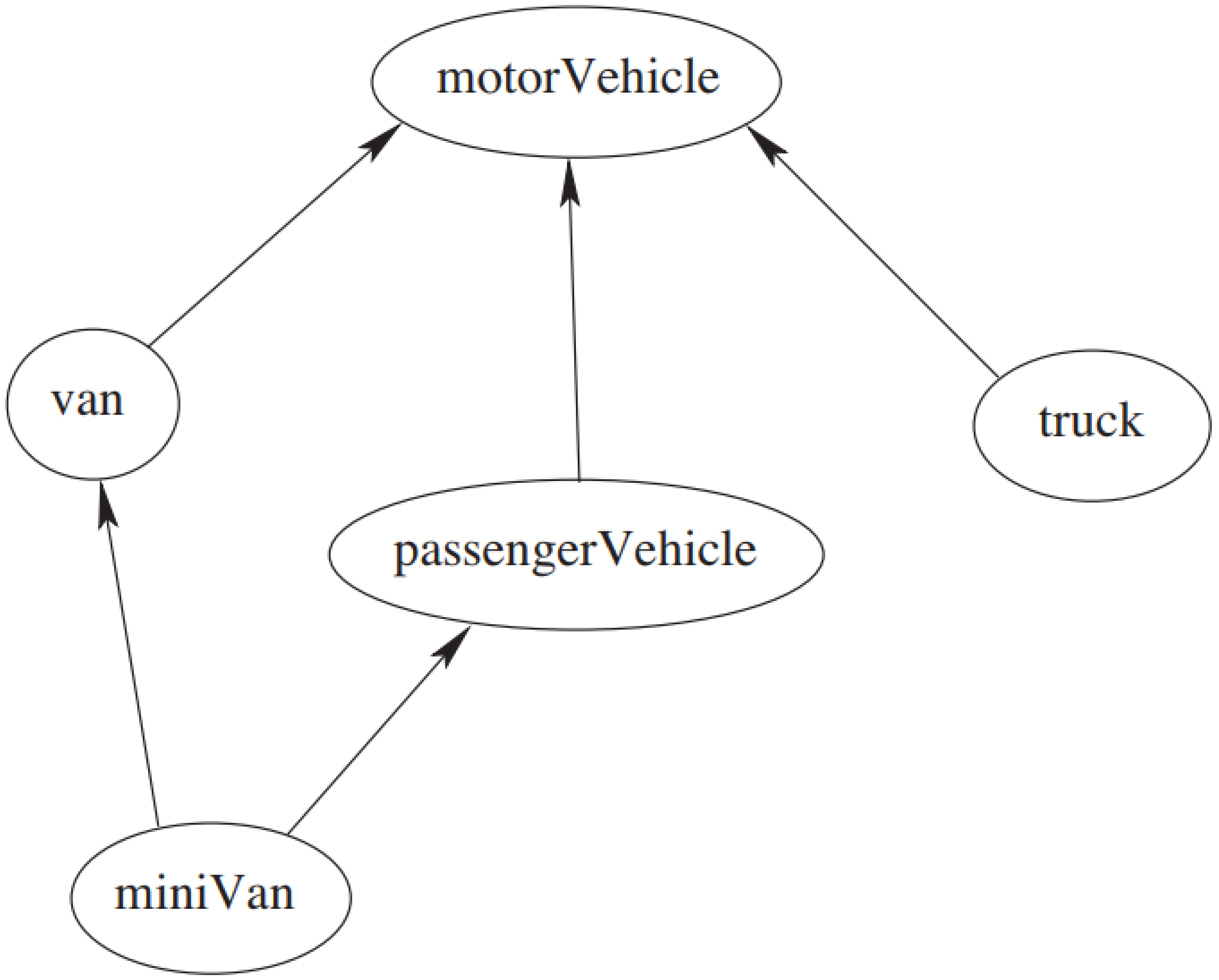
```
swp:House rdf:type rdfs:Class.
swp:House rdfs:subClassOf swp:ResidentialUnit.
swp:House rdfs:comment "The class of houses".

swp:residesAt rdf:type rdfs:Property.
swp:residesAt rdfs:comment "Relates persons to their residence".
swp:residesAt rdfs:domain swp:Person.
swp:residesAt rdfs:range swp:ResidentialUnit.

swp:rents rdf:type rdfs:Property.
swp:rents rdfs:comment "It inherits its domain (swp:Person)
and range (swp:ResidentialUnit) from its superproperty (swp:residesAt)".
swp:rents rdfs:subPropertyOf swp:residesAt.

swp:address rdf:type rdfs:Property.
swp:address rdfs:comment "Is a property of units and takes literals as
its value".
swp:address rdfs:domain swp:Unit.
swp:address rdfs:range rdf:Literal.
```

Example: Motor Vehicles



Example: Motor Vehicles

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<#miniVan> a rdfs:Class ;
      rdfs:subClassOf <#passengerVehicle>, <#van> .

<#motorVehicle> a rdfs:Class .

<#passengerVehicle> a rdfs:Class ;
      rdfs:subClassOf <#motorVehicle> .

<#truck> a rdfs:Class ;
      rdfs:subClassOf <#motorVehicle> .

<#van> a rdfs:Class ;
      rdfs:subClassOf <#motorVehicle> .
```

Example: A University in RDF/XML

```
<rdf:Property rdf:ID="phone">
  <rdfs:comment>
   It is a property of staff members and takes
     literals as values.
  </rdfs:comment>
  <rdfs:domain rdf:resource="#staffMember"/>
   <rdfs:range
 rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
```

RDF and RDF Schema in RDF Schema

- It is useful to see how RDF and RDF Schema are defined themselves in RDF Schema

RDF in RDF Schema (represented in RDF/XML)

```
<?xml version="1.0" encoding="UTF-16"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#¨>

  <rdfs:Class rdf:ID="Statement"
    rdfs:comment="The class of triples consisting of a
      predicate, a subject and an object (that is, a
      reified statement)"/>

  <rdfs:Class rdf:ID="Property"
    rdfs:comment="The class of properties"/>

  <rdfs:Class rdf:ID="Bag"
    rdfs:comment="The class of unordered collections"/>
```

RDF in RDF Schema

```
<rdfs:Class rdf:ID="Seq"
  rdfs:comment="The class of ordered collections"/>

<rdfs:Class rdf:ID="Alt"
  rdfs:comment="The class of collections of alternatives"/>

<rdf:Property rdf:ID="predicate"
    rdfs:comment="Identifies the property of a statement
       in reified form">
  <rdfs:domain rdf:resource="#Statement"/>
  <rdfs:range rdf:resource="#Property"/>
</rdf:Property>

<rdf:Property rdf:ID="subject"
  rdfs:comment="Identifies the resource that a statement is
   describing when representing the statement in reified form¨>
  <rdfs:domain rdf:resource="#Statement"/>
</rdf:Property>
```

RDF in RDF Schema

```
  <rdf:Property rdf:ID="object"
   rdfs:comment="Identifies the object of a statement when
     representing the statement in reified form"/>

  <rdf:Property rdf:ID="type"
   rdfs:comment="Identifies the class of a resource. The
     resource is an instance of that class."/>

</rdf:RDF>
```

RDF Schema in RDF Schema

```
<?xml version="1.0" encoding="UTF-16"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#¨>

  <rdfs:Class rdf:ID="Resource"
    rdfs:comment="The most general class"/>

  <rdf:Property rdf:ID="comment"
      rdfs:comment="Use this for descriptions¨>
    <rdfs:domain rdf:resource="#Resource"/>
    <rdfs:range rdf:resource="#Literal"/>
  </rdfs:Class>

  <rdfs:Class rdf:ID="Class"
      rdfs:comment="The concept of classes.
        All classes are resources.¨>
    <rdfs:subClassOf rdf:resource="#Resource"/>
  </rdfs:Class>
```

RDF Schema in RDF Schema

```
  <rdf:Property rdf:ID="subClassOf¨>
    <rdfs:domain rdf:resource="#Class"/>
    <rdfs:range rdf:resource="#Class"/>
  </rdf:Property>

  <rdf:Property rdf:ID="subPropertyOf¨>
    <rdfs:domain rdf:resource="&rdf;Property"/>
    <rdfs:range rdf:resource="&rdf;Property"/>
  </rdf:Property>

</rdf:RDF>
```

RDF Schema Semantics

- These namespaces do not provide the full definition of RDF and RDF Schema.
  - **`rdfs:subClassOf`** specifies only that it applies to classes and has a class as a value.
    - The meaning of being a subclass, namely, that all instances of one class are also instances of its superclass, is not expressed anywhere.
    - In fact, it cannot be expressed in an RDF document.
    - If it could, there would be no need for defining RDF Schema.
  - We will provide a formal semantics
  - RDF parsers and other software tools for RDF (including query processors) must be aware of the full semantics

An Axiomatic Semantics for RDF and RDF Schema

- We formalize the meaning of the modeling primitives of RDF and RDF Schema
- By translating into first-order logic (FOL) (i.e., predicate logic, universally accepted as the foundation of all (symbolic) knowledge representation)
  - Formulas used in this formalization are referred to as axioms
  - We make the semantics unambiguous and machine accessible
  - We provide a basis for reasoning support by automated reasoners manipulating logical formulas

The FOL Approach

- All language primitives in RDF and RDF Schema are represented by constants:
  - **`Resource`**, **`Class`**, **`Property`**, **`subClassOf`**, etc.
    - A few predefined predicates are used as a foundation for expressing relationships between the constants
- We use predicate logic with equality and an auxiliary theory of lists
- Variable names begin with **`?`**
- All axioms are implicitly universally quantified

An Auxiliary Axiomatization of Lists

- Lists are used to represent containers in RDF
  - They are also needed to capture the meaning of certain constructs (such as cardinality constraints) in richer ontology languages
- Function symbols for lists:
  - **`nil`** (empty list)
  - **`cons(x,l)`** (adds an element to the front of the list)
  - **`first(l)`** (returns the first element)
  - **`rest(l)`** (returns the rest of the list)
- Predicate symbols for lists:
  - **`item(x,l)`** (tests if an element occurs in the list)
  - **`list(l)`** (tests whether **`l`** is a list)

Basic Predicates

- **`PropVal(P,R,V)`**
  - A predicate with 3 arguments, which is used to represent an RDF statement with resource **`R`**, property **`P`** and value **`V`**
    - An RDF statement (triple) **`(R,P,V)`** is represented as **`PropVal(P,R,V)`**.
- **`Type(R,T)`**
  - Specifies that the resource **`R`** has the **`type T`**
  - Short for **`PropVal(type,R,T)`**

`Type(?r,?t)` ↔ `PropVal(type,?r,?t).`

Basic Predicates

- Most axioms provide typing information
  - For example,

    `Type(subClassOf, Property).`

  says that **`subClassOf`** is a property

RDF Classes

- Constants: **`Class`**, **`Resource`**, **`Property`**, **`Literal`**
  - All Constants are instances of **`Class`**

    ```
    Type(Class,Class).
    Type(Resource,Class).
    Type(Property,Class).
    Type(Literal,Class).
    ```

  - **`Resource`** is the most general class: every class and every property is a resource

    ```
    Type(?c,Class) → Type(?c,Resource).
    Type(?p,Property) → Type(?p,Resource).
    ```

  - The predicate in an RDF statement must be a property

    ```
    PropVal(?p,?r,?v) → Type(?p,Property).
    ```

The **`type`** Property

- **`type`** is a property:

  `Type(type, Property).`

  - Note that it is equivalent to
    `PropVal(type,type,Property).`
    - the type of **`type`** is **`Property`**.
  - **`type`** can be applied to resources (domain) and has a class as its value (range)

`Type(?r,?c)` $\rightarrow$

`(Type(?r,Resource)` $\wedge$ **`Type(?c,Class)).`**

The Auxiliary **`FuncProp`** Property

- A *functional property* is a property that is a function: it relates a resource to (at most) one value
  - Functional properties are not a concept of RDF but are used in the axiomatization of other primitives.
- **`P`** is a *functional property* if, and only if,
  - it is a property, and
  - there are no **`x`**, **`y1`** and **`y2`** with **`P(x,y1)`**, **`P(x,y2)`** and **`y1≠y2`**

```
Type(?p, FuncProp) ↔
  (Type(?p, Property) ∧
   ∀?r ∀?v1 ∀?v2
   (PropVal(?p,?r,?v1) ∧ PropVal(?p,?r,?v2) → ?v1=?v2)).
```

Reified Statements

- The constant **`Statement`** represents the class of all reified statements
  - All reified statements are resources, and **`Statement`** is an instance of Class:

```
Type(?s, Statement) → Type(?s, Resource).
Type(Statement, Class).
```

  - A reified statement can be decomposed into the three parts of an RDF triple:

```
Type(?st, Statement) →
 ∃?p ∃?r ∃?v(
    PropVal(Predicate, ?st, ?p) ∧
    PropVal(Subject, ?st, ?r) ∧
    PropVal(Object, ?st, ?v)).
```

Reified Statements

- Every statement has exactly one subject, one predicate, and one object
  - **`Subject`**, **`Predicate`**, and **`Object`** are functional properties

```
Type(Subject, FuncProp).
Type(Predicate, FuncProp).
Type(Object, FuncProp).
```

  - The **`Subject`** and **`Predicate`** values of a statement are **`Resource`**, respectively **`Property`**

```
PropVal(Subject,?st,?r)→
   (Type(?st,Statement) ∧ Type(?r,Resource)).
PropVal(Predicate,?st,?p)→
   (Type(?st,Statement) ∧ Type(?p,Property)).
```

Reified Statements

- The **`Object`** must apply to a reified statements and have as its value either a resource or a literal:

```
PropVal(Object,?st,?v)→
   (Type(?st,Statement) ∧
      (Type(?v,Resource) ∨ Type(?v,Literal))).
```

Containers

- All containers are resources:

```
Type(?c,Container) → Type(?c, Resource).
```

- Containers are lists:

```
Type(?c,Container) → list(?c).
```

- Containers are bags or sequences or alternatives:

```
Type(?c,Container) ↔
  (Type(?c,Bag) ∨ Type(?c,Seq) ∨ Type(?c,Alt)).
```

- Bags and sequences are disjoint:

```
¬(Type(?x,Bag) ∧ Type(?x,Seq)).
```

Containers

- For every natural number **`n > 0`**, there is the selector **`_n`**, which selects the **`n`**th element of a container
  - It is a functional property:

```
Type(_n,FuncProp).
```

  - It applies to containers only:

```
PropVal(_n,?c,?o) →
   Type(?c,Container).
```

RDF Schema Subclass

- **`subClassOf`** is a property:

  `Type(subClassOf,Property).`
- If a class **`C`** is a subclass of a class **`C'`**, then all instances of **`C`** are also instances of **`C'`**:

```
PropVal(subClassOf,?c,?c') ↔
   (Type(?c,Class) ∧ Type(?c',Class) ∧
    ∀?x (Type(?x,?c) → Type(?x,?c'))).
```

RDF Schema Subproperty

- **`P`** is a subproperty of **`P'`** if **`P'(x,y)`** is true whenever **`P(x,y)`** is true:

```
Type(subPropertyOf,Property).
PropVal(subPropertyOf,?p,?p') ↔
   (Type(?p,Property) ∧
    Type(?p',Property) ∧
    ∀?r ∀?v (PropVal(?p,?r,?v) →
                     PropVal(?p',?r,?v)))
```

Constraints

- Every constraint resource is a resource:

```
PropVal(subClassOf,ConstraintResource,Resource).
```

- Constraint properties are all properties that are also constraint resources:

```
Type(?cp, ConstraintProperty) ↔
   (Type(?cp, ConstraintResource) ∧
     Type(?cp, Property)).
```

Domain and Range

- **`domain`** and **`range`** of a property are constraint properties:

```
Type(domain, ConstraintProperty).
Type(range, ConstraintProperty).
```

  - If the domain of **`P`** is **`D`**, then for every **`P(x,y)`**, **`x∈D`**

```
PropVal(domain,?p,?d) →
     ∀?x ∀?y (PropVal(?p,?x,?y) → Type(?x,?d)).
```

  - If the range of **`P`** is **`R`**, then for every **`P(x,y)`**, **`y∈R`**

```
PropVal(range,?p,?r) →
     ∀?x ∀?y (PropVal(?p,?x,?y) → Type(?y,?r)).
```

Domain and Range

- The following formulas that can be inferred from the preceding ones:

```
PropVal(domain,domain,Property).
PropVal(range,domain,Class).
PropVal(domain,range,Property).
PropVal(range,range,Class).
```

Semantics based on Inference Rules

- We have formalized the semantics of RDF and RDFS in first-order logic
  - Software equipped with this knowledge is able to draw interesting conclusions
    - For example, given that the **`range`** of **`rents`** is **`ResidentialUnit`**, that **`ResidentialUnit`** is a subclass of **`Unit`**, and that **`rents(JeffMeyer, BaronWayApartment)`**, the agent can automatically deduce **`Unit(BaronWayApartment)`** using the predicate logic semantics or one of the predicate logic proof systems.

Semantics based on Inference Rules

- The previous axiomatic semantics can be used for automated reasoning with RDF and RDF Schema
  - However, it requires a first-order logic proof system to do so.
  - This is a very heavy requirement and may not scale when millions (or billions) of statements are involved (e.g., millions of statements of the form **`Type(?r, ?c)`**).
- For this reason, RDF has also been given a semantics (and an inference system that is sound and complete for this semantics) directly in terms of RDF triples instead of restating RDF in terms of first-order logic

Semantics based on Inference Rules

- Semantics in terms of RDF triples instead of restating RDF in terms of first-order logic
  - … and sound and complete inference systems
- This inference system consists of inference rules of the form:

  IF **E** contains certain triples

    THEN add to **E** certain additional triples

where **E** is an arbitrary set of RDF triples

  - The total set of these closure rules is no larger than a few dozen and can be efficiently implemented without sophisticated theorem-proving technology

Examples of Inference Rules

- Any resource **`?p`** that is used in the property position of a triple can be inferred to be a member of the class **`rdf:Property`**

IF **`E`** contains the **`triple (?x,?p,?y)`**

THEN **`E`** also contains **`(?p,rdf:type,rdf:property).`**

- The transitivity of the **`subclass`** relation:

IF **`E`** contains the triples **`(?u,rdfs:subClassOf,?v)`** and

`(?v,rdfs:subclassOf,?w)`

THEN **`E`** also contains the triple **`(?u,rdfs:subClassOf,?w)`**

Examples of Inference Rules

- The meaning of **`rdfs:subClassOf`**

IF **`E`** contains the triples **`(?x,rdf:type,?u)`** and

`(?u,rdfs:subClassOf,?v)`

THEN **`E`** also contains the triple **`(?x,rdf:type,?v)`**.

Examples of Inference Rules

- Any resource **`?y`** which appears as the value of a property **`?p`** can be inferred to be a member of the range of **`?p`**

IF **`E`** contains the triples **`(?x,?p,?y)`** and

`(?p,rdfs:range,?u)`

THEN **`E`** also contains the triple **`(?y,rdf:type,?u)`**.

- This shows that range definitions in RDF Schema are not used to restrict the range of a property, but rather to infer the **membership** of the range

Summary

- RDF provides a foundation for representing and processing machine understandable data
- RDF provides a foundation for representing and processing metadata
- RDF has a graph-based data model
- RDF has multiple standard syntaxes (Turtle, RDF/XML, RDFa) to support syntactic interoperability
  - XML and RDF complement each other because RDF supports semantic interoperability
- RDF has a decentralized philosophy and allows incremental building of knowledge, and its sharing and reuse

Summary

- RDF is domain-independent
  - RDF Schema provides a mechanism for describing specific domains
- RDF Schema is a primitive ontology language
  - It offers certain modelling primitives with fixed meaning
- Key concepts of RDF Schema are: class, subclass relations, property, subproperty relations, and domain and range restrictions

Points for Discussion in Subsequent Chapters

- Query languages for RDF and RDFS, including SPARQL
- RDF Schema is quite primitive as a modelling language for the Web
  - Many desirable modelling primitives are missing
  - Therefore we need an ontology layer on top of RDF and RDF Schema

References

- http://ww.w3.org/TR/rdf-syntax-grammar/
- http://www.w3.org/TR/rdf-schema/
- http://www.w3.org/TR/turtle/
- http://www.w3.org/TR/xhtml-rdfa-primer/
- http://www.daml.org/2001/03/axiomatic-semantics.html
- http://www.w3.org/TR/rdf-mt/
- http://www.w3.org/TR/rdf-concepts/
- http://www.w3.org/TR/rdf-primer/
- http://www4.wiwiss.fu-berlin.de/bizer/trig/