CS4211: Advanced Software Engineering
Part I: Introduction and Domain Engineering (Ontology)

Dr. DONG Jin-Song, Associate Professor
(www.comp.nus.edu.sg/~dongjs)
Computer Science Department
National University of Singapore
2005
Course General Issues

• Lecturer: Dr. DONG Jin Song office: SoC1-4-18, ext:4353,

• Lecture Time: Tue, 11:00 - 14:00 (inc: 1 hr tutorial),

• Lecture Location: LT33 (SoC1, level 2)

• CA breakdown:
  – 30% max-3 group project (the deadline: 5pm on 11 Nov 2005, details later)
  – 70% final exam (open book, 21 November 2005 Evening)
The Classical Engineer

- models with calculus, geometry
- analyses using classical theorems (Newton, Fourier, Gauss, ...)
- constructs a *hard* product

The Software Engineer

- models with set theory, logic
- analyses using rules of inference
- constructs a *soft* product
The Aim of This Course

Before Software can be Designed, its Requirements must be well understood. Before Requirements can be expressed the Application Domain must be similarly well understood.

The aim of this course is to introduce some of the latest, sound and advanced techniques of the three major phases of Software Engineering:

- Domain Engineering (e.g. ontology),
- Requirements Engineering (e.g. precise specifications), and
- Software Design (e.g. abstraction and analyzable models).
Overview

- Domain Engineering (Semantically Representing Knowledge Domain)
  - Why not document it on the web? — Good idea, but lack of meaning, need ontology
  - Latest web ontology languages: DAML and OWL
- Requirements Engineering (Precisely Capturing Requirements)
  - Why not document it in a structured English? — Current practice, but not precise enough, need Rigorous Specification Techniques
  - Most successful one: Z (won Oxford/IBM the Queens Award!)
- Software Design (Soundly Constructing Models)
  - Why not to use UML for the job? — Getting popular, but cannot be analyzed or verified, need logical and abstract models
  - Latest abstract design technique: Alloy (developed at MIT, promising!)
Online Information

- DAML http://www.daml.org/, OWL http://www.w3.org/2004/OWL/ (W3C)
- Z http://www.comlab.ox.ac.uk/archive/z.html (Oxford)
- Alloy http://alloy.mit.edu/ (MIT)
Domain Engineering (Ontology)

Domain engineering aims to model common knowledge in a problem domain. Ontologies have also been pointed as holding great promise for semantically capturing knowledge and support software reuse in general.

Reuse and Sharing Advantages:

- Less money
- Less time
- Less resources
Definitions of Ontologies

- An ontology defines the basic terms and relations comprising the vocabulary of a topic area, as well as the rules for combining terms and relations to define extensions to the vocabulary.

- An ontology is an explicit specification of a conceptualization and a hierarchically structured set of terms for describing a domain that can be used as a skeletal foundation for a knowledge base.

- Ontology as
  - a philosophical discipline
  - an informal conceptual system
  - a formal semantic account
  - a representation of a conceptual system via a logical theory
  - the vocabulary used by a logical theory
  - a (meta-level) specification of a logical theory
Semantic Web

“The Semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. It is the idea of having data on the Web defined and linked in a way that it can be used for more effective discovery, automation, integration, and reuse across various applications.” – W3C (www.w3.org/2001/sw)
Semantic Web

• Goals
  – Realizing the full potential of the Web
  – Making it possible for tools (agents) to effectively process information.
  – Ultimate goal - effective and efficient global information/knowledge exchange

• Building on proven ideas
  – Combines XML, RDF, hypertext and metadata approaches to linked information
  – Focuses on general principles of Web automation and data aggregation
RDF, DAML+OIL and OWL

• Resource Description Framework (RDF) — 1999
  – An RDF document is a collection of assertions in subject verb object form for describing web resources
  – Provides interoperability between applications that exchange machine-understandable information on the Web
  – Use XML as a syntax, include XMLNS, and URIs

• DARPA Agent Markup Language (DAML+OIL) — 2001
  – Semantic markup language based on RDF, and
  – Extends RDF(S) with richer modelling primitives
  – DAML combines Ontology Interchange Language (OIL).

• OWL Web Ontology Language — 2003 (become W3C rec)
  – Based on DAML+OIL
  – Three levels support: Lite, DL, Full
HTML and XML

- HTML

```html
<H1> Semantic Web and Software Engineering </H1>
<UL>
 <LI> Teacher: Jin Song Dong 
 <LI> Students: s19908, s20015 
 <LI> Requirements: discrete maths 
</UL>
```

- XML

```xml
<course>
 <title> Semantic Web and Software Engineering </title>
 <teacher> Jin Song Dong </teacher>
 <students> s19908, s20015 </students>
 <req> discrete maths </req>
</course>
```
Lack semantics in XML

• The XML is accepted as the emerging standard for data interchange on the Web. XML allows authors to create their own markup (e.g. `<course>`), which seems to carry some semantics.

• However, from a computational perspective tags like `<course>` carries as much semantics as a tag like `<H1>`. A computer simply does not know, what a course is and how the concept course is related to other concepts.

• XML may help humans predict what information might lie “between the tags” in the case of `<students> </students>`, but XML can only help.

• Only feasible for closed collaboration, e.g., agents in a small and stable community/intranet
RDF Basics

- Resources — Things being described by RDF expressions. Resources are always named by URIs, e.g.
  - HTML Document
  - Specific XML element within the document source.
  - Collection of pages

- Properties — Specific aspect, characteristic, attribute or relation used to describe a resource, e.g. Creator, Title ...

- Statements —
  Resource (Subject) + Property (Predicate) + Property Value (Object)
**RDF Statement Example 1**

Dong, Jin Song is the creator of the web page
http://www.comp.nus.edu.sg/cs4211

- Subject (Resource) - http://www.comp.nus.edu.sg/cs4211
- Predicate (Property) - Creator
- Object (Literal) Dong, Jin Song

![Graphical representation of RDF statement]
RDF Statement Example 2

Dong, Jin Song whose e-mail is dongjs@comp.nus.edu.sg is the creator of the web page http://www.comp.nus.edu.sg/cs4211
RDF in XML syntax

```xml
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/">

  <rdf:Description about="http://www.comp.nus.edu.sg/cs4211">
    <dc:creator>Dong, Jin Song</dc:creator>
    <dc:title>Advanced Software Engineering</dc:title>
    <dc:date>2000-07-01</dc:date>
  </rdf:Description>

</rdf:RDF>
```
RDF Containers

- Bag - An unordered list of resources or literals
- Sequence - An ordered list of resources or literals
- Alternative - A list of resources or literals that represent alternatives for the value of a property
**Container example: Sequence**

Statement: The students of the course CS4211 in alphabetical order are Yuanfang Li, Jun Sun and Hai Wang.

```xml
<rdf:RDF xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
         xmlns:s="http://www.schemas.org/Course/">
  <rdf:Description about="http://www.comp.nus.edu.sg/~cs4211">
    <s:students>
      <rdf:Seq>
      </rdf:Seq>
    </s:students>
  </rdf:Description>
</rdf:RDF>
```
**RDF Schema**

- Basic vocabulary to describe RDF vocabularies, e.g.,
  
  `Class, subClassOf, Property, subPropertyOf, domain, range`

- Defines properties of the resources (e.g., title, author, subject, etc)

- Defines kinds of resources being described (books, Web pages, people, etc)

- **XML Schema** gives specific constraints on the structure of an XML document
  
  RDF Schema provides information about the interpretation of the RDF statements

- **RDF schema** uses XML syntax, but could theoretically use any other syntax
RDF Schema Example (Class)

<?xml version="1.0"?>
<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="Person">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Resource"/>
  </rdfs:Class>

  <rdfs:Class rdf:ID="Student">
    <rdfs:subClassOf rdf:resource="#Person"/>
  </rdfs:Class>

</rdf:RDF>
RDF Schema Example (Property)

```xml
<rdf:Property rdf:ID="teacher">
    <rdfs:domain rdf:resource="#Course"/>
    <rdfs:range rdf:resource="#Person"/>
</rdf:Property>

<rdf:Property rdf:ID="students">
    <rdfs:domain rdf:resource="#Course"/>
    <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq"/>
</rdf:Property>
```
Why RDF(S) is not enough

• Only range/domain constraints on properties (need others)
• No properties of properties (unique, transitive, inverse, etc.)
• No equivalence, disjointness, etc.
• No necessary and sufficient conditions (for class membership)
DAML+OIL

- Europe: Ontology Inference Language (OIL) extends RDF Schema to a fully-fledged knowledge representation language.
- US: DARPA Agent Markup Language (DAML)
- Merged as DAML+OIL in 2001
  - logical expressions
  - data-typing
  - cardinality
  - quantifiers
- Becomes OWL — W3C 2004
DAML: Setting up the namespaces

<rdf:RDF
    xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd = "http://www.w3.org/2000/10/XMLSchema#"
    xmlns:daml = "http://www.daml.org/2001/03/daml+oil#"
/>

DAML: Define Classes

<rdfs:Class rdf:ID="Animal"> </rdfs:Class>

<rdfs:Class rdf:ID="Male">
    <rdfs:subClassOf rdf:resource="#Animal"/> </rdfs:Class>

<rdfs:Class rdf:ID="Female">
    <rdfs:subClassOf rdf:resource="#Animal"/>
    <daml:disjointWith rdf:resource="#Male"/> </rdfs:Class>

<rdfs:Class rdf:ID="Man">
    <rdfs:subClassOf rdf:resource="#Person"/>
    <rdfs:subClassOf rdf:resource="#Male"/> </rdfs:Class>
DAML: Define Properties

<rdf:Property rdf:ID="hasParent">
  <rdfs:domain rdf:resource="#Animal"/>
  <rdfs:range rdf:resource="#Animal"/> </rdf:Property>

<rdf:Property rdf:ID="hasFather">
  <rdfs:subPropertyOf rdf:resource="#hasParent"/>
  <rdfs:range rdf:resource="#Male"/> </rdf:Property>
DAML: Define Restrictions

<rdfs:Class rdf:ID="Person"> <rdfs:subClassOf rdf:resource="#Animal"/>
   <rdfs:subClassOf>
       <daml:Restriction>
           <daml:onProperty rdf:resource="#hasParent"/>
           <daml:toClass rdf:resource="#Person"/>
       </daml:Restriction>
   </rdfs:subClassOf>

   <rdfs:subClassOf>
       <daml:Restriction daml:cardinality="1">
           <daml:onProperty rdf:resource="#hasFather"/>
       </daml:Restriction>
   </rdfs:subClassOf>

   <rdfs:subClassOf>
       <daml:Restriction daml:maxCardinality="1">
           <daml:onProperty rdf:resource="#hasSpouse"/>
       </daml:Restriction>
   </rdfs:subClassOf>
</rdfs:Class>
DAML: UniqueProperty and Transitive

<daml:UniqueProperty rdf:ID="hasMother">
  <rdfs:subPropertyOf rdf:resource="#hasParent"/>
  <rdfs:range rdf:resource="#Female"/>
</daml:UniqueProperty>

<daml:TransitiveProperty rdf:ID="hasAncestor">
</daml:TransitiveProperty>
DAML: hasValue and intersectionOf

<rdfs:Class rdf:ID="TallThing">
   <daml:sameClassAs>
      <daml:Restriction>
         <daml:onProperty rdf:resource="#hasHeight"/>
         <daml:hasValue rdf:resource="#tall"/>
      </daml:Restriction>
   </daml:sameClassAs>
</rdfs:Class>

<rdfs:Class rdf:ID="TallMan">
   <daml:intersectionOf rdf:parseType="daml:collection">
      <rdfs:Class rdf:about="#TallThing"/>
      <rdfs:Class rdf:about="#Man"/>
   </daml:intersectionOf>
</rdfs:Class>
DAML: instances

<Person rdf:ID="Adam">
  <hasHeight rdf:resource="#medium然/
</Person>
OWL: The three sublanguages

- *OWL Lite* supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1.

- *OWL DL* supports those users who want the maximum expressiveness while retaining computational completeness and decidability. OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class).

- *OWL Full* is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right.
**OWL: Changes from DAML+OIL**

- With respect to the three sublanguages, the DAML+OIL semantics is closest to the OWL DL semantics.
- The namespace was changed to [http://www.w3.org/2002/07/owl](http://www.w3.org/2002/07/owl)
- Cyclic subclasses are now allowed
- multiple `rdfs:domain` and `rdfs:range` properties are handled as intersection
- Various properties and classes were renamed, e.g., `daml:UniqueProperty` is replaced by `owl:FunctionalProperty`
- ... [http://www.w3.org/TR/owl-ref/](http://www.w3.org/TR/owl-ref/)
Beyond OWL: Ontology Rule Language (ORL)

- Decidability vs Expressiveness
- OWL is weak in express composite properties
- ORL extends OWL DL with a form of rules while maintaining compatibility with OWLs existing syntax and semantics.
ORL Example

<owlx:Rule>
  <owlx:antecedent>
    <owlx:individualPropertyAtom owlx:property="hasParent">
      <owlx:Variable owlx:name="x1" />
      <owlx:Variable owlx:name="x2" />
    </owlx:individualPropertyAtom>
    <owlx:individualPropertyAtom owlx:property="hasBrother">
      <owlx:Variable owlx:name="x2" />
      <owlx:Variable owlx:name="x3" />
    </owlx:individualPropertyAtom>
  </owlx:antecedent>
  <owlx:consequent>
    <owlx:individualPropertyAtom owlx:property="hasUncle">
      <owlx:Variable owlx:name="x1" />
      <owlx:Variable owlx:name="x3" />
    </owlx:individualPropertyAtom>
  </owlx:consequent>
</owlx:Rule>
Extracting DAML ontology from UML Models

There are a few approaches to transform UML class models to DAML ontology. One particular approach is to extend UML meta model to directly include the notion of ‘Property’ and ‘Restriction’. For example, a property restriction ‘work_for’ can be represented as:

```
Employee <-> work_for <-> Company
```
Process of Building Ontology
Case Study (Tutorial)

A classical ontology, “animal relation” is used to illustrate various modeling and reasoning issues in the web ontology construction.

Developing a DAML+OIL ontology that defines two classes animal and plant which are disjoint. The eats and eaten_by are two properties, which are inverse to each other. The domain of eats is animal. The carnivore is a subclass of animal which can only eat animals.
<daml:Class rdf:ID="animal">  </daml:Class>
<daml:Class rdf:ID="plant">
    <daml:disjointWith rdf:resource="#animal"/>
</daml:Class>

<daml:ObjectProperty rdf:about="eaten_by">  </daml:ObjectProperty>
<daml:ObjectProperty rdf:about="eats">
    <daml:inverseOf rdf:resource="#eaten_by"/>
    <rdfs:domain><daml:Class rdf:about="#animal"/>
</rdfs:domain>
</daml:ObjectProperty>

<daml:Class rdf:ID="carnivore">
    <rdfs:subClassOf rdf:resource="#animal"/>
    <rdfs:subClassOf>
        <daml:Restriction>  <daml:onProperty rdf:resource="#eats"/>
            <daml:toClass rdf:resource="#animal"/>
        </daml:Restriction>  </rdfs:subClassOf>
    </daml:Class>
Class property checking

Defining another class `tastyPlant` which is a subclass of `plant` and eaten by the `carnivore`. There is an inconsistency since by the ontology definition carnivores can only eat animals.

```xml
<daml:Class rdf:ID="tastyPlant">
  <rdfs:subClassOf rdf:resource="#plant"/>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#eat_by"/>
      <daml:toClass rdf:resource="#carnivore"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
```

Any problem with this definition?
Suppose we define that the `polyphagic_animal` eats at least two kind of things i.e `polyphagic_animal` objects have at least two distinct values for the property `eats`. There is also one kind of animal called `picky_animal` which only eats one other kind of animal.

```
<daml:Class rdf:ID="polyphagic_animal"> <rdfs:subClassOf rdf:resource="#animal"/>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#eats"/>
      <daml:minCardinality> 2 </daml:minCardinality>
    </daml:Restriction></rdfs:subClassOf></daml:Class>

<daml:Class rdf:ID="#picky_animal"> <rdfs:subClassOf rdf:resource="#animal"/>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#eats"/>
      <daml:Cardinality> 1 </daml:Cardinality>
    </daml:Restriction></rdfs:subClassOf></daml:Class>
```

Is `picky_animal` a kind of `polyphagic_animal`?
**Subsumption reasoning**

The task of subsumption reasoning is to infer a class is the subclass of another class. We use the relationship between the fish, shark and dolphin as an example to demonstrate this kind of reasoning task. In the animal ontology a property `breathe_by` is defined. The fish is a subclass of the `animal` which `breathe_by` the gill (The classes gill and lung are disjoint).

```xml
<daml:ObjectProperty rdf:ID="breathe_by"/>
<daml:Class rdf:ID="gill"> </daml:Class>
<daml:Class rdf:ID="lung"> <daml:disjointWith rdf:resource="#gill"/></daml:Class>

<daml:Class rdf:ID="fish"> <rdfs:subClassOf rdf:resource="#animal"/>
   <rdfs:subClassOf>
      <daml:Restriction>
         <daml:onProperty rdf:resource="#breathe_by"/>
         <daml:toClass rdf:resource="#gill"/>
      </daml:Restriction></rdfs:subClassOf> </daml:Class>
```

What about frogs and toads?
Shark

A class shark is a subclass of carnivore which breathe by the gill.

<daml:Class rdf:ID="shark">
  <rdfs:subClassOf rdf:resource="#carnivore"/>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#breathe_by"/>
      <daml:toClass rdf:resource="#gill"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

Is shark a kind of fish?
**Dolphin**

Dolphins are a kind of animal which breathe by lungs.

```xml
<daml:Class rdf:ID="dolphin">
    <rdfs:subClassOf rdf:resource="#animal"/>
    <rdfs:subClassOf>
        <daml:Restriction>
            <daml:onProperty rdf:resource="#breathe_by"/>
            <daml:toClass rdf:resource="#lung"/>
        </daml:Restriction>
    </rdfs:subClassOf>
</daml:Class>
```

Is dolphin is a kind of fish ?
**Instantiation**

Instantiation is a reasoning task which tries to check if an individual is an instance of a class. For example, we define two resources `aFeralAnimal` and `aMeekAnimal` as the instances of class `animal`. `aGill` is an instance of class `gill`. `aFeralAnimal` eats `aMeekAnimal` and breathes by `aGill`.

```xml
<animal rdf:ID="aMeekAnimal">
<gill rdf:ID="aGill">
<animal rdf:ID="aFeralAnimal">
  <breathe_by rdf:resource="aGill"/>
  <eats rdf:resource="aMeekAnimal"/>
</animal>
</animal>
```

Is `aFeralAnimal` a carnivore and a fish?
Checking Military Plan Ontology Experience

- Singapore DSO (Defence Science Organisation) has developed an IE engine which has been used to generate ontologies (in DAML) from military formation and plan (in natural language).

- A military ontology is made up of the following four main ingredient sets.
  - military operations and tasks, which define the logic order, type, and phases of a military campaign.
  - military units, which are the participants of the military operations and tasks,
  - geographic locations, where such operations take place and
  - time points for constraining the timing of such operations.
Checking Military Plan Ontology Experience

- The challenge is to ensure: “No military unit is to be assigned to 2 different tasks (at different locations) at the same time”
- Can OWL model and check this property?
- The solution will be presented in the next topic.