
CmpE 593:Multiagent Systems

Pinar Yolum
pinar.yolum@boun.edu.tr

Department of
Computer Engineering
Bogazici University

Web Ontology Language

Based largely on

Service-Oriented Computing: Semantics, Processes, Agents

– Munindar P. Singh and Michael N. Huhns, Wiley, 2004

Web Ontology Language (OWL)

- RDF captures the basics, i.e., an object-oriented type system
- Additional subtleties of meaning are needed for effective KR
- OWL standardizes additional constructs to show how to capture such subtleties of meaning
- OWL builds on RDF

OWL in Brief

- Specifies classes and properties in a form of description logic (DL)
 - Class operators analogous to Boolean operators *and*, *not*, and *or*
 - Constraints on properties: transitive, ...
 - Restrictions: constructs unique to DL
- Has three species: OWL Full, OWL DL, and OWL Lite

Custom Metadata Vocabularies

- Custom vocabularies for such metadata
- The metadata must be given a standard semantics so that different parties interpret it the same way, and so that tools can function appropriately.

```
<Mammal rdf:ID="Mary"/>
```

```
<Mammal rdf:ID="John">
```

```
  <hasParent rdf:resource="#Mary"/>
```

```
</Mammal>
```

Vocabulary Semantics

- A trivial ontology defining our vocabulary
- Uses simple subclasses and properties
 - Disjointness goes beyond RDF
 - Object properties refine RDF properties; relate two objects

```
<owl:Class rdf:ID="Mammal">  
  <rdfs:subClassOf rdf:resource="#Animal"/>  
  <owl:disjointWith rdf:resource="#Reptile"/>  
</owl:Class>
```

```
<owl:ObjectProperty rdf:ID="hasParent">  
  <rdfs:domain rdf:resource="#Animal"/>  
  <rdfs:range rdf:resource="#Animal"/>  
</owl:ObjectProperty>
```

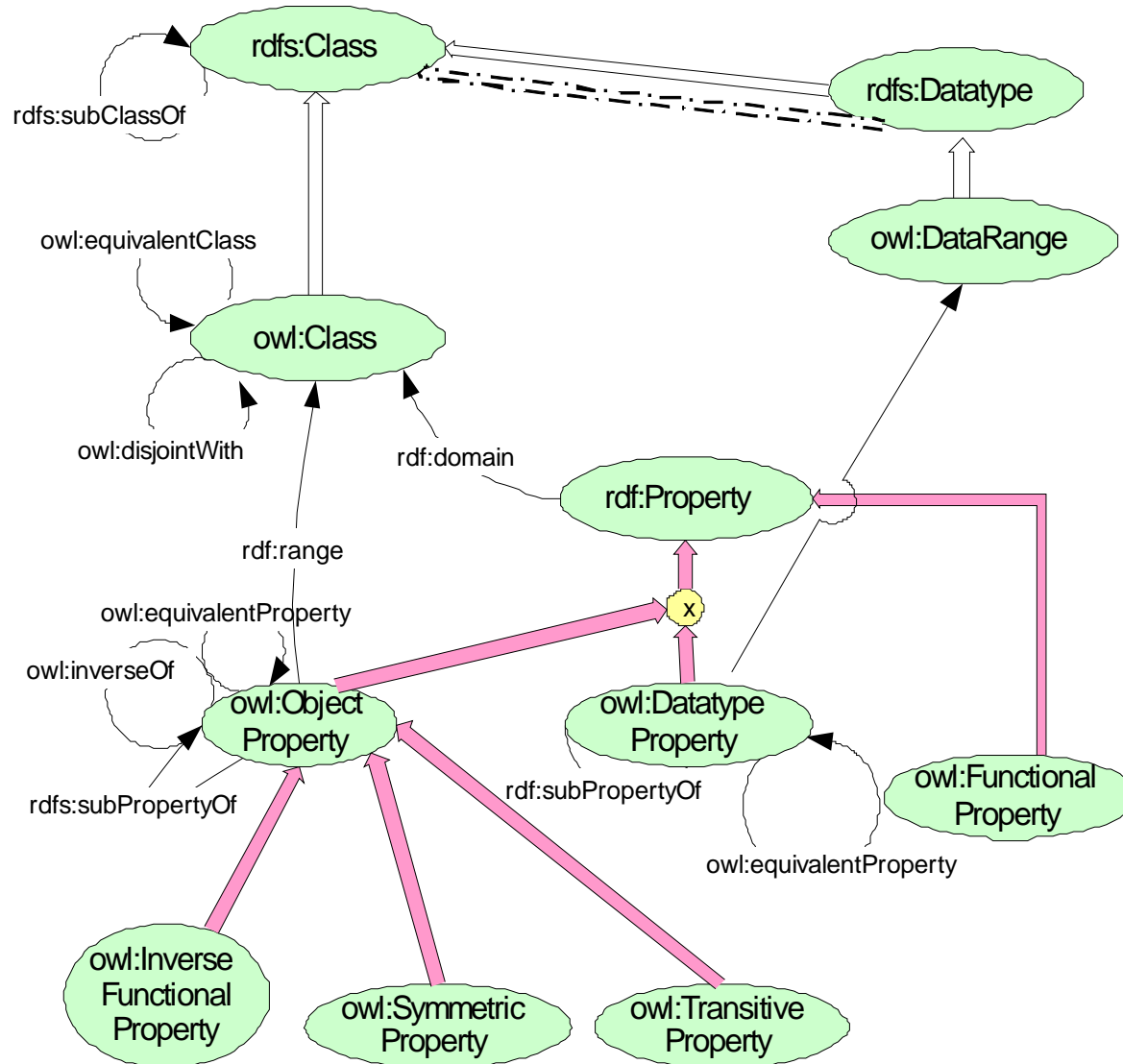
Simple Inference

- Given the definition for the property hasParent and

```
<owl:Thing rdf:ID="Fido">  
  <hasParent rdf:resource="#Rover"/>  
</owl:Thing>
```

we can infer that Fido is an Animal

OWL Entities and Relationships



OWL Object Properties

- Transitive Property
 - $P(x,y)$ and $P(y,z)$ implies $P(x, z)$

```
<owl:ObjectProperty rdf:ID="locatedIn">  
  <rdf:type rdf:resource="&owl;TransitiveProperty" /> <rdfs:domain  
  rdf:resource="&owl;Thing" />  
  <rdfs:range rdf:resource="#Region" />  
</owl:ObjectProperty>  
<Region rdf:ID="SantaCruzMountainsRegion">  
  <locatedIn rdf:resource="#CaliforniaRegion" />  
</Region>  
<Region rdf:ID="CaliforniaRegion">  
  <locatedIn rdf:resource="#USRegion" />  
</Region>
```

OWL Object Properties

- Symmetric Property
 - $P(x,y)$ iff $P(y,x)$

```
<owl:ObjectProperty rdf:ID="adjacentRegion">  
  <rdf:type rdf:resource="&owl;SymmetricProperty" />  
  <rdfs:domain rdf:resource="#Region" />  
  <rdfs:range rdf:resource="#Region" />  
</owl:ObjectProperty>  
<Region rdf:ID="MendocinoRegion">  
  <locatedIn rdf:resource="#CaliforniaRegion" />  
  <adjacentRegion rdf:resource="#SonomaRegion" />  
</Region>
```

OWL Object Properties

- Functional Property
 - $P(x, y)$ and $P(x, z)$ implies $y = z$

```
<owl:Class rdf:ID="VintageYear" />
```

```
<owl:ObjectProperty rdf:ID="hasVintageYear">
```

```
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
```

```
  <rdfs:domain rdf:resource="#Vintage" />
```

```
  <rdfs:range rdf:resource="#VintageYear" />
```

```
</owl:ObjectProperty>
```

OWL Object Properties

- inverseOf
 - $P1(x,y)$ iff $P2(y,x)$

```
<owl:ObjectProperty rdf:ID="hasMaker">  
  <rdf:type rdf:resource="&owl;FunctionalProperty" />  
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID="producesWine">  
  <owl:inverseOf rdf:resource="#hasMaker" />  
</owl:ObjectProperty>
```

OWL Object Properties

- inverseFunctionalProperty
 - $P(y,x)$ and $P(z,x)$ implies $y=z$

```
<owl:ObjectProperty rdf:ID="hasMaker" />
```

```
<owl:ObjectProperty rdf:ID="producesWine">
```

```
  <rdf:type rdf:resource="&owl;InverseFunctionalProperty" />
```

```
  <owl:inverseOf rdf:resource="#hasMaker" />
```

```
</owl:ObjectProperty>
```

Constructing OWL Classes

- Explicitly (as in the examples above)
or
- Anonymously, using
 - Restrictions
 - Set operations

Restrictions: 1

- A unique feature of description logics
- Like division: define classes in terms of a restriction that they satisfy with respect to a given property
- Anonymous: typically included in a class def to enable referring them
- Key primitives are
 - someValuesFrom a specified class
 - allValuesFrom a specified class
 - hasValue equal to a specified individual or data type
 - minCardinality
 - maxCardinality
 - Cardinality (when maxCardinality equals minCardinality)

Restrictions: 2

Examples of restriction fragments

```
<owl:Restriction>
```

```
  <owl:onProperty rdf:resource="#hasFather"/>
```

```
  <owl:maxCardinality rdf:datatype="xsd:nonNegativeInteger"> 1
```

```
  </owl:maxCardinality>
```

```
</owl:Restriction>
```

```
<owl:Restriction>
```

```
  <owl:onProperty rdf:resource='#bakes'/>
```

```
  <owl:someValuesFrom rdf:resource='#Bread'/>
```

```
</owl:Restriction>
```

Restrictions: 3

- ```
<owl:Class rdf:ID="Wine">
 <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
 <rdfs:subClassOf>
 <owl:Restriction>
 <owl:onProperty rdf:resource="#hasMaker" />
 <owl:allValuesFrom rdf:resource="#Winery" />
 </owl:Restriction>
 </rdfs:subClassOf>
 ...
</owl:Class>
```

The maker of a Wine must be a Winery. The `allValuesFrom` restriction is on the `hasMaker` property of this Wine class *only*. Makers of Cheese are not constrained by this local restriction

# Set operators

- intersectionOf, unionOf, complementOf

```
<owl:Class rdf:ID='SugaryBread'>
 <owl:intersectionOf rdf:parseType='Collection'>
 <owl:Class rdf:about='#Bread' />
 <owl:Class rdf:about='#SweetFood' />
 </owl:intersectionOf>
</owl:Class>
```

```
<owl:Class rdf:ID="Fruit">
 <rdfs:subClassOf rdf:resource="#SweetFruit" />
 <rdfs:subClassOf rdf:resource="#NonSweetFruit" />
</owl:Class>
```

- Assertions that are given to be true
- Can be especially powerful in combination with other axioms, which may come from different documents
- Some primitives
  - `rdfs:subClassOf`
  - `owl:equivalentClass`

# Axioms: 2

```
<owl:AllDifferent> <!-- in essence, pair-wise inequalities -->
 <owl:distinctMembers rdf:parseType='Collection'>
 <ex:Country rdf:ID='Russia'/>
 <ex:Country rdf:ID='India'/>
 <ex:Country rdf:ID='USA'/>
 </owl:distinctMembers/>
</owl:AllDifferent>
```

```
<ex:Country rdf:ID='Iran'/>
<ex:Country rdf:ID='Persia'>
 <owl:sameIndividualAs rdf:resource='#Iran'/>
</ex:Country>
```

# Restrictions versus Axioms

---

- Axioms are global assertions that can be used as the basis for further inference
- Restrictions are constructors
  - When we state that hasFather has a maxCardinality of 1, we are
    - Defining the class of animals who have zero or one fathers: this class may or may not have any instances
    - Not stating that all animals have zero or one fathers
- Often, to achieve the desired effect, we would have to combine restrictions with axioms (such as based on equivalentClass)

# Inference

---

- OWL is about content, not the syntax
- Statements from different documents about the same URI are automatically conjoined
- OWL declarations may seem conflicting
  - Declare that no one can have more than one mother
  - Declare Mary is John's mother
  - Declare Jane is John's mother
- A DBMS would declare an integrity violation
- An OWL reasoner would say  $Mary = Jane$

# Dialects Compared

---

- *OWL DL*: the core dialect, includes DL primitives; may not be tractable, decidable
- *OWL Lite*: adds restrictions to OWL DL make it tractable
- *OWL Full*: lifts restrictions to allow other interpretations; extremely general; intractable; undecidable; included just for fancy expressiveness needs

# Resources

- Wine ontology: <http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf>
- Food ontology: <http://www.w3.org/TR/2004/REC-owl-guide-20040210/food.rdf>
- W3C: <http://www.w3.org/2004/OWL/>
- Editor: Protégé <http://protege.stanford.edu/>
- Reasoners:
  - Pellet: <http://www.mindswap.org/2003/pellet/index.shtml>
  - Jena: <http://jena.sourceforge.net/>

# Expressiveness Limitations: 1

---

OWL DL cannot express some simple requirements

- *Non-tree models*: because instance variables are implicit in OWL restrictions, OWL cannot express conditions that require that two variables be identified
  - Think of siblings – two people who have the *same* parents – but in terms of classes
  - Do the same thing with class definitions

# Expressiveness Limitations: 2

---

## Specialized properties

- Cannot state that the child of a mammal must be a mammal and so on without
  - Defining new child properties for each class
  - Adding an axiom for each class stating that it is a subClassOf the restriction of hasChild to itself
- Analogous to the problem in a strongly typed object-oriented language without generics
  - You have to typecast the contents of a hash table or linked list

# Expressiveness Limitations: 3

---

- Constraints among individuals
  - Cannot define tall person: class of persons whose height is above a certain threshold
  - Can define ETHusband: class of persons who have been married to Elizabeth Taylor
- Cannot capture defeasibility (also known as nonmonotonicity)
  - Birds fly
  - Penguins are birds
  - Penguins don't fly

# OWL Summary

---

- OWL builds on RDF to provide a rich vocabulary for capturing knowledge
  - Synthesizes a lot of excellent work on discrete, taxonomic knowledge representation
  - Fits well with describing information resources
    - a basis for describing metadata vocabularies
  - Critical for unambiguously describing services so they can be selected and suitably engaged

# Ontology Management

---

- Descriptions of services are improved through the use of ontologies
  - But how do we make sure the parties involved agree upon and understand the ontologies needed?
- Traditional approach: standardize the ontologies via a formal process
- Emerging approach: be more like the Web; figure out the “correct” ontology via consensus

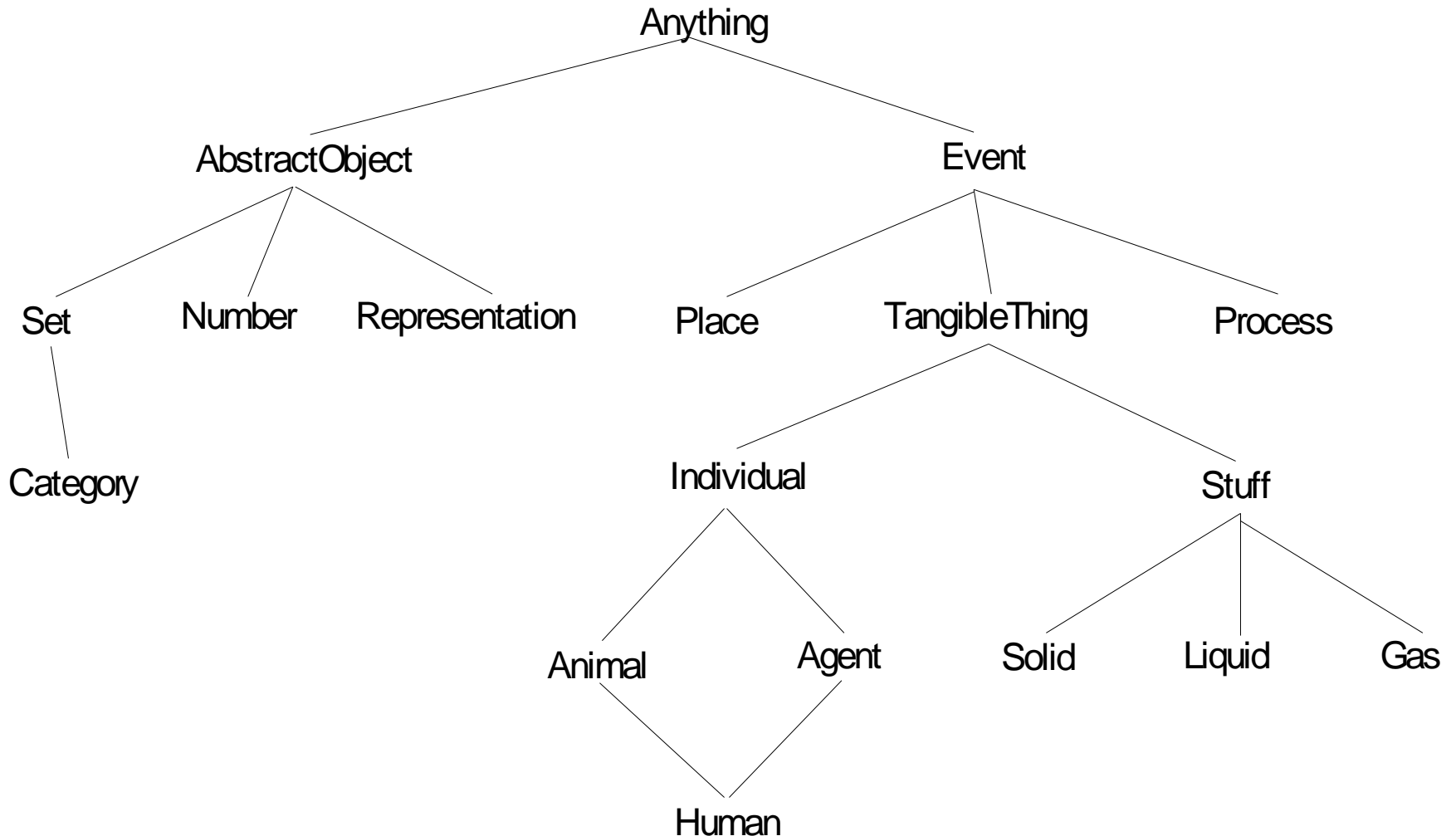
# Standard Ontologies

---

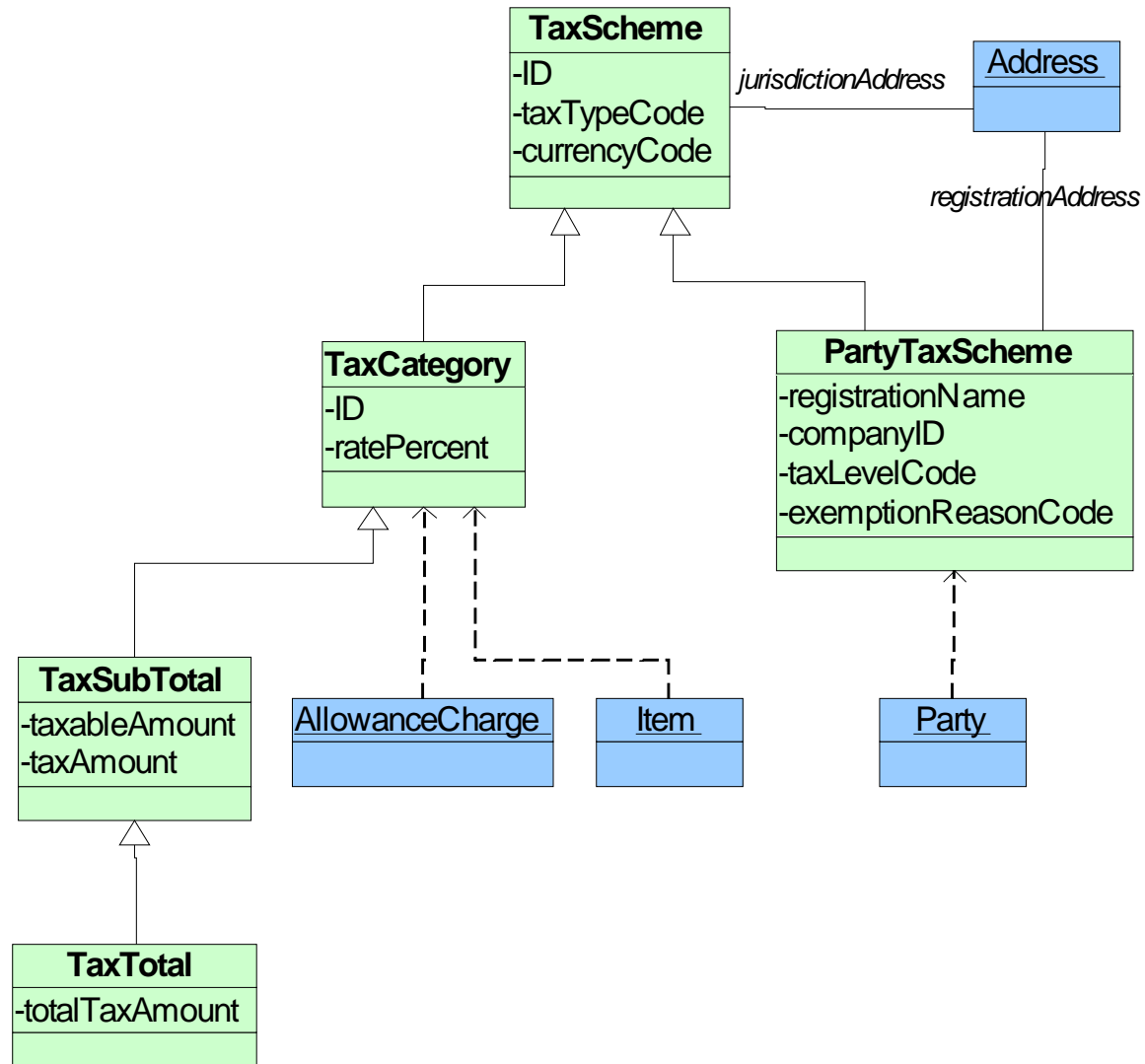
Standardization is more a sociopolitical than a technical process

- IEEE Standard Upper Ontology
- Common Logic (language and upper-level ontology)
- Process Specification Language
- Space and time ontologies
- Domain-specific ontologies, such as health care, taxation, shipping, ...

# An Example Upper Ontology



# OASIS Universal Business Language (UBL)



# Standardization Pros

---

- Where standards exist and are agreed upon, they (even if imperfect)
  - Save time and improve effectiveness
  - Enable specialized tools where appropriate
  - Improve longevity of solution over time and space
  - Suggest directions for improvement

# Standardization Cons

---

- Standardization of domain-specific ontologies is
  - Cumbersome
  - Often out of date by the time completed
  - Difficult to maintain
  - Often violated for competitive reasons

# Standardization Proposal

---

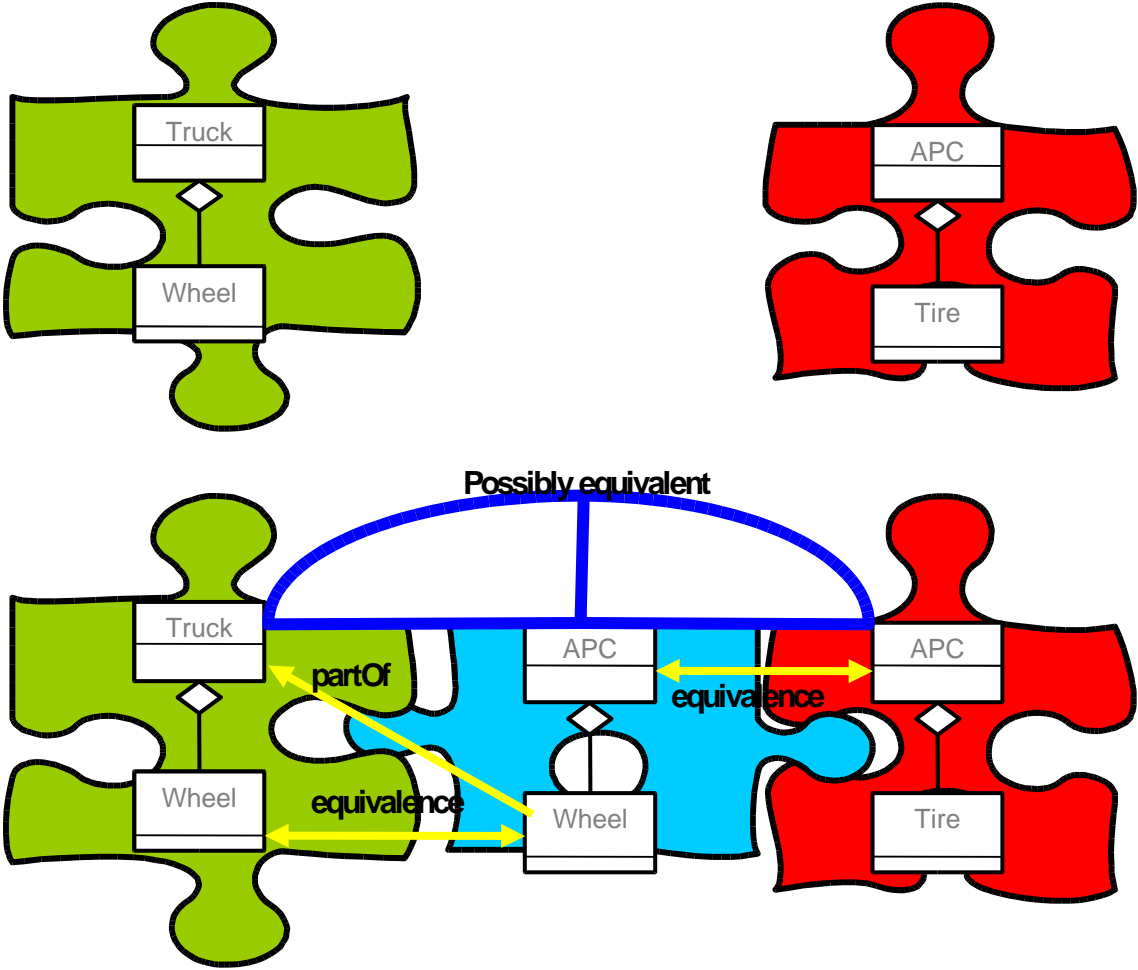
- Always use standard languages (XML, RDF, OWL, ...)
- Take high-level concepts from standard models:
  - Domain experts are not good at KR
  - Lot of work in the best of cases
- Work toward consensus in chosen domain

# Inducing Common Ontologies

---

- Instead of beginning with a standard, develop consensus to induce common ontologies
- Assumptions:
  - No global ontology
  - Individual sources have local ontologies
  - Which are heterogeneous and inconsistent
- Motivation: Exploit richness of variety in ontologies
  - To see where they reinforce each other
  - To make indirect connections (next page)

# Relating Ontologies

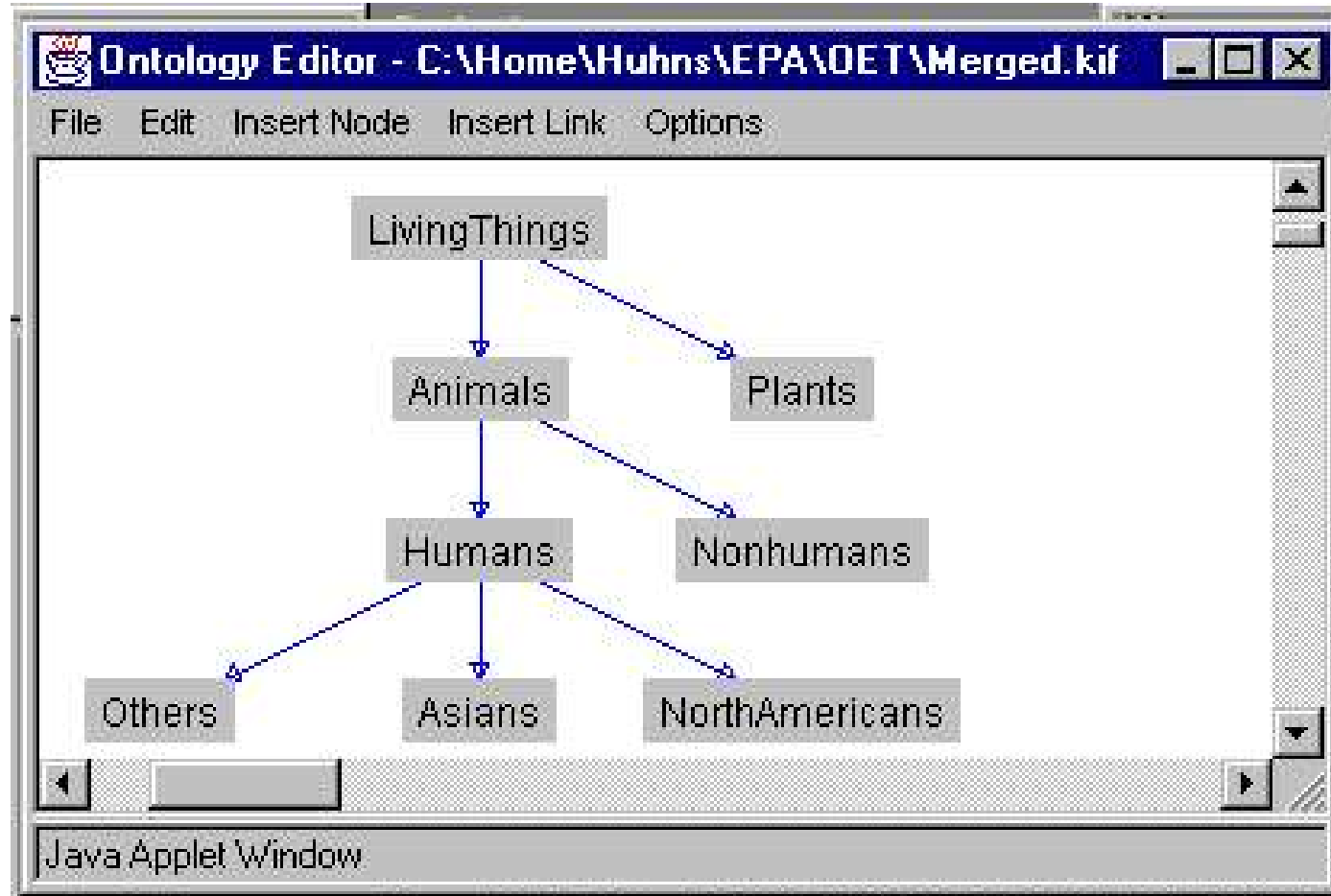


# Relating Ontologies

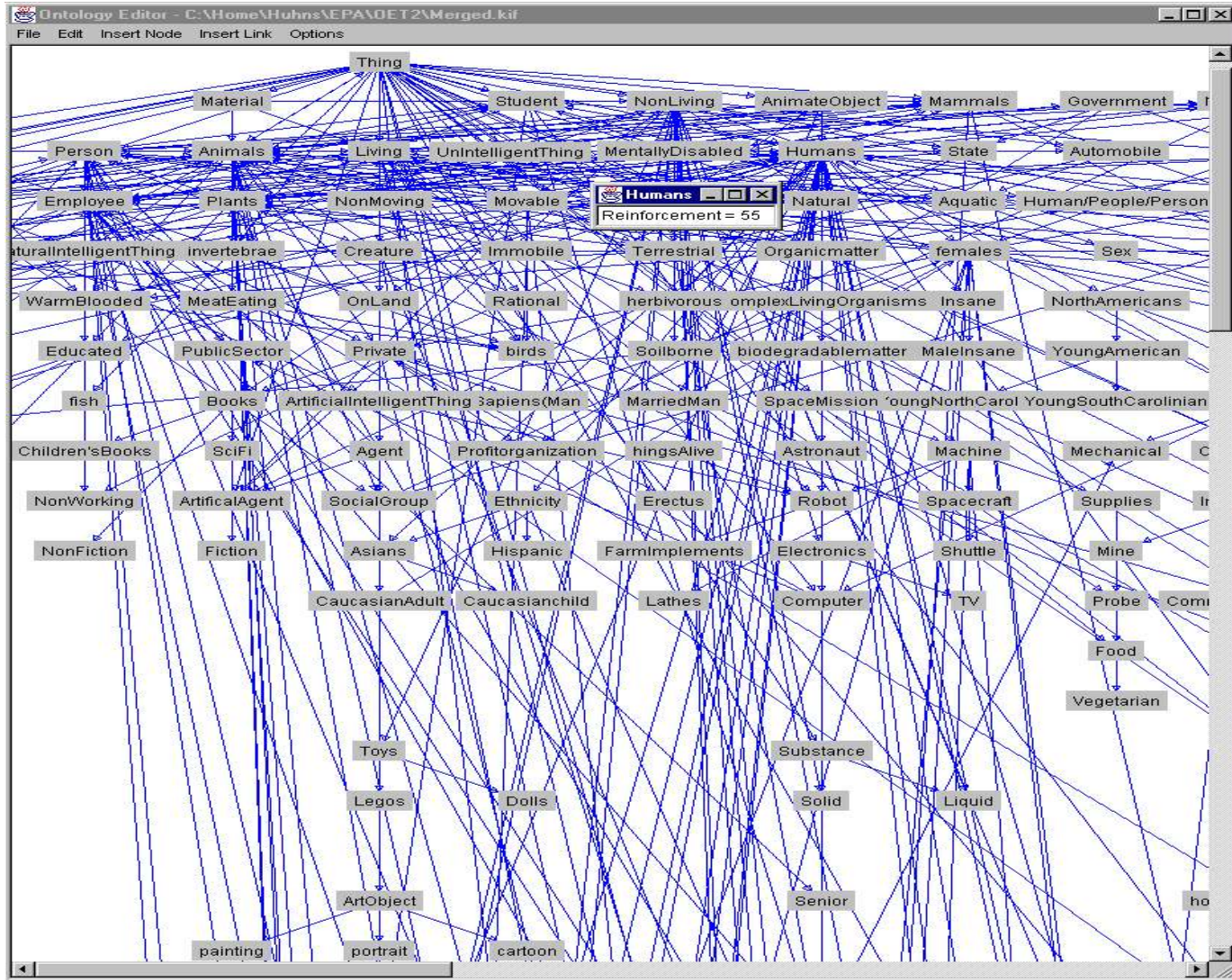
---

- A concept in one ontology can have one of seven mutually exclusive relationships with a concept in another:
  1. subclassOf
  2. superclassOf
  3. partOf
  4. hasPart
  5. siblingOf
  6. equivalentTo
  7. other
- Each ontology adds constraints that can help to determine the most likely relationship

# Initial Experiment: 55 Individual Simple Ontologies about Life



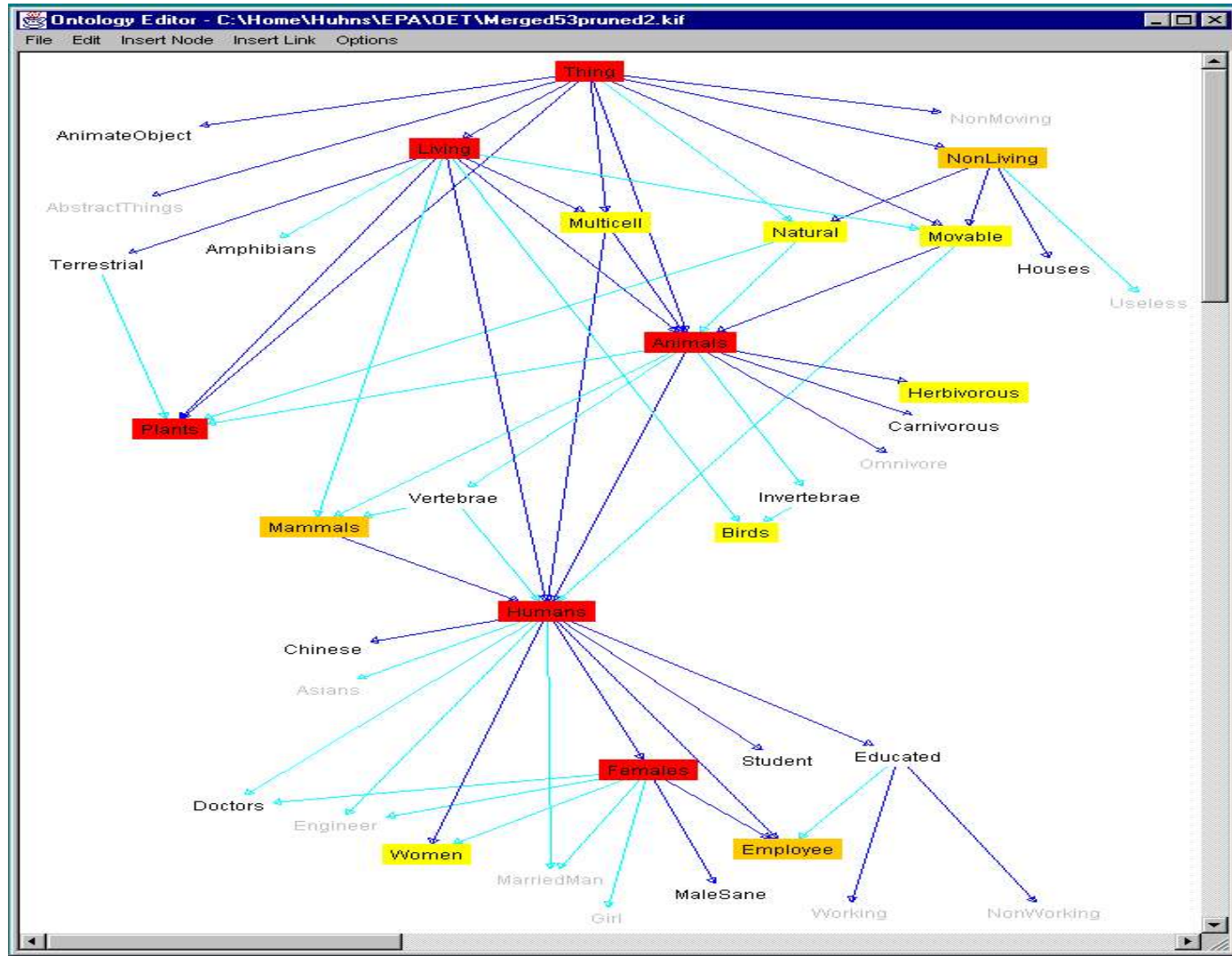
# 55 Merged Ontologies



# Methodology for Merging and Reinforcement

- Merging used smart substring matching and subsumption  
For example, *living*  $\leftrightarrow$  *livingThing*  
However, *living*  $\nleftrightarrow$  *livingRoom*  
because they have disjoint subclasses
  - 864 classes with more than 1500 subclass links were merged into 281 classes related by 554 subclass links
- We retained the classes and subclass links that appeared in more than 5% of the ontologies
  - 281 classes were reduced to 38 classes with 71 subclass links
- We merged concepts that had the same superclass and subclass links
  - Result has 36 classes related by 62 subclass links

# Consensus Ontology for Mutual Understanding



# Consensus Directions

---

- The above approach considered lexical and syntactic bases for similarity
- Other approaches can include
  - Richer dictionaries
  - Richer voting mechanisms
  - Richer forms of structure within ontologies, not just taxonomic structure
  - Models of authority as in the WWW

# Alternative Approaches

---

We may construct large ontologies by

- Inducing classes from large numbers of instances using data-mining techniques
- Building small specialized ontologies and merging them (Ontolingua)
- Top-down construction from first principles (Cyc and IEEE SUO)

# Aside: Categorizing Information

---

Consensus is driven by practical considerations

- Should service providers classify information where it
  - Belongs in the “correct” scientific sense?
  - Where users will look for it?
- Case in point: If most people think a whale is a kind of fish, then should you put information about whales in the fish or in the mammal category?