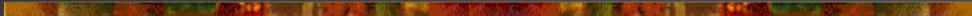


Ontology



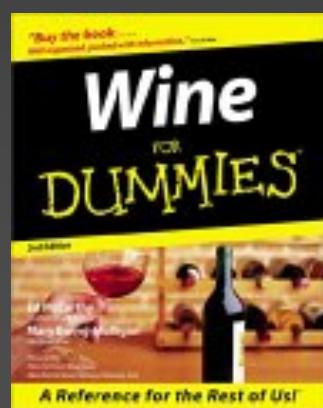
Based on

*“Ontology Development 101: A Guide to Creating Your First Ontology” by
Natalya F. Noy and Deborah L. McGuinness*

http://protege.stanford.edu/publications/ontology_development/ontology101.html



A shared **ONTOLOGY** of **wine and food**



Outline

- What is an ontology?
 - Why develop an ontology?
 - Step-By-Step: Developing an ontology
 - Going deeper: Common problems and solutions
 - Ontologies in the Semantic Web languages
 - Current research issues in ontology engineering
-

What Is An Ontology

- An ontology is an explicit description of a domain:
 - concepts
 - properties and attributes of concepts
 - constraints on properties and attributes
 - Individuals (*often, but not always*)
- An ontology defines
 - a common vocabulary
 - a shared understanding

Ontology Examples

- Taxonomies on the Web
 - Yahoo! categories
- Catalogs for on-line shopping
 - Amazon.com product catalog
- Domain-specific standard terminology
 - Unified Medical Language System (UMLS)
 - UNSPSC - terminology for products and services

Example Ontologies

see <http://www.cs.utexas.edu/users/mfkb/related.html>

- * ARPI Planning and Scheduling ontologies
- * Aviation Ontology
- * BPMO - The Business Processes Management Ontology
- * CYC (and the derivative PDKB)
- * DOLCE- a Descriptive Ontology for Linguistic and Cognitive Engineering .
- * Dublin Core (bibliographic organization)
- * The Enterprise Ontology (for business enterprises)
- * Ontologies for ethology (animal behavior) , e.g. Loggerhead Turtle
- * FrameNet (lexical reference)
- * Generalized Upper Model (for NLP)
- * Mikrokosmos (for NLP)
- * ON9 (the CNR-ITBM Ontology Library)
- * OWL-S - The OWL(formerly DAML) Services ontology.
- * Ontolingua Ontology Library
- * OpenMind database and OM CSNet Semantic Network
- * PharmGKB - Pharmacogenetics and Pharmacogenomic Knowledge Base
- * PSL (process specification)
- * QoS (computers and networks)
- * SENSUS(for NLP)
- * STEP (for product data exchange)
- * SUMO (the Suggested Upper Merged Ontology)
- * the Twente Ontology Collection
- * UMLS (biomedicine)
- * Wilkins' ontology (17th century!)
- * WordNet (lexical reference)

Example Tools for Ontologies

see <http://www.cs.utexas.edu/users/mfkb/related.html>,

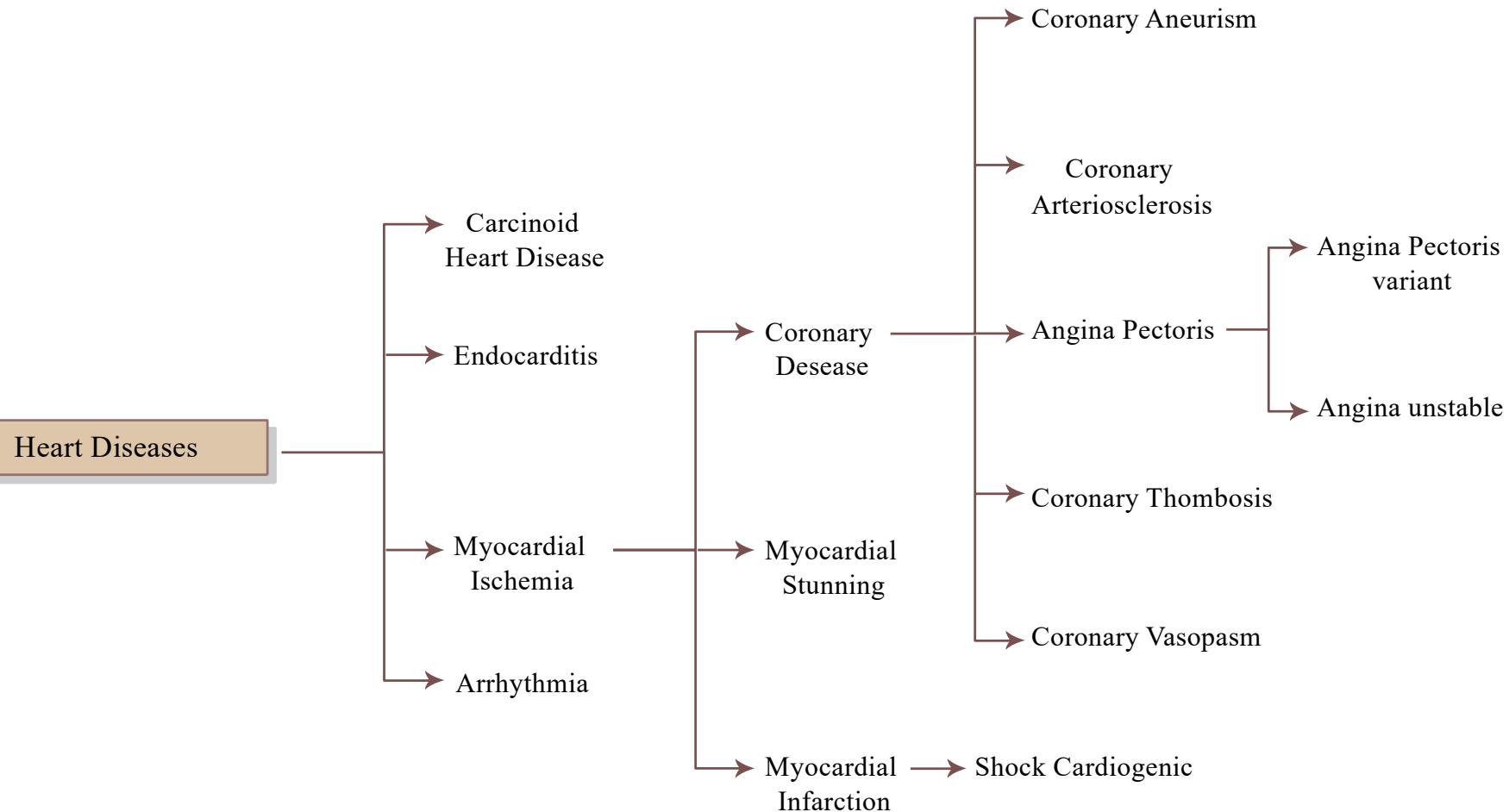
<http://www.xml.com/pub/a/2002/11/06/ontologies.html>

- * Chimaera
- * CODE4
- * Generic Knowledge-Base Editor
- * Ika rus
- * JOE (Java Ontology Editor)
- * KAON
- * KACTUS
- * OilEd
- * OntoEdit
- * Ontosaurus
- * Protege
- * Snobase
- * Stanford Ontology Editor
- * SymOntos
- * Word Map

Some Large Ontologies

CYC commonsense	CYCL	PartiallyOnline: 6000Top Concepts
SUMO Upper ontology	KIF AlsoLOOM, OWL,Protege	PublishedOnline
WordNet Lexical memory	Semantic Network	PublishedOnline
Sensus Text understanding	Semantic Network	PublishedOnline
UMLS Bio medicine	Semantic Network	PublishedOnline

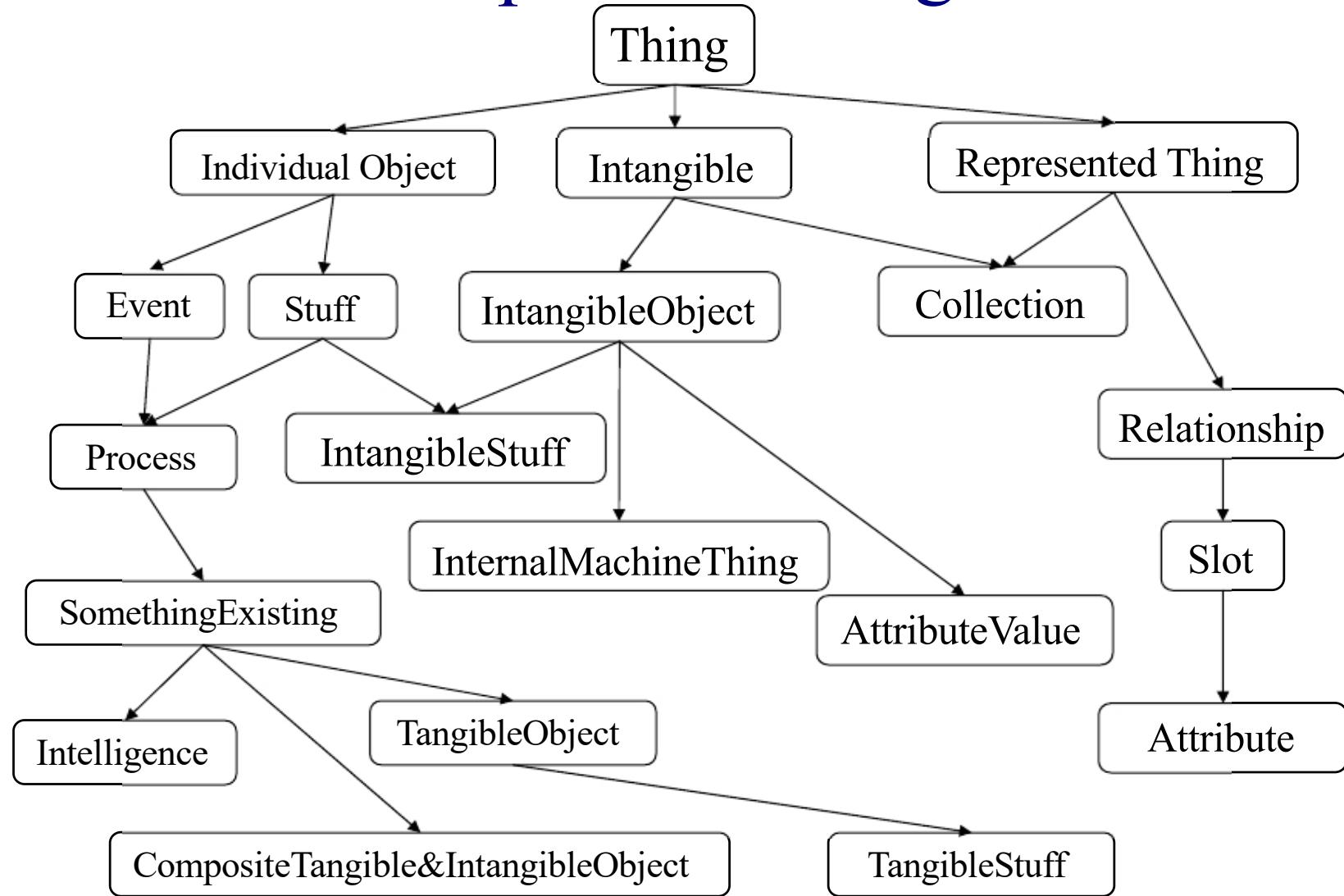
UMLS



CYC

- Goal: Encode all of human common sense knowledge
- Mechanization: human-entered axioms
- Periodic review, reorganization, compaction, separation into distinct mini-theories, not mutually consistent
- Driven by application domains
- Often seems ad-hoc

CYC Top Level Categories



CYC Ontology

- [Alphabetized List of CYC® Constants](#)
- [Search for Constants by Name](#)
- The Entries for the CYC® Constants (divided up among 43 pages)
 - [Fundamentals](#)
 - [Top Level](#)
 - [Time and Dates](#)
 - [Types of Predicates](#)
 - [Spatial Relations](#)
 - [Quantities](#)
 - [Mathematics](#)
 - [Contexts](#)
 - [Groups](#)
 - ["Doing"](#)
 - [Transformations](#)
 - [Changes Of State](#)
 - [Transfer Of Possession](#)
 - [Movement](#)
 - [Parts of Objects](#)
 - [Composition of Substances](#)
 - [Agents](#)
 - [Organizations](#)
 - [Actors](#)
 - [Roles](#)
 - [Professions](#)
 - [Emotion](#)
 - [Propositional Attitudes](#)
 - [Social](#)
 - [Biology](#)
 - [Chemistry](#)
 - [Physiology](#)
 - [General Medicine](#)
 - [Materials](#)
 - [Waves](#)

CYC Examples

[#\\$Professional](#)

A set of agents. Elements of [#\\$Professional](#) are agents who spend a significant part of their waking hours doing activities that are characteristic of some occupation, skilled or unskilled. However, elements of [#\\$Professional](#) need not be working the entire duration of when they are a professional, such as a [#\\$Professor](#) on summer break, or someone who is temporarily unemployed. The elements of [#\\$Professional](#) are persons, most of whom belong to at least one such collection during some portion of their lives. Typically their actions are performed for pay, but not always (e.g., [#\\$Artist-Visual](#)). What are colloquially considered professions or occupations are subsets of [#\\$Professional](#); for example, [#\\$LumberJack](#), [#\\$Scientist](#), [#\\$Lifeguard](#), [#\\$StockBroker](#), [#\\$Technician](#), [#\\$CraftWorker](#), [#\\$Housekeeper](#), [#\\$SportsCoach](#), [#\\$Athlete](#), [#\\$LegalProfessional](#), [#\\$Publicist](#), [#\\$CrewMemberOnShip](#), [#\\$SelfEmployedWorker](#) (and many more). Additionally, other subsets of [#\\$Professional](#) classify workers according to other features of their working life besides skills; e.g., [#\\$SelfEmployedWorker](#), [#\\$DeskWorker](#). Subsets may be general (e.g., [#\\$DeskWorker](#), [#\\$Doctor-Medical](#)) or specialized (e.g., [#\\$ContinuingEdProgramCoordinator](#), [#\\$PediatricNeuroSurgeon](#)). Elements of [#\\$Professional](#) are people: [#\\$MaryShepherd](#) ([#\\$HumanCyclist](#)), [#\\$MichaelJordan](#) ([#\\$Athlete](#)), [#\\$AlfredNorthWhitehead](#) ([#\\$Philosopher](#)), [#\\$Michelangelo](#) ([#\\$Sculptor](#), [#\\$PainterFineArtist](#)), [#\\$BillClinton](#) ([#\\$UnitedStatesPresident](#)), etc.

[isa: #\\$ExistingObjectType](#)

[genls: #\\$Person](#)

[some subsets: #\\$Researcher](#) [#\\$Athlete](#) [#\\$Executive](#) [#\\$MilitaryPerson](#) [#\\$Employee](#) [#\\$DeskWorker](#) [#\\$PublicSectorEmployee](#) [#\\$Consultant](#) [#\\$AcademicProfessional](#) [#\\$NonProfitEmployee](#) [#\\$SalesPerson](#) [#\\$EntertainmentOrArtsProfessional](#) [#\\$PrivateSectorEmployee](#) [#\\$Farmer](#) [#\\$SelfEmployedWorker](#) (plus 31 more public subsets, 596 unpublished subsets)

[#\\$OccupationType](#)

A collection of collections. Each element of [#\\$OccupationType](#) is a collection of workers, based on their kinds of work; each of those workers is an element of ([#\\$isa](#)) [#\\$Professional](#). Elements of [#\\$OccupationType](#) represent all kinds of jobs, not just the kinds of occupations colloquially considered "professional". Elements of [#\\$OccupationType](#) include the collections [#\\$ComputerProgrammerProfessional](#), [#\\$FoodServiceEmployee](#), [#\\$MedicalCareProfessional](#), [#\\$BaseballUmpire](#), [#\\$SalesRepresentative](#), [#\\$Brewer](#), [#\\$Gymnast](#), and many others. Also see [#\\$Professional](#), [#\\$PositionType](#).

[isa: #\\$SlicingDisjointCollection](#) [#\\$Collection](#)

[genls: #\\$PositionType](#) [#\\$PersonByActivityType](#)

[some subsets: #\\$MedicalSpecialtyType](#)

CYC Examples (cont'd)

[#Employees : <#Agent> <#Agent>](#)

The predicate [#Employees](#) relates a particular employer to one of its paid employees. ([#Employees EMPLOYER WORKER](#)) means WORKER regularly performs work for EMPLOYER, and EMPLOYER pays WORKER for that activity (often by paycheck). EMPLOYER directs the manner in which WORKER performs the work and may provide the workplace, tools, capital, and other assistance for the work. EMPLOYER is commonly an organization but may be a person. E.g., ([#Employees PerryMason PaulDrake](#)); ([#Employees \\$\\$Cycorp \\$\\$Lenat](#)). This predicate is true during all or any part of the period that the employment continues; e.g., ([#\\$holdsIn \(#\\$YearFn 1995\) \(#Employees \\$\\$CarnegieMellonUniversity \\$\\$DeRidder\)](#)).

isa: [#\\$AsymmetricBinaryPredicate \\$\\$CotemporalObjectsSlot](#)

genlPreds: [#\\$hasWorkers #\\$affiliatedWith #\\$cotemporal](#)

[#\\$hasTitle : <#Person> <#Title> <#Organization>](#)

The predicate [#\\$hasTitle](#) relates a person to a title that s/he holds in an organization. ([#\\$hasTitle PER TITLE ORG](#)) means that the [#Person](#) PER has the [#\\$Title](#) TITLE in the [#\\$Organization](#) ORG. Elements of [#\\$Title](#) are linguistic objects usually related to positions or other qualifications that a person has. A person generally has a title only while actually holding the related position; e.g., ([#\\$hasTitle \\$\\$Lenat \\$\\$PrincipalScientist-Title \\$\\$CycGroup](#)) tells us Doug Lenat's title at the Cyc Project while under MCC's organizational structure. A noteworthy class of exceptions is [#\\$CourtesyTitle](#) (q.v.), which include forms of address such as 'Mr.' and 'Ms.', plus some titles which by courtesy the holders retain for life, such as (in the U.S.) 'President' and military rank designations (e.g., officers retired from the armed services).

Note: Elements of [#\\$Title](#) belong to the set [#\\$LinguisticObject](#), while positions themselves are represented by persons (cf. [#\\$PositionType](#)). Cf. [#\\$hasPositionIn](#).

isa: [#\\$TernaryPredicate](#)

[#\\$insIsJobOf : <#\\$ScriptType> <#\\$Professional>](#)

The predicate [#\\$insIsJobOf](#) indicates a type of work done by a particular individual. ([#\\$insIsJobOf SCRIPT-TYPE PER](#)) means that the person PER performs instances of SCRIPT-TYPE as part of his or her job. E.g., [#\\$KeithRichards](#) performs instances of [#\\$WritingMusic](#) as part of his work; [#\\$Goolsbey](#) performs instances of [#\\$ProgrammingAComputer](#) in his job at Cycorp; a [#\\$SecurityGuard](#) performs instances of [#\\$ProtectingSomething](#). Note that assertions using [#\\$insIsJobOf](#) are true for some specific period of time, which may be indicated with [#\\$holdsIn](#).

isa: [#\\$TypePredicate #\\$BinaryPredicate](#)

Aspects of an Ontology

- Content
- Form
- Purpose
- Development

Aspects of an Ontology

- Content
 - types of objects, relationships
 - e.g. the blocks world conceptualization includes:
 - Object Classes: Blocks, Robot Hands
 - Properties: shapes of blocks, color of blocks
 - Relationships: On, Above, Below, Grasp
 - Processes: stacking plan for a tower
- Form
- Purpose
- Development

Aspects of an Ontology

- Content
- Form
 - Is the taxonomic relationship (instance-of, subclass) primary?
 - Are definitions of, or constraints on, terms provided?
 - Is the definitional language as rich as a full logic?
 - Is it process-centric or object-centric?
- Purpose
- Development

Aspects of an Ontology

- Content
- Form
- Purpose
 - Knowledge sharing
 - E.g. Between people, software systems, agents
 - Knowledge reuse
 - E.g. When models or systems change
 - General (common sense) or domain specific
- Development

Aspects of an Ontology

- Content
- Form
- Purpose
- Development
 - Is it acquired or engineered?
 - If acquired, what about:
 - Quality of knowledge
 - Diversity of content
 - Trust in knowledge
 - Unpredictable use

Ontology vs KB?

- Can think of an ontology as a kind of KB, but:
- **Ontology** serves different purpose:
 - Only needs to describe vocabulary, axioms
 - E.g. database schema is an ontology
- **KB** includes:
 - Specific knowledge needed for problem-solving

What Is “Ontology Engineering”?

Ontology Engineering: Defining terms in the domain and relations among them

- Defining concepts in the domain (**classes**)
 - Arranging the concepts in a hierarchy (**subclass-superclass hierarchy**)
 - Defining which attributes and **properties (slots)** classes can have and constraints on their values
 - Defining **individuals** and filling in slot values
-

Outline

- What is an ontology?
 - Why develop an ontology?
 - Step-By-Step: Developing an ontology
 - Going deeper: Common problems and solutions
 - Ontologies in the Semantic Web languages
 - Current research issues in ontology engineering
-

Why Develop an Ontology?

- To share **common understanding** of the structure of information
 - among people
 - among software agents
 - To enable **reuse** of domain knowledge
 - to avoid “re-inventing the wheel”
 - to introduce standards to allow interoperability
-

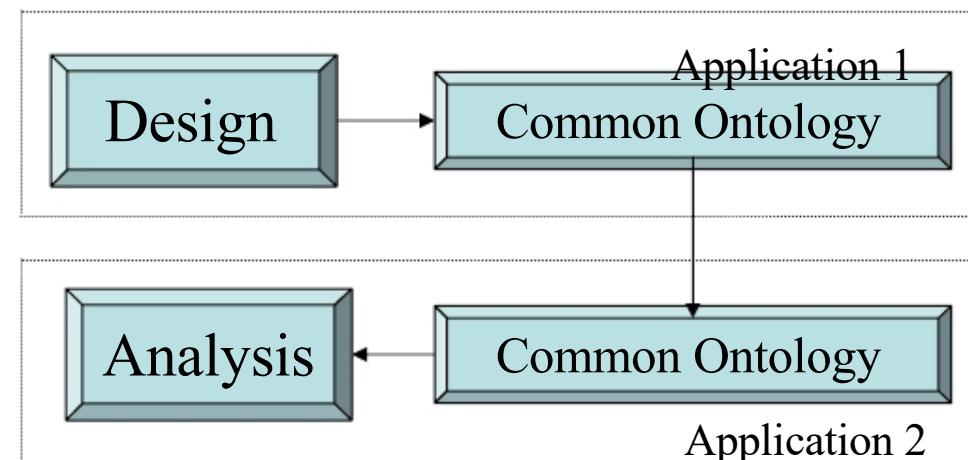
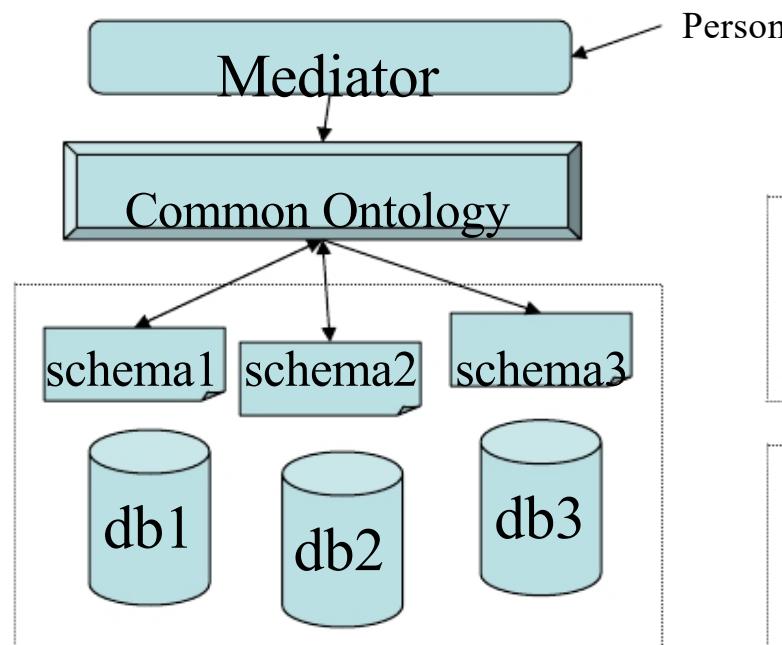
Motivations

- Engineering motivation:
 - Every knowledge-based system is based on an ontology of its domain
 - Explication of the ontology is a time-consuming component of the development process
 - Why not amortize the effort and share ontologies?
 - E.g. “core ontologies” such as space, time, quantities
- Scientific motivation:
 - Understand fundamental issues about human conceptualizations

Pragmatic Motivations

- Responding to the unexpected
- Distributed Databases
- Distributed Applications
- Communicating Agents
- Semantic Web

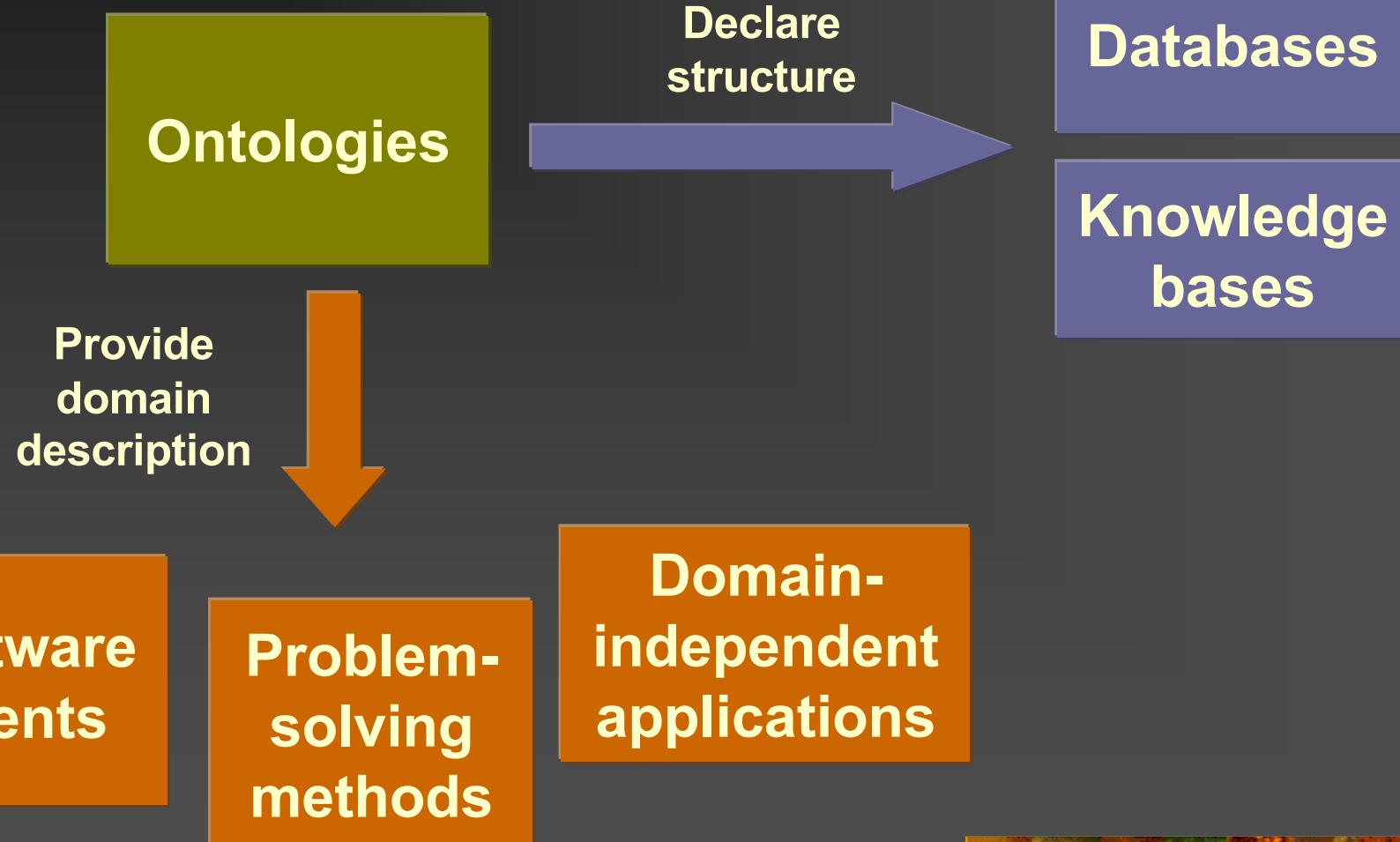
Key Question: What does he mean when he says <...>?



More Reasons

- To make domain assumptions **explicit**
 - easier to change domain assumptions
(consider a genetics knowledge base)
 - easier to understand and update legacy data
 - To **separate** domain knowledge from the operational knowledge
 - re-use domain and operational knowledge separately (e.g., configuration based on constraints)
-

An Ontology Is Often Just the Beginning



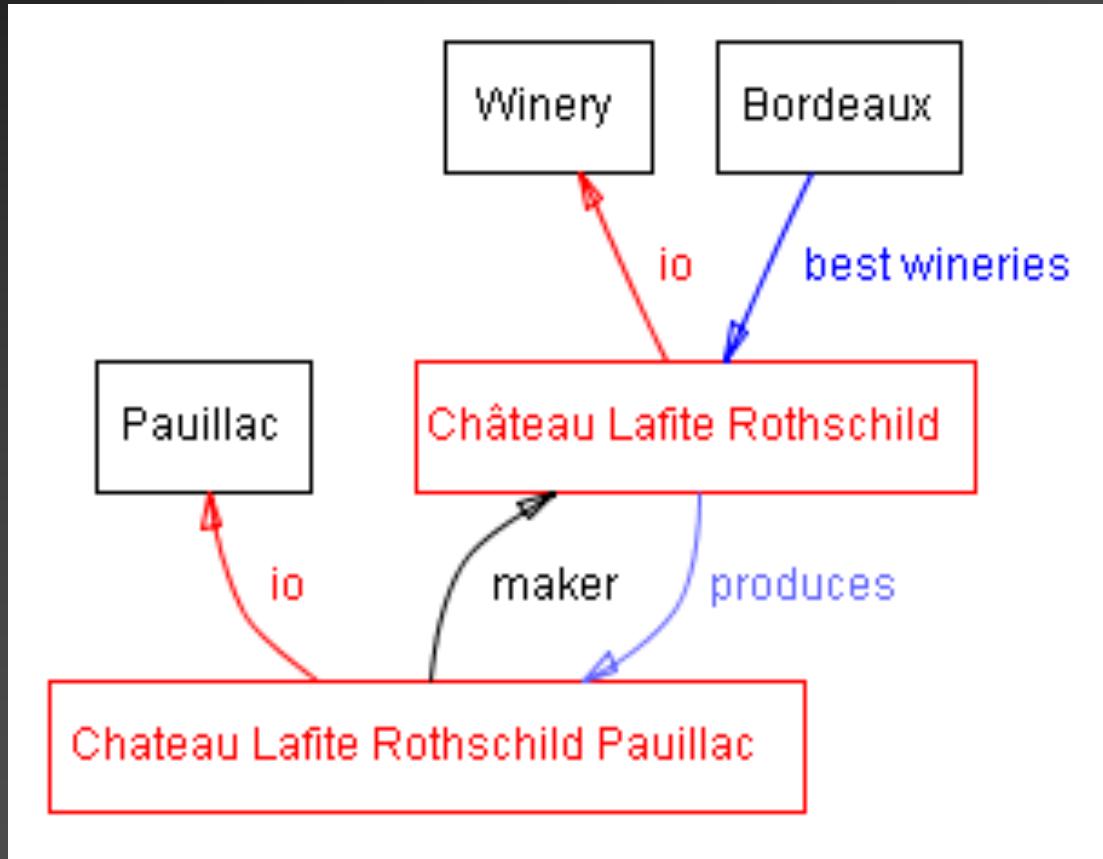
Outline

- What is an ontology?
- Why develop an ontology?
- Step-By-Step: Developing an ontology
- Going deeper: Common problems and solutions
- Ontologies in the Semantic Web languages
- Current research issues in ontology engineering

Building an Ontology

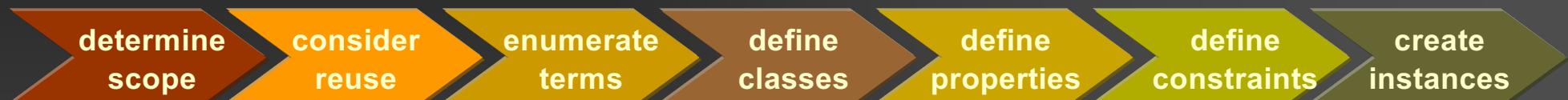
- Planning
- Specification - consider scope and purpose
- Knowledge Acquisition
- Conceptualization - glossary of terms, top-down, bottom-up, middle-out
- Integration - of existing relevant ontologies
- Implementation
- Evaluation - Clarity, Coherence, Extensibility, Minimal Encoding Bias, Minimal Ontological Commitment

Wines and Wineries

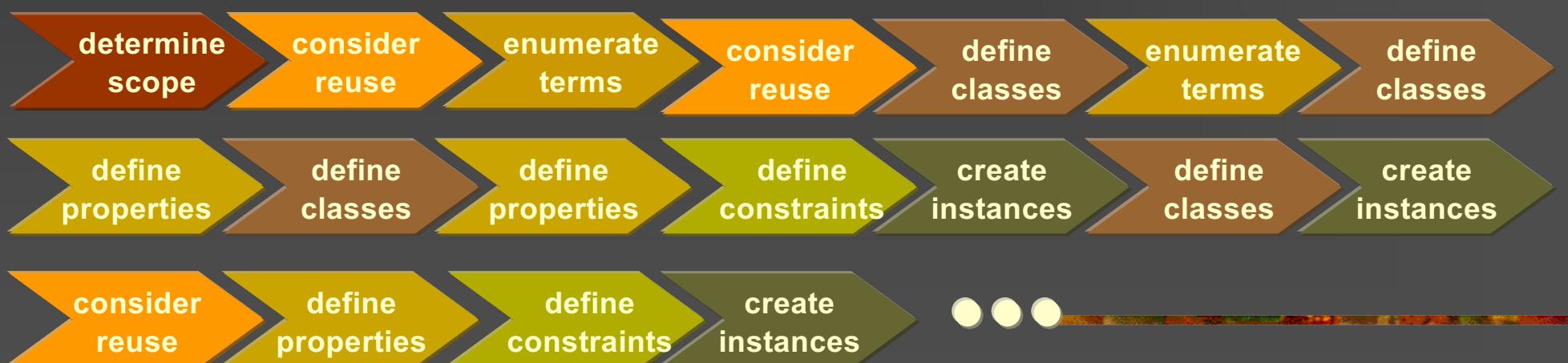


Ontology-Development Process

In this tutorial:



In reality - an iterative process:



Ontology Engineering versus Object-Oriented Modeling



An ontology

- reflects the structure of the world
- is often about **structure** of concepts
- actual physical representation is not an issue

An OO class structure

- reflects the structure of the data and code
 - is usually about **behavior** (methods)
 - describes the physical representation of data (long int, char, etc.)
- 

Preliminaries - Tools

- All screenshots in this tutorial are from Protégé-2000, which:
 - is a graphical ontology-development tool
 - supports a rich knowledge model
 - is open-source and freely available (<http://protege.stanford.edu>)
- Some other available tools:
 - Ontolingua and Chimaera
 - OntoEdit
 - OilEd

Determine Domain and Scope



- What is the domain that the ontology will cover?
- For what we are going to use the ontology?
- For what types of questions the information in the ontology should provide answers (competency questions)?

Answers to these questions may change during the lifecycle

Competency Questions

- Which wine characteristics should I consider when choosing a wine?
 - Is Bordeaux a red or white wine?
 - Does Cabernet Sauvignon go well with seafood?
 - What is the best choice of wine for grilled meat?
 - Which characteristics of a wine affect its appropriateness for a dish?
 - Does a flavor or body of a specific wine change with vintage year?
 - What were good vintages for Napa Zinfandel?
-

Consider Reuse



- Why reuse other ontologies?
 - to save the effort
 - to interact with the tools that use other ontologies
 - to use ontologies that have been validated through use in applications

What to Reuse?

- Ontology libraries
 - DAML ontology library (www.daml.org/ontologies)
 - Ontolingua ontology library
(www.ksl.stanford.edu/software/ontolingua/)
 - Protégé ontology library
(protege.stanford.edu/plugins.html)
 - Upper ontologies
 - IEEE Standard Upper Ontology (suo.ieee.org)
 - Cyc (www.cyc.com)
-

What to Reuse? (II)

- General ontologies
 - DMOZ (www.dmoz.org)
 - WordNet (www.cogsci.princeton.edu/~wn/)
- Domain-specific ontologies
 - UMLS Semantic Net
 - GO (Gene Ontology) (www.geneontology.org)

Enumerate Important Terms



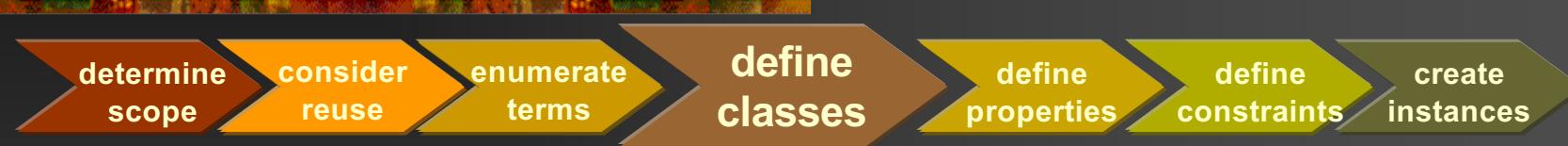
- What are the terms we need to talk about?
- What are the properties of these terms?
- What do we want to say about the terms?

Enumerating Terms - The Wine Ontology

*wine, grape, winery, location,
wine color, wine body, wine flavor, sugar
content*

*white wine, red wine, Bordeaux wine
food, seafood, fish, meat, vegetables,
cheese*

Define Classes and the Class Hierarchy



- A class is a **concept** in the domain
 - a class of wines
 - a class of wineries
 - a class of red wines
- A class is a **collection** of elements with similar properties
- **Instances** of classes
 - a glass of California wine you'll have for lunch

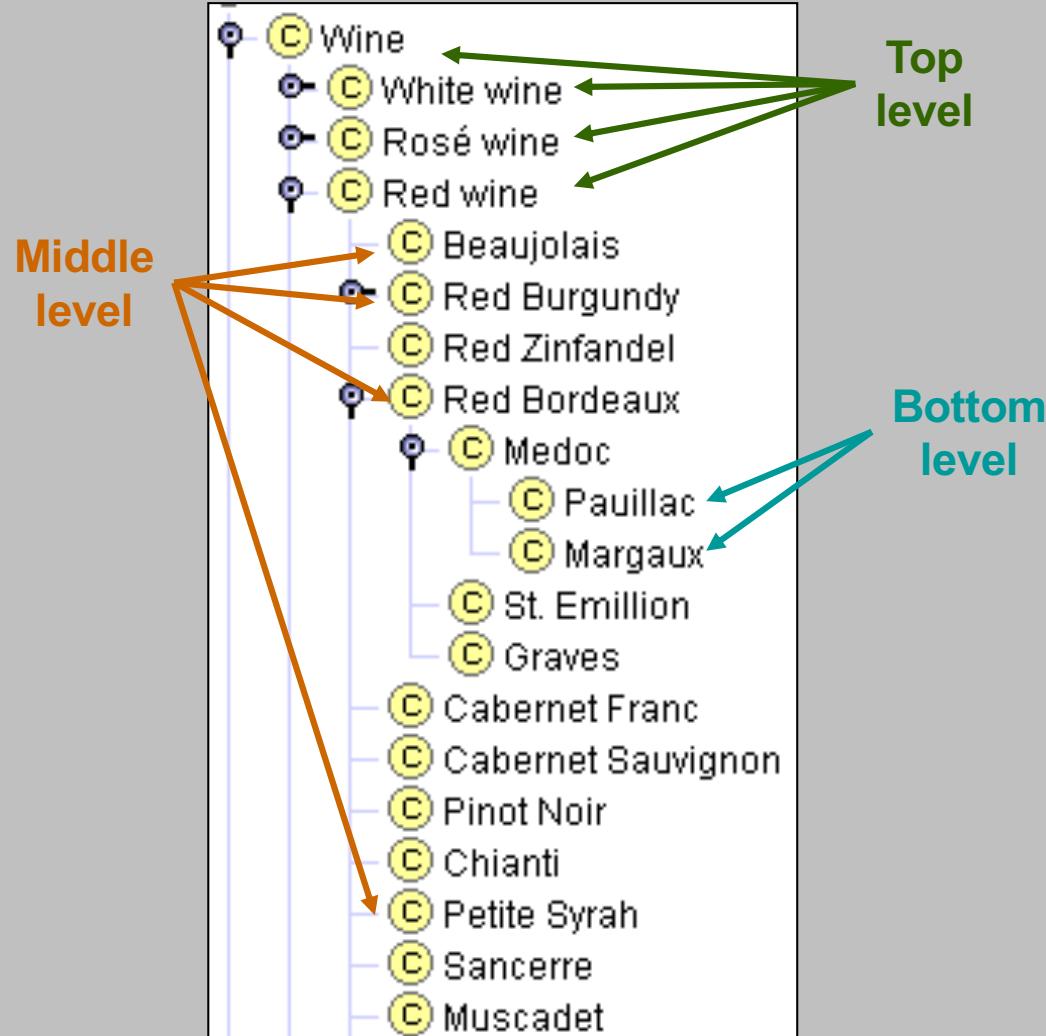
Class Inheritance

- Classes usually constitute a **taxonomic hierarchy** (a subclass-superclass hierarchy)
- A class hierarchy is usually an **IS-A** hierarchy:
an instance of a subclass is an instance of a superclass
- If you think of a class as a **set** of elements, a subclass is a **subset**

Class Inheritance - Example

- Apple is a subclass of Fruit
Every apple is a fruit
- Red wines is a subclass of Wine
Every red wine is a wine
- Chianti wine is a subclass of Red wine
Every Chianti wine is a red wine

Levels in the Hierarchy



Modes of Development

- top-down – define the most general concepts first and then specialize them
 - bottom-up – define the most specific concepts and then organize them in more general classes
 - combination – define the more salient concepts first and then generalize and specialize them
-

Documentation

- Classes (and slots) usually have documentation
 - Describing the class in natural language
 - Listing domain assumptions relevant to the class definition
 - Listing synonyms
 - Documenting classes and slots is as important as documenting computer code!
-

Define Properties of Classes – Slots



- Slots in a class definition describe attributes of instances of the class and relations to other instances

Each wine will have color, sugar content, producer, etc.

Properties (Slots)

- Types of properties
 - “intrinsic” properties: flavor and color of wine
 - “extrinsic” properties: name and price of wine
 - parts: ingredients in a dish
 - relations to other objects: producer of wine (winery)
- Simple and complex properties
 - simple properties (attributes): contain primitive values (strings, numbers)
 - complex properties: contain (or point to) other objects (e.g., a winery instance)

Slots for the Class Wine

Template Slots				▼	▼	C	X	+	-
Name	Type	Cardinality		Other Facets					
S body	Symbol	single		allowed-values={FULL,MEDIUM,LIGHT}					
S color	Symbol	single		allowed-values={RED,ROSÉ,WHITE}					
S flavor	Symbol	single		allowed-values={DELICATE,MODERATE,STRONG}					
S grape	Instance	multiple		classes={Wine grape}					
S maker <small>I</small>	Instance	single		classes={Winery}					
S name	String	single							
S sugar	Symbol	single		allowed-values={DRY,SWEET,OFF-DRY}					

(in Protégé-2000)

Slot and Class Inheritance

- A subclass **inherits** all the slots from the superclass
If a wine has a name and flavor, a red wine also has a name and flavor
- If a class has **multiple** superclasses, it inherits slots from all of them
Port is both a dessert wine and a red wine. It inherits “sugar content: high” from the former and “color:red” from the latter

Property Constraints



- Property constraints (**facets**) describe or limit the set of possible values for a slot

The name of a wine is a string

The wine producer is an instance of Winery

A winery has exactly one location

Facets for Slots at the Wine Class

Template Slots				▼	▼	C	X	+	-
Name	Type	Cardinality	Other Facets						
S body	Symbol	single	allowed-values={FULL,MEDIUM,LIGHT}						
S color	Symbol	single	allowed-values={RED,ROSÉ,WHITE}						
S flavor	Symbol	single	allowed-values={DELICATE,MODERATE,STRONG}						
S grape	Instance	multiple	classes={Wine grape}						
S maker <small>I</small>	Instance	single	classes={Winery}						
S name	String	single							
S sugar	Symbol	single	allowed-values={DRY,SWEET,OFF-DRY}						

Common Facets

- Slot **cardinality** – the number of values a slot has
 - Slot **value type** – the type of values a slot has
 - **Minimum and maximum value** – a range of values for a numeric slot
 - **Default value** – the value a slot has unless explicitly specified otherwise
-

Common Facets: Slot Cardinality

- **Cardinality**
 - Cardinality N means that the slot **must** have N values
 - **Minimum cardinality**
 - Minimum cardinality 1 means that the slot must have a value (**required**)
 - Minimum cardinality 0 means that the slot value is **optional**
 - **Maximum cardinality**
 - Maximum cardinality 1 means that the slot can have at most one value (**single-valued slot**)
 - Maximum cardinality greater than 1 means that the slot can have more than one value (**multiple-valued slot**)
-

Common Facets: Value Type

- **String:** a string of characters (“Château Lafite”)
- **Number:** an integer or a float (15, 4.5)
- **Boolean:** a true/false flag
- **Enumerated type:** a list of allowed values (high, medium, low)
- **Complex type:** an instance of another class
 - Specify the class to which the instances belong

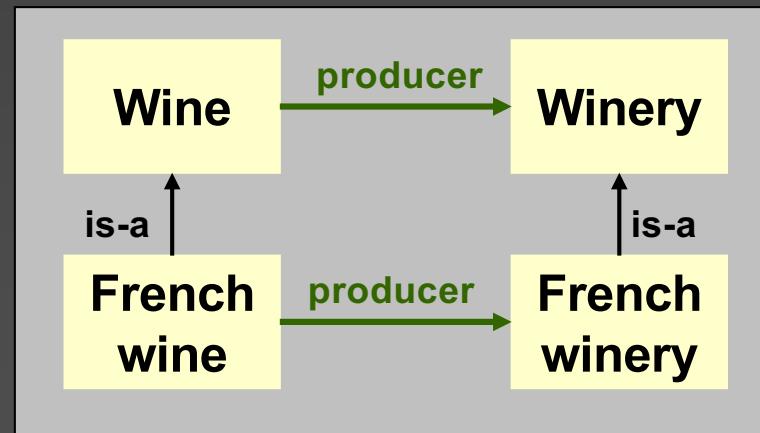
The Wine class is the value type for the slot “produces” at the Winery class

Domain and Range of Slot

- **Domain** of a slot – the class (or classes) that have the slot
 - More precisely: class (or classes) instances of which can have the slot
- **Range** of a slot – the class (or classes) to which slot values belong

Facets and Class Inheritance

- A subclass **inherits** all the slots from the superclass
- A subclass can **override** the facets to “narrow” the list of allowed values
 - Make the cardinality range smaller
 - Replace a class in the range with a subclass



Create Instances



- Create an instance of a class
 - The class becomes a **direct type** of the instance
 - Any superclass of the direct type is a **type** of the instance
- Assign slot values for the instance frame
 - Slot values should conform to the facet constraints
 - Knowledge-acquisition tools often check that

Creating an Instance: Example

I Chateau Morgan Beaujolais (Beaujolais)

Name	Area	V + -
Chateau Morgan Beaujolais	Beaujolais region	
Body	Color	Maker
LIGHT	RED	Chateau Morgan
Flavor	Sugar	Grape
DELICATE	DRY	Gamay grape
Tannin Level		
LOW		

Outline

- What is an ontology?
 - Why develop an ontology?
 - Step-By-Step: Developing an ontology
 - Going deeper: Common problems and solutions
 - Ontologies in the Semantic Web languages
 - Current research issues in ontology engineering
-

Going Deeper

- Breadth-first coverage



- Depth-first coverage



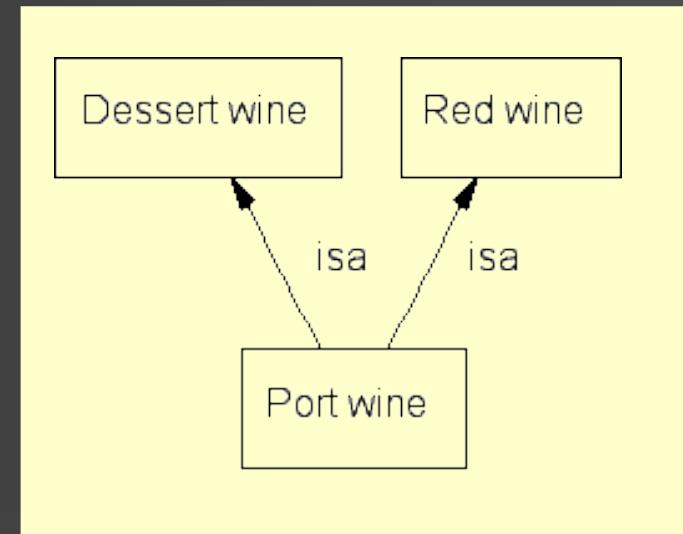
Defining Classes and a Class Hierarchy

- The things to remember:
 - There is no single correct class hierarchy
 - But there are some guidelines
- The question to ask:

“Is each instance of the subclass an instance of its superclass?”

Multiple Inheritance

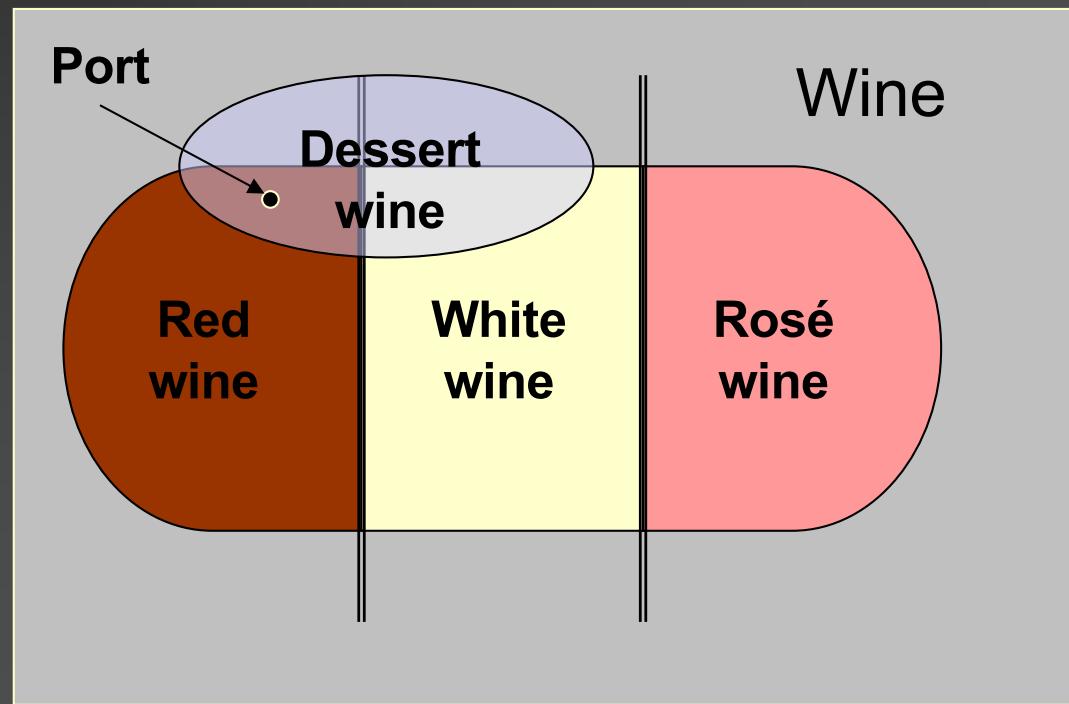
- A class can have more than one superclass
- A subclass inherits slots and facet restrictions from all the parents
- Different systems resolve conflicts differently



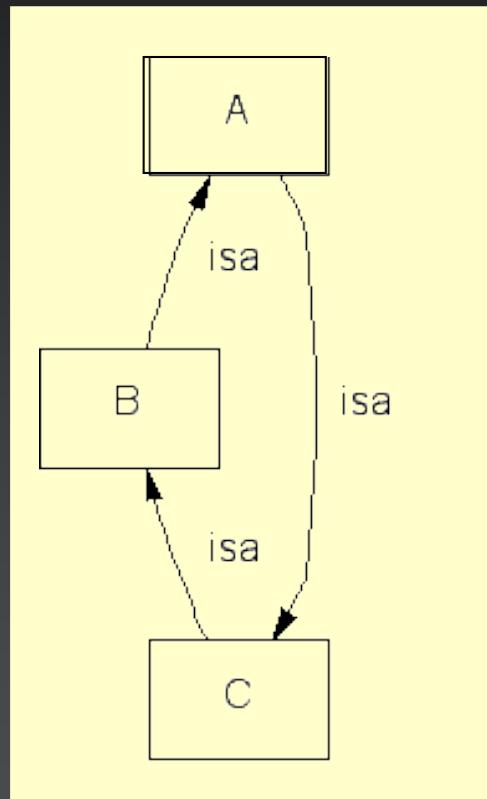
Disjoint Classes

- Classes are **disjoint** if they cannot have common instances
- Disjoint classes cannot have any common **subclasses** either

*Red wine, White wine,
Rosé wine are disjoint
Dessert wine and Red
wine are not disjoint*

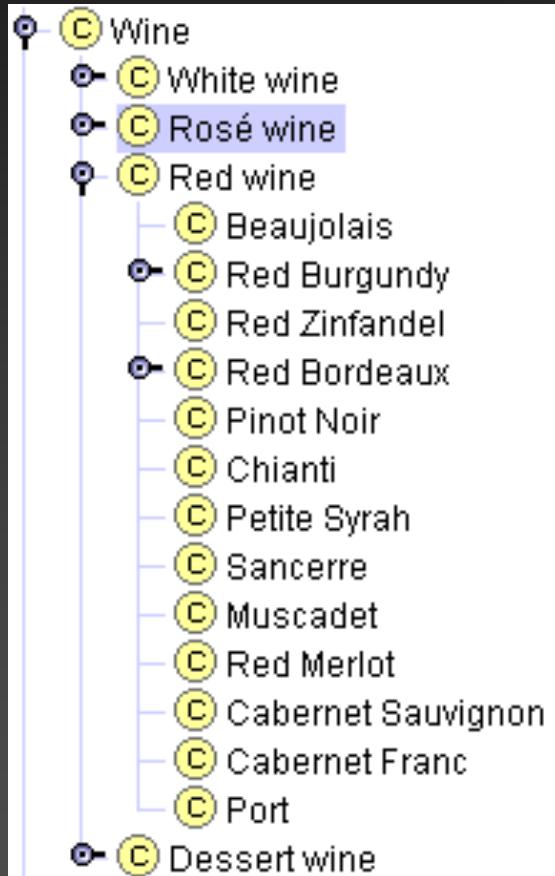


Avoiding Class Cycles



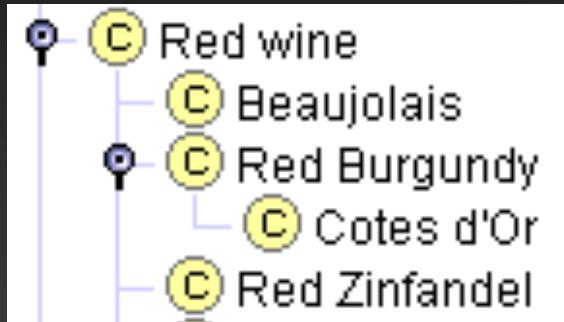
- Danger of multiple inheritance: cycles in the class hierarchy
- Classes A, B, and C have equivalent sets of instances
 - By many definitions, A, B, and C are thus equivalent

Siblings in a Class Hierarchy



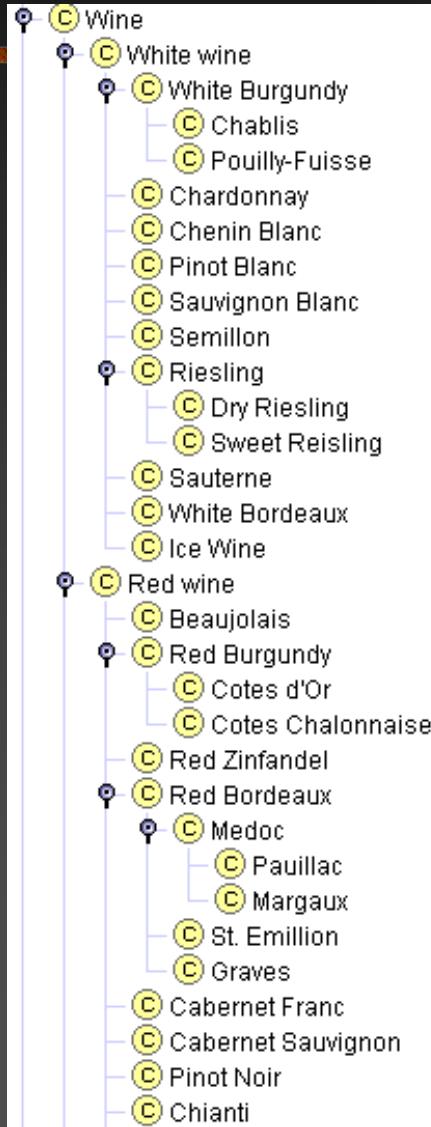
- All the **siblings** in the class hierarchy must be at the same level of generality
- Compare to section and subsections in a book

The Perfect Family Size



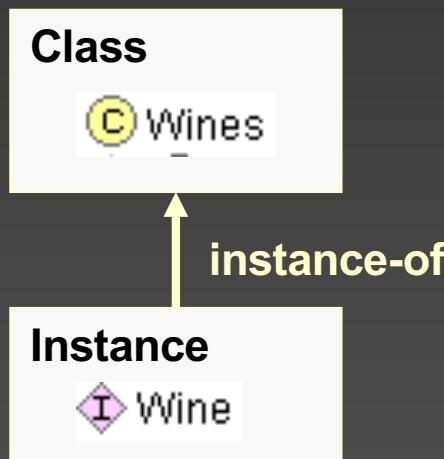
- If a class has only one child, there may be a modeling problem
 - If the only Red Burgundy we have is Côtes d'Or, why introduce the subhierarchy?
 - Compare to bullets in a bulleted list
-

The Perfect Family Size (II)



- If a class has more than a dozen children, additional subcategories may be necessary
- However, if no natural classification exists, the long list may be more natural

Single and Plural Class Names

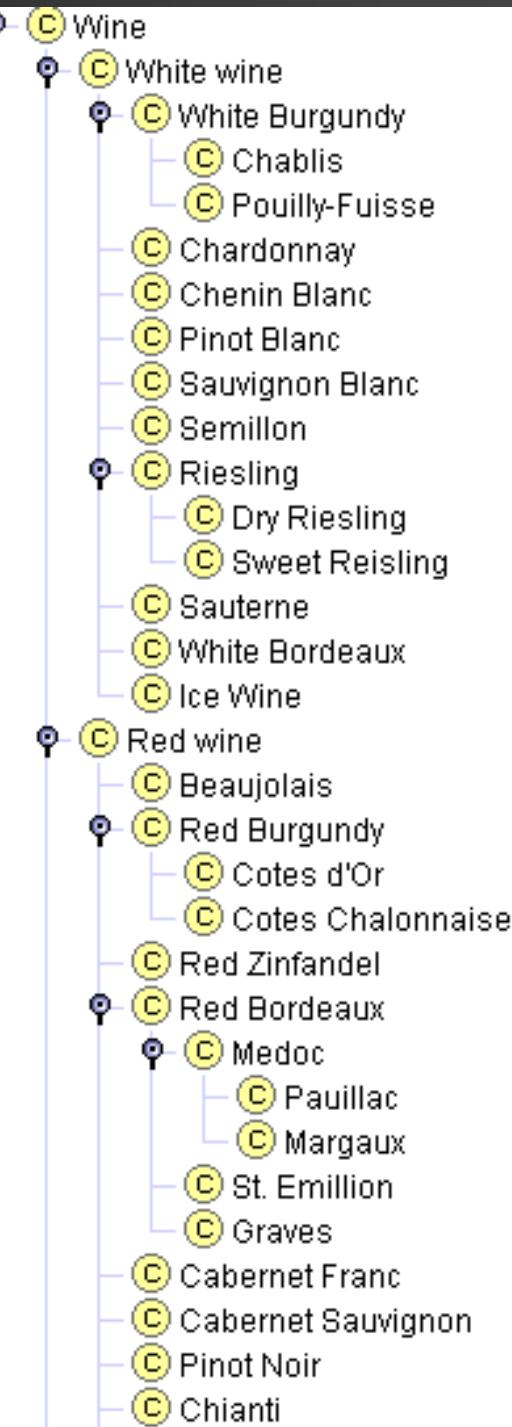


- A “wine” is not a kind-of “wines”
- A wine is an instance of the class Wines
- Class names should be either
 - all singular
 - all plural

Classes and Their Names

- Classes represent **concepts** in the domain, **not their names**
- The class name can change, but it will still refer to the same concept
- **Synonym names** for the same concept are not different classes
 - Many systems allow listing synonyms as part of the class definition

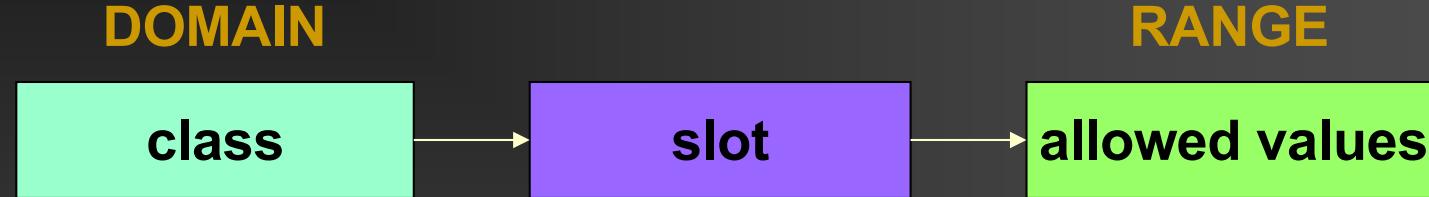
A Completed Hierarchy of Wines



Back to the Slots: Domain and Range

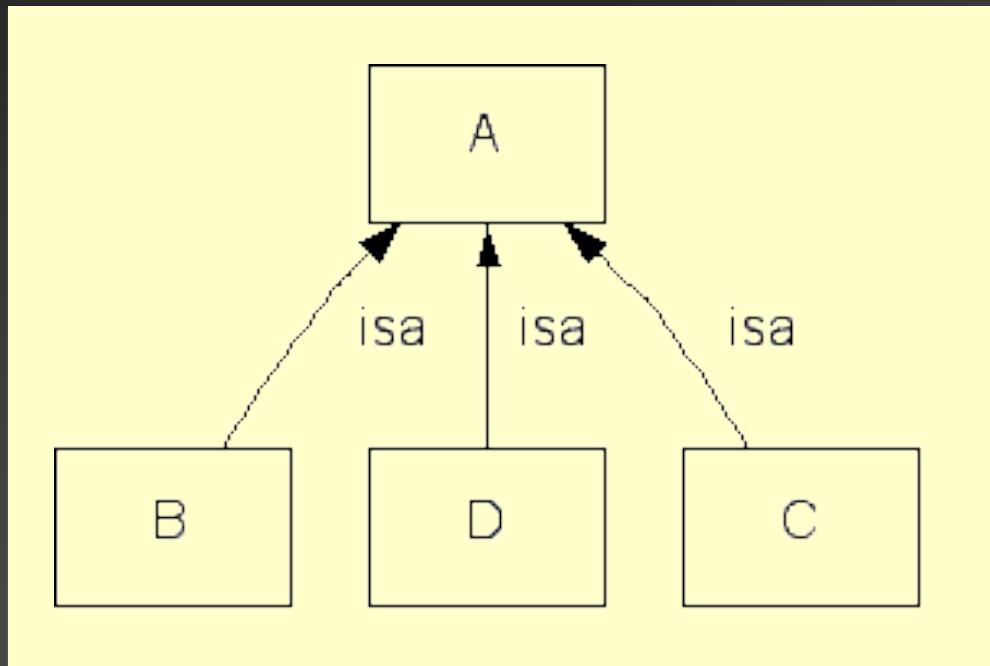
- When defining a domain or range for a slot, find the most general class or classes
- Consider the flavor slot
 - Domain: Red wine, White wine, Rosé wine
 - Domain: Wine
- Consider the produces slot for a Winery:
 - Range: Red wine, White wine, Rosé wine
 - Range: Wine

Back to the Slots: Domain and Range



- When defining a domain or range for a slot, find the most general class or classes
- Consider the flavor slot
 - Domain: Red wine, White wine, Rosé wine
 - Domain: Wine
- Consider the produces slot for a Winery:
 - Range: Red wine, White wine, Rosé wine
 - Range: Wine

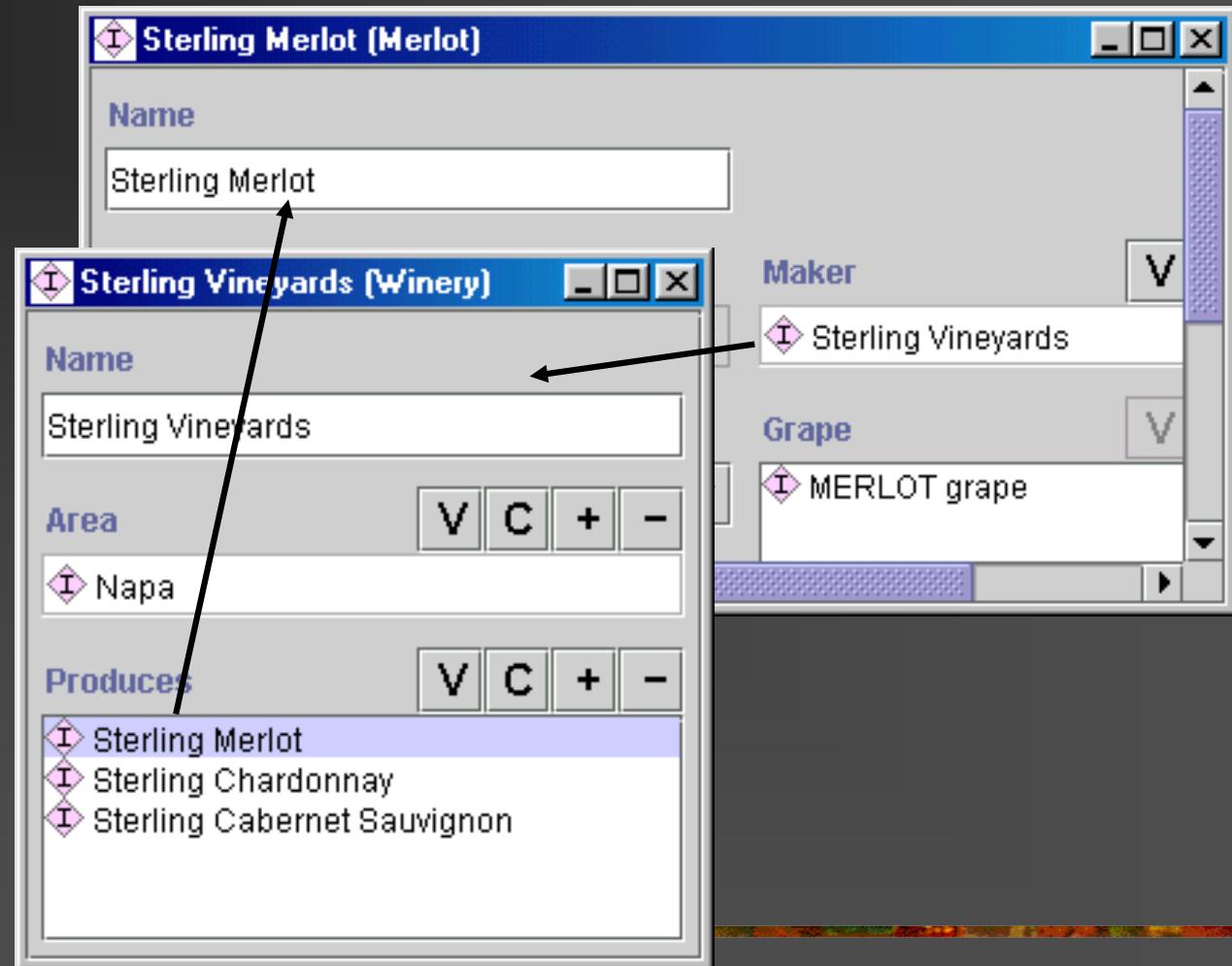
Defining Domain and Range



- A class and a superclass – replace with the superclass
 - All subclasses of a class – replace with the superclass
 - Most subclasses of a class – consider replacing with the superclass
-

Inverse Slots

Maker and
Producer
are inverse slots



Inverse Slots (II)

- Inverse slots contain redundant information, but
 - Allow acquisition of the information in either direction
 - Enable additional verification
 - Allow presentation of information in both directions
 - The actual implementation differs from system to system
 - Are both values stored?
 - When are the inverse values filled in?
 - What happens if we change the link to an inverse slot?
-

Default Values

- Default value – a value the slot gets when an instance is created
 - A default value can be changed
 - The default value is a **common** value for the slot, but is not a **required value**
 - For example, the default value for wine body can be FULL
-

Limiting the Scope

- An ontology should not contain **all** the possible information about the domain
 - No need to specialize or generalize more than the application requires
 - No need to include all possible properties of a class
 - Only the most salient properties
 - Only the properties that the applications require

Limiting the Scope (II)

- Ontology of wine, food, and their pairings probably will not include
 - Bottle size
 - Label color
 - My favorite food and wine
 - An ontology of biological experiments will contain
 - Biological organism
 - Experimenter
 - Is the class Experimenter a subclass of Biological organism?
-

Outline

- What is an ontology?
 - Why develop an ontology?
 - Step-By-Step: Developing an ontology
 - Going deeper: Common problems and solutions
 - **Ontologies in the Semantic Web languages**
 - Current research issues in ontology engineering
-

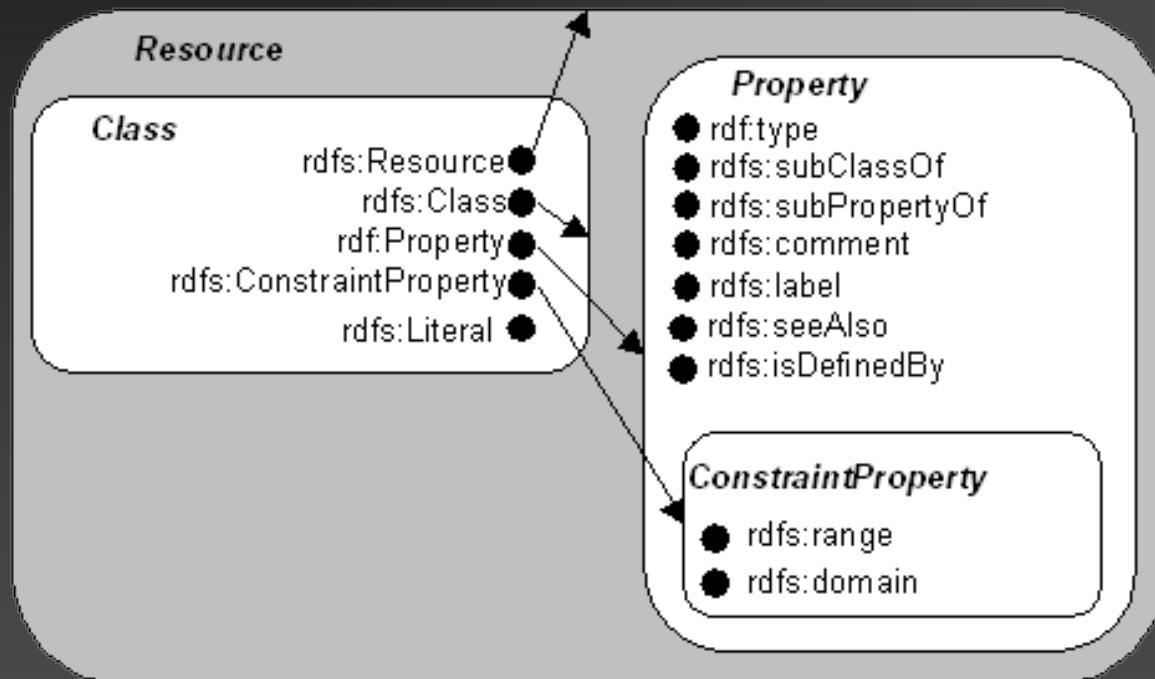
Ontologies and the SW Languages

- Most Semantic Web languages are designed explicitly for representing ontologies
 - RDF Schema
 - DAML+OIL
 - SHOE
 - XOL
 - XML Schema
-

SW Languages

- The languages differ in their
 - syntax
 - We are not concerned with it here – An ontology is a conceptual representation
 - terminology
 - Class-concept
 - Instance-object
 - Slot-property
 - expressivity
 - What we can express in some languages, we cannot express in others
 - semantics
 - The same statements may mean different things in different languages

RDF and RDF Schema Classes



RDF Schema Specification 1.0 (<http://www.w3.org/TR/2000/CR-rdf-schema-20000327/>)

RDF(S) Terminology and Semantics

- Classes and a class hierarchy
 - All classes are instances of **rdfs:Class**
 - A class hierarchy is defined by **rdfs:subClassOf**
- Instances of a class
 - Defined by **rdf:type**
- Properties
 - Properties are global:
A property **name** in one place is the same as the property **name** in another (assuming the same namespace)
 - Properties form a hierarchy, too (**rdfs:subPropertyOf**)

Property Constraints in RDF(S)

- Cardinality constraints
 - No explicit cardinality constraints
 - Any property can have multiple values
 - Range of a property
 - a property can have only one range
 - Domain of a property
 - a property can have more than one domain (can be attached to more than one class)
 - No default values
-

DAML+OIL: Classes And a Class Hierarchy

- Classes
 - Each class is an instance of **daml:Class**
 - Class hierarchy
 - Defined by **rdfs:subClassOf**
 - More ways to specify organization of classes
 - Disjointness (**daml:disjointWith**)
 - Equivalence (**daml:sameClassAs**)
 - The class hierarchy can be computed from the properties of classes
-

More Ways To Define a Class in DAML+OIL

- Union of classes

A class Person is a union of classes Male and Female

- Restriction on properties

A class Red Thing is a collection of things with color: Red

- Intersection of classes

A class Red Wine is an intersection of Wine and Red Thing

- Complement of a class

Carnivores are all the animals that are not herbivores

- Enumeration of elements

A class Wine Color contains the following instances: red, white, rosé

Property Constraints in DAML+OIL

- Cardinality
 - Minimum, maximum, exact cardinality
 - Range of a property
 - A property range can include multiple classes: the value of a property must be an instance of each of the classes
 - Can specify explicit union of classes if need different semantics
 - Domain of a property – same as range
 - No default values
-

Outline

- What is an ontology?
 - Why develop an ontology?
 - Step-By-Step: Developing an ontology
 - Going deeper: Common problems and solutions
 - Ontologies in the Semantic Web languages
 - Current research issues in ontology engineering
-

Research Issues in Ontology Engineering

- Content generation
- Analysis and evaluation
- Maintenance
- Ontology languages
- Tool development

Content: Top-Level Ontologies

- What does “top-level” mean?
 - Objects: tangible, intangible
 - Processes, events, actors, roles
 - Agents, organizations
 - Spaces, boundaries, location
 - Time
 - IEEE Standard Upper Ontology effort
 - Goal: Design a single upper-level ontology
 - Process: Merge upper-level of existing ontologies
-

Content: Knowledge Acquisition

- Knowledge acquisition is a bottleneck
 - Sharing and reuse alleviate the problem
 - But we need automated knowledge acquisition techniques
 - Linguistic techniques: ontology acquisition from text
 - Machine-learning: generate ontologies from structured documents (e.g., XML documents)
 - Exploiting the Web structure: generate ontologies by crawling structured Web sites
 - Knowledge-acquisition templates: experts specify only part of the knowledge required
-

Analysis



- Analysis: semantic consistency
 - Violation of property constraints
 - Cycles in the class hierarchy
 - Terms which are used but not defined
 - Interval restrictions that produce empty intervals
(min > max)
 - Analysis: style
 - Classes with a single subclass
 - Classes and slots with no definitions
 - Slots with no constraints (value type, cardinality)
 - Tools for automated analysis
 - Chimaera (Stanford KSL)
 - DAML validator
- 

Evaluation

- One of the hardest problems in ontology design
 - Ontology design is **subjective**
 - What does it mean for an ontology to be correct (**objectively**)?
 - The best test is the application for which the ontology was designed
-

Ontology Maintenance

- Ontology merging
 - Having two or more overlapping ontology, create a new one
 - Ontology mapping
 - Create a mapping between ontologies
 - Versioning and evolution
 - Compatibility between different versions of the same ontology
 - Compatibility between versions of an ontology and instance data
-

Ontology Languages

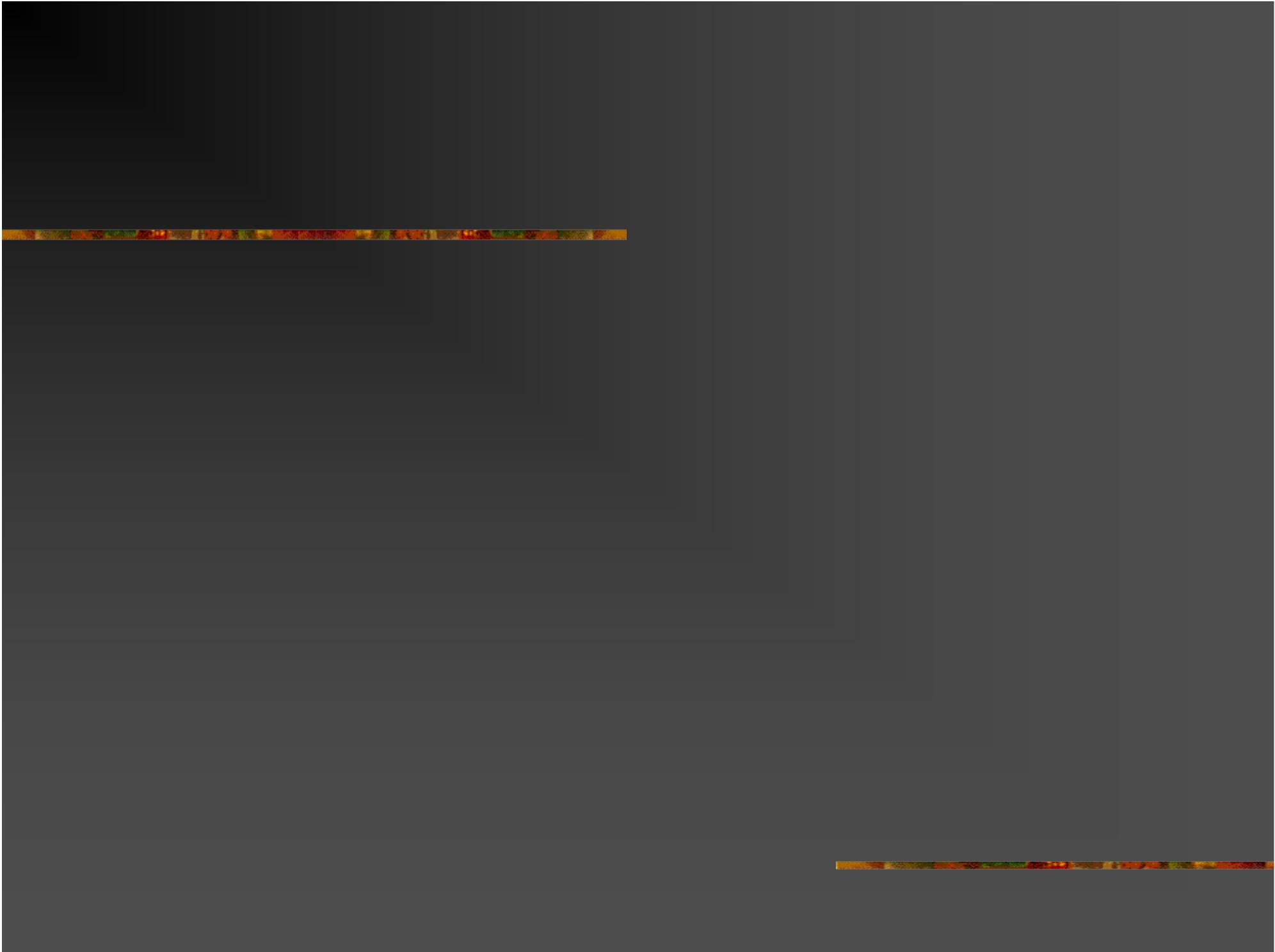
- What is the “right” level of expressiveness?
 - What is the “right” semantics?
 - When does the language make “too many” assumptions?
-

Ontology-Development Tools

- Support for various ontology language (knowledge interchange)
 - Expressivity
 - Usability
 - More and more domain experts are involved in ontology development
 - Multiple parentheses and variables will no longer do
-

Where to Go From Here?

- Tutorials
 - Natalya F. Noy and Deborah L. McGuinness (2001) “Ontology Development 101: A Guide to Creating Your First Ontology”
http://protege.stanford.edu/publications/ontology_development/ontology101.html
 - Farquhar, A. (1997). Ontolingua tutorial.
<http://ksl-web.stanford.edu/people/axf/tutorial.pdf>
 - We borrowed some ideas from this tutorial
- Methodology
 - Gómez-Pérez, A. (1998). Knowledge sharing and reuse. Handbook of Applied Expert Systems. Liebowitz, editor, CRC Press.
 - Uschold, M. and Gruninger, M. (1996). Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review* 11(2)



Transitivity of the Class Hierarchy

- The is-a relationship is transitive:

B is a subclass of A

C is a subclass of B

C is a subclass of A

- A direct superclass of a class is its “closest” superclass

