Intelligent Internet Technologies

Lectures 18-19.

Ontology Engineering

Alexandra V. Vitko

KNURE, AI Department, alexandra_vitko@yahoo.com

What is “Ontology Engineering”?

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

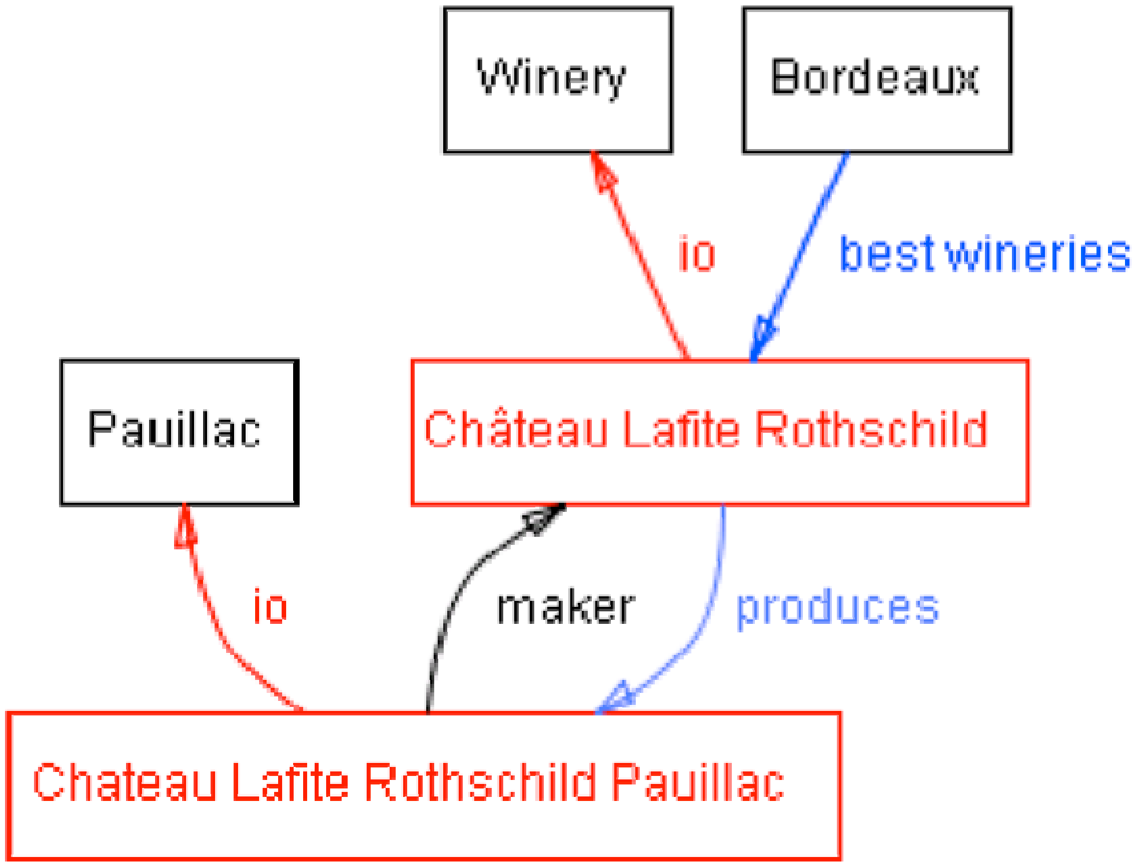
- Defining concepts in the domain (**classes**)
- Arranging the concepts in a taxonomy (**subclass-superclass hierarchy**)
- Defining which **properties** (**slots**) classes can have and constraints on their values (**facets**)
- Defining **instances** and filling in slot values

Tool for Ontology Engineering

Screenshots in further examples 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/index.shtml)

Ontology Example Used



Fundamental rules in ontology design

- There is **no one correct way to model a domain**— there are always viable alternatives. The best solution almost always depends on the application that you have in mind and the extensions that you anticipate.
- Ontology development is necessarily an **iterative** process.
- Concepts in the ontology should be close to objects (physical or logical) and relationships in your domain of interest. These are most likely to be nouns (objects) or verbs (relationships) in sentences that describe your domain.

Often an ontology of the domain is not a goal in itself.

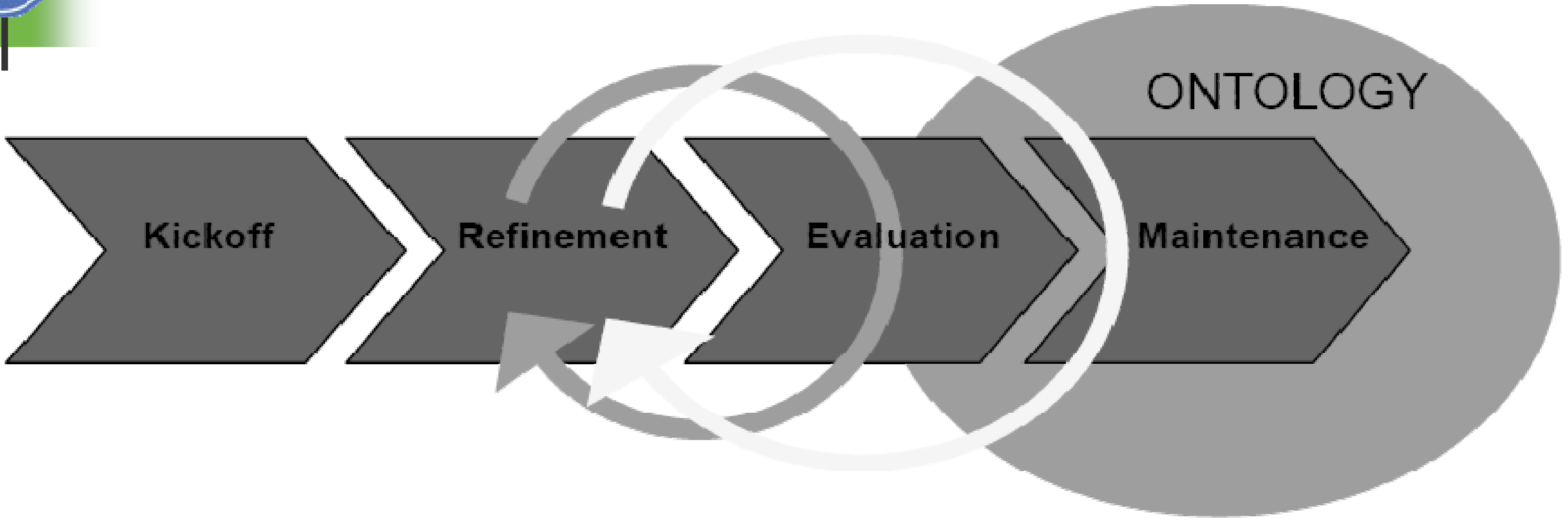
Design criteria for ontologies (Gruber,1993)

- Clarity
- Coherence
- Extendibility
- Minimal encoding bias
- Minimal ontological commitment

Methodology

- The task of developing an ontology is a typical knowledge acquisition task
- Existing methodologies for setting up knowledge-based systems may be adopted

OTK Methodology (http://www.ontoknowledge.org)



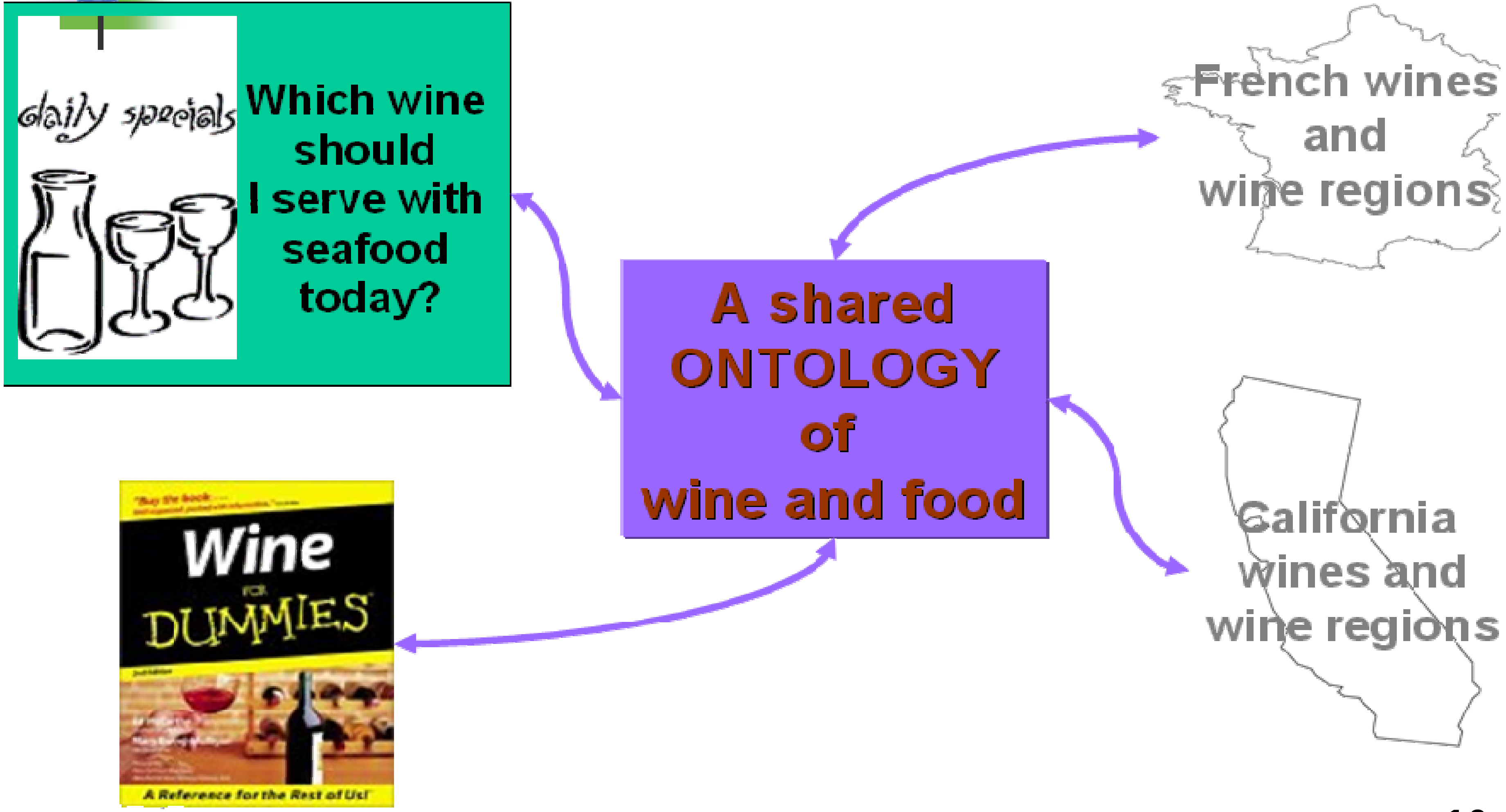
• Requirement specification • Analyze input sources • Develop baseline taxonomy	• Concept elicitation with domain experts • Conceptualize and formalize • Add relations and axioms	• Revision and expansion based on feedback • Analyze usage patterns • Analyze competency questions	• Manage organizational maintenance process

Ontology Development
Step 1. Determine the 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)
- Who will use and maintain the ontology?

The answers to these questions may change during the ontology-design process, but at any given time they help limit the scope of the model.

Ontology Example


daily specials
Which wine should I serve with seafood today?
A shared ONTOLOGY of wine and food
French wines and wine regions
California wines and wine regions
Wine for DUMMIES
A Reference for the Rest of Us!

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?

Step 2. Consider reusing existing ontologies

- It is almost always worth considering what someone else has done and checking if we can refine and extend existing sources for our particular domain and task.

Step 3. Enumerating Terms

wine, grape, winery, location, ...

wine color, wine body, wine flavor, sugar content, ...

white wine, red wine, Bordeaux wine, ...

food, seafood, fish, meat, vegetables,

cheese, ...

Step 4. Define the 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

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

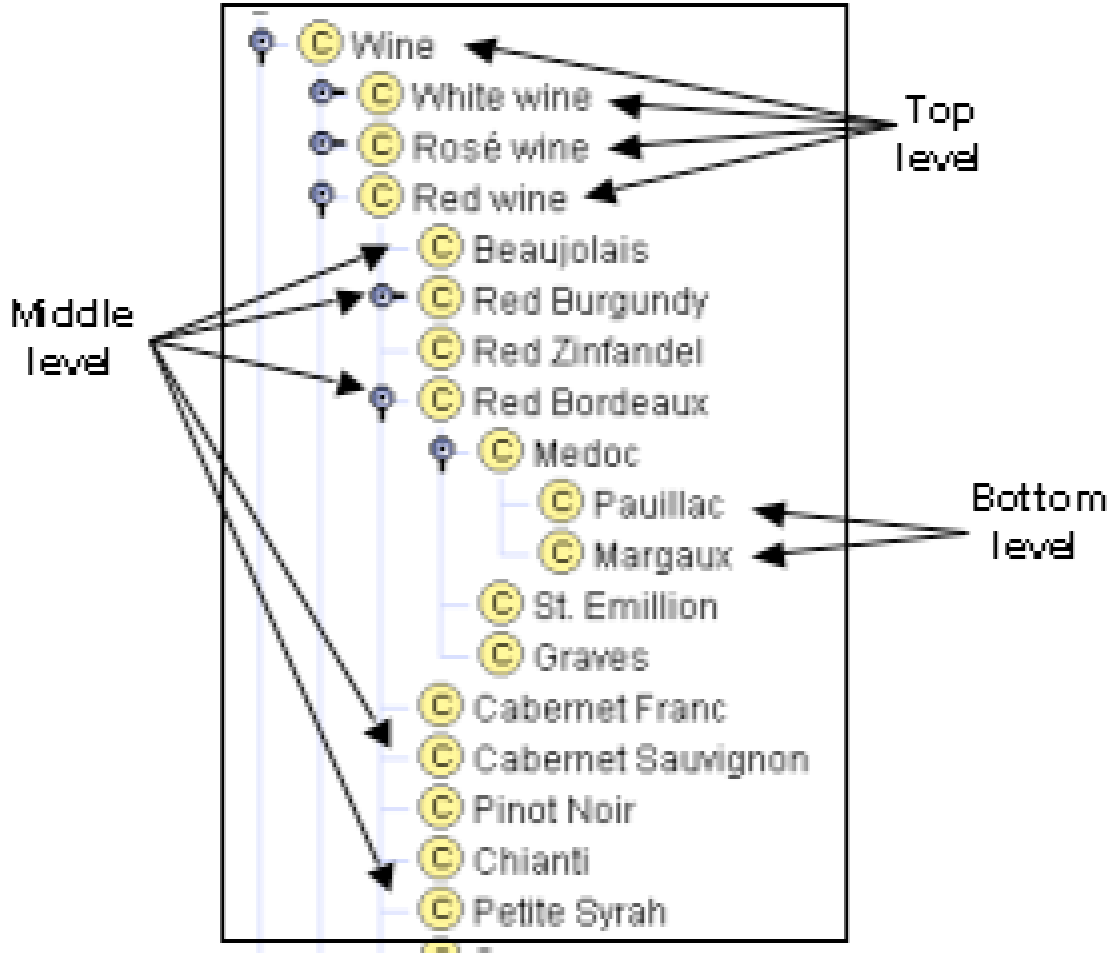
- Apple is a subclass of Fruit

Every apple is a fruit

- Red wine is a subclass of Wine

Every red wine is a wine

Levels in the Hierarchy



Models 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 appropriately

Documentation

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

Step 5. Define the properties of classes - slots

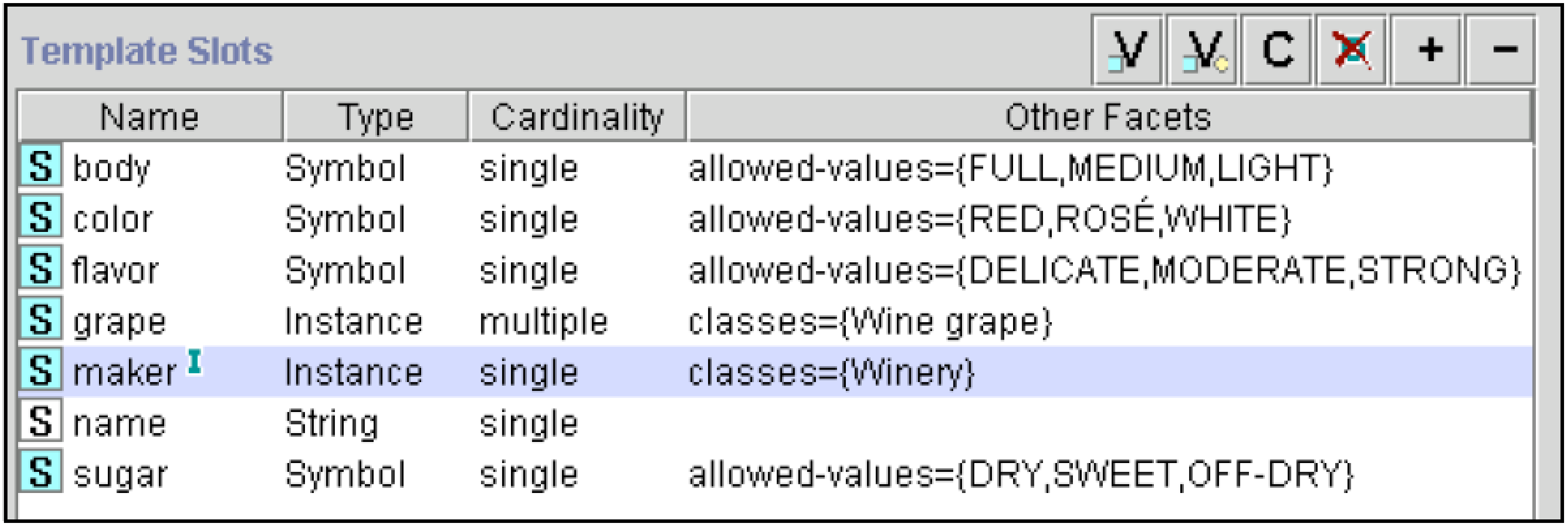
- Slots is a class definition describe attributes of instances of the class and relations to other instances

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

Properties (Slots)

- **Types of properties**
  - “intrinsic” properties: flavor and color of a wine;
  - “extrinsic” properties: name and price of a wine;
  - parts: ingredients in a dish;
  - relationships 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

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	Instance	single	classes={Winery}
S name	String	single	
S sugar	Symbol	single	allowed-values={DRY,SWEET,OFF-DRY}

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

Step 6. Define the facets of the slots (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

Common Facets

- **Slot cardinality** – the number of values a slot can have
- **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

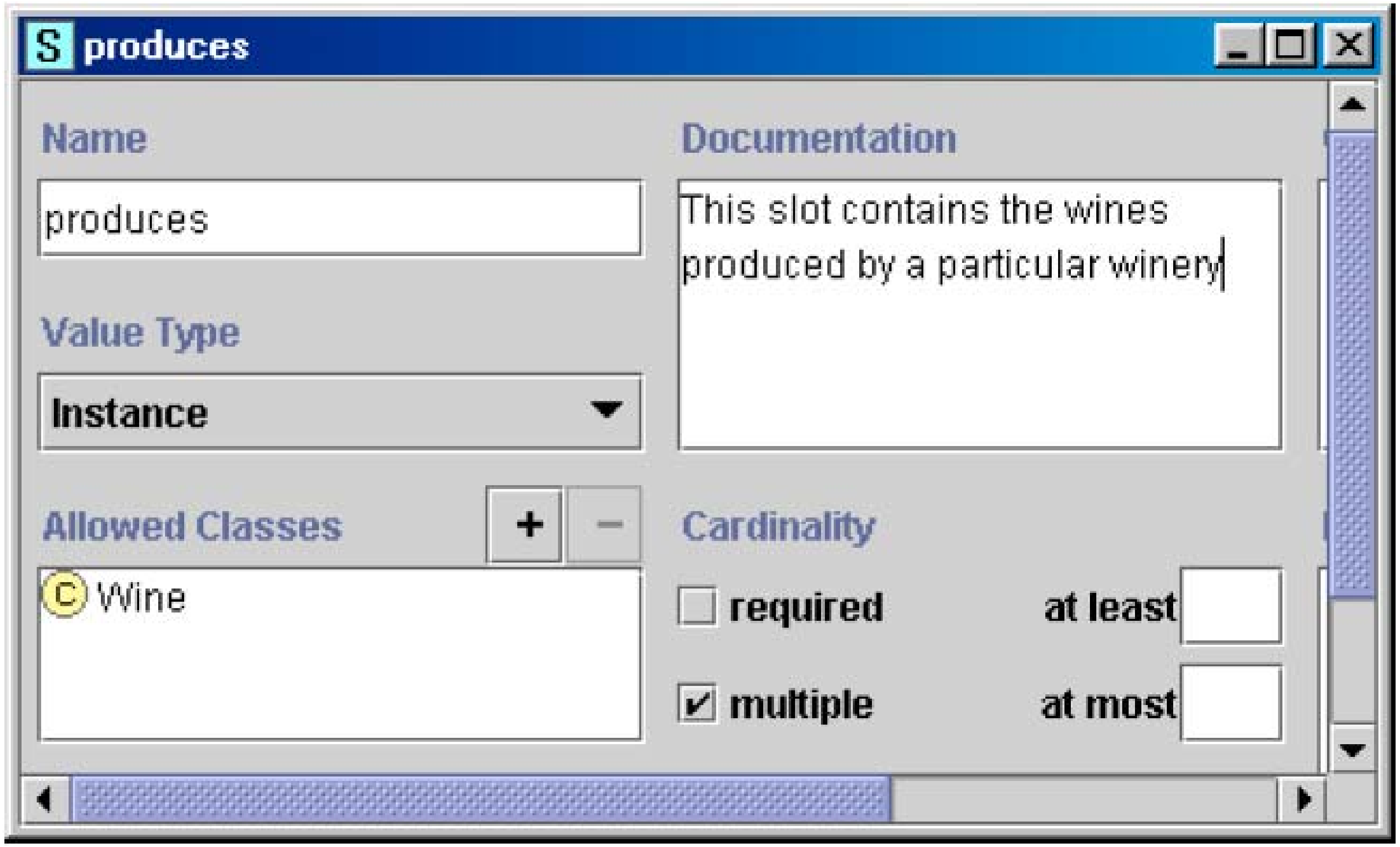
- **Cardinality**
  - Cardinality N means that the slot must have N values.
- **Minimum cardinality**
  - Minimum cardinality 1 means that the slot must have a values (required).
  - Minimum cardinality 0 means that the slot value is optional.
- **Maximum cardinality**
  - Maximum cardinality 1 means 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 ("Cabertnet")
- **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 instance belong

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

Example (definition of a slot "produces")



produces

Name: produces

Documentation: This slot contains the wines produced by a particular winery

Value Type: Instance

Allowed Classes: Wine

Cardinality: [ ] required [✓] multiple at least at most

Domain and Range of property

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

Basic rules for determining a domain and a range of a slot

- When defining a domain or a range for a slot, find the most general classes or class that can be respectively the domain or the range for the slots .
- On the other hand, do not define a domain and range that is overly general: all the classes in the domain of a slot should be described by the slot and instances of all the classes in the range of a slot should be potential fillers for the slot.

Basic rules for determining a domain and a range of a slot (2)

- If a list of classes defining a range or a domain of a slot includes a class and its subclass, remove the subclass.
- If a list of classes defining a range or a domain of a slot contains all subclasses of a class A, but not the class A itself, the range should contain only the class A and not the subclasses.
- If a list of classes defining a range or a domain of a slot contains all but a few subclasses of a class A, consider if the class A would make a more appropriate range definition.

Domain and Range (2)

- Range can be a class or a literal
- Consider the *colour* property
  - Domain: Wine
  - Range: literal (string) or
  - Range: class colour

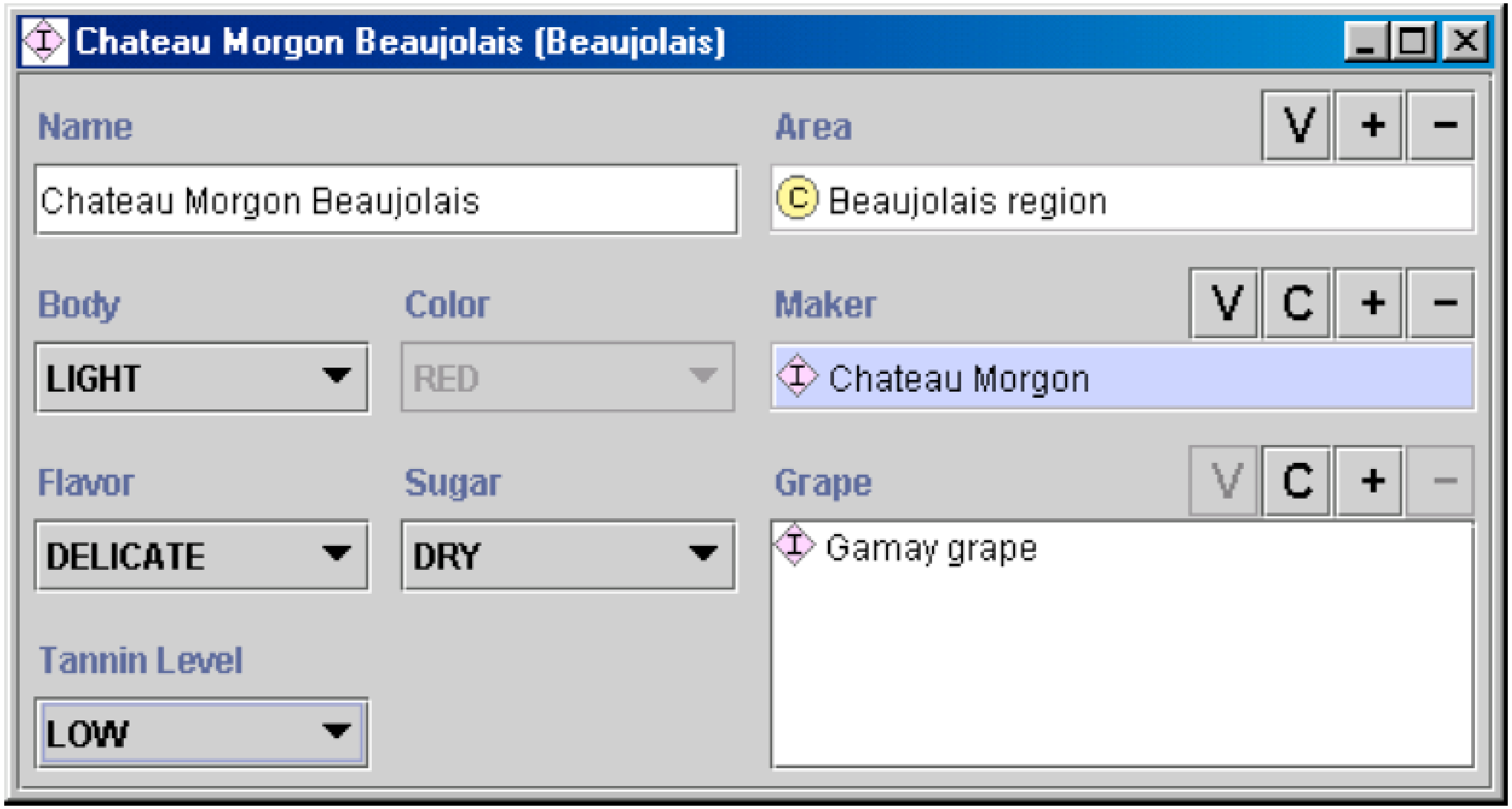
Default Values

- Default value – the 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

Step 7. Create Instances

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

Creating an Instance: Example



Chateau Morgon Beaujolais (Beaujolais)

Name: Chateau Morgon Beaujolais

Area: Beaujolais region

Body: LIGHT

Color: RED

Maker: Chateau Morgon

Flavor: DELICATE

Sugar: DRY

Grape: Gamay grape

Tannin Level: LOW

Defining Classes and a Class Hierarchy

- The things to remember:
  - There is no single correct class hierarchy for any given domain
  - But there are some guidelines

Ensuring that the class hierarchy is correct

- **An “is-a” relation**
  - *A subclass of a class represents a concept that is a “kind of” the concept that the superclass represents.*
- **A single wine is not a subclass of all wines**

Transitivity of the hierarchical relations

- A subclass relationship is transitive:

  If B is a subclass of A and

  C is a subclass of B, then

  C is a subclass of A

- A direct subclass of a class is its "closest" subclass

Classes and Their Names

- Classes represent **concepts** in the domain and **not their name**
- 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

Avoiding Class Cycles

- **cycles** in the class hierarchy when
  - some class A has a subclass B and at the same time B is a superclass of A
- then the classes A and B are equivalent:
  - all instances of A are instances of B and all instances of B are also instances of A

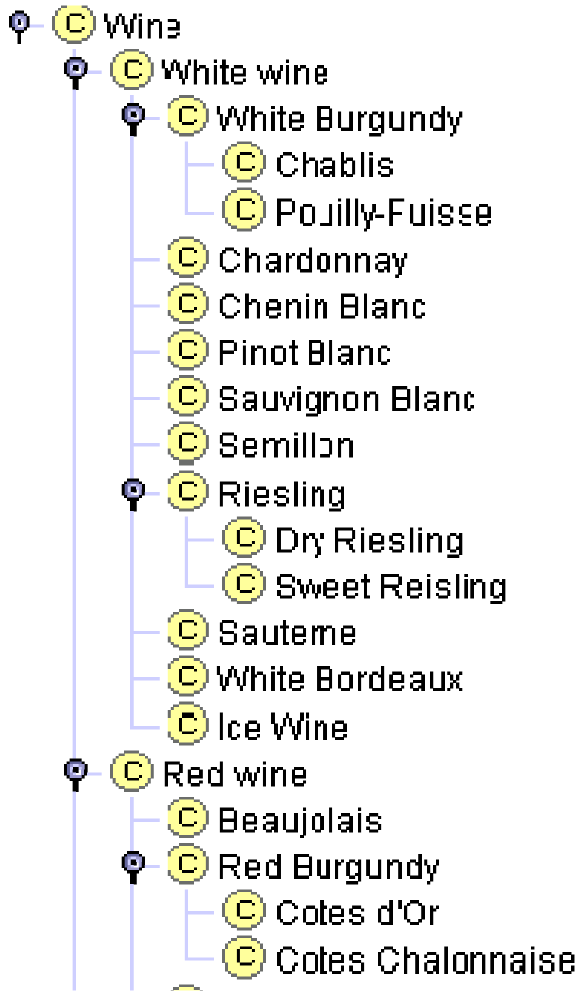
Siblings in a Class Hierarchy

- All the siblings in the hierarchy (except for the ones at the root) must be at the same level of generality
- Compare to section and subsections in a book

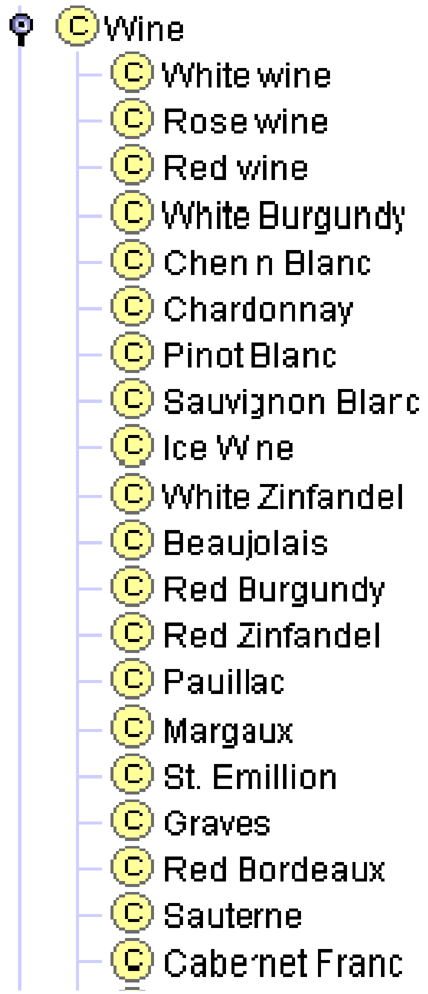
How many is too many and how few are too few?

- If a class has only one direct subclass there may be a modeling problem or the ontology is not complete
- If there are more than a dozen subclasses for a given class then additional intermediate categories may be necessary

How many is too many and how few are too few?



- Wine
  - White wine
    - White Burgundy
      - Chablis
      - Pouilly-Fuisse
    - Chardonnay
    - Chenin Blanc
    - Pinot Blanc
    - Sauvignon Blanc
    - Semillon
    - Riesling
      - Dry Riesling
      - Sweet Reisling
    - Sauterne
    - White Bordeaux
    - Ice Wine
  - Red wine
    - Beaujolais
    - Red Burgundy
      - Cotes d'Or
      - Cotes Chalonnaise



- Wine
  - White wine
  - Rose wine
  - Red wine
  - White Burgundy
  - Chenin Blanc
  - Chardonnay
  - Pinot Blanc
  - Sauvignon Blanc
  - Ice Wine
  - White Zinfandel
  - Beaujolais
  - Red Burgundy
  - Red Zinfandel
  - Pauillac
  - Margaux
  - St. Emillion
  - Graves
  - Red Bordeaux
  - Sauterne
  - Cabernet Franc

Multiple inheritance

- A class can have more then one superclass
- A subclass inherits slots and facet restrictions from all the parents

When to introduce a new class (or not)

- Subclasses of a class usually
  - have additional properties that the superclass dos not have, or
  - restrictions different from those of the superclass, or
  - participate in different relationships than the superclass
- Classes in terminological hierarchies do not have to introduce new properties

A new class or a property value?

- If the concepts with different slot values become restrictions for different slots in other classes. Otherwise, we represent the distinction in a slot value
- If a distinction is important in the domain and we think of the objects with different value for the distinction as different kinds of objects, then we should create a new class for the distinction
- A class to which an individual instance belongs should not change often

An instance or a class?

- Individual instances are the most specific concepts represented in a knowledge base
- If concepts form a natural hierarchy, then we should represent them as classes

Limiting the Scope (1)

- 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 (2)

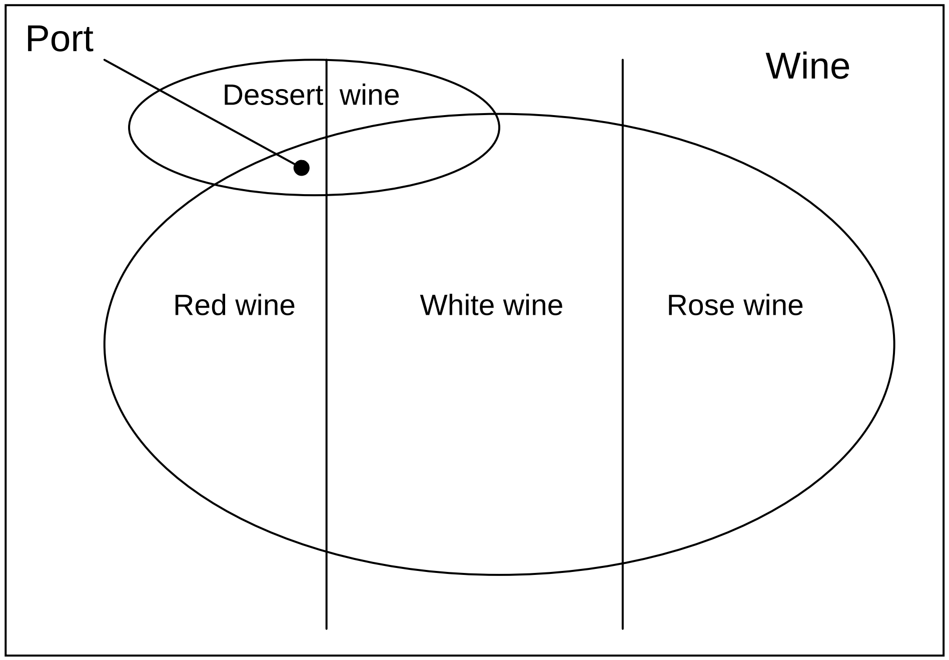
- Ontology of wine, food and there pairings probably will not include
  - Bottle size
  - Label color
- An ontology of biological experiments will contain
  - Biological organism
  - Experimenter
- Is the class Experimenter a subclass of Biological organism?

Disjoint Classes

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

Red wine, White wine, Rose wine are disjoint

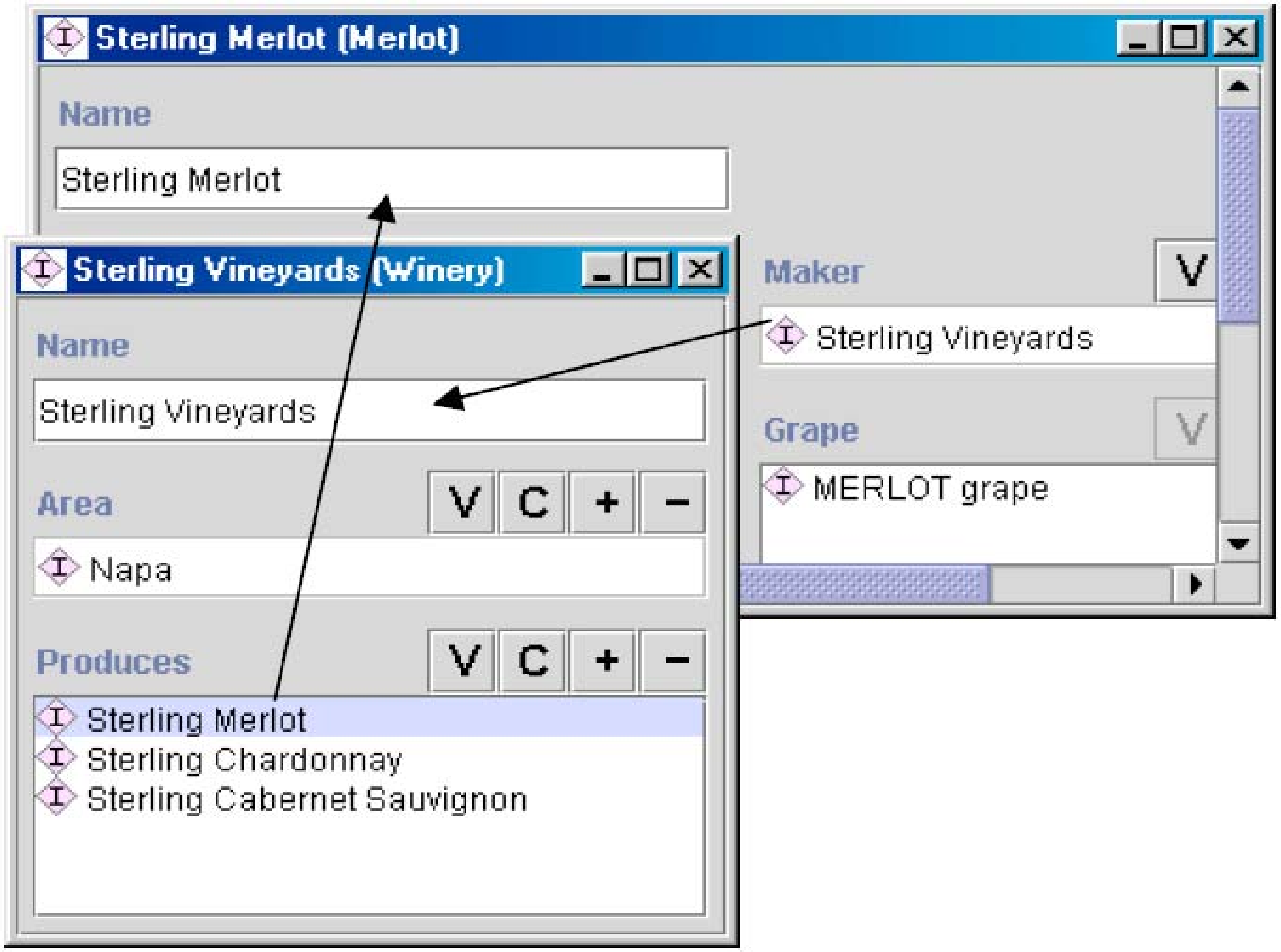
Dessert wine and Red wine are not disjoint



Inverse slots

- If a *wine* was *produced by* a *winery*, then the *winery produces* that *wine*
- *maker* and *produces* are called **inverse relations**
- Filling in one of the inverse slots triggers an automatic update of the other

Instances with inverse slots



Default values

- If a particular slot value is the same for most instances of a class, we can define this value to be a **default value** for the slot
- We can then change the value to any other value that the facets will allow
- Default values are there for convenience:
  - they do not enforce any new restrictions on the model
  - they do not change the model in any way

***Default values** is different from **slot values:** Slot values cannot be changed.*

What’s in a name?

Define a naming convention for classes and slots and adhere to it

- Take into account he following features of a knowledge representation system:
  - Does the system have the same name space for classes, slots, and instances? (class *winery* and a slot *winery*)
  - Is the system case-sensitive? (such as *Winery* and *winery*)
  - What delimiters does the system allow in the names? (spaces, commas, asterisks, and so on)
- Protégé-2000
  - maintains a single name space for all its frames (we cannot have a class *winery* and a slot *winery*)
  - is case-sensitive (we can have a class *Winery* and a slot *winery*)
- CLASSIC
  - is not case sensitive and maintains different name spaces for classes, slots, and individuals (we can have both a class and a slot *Winery*)

Capitalization and delimiters

It is common to capitalize class names and use lower case for slot names (assuming the system is case-sensitive)

- Some possible choices of delimiting the words in concept name:
  - Use Space: *Meal course* (Protégé allow spaces in concept names)
  - Run the words together and capitalize each new word: *MealCourse*
  - Use an underscore or dash or other delimiter in the name: *Meal_Course, Meal_course, Meal-Course, Meal-course*

Singular or plural

- How to call the class: *Wines* or *Wine*?
- No alternative is better or worse than the other
- Singular for class names is used more often in practice
- Whatever the choice, it should be consistent throughout the whole ontology

Do not create a class *Wines* and then create a class *Wine* as its subclass

Prefix and suffix conventions

- Two common practices are:

  to add a *has-* or a suffix *–of* to slot names (*has-maker* and *has-winery*, or *maker-of* and *winery-of*)
- This approach allows anyone looking at a term to determine immediately if the term is a class or a slot
- But the term names become slightly longer.

Other naming considerations

- Do not add strings such as “class”, “property”, “slot”, and so on to concept names
- It is usually a good idea to avoid abbreviations in concept names (use *Cabernet Sauvignon* rather than *Cab*)
- Names of direct subclasses of a class should either all include or not include the name of the superclass (*Red Wine* and *White Wine* or *Red* and *White*, but not *Red Wine* and *White*)