Ontology

Dr.Khaled ElBahnasy

What is in an ontology

An **ontology** is a formal explicit description of concepts in a domain of discourse (**classes** (sometimes called **concepts**)), properties of each concept describing various features and attributes of the concept (**slots** (sometimes called **roles** or **properties**)), and restrictions on slots (**facets** (sometimes called **role restrictions**)).

An ontology together with a set of individual **instances** of classes constitutes a **knowledge base**. In reality, there is a fine line where the ontology ends and the knowledge base begins.

Developing an ontology

In practical terms, developing an ontology includes:

- defining classes in the ontology
- arranging the classes in a taxonomic (subclass–superclass) hierarchy,
- defining slots and describing allowed values for these slots,
- filling in the values for slots for instances.

A Simple Knowledge-Engineering Methodology

- First, we would like to emphasize some fundamental rules in ontology design to which we will refer many times. These rules may seem rather dogmatic. They can help, however, to make design decisions in many cases.
- *1) 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.*
- *2) Ontology development is necessarily an iterative process.*
- *3) 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.*

A Simple Knowledge-Engineering Methodology

- That is, deciding what we are going to use the ontology for, and how detailed or general the ontology is going to be will guide many of the modeling decisions down the road. Among several viable alternatives, we will need to determine which one would work better for the projected task, be more intuitive, more extensible, and more maintainable. We also need to remember that an ontology is a model of reality of the world and the concepts in the ontology must reflect this reality. After we define an initial version of the ontology, we can evaluate and debug it by using it in applications or problem-solving methods or by discussing it with experts in the field, or both. As a result, we will almost certainly need to revise the initial ontology. This process of iterative design will likely continue through the entire lifecycle of the ontology.

Step 1.Determine the domain and scope of the ontology

- We suggest starting the development of an ontology by defining its domain and scope. That is, answer several basic questions:
- · 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?
- · 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.

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. Reusing existing ontologies may be a requirement if our system needs to interact with other applications that have already committed to particular ontologies or controlled vocabularies. Many ontologies are already available in electronic form and can be imported into an ontology-development environment that you are using. The formalism in which an ontology is expressed often does not matter, since many knowledge-representation systems can import and export ontologies. Even if a knowledge-representation system cannot work directly with a particular formalism, the task of translating an ontology from one formalism to another is usually not a difficult one.

Step 2.Consider reusing existing ontologies

- There are libraries of reusable ontologies on the Web and in the literature. For example, we can use the Ontolingua ontology library (http://www.ksl.stanford.edu/software/ontolingua/) or the DAML ontology library (http://www.daml.org/ontologies/). There are also a number of publicly available commercial ontologies (e.g., UNSPSC (www.unspsc.org), RosettaNet (www.rosettanet.org), DMOZ (www.dmoz.org)).
- For this guide however we will assume that no relevant ontologies already exist and start developing the ontology from scratch.

Step 3.Enumerate important terms in the ontology

- It is useful to write down a list of all terms we would like either to make statements about or to explain to a user. What are the terms we would like to talk about? What properties do those terms have? What would we like to say about those terms? For example, important wine-related terms will include wine, grape, winery, location, a wine's color, body, flavor and sugar content; different types of food, such as fish and red meat; subtypes of wine such as white wine, and so on. Initially, it is important to get a comprehensive list of terms without worrying about overlap between concepts they represent, relations among the terms, or any properties that the concepts may have, or whether the concepts are classes or slots..

Step 3.Enumerate important terms in the ontology

- The next two steps—developing the class hierarchy and defining properties of concepts (slots)—are closely intertwined. It is hard to do one of them first and then do the other. Typically, we create a few definitions of the concepts in the hierarchy and then continue by describing properties of these concepts and so on. These two steps are also the most important steps in the ontology-design process. We will describe them here briefly and then spend the next two sections discussing the more complicated issues that need to be considered, common pitfalls, decisions to make, and so on.

Step 4.Define the classes and the class hierarchy

- There are several possible approaches in developing a class hierarchy (Uschold and Gruninger 1996):
- · A **top-down** development process starts with the definition of the most general concepts in the domain and subsequent specialization of the concepts. For example, we can start with creating classes for the general concepts of Wine and Food. Then we specialize the Wine class by creating some of its subclasses: White wine, Red wine, Rosé wine. We can further categorize the Red wine class, for example, into Syrah, Red Burgundy, Cabernet Sauvignon, and so on.
- · A **bottom-up** development process starts with the definition of the most specific classes, the leaves of the hierarchy, with subsequent grouping of these classes into more general concepts. For example, we start by defining classes for Pauillac and Margaux wines. We then create a common superclass for these two classes—Medoc—which in turn is a subclass of Bordeaux.
- · A **combination** development process is a combination of the top-down and bottom-up approaches: We define the more salient concepts first and then generalize and specialize them appropriately. We might start with a few top-level concepts such as Wine, and a few specific concepts, such as Margaux . We can then relate them to a middle-level concept, such as Medoc. Then we may want to generate all of the regional wine classes from France, thereby generating a number of middle-level concepts.

Step 4.Define the classes and the class hierarchy

- None of these three methods is inherently better than any of the others. The approach to take depends strongly on the personal view of the domain. If a developer has a systematic top-down view of the domain, then it may be easier to use the top-down approach. The combination approach is often the easiest for many ontology developers, since the concepts “in the middle” tend to be the more descriptive concepts in the domain (Rosch 1978).
- If you tend to think of wines by distinguishing the most general classification first, then the top-down approach may work better for you. If you’d rather start by getting grounded with specific examples, the bottom-up approach may be more appropriate.
- Whichever approach we choose, we usually start by defining classes. From the list created in Step 3, we select the terms that describe objects having independent existence rather than terms that describe these objects. These terms will be classes in the ontology and will become anchors in the class hierarchy.[2] We organize the classes into a hierarchical taxonomy by asking if by being an instance of one class, the object will necessarily (i.e., by definition) be an instance of some other class.

Step 4.Define the classes and the class hierarchy

- *If a class A is a superclass of class B, then every instance of B is also an instance of A*
- In other words, the class B represents a concept that is a “kind of” A.

Step 5.Define the properties of classes—slots

- The classes alone will not provide enough information to answer the competency questions from Step 1. Once we have defined some of the classes, we must describe the internal structure of concepts.
- We have already selected classes from the list of terms we created in Step 3. Most of the remaining terms are likely to be properties of these classes. These terms include, for example, a wine's color, body, flavor and sugar content and location of a winery.
- For each property in the list, we must determine which class it describes. These properties become slots attached to classes. Thus, the Wine class will have the following slots: color, body, flavor, and sugar. And the class Winery will have a location slot.
- In general, there are several types of object properties that can become slots in an ontology:
- · "intrinsic" properties such as the flavor of a wine;
- · "extrinsic" properties such as a wine's name, and area it comes from;
- · parts, if the object is structured; these can be both physical and abstract "parts" (e.g., the courses of a meal)
- · relationships to other individuals; these are the relationships between individual members of the class and other items (e.g., the maker of a wine, representing a relationship between a wine and a winery, and the grape the wine is made from.)

Step 6.Define the facets of the slots

- Slots can have different facets describing the value type, allowed values, the number of the values (cardinality), and other features of the values the slot can take. For example, the value of a name slot (as in “the name of a wine”) is one string. That is, name is a slot with value type String. A slot produces (as in “a winery produces these wines”) can have multiple values and the values are instances of the class Wine. That is, produces is a slot with value type Instance with Wine as allowed class.
- We will now describe several common facets.
- **Slot cardinality**
- Slot cardinality defines how many values a slot can have. Some systems distinguish only between single cardinality (allowing at most one value) and multiple cardinality (allowing any number of values). A body of a wine will be a single cardinality slot (a wine can have only one body). Wines produced by a particular winery fill in a multiple-cardinality slot produces for a Winery class.
- Some systems allow specification of a minimum and maximum cardinality to describe the number of slot values more precisely. Minimum cardinality of N means that a slot must have at least N values. For example, the grape slot of a Wine has a minimum cardinality of 1: each wine is made of at least one variety of grape. Maximum cardinality of M means that a slot can have at most M values. The maximum cardinality for the grape slot for single varietal wines is 1: these wines are made from only one variety of grape. Sometimes it may be useful to set the maximum cardinality to 0. This setting would indicate that the slot cannot have any values for a particular subclass.

Slot-value type

- A value-type facet describes what types of values can fill in the slot. Here is a list of the more common value types:
- · **String** is the simplest value type which is used for slots such as name: the value is a simple string
- · **Number** (sometimes more specific value types of Float and Integer are used) describes slots with numeric values. For example, a price of a wine can have a value type Float
- · **Boolean** slots are simple yes–no flags. For example, if we choose not to represent sparkling wines as a separate class, whether or not a wine is sparkling can be represented as a value of a Boolean slot: if the value is “true” (“yes”) the wine is sparkling and if the value is “false” (“no”) the wine is not a sparkling one.
- · **Enumerated** slots specify a list of specific allowed values for the slot. For example, we can specify that the flavor slot can take on one of the three possible values: strong, moderate, and delicate. In Protégé-2000 the enumerated slots are of type Symbol.
- · **Instance**-type slots allow definition of relationships between individuals. Slots with value type Instance must also define a list of allowed classes from which the instances can come. For example, a slot produces for the class Winery may have instances of the class Wine as its values.

Domain and range of a slot

- Allowed classes for slots of type Instance are often called a **range** of a slot. In the example, the class Wine is the range of the produces slot. Some systems allow restricting the range of a slot when the slot is attached for a particular class.
- The classes to which a slot is attached or a classes which property a slot describes, are called the **domain** of the slot..In the systems where we *attach* slots to classes, the classes to which the slot is attached usually constitute the domain of that slot. There is no need to specify the domain separately.
- The basic rules for determining a domain and a range of a slot are similar:
- *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. an overly general class for range (i.e., one would not want to make the range THING) but one would want to choose a class that will cover all fillers*
- Instead of listing all possible subclasses of the Wine class for the range of the produces slot, just list Wine. At the same time, we do not want to specify the range of the slot as THING—the most general class in an ontology.
- In more specific terms:

Step 7.Create instances

- The last step is creating individual instances of classes in the hierarchy. Defining an individual instance of a class requires (1) choosing a class, (2) creating an individual instance of that class, and (3) filling in the slot values. For example, we can create an individual instance Chateau-Morgon-Beaujolais to represent a specific type of Beaujolais wine. Chateau-Morgon-Beaujolais is an instance of the class Beaujolais representing all Beaujolais wines. This instance has the following slot values defined :
- · Body: Light
- · Color: Red
- · Flavor: Delicate
- · Tannin level: Low
- · Grape: Gamay (instance of the Wine grape class)
- · Maker: Chateau-Morgon (instance of the Winery class)
- · Region: Beaujolais (instance of the Wine-Region class)
- · Sugar: Dry