Just What Is an Ontology, Anyway?

Thomas C. Jepsen
IT consultant

This tutorial article describes some definitions of "ontology" as it relates to computer applications and gives an overview of some common ontology-based applications.

Finding a precise definition of an ontology as it applies to computer applications can be a trying experience. According to *Merriam-Webster's Dictionary*, an ontology is "the branch of metaphysics dealing with the nature of being or reality." Moving on to the Object Management Group's *Ontology Definition Metamodel* document,[1] we learn that an ontology is a "specification of a conceptualization." Finally, turning in desperation to the W3C's Web Ontology Language Overview,[2] we discover that an ontology is "the representation of meaning of terms in vocabularies and the relationships between those terms." All of this is so confusing—just what is an ontology, anyway?

Perhaps a good practical definition would be that an ontology is a method of representing items of knowledge (ideas, facts, things—whatever) in a way that defines the relationships and classifications of concepts within a specified domain of knowledge. It's this ability to define a variety of useful relationships among items of knowledge, and to implement these relationships in software, that make an ontology such a powerful gadget in the knowledge manager's toolkit.

History of Ontologies

Part of the difficulty in defining just what an ontology is lies in the fact that the word ontology comes from philosophy, where it refers to the study of being or existence. European medieval scholars, for example, devised an argument for the existence of God that they referred to as the *ontological argument*. (The ontological argument for God's existence, as stated by Anselm of Canterbury [1033–1109] and others, asserted that the Supreme Being was the highest term in a scale of terms ranging from the lowest form of being to infinity, referred to as the "Great Chain of Being." So if we use modern ontology language to describe Anselm's argument, God is the ultimate Thing class, and all lower beings, from the angels down to the microbes, are nested subclasses of Thing.) From a more modern perspective, ontologies came to be of interest to computer scientists in the 1970s as they began to develop the field of artificial intelligence. They realized that if you could create a domain of knowledge and establish formal relationships among the items of knowledge in the domain, you could perform certain types of automated reasoning. Tom Gruber,

 Published by the IEEE Computer Society

a computer scientist as Stanford University, formally introduced the term ontology to computer science in a 1993 paper.[3]

Gruber's work and that of other computer scientists led to the development of several formal ontology languages. DARPA, for example, developed DAML (the DARPA Agent Markup Language) for Semantic Web applications; DAML + OIL (Ontology Inference Layer), a syntax layered on RDF and XML, served as the basis for the W3C's Web Ontology Language, or OWL. (As you've probably noted by now, ontologies rely heavily on nested acronyms to define their lineage, sort of a tip of the hat to the original ontological argument for the Great Chain of Being.) The W3C's Web Ontology Working Group formed in 2001 and released the first OWL specifications in 2004. I will refer to OWL as a typical ontology language for the remainder of this article. (But, really, it's not that simple because OWL has three separate flavors: OWL Lite, a simple syntax that provides classification capabilities; OWL DL, a computationally complete version that supports description logic; and OWL Full, a version that provides compatibility with RDF schema. Unless otherwise noted, the descriptions in this article refer to OWL Lite.)

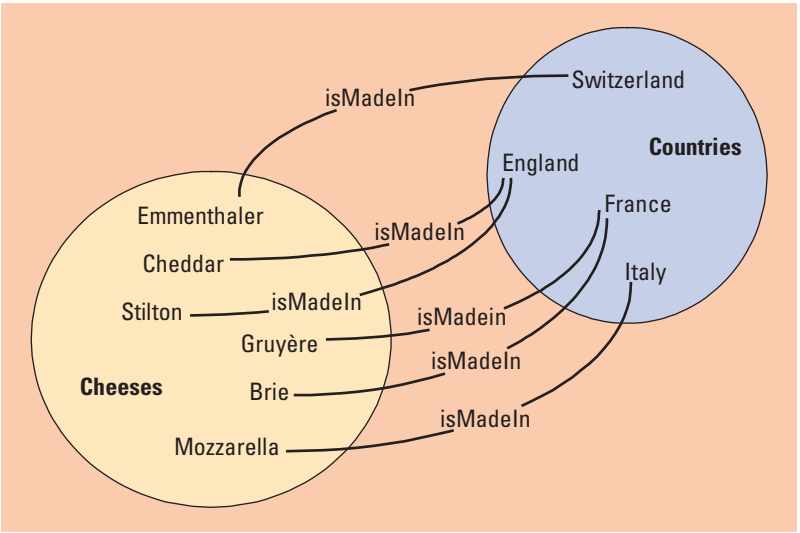

Figure 1. Cheeses and countries. Relationships among classes and instances illustrate the use of properties.

Ontology Properties and Characteristics

Like object-oriented programming, ontologies use classes and instances to represent knowledge items, but implementations of these two metamodels use classes in significantly different ways. For example, a class is a fundamental artifact (classifier) in the Unified Modeling Language (UML) that describes a set of objects that share the same specifications of features, constraints, and semantics. An instance of a class is an object; instances of UML classes inherit their behavior from the class definition, and all objects in UML are instances of named classes. In OWL, we can think of a class as a set, and the instances of the OWL class (OWLThings), referred to collectively as the class extension, as members of that set. OWL also has a universal class, Thing; all individuals are members of Thing, and all classes are subclasses of it. However, in OWL, an individual can be an instance of Thing and not an instance of any other class.

The real power of ontologies lies in the ability to create relationships among classes and instances, and to assign properties to those relationships that let us make inferences about them. Consider, for example, a set of cheeses—Brie, cheddar, Emmenthaler, Gruyère, mozzarella, and Stilton. Consider also a set of countries—England, Switzerland, Italy, and France. It's easy to imagine a relationship labeled "is made in" between each cheese and a country, as in Figure 1.

We can infer several facts from these relationships—for example, we can see that an instance either belongs to the set of cheeses or the set of countries, but can't be both a cheese and a country. In ontology language, we can say that the class of cheeses and the class of countries are disjoint—there are no instances that belong to both. These relationships can help us do some reasoning: the property isMadeIn tells us that if we're looking for Emmenthaler, we know that it's made in Switzerland, so we can expect to find it there. Conversely, we know that if we're in England looking for a cheese, we can expect to find Stilton and cheddar. This is because we can infer that isMadeIn has an inverse property (we can call it hasProduct, for example) that relates the two instances in the opposite direction.

Classes can also be subclasses of another class. Thus, Cheeses can be a subclass of Food, which might include meat, fish, vegetables, and fruits. This lets us say with certainty that because Brie

is a Cheese, and all Cheeses are Food, then Brie is also a Food. (This relationship is called *necessary implication*.) OWL allows equivalent classes to exist—that is, two classes can be stated to be the same and to contain the same instances. For example, we might create a class called Fromages, which can contain all the same instances as Cheeses.

We can see from Figure 1 that cheddar is made in England; however, the figure doesn't tell us if cheddar is also made in the US or in Bulgaria. In OWL, this is unknowable because like most ontology languages, OWL follows the *open world assumption*—any assertion not explicitly stated is undecidable; the *closed world assumption* is that any assertion not explicitly stated is assumed to be false. Hotel reservation systems, for example, are based on the closed world assumption—if my name isn't on the reservation list, I don't have a room for the night. Similarly, although Figure 1 indicates that Brie and Gruyère are made in France, it doesn't tell us if these are the only cheeses made in France. We would need additional information to determine this. In the US, we often refer to Emmenthaler as "Swiss cheese." This poses no problem for OWL, which is quite happy to refer to the same cheese by either name. This is because OWL doesn't use the *unique name assumption* (UNA), which states that all individuals must have a unique name.

An example of an immanent ontology would be an ontology of all the items in a daily newspaper; its structure would change on a daily basis, dependent on that day's news.

Types of Ontologies

Ontologies tend to be of two general types: *transcendent* ontologies, which are authoritative and defined externally from the applications that use them, and *immanent* ontologies, in which the structure is defined by the domain's knowledge content. An example of a transcendent ontology would be the periodic table used in chemistry; the structure is rigidly defined by a long-standing body of scientific knowledge and rarely altered. If we look at the periodic table, we note that it's organized in columns and rows—the noble gases, such as helium and neon, the halogens, such as chlorine and fluorine, and so forth. Any new element would have to be added to an already existing category, and all the chemists in the world would have to agree that, yes, this really is a new element and it belongs in this specific category. Adding a new element is a relatively rare occurrence, and adding a new family of elements seems extremely unlikely. It's a tribute to Dmitri Mendeleev's intuition that the structure he created 140 years ago is still in use in essentially its original form. An example of an immanent ontology would be an ontology of all the items in a daily newspaper; its structure would change on a daily basis, dependent on that day's news. One day's ontology might include articles on a natural disaster; the next day's might include a report of an economic crisis.

You might ask, which kind is best? As usual, it all depends. Many designers start out to implement what they believe to be transcendent ontologies, only to discover that what they believed to be fixed categories have suddenly grown, shrunk, or morphed into something else. As Clay Shirky points out in his much-cited blog piece, "Ontology Is Overrated: Categories, Links and Tags," designers of classification systems often make the mistake of unconsciously reflecting their own prejudices in the systems they implement (www.shirky.com/writings/ontology_overrated.html). For example, the Dewey Decimal system, long used by librarians to classify books by subject, reflected the Eurocentric bias of its creators by allocating most of its codes to subjects related to European history and culture. The importance of categories tends to change over time as cultures change and new innovations arise. An ontology of programming languages created in 1980, for example, would include COBOL, Fortran, and C, but C++, Java, Python, and Ruby on Rails would be conspicuously absent. Very few ontologies are truly transcendent in the sense of having fixed categories that rarely, if ever, change.

How Is an Ontology Different from a Hierarchy?

We're all aware that big things can contain small things; this concept is easily accommodated with-

in ontologies by using classes, subclasses, and superclasses to show hierarchical relationships. Thus, as I've already shown, we can create a class of foods that includes subclasses of fruits, vegetables, meat, fish, and cheeses. Within each subclass, we might identify a property found in certain individuals—for instance, we might find examples of fruits, vegetables, and cheeses that are orange in color. Within a true hierarchical structure, it's difficult to show that this property is common across the subclasses: we might end up creating separate subclasses of orange fruits, orange vegetables, and orange cheeses. In an ontology, however, we can simply create a new class of orange food items and make it a subclass of fruits, vegetables, and cheeses, all at the same time, as Figure 2 illustrates.

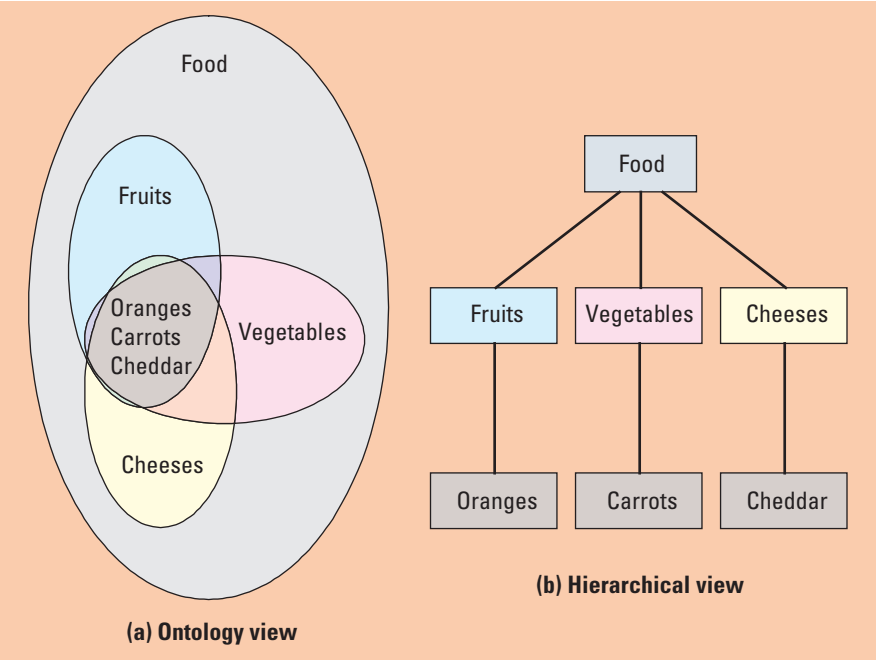

Figure 2. Hierarchical view vs. ontological view. (a) Ontologies represent common properties across subclasses, whereas (b) hierarchies reflect a more top-down approach.

Reasoners

Because the relationships used in ontologies are formally defined, it's possible to use a reasoner to perform automated reasoning—for example, a reasoner can determine if one class is a subclass of another class. This makes it possible to find the ontology's inferred class hierarchy and determine if a given class has any possible instances. In the orange food ontology example, a reasoner can determine that the class of oranges, carrots, and cheddar is a valid subclass of Food, and that it contains at least three members.

Visual Representations

Of the many good reasons for using visual languages for knowledge representation, one is that an ontology might contain relationships that aren't easily expressible in pure text or formal logic. Another is that it's often easier to represent complex relationships with a picture that can be understood on an intuitive level. Generally speaking, visual languages for knowledge representation consist of nodes connected by arcs. However, to eliminate ambiguity, we must apply some formal rules—starting and ending points, directionality, node function, and cardinality of relationships must all be defined.

One form of visual knowledge representation for ontologies is the semantic network, or concept map. A semantic network is a graph made of nodes connected by arcs. The nodes represent objects or concepts in a domain, and the arcs represent relationships among the nodes. Typical relationships might include (but are not limited to) is-a-type-of, is-related-to, or is-an-instance-of.

One of the most common tools for visual representation of ontologies is Protégé, a free, open source ontology editor and knowledgebase framework (http://protege.stanford.edu). Protégé is based on a Meta Object Facility- (MOF-) compatible metamodel, and its ontologies can be exported in RDF, OWL, or XML schema format. Figure 3 shows an example in Protégé of an OWL semantic network that shows the relationship of different types of pizza.

Protégé provides tools for visualizing ontologies as well as for constructing them; Figure 3, for example, uses a visualization tool called OWLViz to provide a visual representation of an ontology and its relationships. Protégé also allows for automated reasoning; RACER is a reasoner frequently used with it (www.sts.tu-harburg.de/~r.f.moeller/racer/). In fact, the

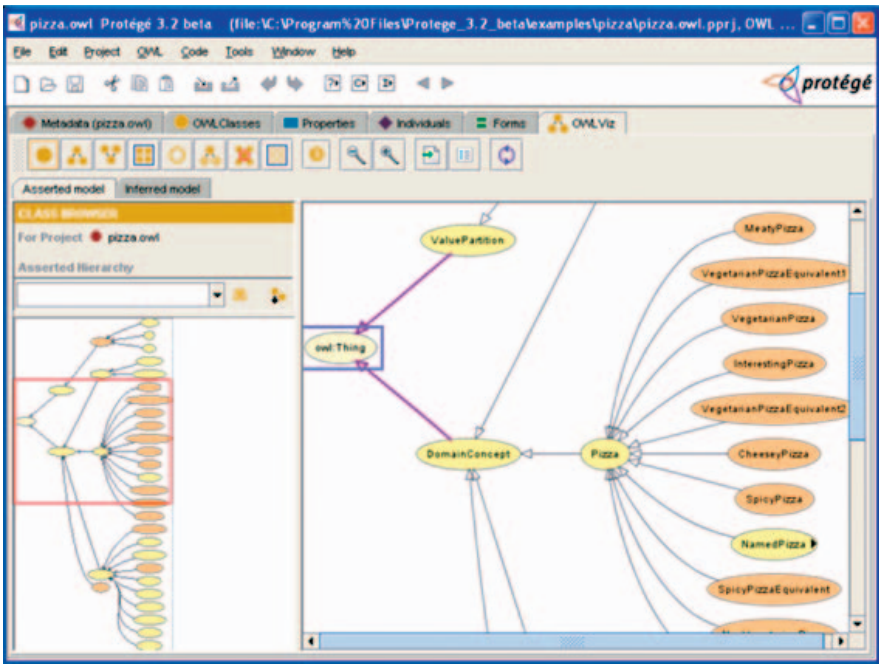

Figure 3. Example semantic network representation in Protégé. This visual representation of an ontology uses an OWL semantic network to show the relationships among different types of pizza.

Protégé user interface allows for side-by-side viewing of asserted and inferred class hierarchies using RACER.

Practical Applications for Ontologies

Ontologies are particularly well-suited for research in areas with vast amounts of available data, where the relationships to be explored aren't hierarchical, such as in biomedical research. A good example is pharmaceutical research—it's possible to create an ontology of pharmaceuticals and their chemical composition, their intended therapeutic use, and genetic variations among individual patients. Using this approach, scientists can see relationships between a patient's individual genome and the efficacy of a specific medication in a treatment regimen, and then fine-tune the treatment to make it more effective.

Gene Ontology

Genetic information tends to be scattered across multiple databases, making retrieval and analysis more difficult. One project that attempts to deal with this is the Gene Ontology (www.geneontology.org/GO.doc.shtml), which began in 1998 as a collaborative effort among three model organism databases—FlyBase, the Saccharomyces Genome Database (SGD), and the Mouse Genome Database (MGD)—to merge the information contained in separate databases for drosophila (fruitfly), saccharomyces (yeast), and mouse genomes. It has since grown to include more than 17 individual databases of genetic information. The Gene Ontology maintains three ontologies that describe gene products in terms of their associated biological processes, cellular components, and molecular functions in a species-independent manner. In addition to maintaining the ontologies, the project's researchers provide gene annotation and develop tools to facilitate access and search.

The Gene Ontology makes extensive use of some of the fundamental characteristics of ontologies. Because we might refer to a single genetic entity by multiple names, the ontologies use synonyms to broaden or narrow the scope of inquiry as necessary. Child terms (subclasses) might be related to multiple parent terms (superclasses), for example. The ontologies use five basic types of relationships: is_a, part_of, regulates, positively_regulates, and negatively_regulates.

SNOMED-CT

Standardized Nomenclature for Medicine-Clinical Terminology (SNOMED-CT) is an ontology of clinical terminology used in healthcare. It's the world's most comprehensive clinical terminology database, comprising 350,000+ terms, and is maintained by the International Health Terminology Standards Development Organization (IHTSDO; www.ihtsdo.org).

SNOMED-CT is a good example of a transcendent ontology. It consists of concepts that represent clinical meanings, each with a numerical concept ID and a human-readable fully specified name (FSN). Each FSN contains a semantic tag that indicates the clinical category (such as Person, Disorder) it belongs to, and each concept has a *preferred term* defined for it that captures the common expression used by clinicians. Unlike FSNs, however, preferred terms aren't

uniquely identified because a term can have different meanings in different contexts. A concept can also have multiple synonyms associated with it, which represent different names for the same thing—for example, "heart attack" and "cardiac infarction" are synonyms for the same concept/FSN, "myocardial infarction." Relationships in SNOMED-CT can be of four types: defining, qualifying, historical, and additional. The main defining relationship is the IS_A relationship, which SNOMED-CT uses to define subtype/supertype hierarchical relationships.

SNOMED-CT uses defining attributes to model concept definitions. An attribute's domain is the hierarchy (or hierarchies) to which the attribute applies. An attribute's range is the set of values that it can possess. Some common attributes include finding site, which describes an anatomical structure; causative agent, which defines the direct causative agent of a disease; and severity, which represents the level of severity for a clinical finding.

SNOMED-CT's value lies not only in the way it helps standardize medical terminology, but in its ability to automate many functions related to medical record administration. Use of SNOMED-CT concept IDs and FSNs enables much of the work involved in patient billing to be automated and makes it easier to collect data for research purposes.

Ontologies, as I pointed out earlier, have become an important tool in the knowledge manager's toolkit. Their ability to find needles of pertinent relationships in haystacks of data is particularly critical in an age when we're all being inundated with a firehose of information on a daily basis. You have to wonder what the medieval philosophers would say if they could see what their ruminations on existence have evolved into.

References

1. *Ontology Definition Metamodel*, OMG document number ptc/2007-09-09, Object Management Group, Nov. 2007.
2. "OWL Web Ontology Overview," W3C Recommendation, 10 Feb. 2004; www.w3.org/TR/2004/REC-owl-features-20040210/.
3. T. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing," *Int'l J. Human-Computer Studies*, vol. 43, nos. 5–6, 1995, pp. 907–928.

Thomas C. Jepsen *is an IT consultant in Chapel Hill, North Carolina. He also serves as the programming languages editor for* IT Professional *and chair of IEEE-USA's Medical Technology Policy Committee. Jepsen's current research interests include healthcare IT standards and development of service-oriented architecture (SOA) applications. He's the author of* Distributed Storage Networks: Architecture, Protocols and Management *(John Wiley & Sons, 2003), editor of* Java in Telecommunications: Solutions for Next Generation Networks *(John Wiley & Sons, 2001), and contributing author of* Systems Engineering Approach to Medical Automation *(Artech House, 2008). Contact him at tjepsen@ieee.org.*