Ontologies and Their Practical Implementation

Gian Piero Zarri
University of Paris IV/Sorbonne, France

INTRODUCTION:
“ONTOLOGIES” AND “TAXONOMIES”

Starting from the ‘90s, ontologies have emerged as an important research topic investigated by several research communities (including the database community) and used especially in defining standards for data exchange, information integration, and interoperability. The word “ontology” comes from medieval philosophy, where it was used to talk about the existence of beings in the world (Guarino & Giaretta, 1995). According to its modern computer-science technical meaning (Gruber, 1993), a consensus definition says that, “Ontologies represent a formal and explicit specification of a shared conceptualization,” (p. 199) where:

- **Conceptualization** refers to an abstract model of some phenomenon/situation in the world, where the model results by the identification of the relevant concepts that characterize this particular phenomenon/situation. To avoid any hype, “concepts” can be simply understood here as the discrete, important notions that must be necessarily utilized to describe the phenomenon/situation under consideration.
- **Explicit** means that the type of concepts used and the constraints on their use are explicitly defined.
- **Formal** refers to the fact that the ontology should be machine-usable.
- **Shared** reflects the notion that an ontology captures consensual knowledge, that is, this knowledge is not private to some individual but must be accepted by a group.

A definition like this needs, however, some further discussion. Apart from the requirement of being usable on a computer, there is nothing in the previous characterization of an ontology that, for example, could not be applied also to a “taxon” in the classical Linnaean meaning: It is obvious, in fact, that Linnaeae’s classifications for biology were intended to give an exhaustive definition of some phenomena/situations and that they were intended to be explicit and shared. And surely a “taxon” for a notion like “mammal” is not so different from a “concept” for the same notion. Moreover, both the concepts and the taxa are organized into a hierarchy that takes the form of a tree—of a DAG (direct acyclic graph) if multi-inheritance (see the next section) is admitted. For a (pragmatic) distinction between “taxonomies” and “ontologies,” we must then rely on the “scope” of the definitions associated with the taxa/concepts.

Speaking for simplicity’s sake, from this time onward, of concepts, in a taxonomy (and in the most simple types of ontologies) the implicit definition of a concept derives simply by the fact of being inserted in a network of specific/generic relationships with the other concepts of the taxonomy/hierarchy. This means that a concept like company_ is defined by the fact of being, at the same time, a specific term of a higher order concept like social_body (company_ is subsumed by social_body) and a generic term with respect to a specialized concept like computer_company (company_ subsumes computer_company).

To get now a “real” ontology, we must supply also some explicit definitions for the concepts—or at least for a majority among them. This can be obtained, e.g., by associating a “frame” (a set of properties(attributes with associated classes of admitted values; see the next section) with these concepts. For example, if we consider that properties useful to better specify the concept company_ could be, among other things, DateOfCreation and DomainOfActivity, we will associate such properties (slots) with the concept. We will also impose, at the same time, that when specific examples (“instances”) of the concept company_ will be created, the slot DateOfCreation will only be filled by instances of another concept of the hierarchy, like date_, and the DomainOfActivity slot with instances of a second concept, like market_sector. A definition mechanism like this is totally extraneous to a classical Linnaean taxonomy.

In the next sections, we will outline the main principles of the “classic” ontology’s theory as it has been developed in the artificial intelligence domain. A companion article of this encyclopedia, “Using Semantic Web Tools for Ontologies Construction” deals, on the contrary, with the new developments originated by the use of ontologies as the basic knowledge representation tool in a Semantic Web context.
BACKGROUND

Inheritance Hierarchies

In this subsection, we will deal with the general “architectural” issues related to the construction of well-formed hierarchies of concepts—both ontologies and taxonomies. Ontologies/taxonomies are structured as “inheritance” hierarchies, making use of the well-known IsA link—called also AKindOf (AKo), SuperC, etc.; see Figure 1. A relatively unchallenged—see, however, Brachman (1983)—semantic interpretation of IsA states that this relationship among concepts, when noted as (IsA B A), means that concept B is a specialization of the more general concept A. In other terms, A subsumes B. This assertion can be expressed in logical form as:

\[ \forall x (B(x) \rightarrow A(x)) \]  

(1) says that if any elephant_ _ (B) IsA mammal_ _ (A), and if clyde_ is an elephant_, then clyde_ is also a mammal_. In this section, we will adopt the convention of writing down the concepts_ in italics and their instances_ (e.g., clyde_, an “individual”) in roman characters. When (1) is interpreted strictly, it also implies that a given concept B and all its instances must inherit all the features (properties) and their values of all the concepts C in the hierarchy that have B as a specialization; we speak in this case of “strict inheritance.” Note that, under the strict inheritance hypothesis, totally new properties can be added to B to differentiate it (specialize it) with respect to its parents.

Relation IsA is transitive: This means that, e.g., having both \( \forall x (C(x) \rightarrow B(x)) \) and \( \forall x (B(x) \rightarrow A(x)) \), we can deduce from this that \( \forall x (C(x) \rightarrow A(x)) \). This property is particularly important because it allows, in an inheritance hierarchy like that of Figure 1, one to represent explicitly only the IsA relationships that associate directly two nodes (i.e., without the presence of intermediary nodes). All the residual IsA relationships are then explicitly derived only when needed: E.g., from Figure 1 and from the transitive property of IsA, we can explicitly assert that (IsA chow_ mammal_).

The necessary complement of IsA for the construction of well-formed hierarchies concerns some form of InstanceOf link, used to introduce the “instances” (concrete examples) of the general notions represented by the concepts. The difference between (IsA B A) and (InstanceOf C B) is normally explained in terms of the difference between the two options of (i) considering B as a subclass of A in the first case, operator “\( \subset \),” and (ii) considering C as a member of the class B in the second, operator “\( \epsilon \).” Unfortunately, this is not sufficient to eliminate any ambiguity about the notion of instance, which is much more controversial than that of concept. Problems about the definition of instances concern, e.g., (i) the possibility of accepting that concepts (to the exclusion of the root) could also be considered as “instances” of their generic concepts and (ii) the possibility of admitting several levels of instances, i.e., instances of an instance. For a discussion about these problems and the possible solutions, see Bertino, Catania, and Zarri (2001, p. 138).

The precise definition of the meaning of InstanceOf is not the only problem that affects the construction and use of inheritance hierarchies, especially when the inheritance considered is more a “behavioral” than a “structural” one, i.e., is more interested in the actual behavior and meaning of the properties inherited than in the pure mechanical aspects of the propagation. From this point of view, we have to face two main problems: “overriding” (or “defeasible inheritance,” or “inheritance with exceptions”) and “multiple inheritance”; we will discuss overriding in some depth.

Overriding consists in the possibility of admitting exceptions to the “strict inheritance” interpretation of (1). Let us consider this group of assertions:

- a. Elephants are grey, except for royal elephants.
- b. Royal elephants are white.
- c. All the royal elephants are elephants.

Assertion (c) introduces a new concept, royal_elephant, as a specialization of elephant_ of Figure 1. If now (InstanceOf clyde_ royal_elephant), the strict inheritance law would lead us to conclude that the property (slot; see the next subsection) ColourOf of clyde_ is “filled” with the value gray_, but from (a) and (b) we know that the correct filler is instead white_. This means that royal_elephant has an “overriding property.”
Related Content

**Natural Language Front-End for a Database**
[www.igi-global.com/chapter/natural-language-front-end-database/11180?camid=4v1a](www.igi-global.com/chapter/natural-language-front-end-database/11180?camid=4v1a)

**Theories of Meaning in Schema Matching: A Review**
[www.igi-global.com/article/theories-meaning-schema-matching/3391?camid=4v1a](www.igi-global.com/article/theories-meaning-schema-matching/3391?camid=4v1a)

**A Study of Open Source Software Development from Control Perspective**
[www.igi-global.com/article/study-open-source-software-development/49722?camid=4v1a](www.igi-global.com/article/study-open-source-software-development/49722?camid=4v1a)

**CODAR: A POA-Based CORBA Database Adapter for Web Service Infrastructures**
[www.igi-global.com/chapter/codar-poa-based-corba-database/31431?camid=4v1a](www.igi-global.com/chapter/codar-poa-based-corba-database/31431?camid=4v1a)