Ontology Languages for the Semantic Web

Asunción Gómez-Pérez and Oscar Corcho, *Universidad Politécnica de Madrid*

The authors analyze the most representative ontology languages created for the Web and compare them using a common framework.

Ontologies have proven to be an essential element in many applications. They are used in agent systems, knowledge management systems, and e-commerce platforms. They can also generate natural language, integrate intelligent information, provide semantic-based access to the Internet, and extract information from texts in addition to being used in many other applications to explicitly declare the knowledge embedded in them.

However, not only are ontologies useful for applications in which knowledge plays a key role, but they can also trigger a major change in current Web contents. This change is leading to the third generation of the Web—known as the Semantic Web—which has been defined as "the conceptual structuring of the Web in an explicit machine-readable way."[1] This definition does not differ too much from the one used for defining an ontology: "An ontology is an explicit, machine-readable specification of a shared conceptualization."[2]

In fact, new ontology-based applications and knowledge architectures are developing for this new Web. A common claim for all of these approaches is the need for languages to represent the semantic information that this Web requires—solving the heterogeneous data exchange in this heterogeneous environment. Here, we don't decide which language is best of the Semantic Web. Rather, our goal is to help developers find the most suitable language for their representation needs.

Ontology languages

Several ontology languages have been developed during the last few years, and they will surely become ontology languages in the context of the Semantic Web. Some of them are based on XML syntax, such as Ontology Exchange Language (XOL),[3] SHOE[4] (which was previously based on HTML), and Ontology Markup Language (OML),[5] whereas Resource Description Framework (RDF)[6] and RDF Schema[7] are languages created by World Wide Web Consortium (W3C) working groups. Finally, two additional languages are being built on top of RDF(S)—the union of RDF and RDF Schema—to improve its features: Ontology Inference Layer (OIL)[8] and DAML+OIL[9] (see Figure 1). Other languages have also been used, traditionally, for building ontologies, but that analysis is out of the scope of this article.

XML-based Ontology Exchange Language

The US bioinformatics community designed XOL for the exchange of ontology definitions among a heterogeneous set of software systems in their domain. Researchers created it after studying the representational needs of experts in bioinformatics. They selected Ontolingua and OML as the basis for creating XOL, merging the high expressiveness of OKBC-Lite, a subset of the Open Knowledge Based Connectivity protocol, and the syntax of OML, based on XML. There are no tools that allow the development of ontologies using XOL. However, since XOL files use XML syntax, we can use an XML editor to author XOL files.

Simple HTML Ontology Extension

SHOE, developed at the University of Maryland and used to develop OML, was created as an extension of HTML, incorporating machine-readable semantic knowledge in HTML documents or other Web documents. Recently, the University of Maryland has adapted the SHOE syntax to XML. SHOE makes it possible for agents to gather meaningful infor-

1094-7167/02/$17.00 © 2002 IEEE

mation about Web pages and documents, improving search mechanisms, and knowledge gathering. This process consists of three-phases: Define an ontology, annote HTML pages with ontological information to describe themselves and other pages, and have an agent semantically retrieve information by searching all the existing pages and keeping information updated.

The Knowledge Annotator annotates ontological information in HTML pages. [4]

Ontology Markup Language

OML, developed at the University of Washington, is partially based on SHOE. In fact, it was first considered an XML serialization of SHOE. Hence, OML and SHOE share many features.

Four different levels of OML exist: *OML Core* is related to logical aspects of the language and is included by the rest of the layers; *Simple OML* maps directly to RDF(S); *Abbreviated OML* includes conceptual graphs features; and *Standard OML* is the most expressive version of OML. We selected *Simple OML*, because the higher layers don't provide more components than the ones identified in our framework. These higher layers are tightly related to the representation of conceptual graphs.

There are no other tools for authoring OML ontologies other than existing general-purpose XML edition tools.

Resource Description Framework and RDF Schema

RDF, developed by the W3C for describing Web resources, allows the specification of the semantics of data based on XML in a standardized, interoperable manner. It also provides mechanisms to explicitly represent services, processes, and business models, while allowing recognition of nonexplicit information.

The RDF data model is equivalent to the semantic networks formalism. It consists of three object types: *resources* are described by RDF expressions and are always named by URIs plus optional anchor IDs; *properties* define specific aspects, characteristics, attributes, or relations used to describe a resource; and *statements* assign a value for a property in a specific resource (this value might be another RDF statement).

The RDF data model does not provide mechanisms for defining the relationships between properties (attributes) and resources—this is the role of RDFS. RDFS offers primitives for defining knowledge models that are closer to frame-based approaches. RDF(S) is widely used as a representation format in many tools and projects, such as Amaya, Protégé, Mozilla, SilRI, and so on.

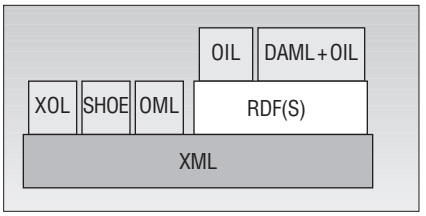


Figure 1. The languages stack in the Semantic Web.

Ontology Interchange Language

OIL, developed in the OntoKnowledge project (www.ontoknowledge.org/OIL), permits semantic interoperability between Web resources. Its syntax and semantics are based on existing proposals (OKBC, XOL, and RDF(S)), providing modeling primitives commonly used in frame-based approaches to ontological engineering (concepts, taxonomies of concepts, relations, and so on), and formal semantics and reasoning support found in description logic approaches (a subset of first order logic that maintains a high expressive power, together with decidability and an efficient inference mechanism).

OIL, built on top of RDF(S) (see Figure 1), has the following layers: *Core OIL* groups the OIL primitives that have a direct mapping to RDF(S) primitives; *Standard OIL* is the complete OIL model, using more primitives than the ones defined in RDF(S); *Instance OIL* adds instances of concepts and roles to the previous model; and *Heavy OIL* is the layer for future extensions of OIL.

OILEd, Protégé2000, and WebODE can be used to author OIL ontologies. OIL's syntax is not only expressed in XML but can also be presented in ASCII. We use ASCII for our examples.

DARPA Agent Markup Language+OIL

DAML+OIL has been developed by a joint committee from the US and the European Union (IST) in the context of DAML, a DARPA project for allowing semantic interoperability in XML. Hence, DAML+OIL shares the same objective as OIL.

DAML+OIL is built on RDF(S). Its name implicitly suggests that there is a tight relationship with OIL. It replaces the initial specification, which was called DAML-ONT, and was also based on the OIL language. OILEd, OntoEdit, Protégé2000, and WebODE are tools that can author DAML+OIL ontologies.

Comparison criteria

We focus on the identifying components that can be represented in each language and their meaning and which components can be specified, because inference depends not only on the language but also on the tools that are being currently developed around the language (which is out of the scope of this paper).[10] We will use the same set of criteria to compare different ontology languages, based on different representation formalisms.

Ontology knowledge can be specified using five components: concepts (which are usually organized by taxonomies), relations, functions, axioms, and instances.

Concepts

Concepts can be abstract or concrete, elementary or composite, real or fictitious; in short, a concept can be anything about which something is said, and, therefore, could also be the description of a task, function, action, strategy, reasoning process, and so on. Concepts are also known as classes (XOL, RDF(S), OIL, DAML+OIL), objects (OML) or categories (SHOE). When we evaluate concepts, we consider several questions.

Is it possible to define partitions *(sets of disjoint concepts)?* This is an important feature, especially for agents that reason with the information in the ontology. They won't allow an instance to be an instance of two concepts that belongs to a partition. For example, concepts *Table* and *Chair* can define a partition in a furniture ontology.

Is it possible to define the documentation *of the concept?* This includes a natural language definition for it. This is a desirable feature for maintenance and readability purposes.

Does the language provide mechanisms to define concept attributes? Attributes are also known as slots (XOL), functions (OML), or properties (RDF(S) and DAML+OIL). Binary relations and roles (in SHOE and OIL) can be used to specify attributes. The following attributes have been identified:

- *Instance attributes:* Attributes whose value might be different for each instance of the concept.
- *Class attributes:* Attributes whose value is attached to the concept, that is, its value will be the same for all instances of the concept.
- *Local attributes:* Same-name attributes attached to different concepts. For example, concepts *Chair* and *Table* might have an attribute with name *color* without clashes.

- *Global attributes:* Ones in which the domain is not specified. They can be applied to any concept in the ontology. For example, if we decide to create attribute *color* as a global attribute, it will be applied to all concepts in the ontology.

Instance and class attributes are commonly used in concept descriptions. The need for local and global attributes will depend on the application's representation needs.

Does the language provide the following predefined facets for attributes at a class?

- *Default slot value* (used to assign a value to the attribute in case there is no explicit value defined for it)
- *Type* (used to constrain the type—also called range—of the attribute)
- *Cardinality constraints* (used to constrain the minimum and maximum number of values)
- *Slot documentation* (includes a natural language definition for the attribute)

Constraints on the type of attributes and their cardinality are used to determine what type of values an attribute might have and how many values it can have when applied to a class. For example, a *Product* has just one *price* (which is an integer value) and from one to five *colors* represented by strings. The default values are used in case we do not have explicit information about the value of a slot. For example, we can assume that the *discount* of a *Product* is 0 if it is not specified.

Taxonomies

Taxonomies are used to organize ontological knowledge using generalization and specialization relationships through which simple and multiple inheritance could be applied. We analyze whether the following primitives are predefined in the languages:

- *Subclass of* (also called subsumption relationship) specializes general concepts in more specific concepts.
- *Disjoint decomposition* (a partition where all its concepts are subclasses of a common concept) does not necessarily have to be complete—that is, there might be instances of this concept that are not included in any of its subclasses. For example, the partition composed of concepts *Table* and *Chair* is a disjoint decomposition of concept *Furniture*, and instances of concept *Furniture* can exist that are not instances of concepts *Table* or *Chair* (wardrobes are also pieces of furniture).
- *Exhaustive subclass decomposition* is a complete disjoint decomposition—that is, any instance of the superclass concept must be an instance of any of the concepts in the partition. For example, the partition composed of concepts *RISCMicroprocessor* and *CISCMicroprocessor* is an exhaustive subclass decomposition of the concept *Microprocessor*, as there is no instance of concept *Microprocessor* that is not an instance of concept *RISCMicroprocessor* or concept *CISCMicroprocessor*.
- *Not subclass of* might be used to state that a concept is not a specialization of another concept. This kind of knowledge is usually represented using the denial of the *subclass of* primitive.

Table 1. Definition of concepts (+ indicates a supported feature in the language; - indicates unsupported features, and +/- indicates features that need further explanation).

Concepts	XOL	SHOE	OML	RDF(S)	OIL	DAML+OIL
General Issues						
Partitions	-	-	+	-	+	+
Documentation	+	+	+	+	+	+
Attributes						
Instance attributes	+	+	+	+	+	+
Class attributes	+	-	+	-	+	+
Local scope	+	+	+	+	+	+
Global scope	+	-	+	+	+	+
Facets						
Default value	+	-	-	-	-	-
Type constraints	+	+	+	+	+	+
Cardinality constraints	+	-	-	-	+	+
Documentation	+	+	+	+	+	+

Relations and functions

Relations are an interaction between concepts of the domain and attributes. In fact, attributes can be used to define binary relations in XOL, RDF(S) and DAML+OIL. Relations are called relations in SHOE and OML, and roles in OIL.

Functions are a special kind of relation where the value of the last argument is unique for a list of values of the n–1 preceding arguments. We ask the following questions:

- Is it possible to define *arbitrary n-ary relations or functions*? If this is not possible, which is the maximum number of arguments? Examples of n-ary relations and functions can be, respectively, *purchase* (a buyer purchases a product from a seller for an amount of money) and *getTelephoneNumber* (given a name, surname, and address, the function will return the person's telephone number).
- Might the *type* of arguments be *constrained*? In the *purchase* relation just described, the types of arguments are, *Person*, *Product*, and *Number*.
- Is it possible to define any kind of *integrity constraints* to check the correctness of the arguments' value? For example, define that the *price* of a product for the *purchase* relation must be positive, or that the product that is sold must have an attribute called *soldOut* with value *False*.
- Is it possible to define *operational definitions* to infer values of arguments with procedures, formulas, or rules, or to define its semantic using axioms or rules? For example, let's think of a function called *isUnderMinimumStock*, applicable to a *Product* and a *Store*, which returns *True* if the amount of the product's pieces is under a minimum number, which is the value of the product's attribute *minimumStockPerStore*. There is a need for an operational definition (a formula or rule) to obtain the function's output value.

Axioms

Axioms model sentences that are always true and can be used for several purposes, such as constraining information, verifying correctness, or deducting new information. Axioms are also known as assertions (OML). Axioms are not widely used currently in the context of Semantic Web applications.

The need for axioms is application-dependent. Although they are not currently widely used, they will become an important factor in Semantic Web applications, because new knowledge will be deducted when looking for information, inconsistencies will be detected when processing millions of Web pages, and so on.

We focus on the following characteristics:

- Is it possible to build axioms in *first-order logic*? The order of a logic specifies what entities *ForAll* and *Exists* might quantify over. First-order logic can only quantify over sets of atomic propositions (for example, *ForAll* p . p => p), while second-order logic can quantify over functions on propositions (for example, *ForAll* f(p) . g(p) => h(p)).[11]
- Is it possible to build second-order logic axioms?
- Can we define axioms as independent elements in the ontology (independent axioms)?
- Can we embed axioms inside the definition of other elements, such as relations, concepts, and so on?

Instances

Instances represent elements in the domain attached to a specific concept. *Facts* represent a relation that holds between elements, and c*laims* represent assertions of a fact made by an instance. All these terms are used to represent elements in the domain. We asked the following questions:

- Is it possible to define *instances of concepts*? For example, *Peter* is an instance of class *Buyer*.
- Is it possible to define instances of relations (*facts*)? For example, "*Peter buys Computer2*."
- Does the language provide special mechanisms to define *claims*? For example, "*John* says that *Peter has bought Computer2*."

We will describe information in distributed resources in the Semantic Web using instances of concepts and relations. Claims are also important, because in the distributed environment of the Semantic Web, resources will be able to make whatever claims they want. Agents shouldn't interpret them as facts of knowledge, but as claims being made by a particular instance about itself or about other instances or data, which might prove to be inconsistent with others.[4]

Table 2. Definition of taxonomies (+ indicates a supported feature in the language; - for unsupported features, and +/- for features that need further explanation).

Taxonomies	XOL	SHOE	OML	RDF(S)	OIL	DAML+OIL
Subclass of	+	+	+	+	+	+
Exhaustive decompositions	-	-	+/-	-	+	+
Disjoint decompositions	-	-	+	-	+	+
Not subclass of	-	-	-	-	+	+

Results for concepts

Table 1 shows that *partitions* are only definable in OML, OIL, and DAML+OIL. XOL does not allow defining partitions, because it uses a restricted knowledge model taken from OKBC (which does not support partitions). SHOE and RDF(S) do not include it as well.

Documentation is included as a feature in all languages. In SHOE, OML, RDF(S), and DAML+OIL we can find more types of documentation that might be provided for a concept, such as short labels in all the languages (which are used to present the information about the concept's name to an agent) or *comment*, in RDF(S) and DAML+OIL.

Most languages distinguish between *instance* and *class attributes* and specify local or global scopes for classes. XOL uses labels *own* and *template* (the default is *own*) for expressing class and instance attributes. Neither SHOE nor RDF(S) distinguish instance and class attributes, and it is not possible to establish a fixed value for class attributes. In OML, there is no syntactic difference between both kinds of attributes, but assertions can be added to add a value to the class attributes and avoid instances filling them with different values. The same applies to OIL, although the value is fixed using the *has-filler* primitive. It is also necessary to fix the cardinality of the attribute to avoid instances adding more values to the attribute. Finally, DAML+OIL allows defining restrictions on attributes in a similar way to OIL, although it is not necessary to fix its cardinality.

Local scope is achieved by restricting the domain of the attributes to the specific concept where they can be applied (in OML, this is implicit, as attributes are defined inside the definition of the class). An attribute with *global scope* might be represented in all languages, except for SHOE, as an attribute or binary relation where the domain is not specified. In SHOE, you cannot define an attribute or relation without a domain. Hence, you must define it for each concept in the top of all the taxonomies of concepts, so that it is specified for all concepts through inheritance.

Only XOL supports *default values*. In OIL, fillers could be used for representing default values, but you must be careful when using them, as their behavior is different from that of default values. If you create an instance of a concept and assign a value to an attribute that had a default value, the default value disappears, whereas a value included as a filler will not disappear as a value for the attribute in the instance.

The *type* and *documentation* of attributes can be declared explicitly in all the languages. *Cardinality constraints* cannot be established in SHOE, OML, or RDF(S).

Results for taxonomies

There is just one primitive predefined in all languages and correctly handled by them, *subclass of*. Related to the rest of the primitives for taxonomies, Table 2 shows that the first languages that were created did not take into account the suitability of partitions for representing taxonomies. Hence, they didn't include primitives to model them.

Table 2 shows the exhaustive subclass decomposition (in OML, OIL, and DAML+OIL) of concept *Book* into its subclasses *Manual* and *GenericBook* (see the "Experiment" sidebar also). OIL and DAML+OIL provide built-in primitives that allow defining these kinds of decomposition. In OML, it is represented as a disjoint decomposition, and an additional assertion is needed to establish its exhaustiveness. In the rest of the languages (XOL, SHOE, and RDF(S)), we can only represent the subclass-of relationship between the concepts *Manual* and *Book*, and *GenericBook* and *Book*. There is a loss of information that should be handled with care in case we choose any of them to represent our ontologies.

OIL and DAML+OIL are the only languages that allow representing *not-subclass-of* relationships.

Results for relations and functions

Relations and functions' *arity* is the first feature that we studied. Table 3 shows that only SHOE and OML include *n-ary* relations

Table 3. Definition of relations and functions (+ indicates a supported feature in the language; - indicates unsupported features, and +/- indicates features that need further explanation).

Relations and functions	XOL	SHOE	OML	RDF(S)	OIL	DAML+OIL
n-ary relations/functions	+/-	+	+	+/-	+/-	+/-
Type constraints	+	+	+	+	+	+
Integrity constraints	-	-	+	-	-	-
Operational definitions	-	-	-	-	-	-

Table 4. Definition of axioms (+ indicates a supported feature in the language; - indicates unsupported features, and +/- indicates features that need further explanation).

Axioms	XOL	SHOE	OML	RDF(S)	OIL	DAML+OIL
First-order logic	-	+/-	+	-	+/-	+/-
Second-order logic	-	-	-	-	-	-
Independent axioms	-	-	-	-	-	-
Embedded axioms	-	-	+	-	-	-

Table 5. Definition of instances (+ indicates a supported feature in the language; - indicates unsupported features, and +/- indicates features that need further explanation).

Instances	XOL	SHOE	OML	RDF(S)	OIL	DAML+OIL
Instances of concepts	+	+	+	+	+	+
Facts	+	+	+	+	+	+
Claims	-	+	-	+/-	+/-	+/-

and functions as built-in primitives in the language. The rest of the languages do not provide primitives for declaring them: they just provide built-in primitives for binary relations, which correspond to the attributes for the concepts in the ontology. Hence, *n-ary* relations must be specified by means of concepts whose attributes correspond to the arguments of the relation.

Type constraints can be defined for arguments in all the languages. More general constraints (*integrity constraints*) cannot be defined, except for OML, which lets us define general axioms that can be applied to these arguments. OIL and DAML+OIL also provide some rich methods for establishing constraints on arguments, such as qualified constraints (for example, cardinality constraints that are only applied when the range of the relation belongs to a specific class).

Finally, *operational definitions* cannot be defined for relations and functions in any language. Other means of representing them should be used that are out of the scope of these ontology specification languages.

Results for axioms

Table 4 shows that we were just able to represent all the axioms of the ontology in OML. We could also represent the deductive axiom in SHOE, using a SHOE inference rule, as these rules allow the deduction of values for attributes.

Although they are not considered in our framework, we must comment that OIL and DAML+OIL allow defining axioms about algebraic properties of relations (symmetry, transitivity, and so on). Studies on semantic patterns are also available that present a way to include these types of axioms for RDF(S).

Results for instances

Table 5 shows that instances and facts are easily represented in all the languages. This is not the case of claims, which are just provided as built-in primitives in SHOE, by including definitions inside an instance definition (this means that the definitions are claims of the instance in which they are embedded). Claims are not predefined in RDF(S), OIL, or DAML+OIL, but their specifications show how claims could be defined using the language constructs. Additionally, any agent collecting information from a resource on the Web could consider this information as a claim made by the resource in which the information is presented.

We have considered representation and information exchange needs and the current advantages and limitations of each existing language. Representational needs vary depending on the use of ontologies in applications. The terms *lightweight* and *heavyweight* refer to two different kinds of ontologies: those ontologies where concepts (described by their attributes and are organized in taxonomies using only the *subclass-of* relationship), relations and functions, and possibly instances are the only components that are represented, and those ontologies that also contain axioms.

If we could measure the expressiveness of languages from languages that allow defining lightweight ontologies to languages that allow heavyweight ontologies, the order would be: XOL, RDF(S), SHOE, OML, OIL, and DAML+OIL.

If you need to define a lightweight ontology for your application (you are just worried about describing concepts and organizing them in taxonomies), you can use any of the languages we presented in this article. You can decide which language to use by considering existing tools for editing ontologies in it, available software that handles the language (APIs, inference engines, and so on), familiarity with the representation formalism (XOL and SHOE use frames, which might be easier to use than semantic networks (RDF(S)), conceptual graphs (OML), or description logic (OIL and DAML+OIL)), and so on.

In defining heavyweight ontologies, you must carefully select the language—a wrong selection could prevent the success of your application. We recommend starting with our framework to determine your representation needs. Once the tables have been filled with this information, you can compare them with the ones provided in this article, so you can drop languages without enough expressiveness capabilities for your application.

The second step should be based on the reasoning support needed for the application. XOL and OML have no reasoning support available, while inference engines for RDF(S), SHOE, OIL, and DAML+OIL exist. In this case, you should consider what kind of reasoning you need. Query services have been created for RDF(S), SHOE, and DAML+OIL. OIL and DAML+OIL also provide automatic classifications for detecting inconsistencies in the ontology and organizing concepts in the taxonomy.

If you still have several candidate lan-

Experiment: An Ontology for an E-Commerce Platform

We used a set of ontologies as a test set to choose the most suitable language during the first phase of the European Information Society Technologies project Multilingual Knowledge-Based European Electronic Marketplace. The tests implemented the ontologies in all the languages, analyzed the information that could be represented, and obtained guidelines on how to implement different kinds of information in all of them. The implemented codes of this ontology can be downloaded at http://delicias.dia.fi.upm.es/RoadMap/ontologies.html.

The taxonomy of concepts, their attributes (Instance attributes are preceded by * and class attributes are not preceded by any sign), and the ad-hoc relations are presented graphically (in a frame-based fashion) in Figure A.

Our aim was to build a general ontology about office material—desks, computers, and books were included, among other products. Computers and books are more specialized, so there is a disjoint decomposition of computers into Unix servers and PCs, and an exhaustive decomposition of books into manuals and generic books (although representing generic concepts is not a good practice in ontology modeling). We have created this concept for the sake of having an exhaustive partition, so there are only manuals or generic books. Several attributes have been defined for concepts in the ontology: *price* in *OfficeMaterial* can have at most one value (and by default 0); *platform* in *UnixServer* and *PC* has value UNIX and PC; *type* in *Manual*, and *theme* in *GenericBook*. All concepts in the ontology have an attribute called *name* considered as an attribute with global scope.

There are three ad-hoc relationships: *hasManual* from *Computer* to *Manual*; *purchases*, between *Buyer*, *Seller* and *OfficeFurniture*, and *isRecommended*, from *Book* to *Buyer*.

We have also defined instances for some concepts. *John* and *Peter*, as instances of *Buyer* and *Seller*, *UnixV5* as an instance of *Manual*, *UnixForDummies* as an instance of *GenericBook*, and facts between instances, such as *John purchases* the manual *UnixV5* from *Peter*.

Finally, we have included several axioms expressing constraints in our e-commerce platform. For example, "a computer that is a Unix Server has a manual that is of type Unix", and "when a person purchases a Unix Server, he/she must also purchase a manual of type Unix." There is also one axiom included for deductive purposes: "if a person buys a manual of type Unix, he or she must be recommended other generic books with theme computers."

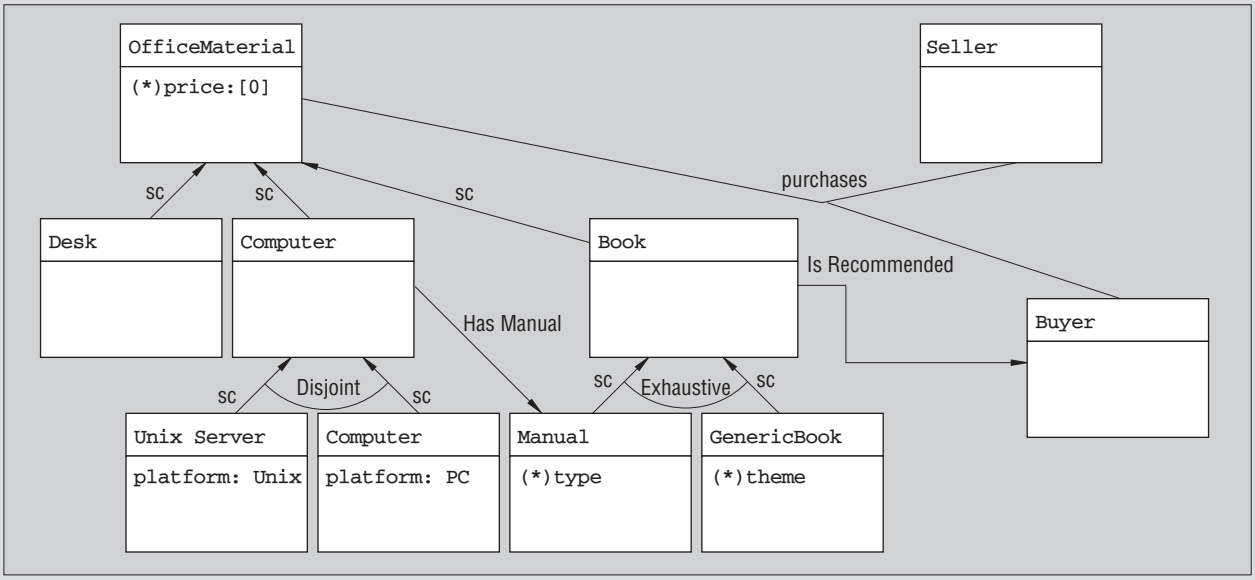


Figure A. Part of one of the sample ontologies that have been implemented in all the languages.

guages, your decision should be based on existing tools for creating ontologies and annotating resources (creating instances and facts). OIL and DAML+OIL are supported by a wide range of ontology development tools (such as OILEd, OntoEdit (only DAML+OIL), Protégé and WebODE). RDF(S) is not supported by any of these ontology tools, but by more general tools for metadata generation. There are also annotators for RDF(S), which can be also used for OIL and DAML+OIL, as instances are stored as RDF instances. The SHOE knowledge annotator allows annotating Web pages with SHOE, but no ontology editor is available for building ontologies. Finally, the rest of the languages are not supported by any specific tool.

This roadmap has been successfully used in the selection of a language for the specification of ontologies in the IST project MKBEEM (see the "Experiment" sidebar). ■

Acknowledgments

This work is supported by the CICYT project "Metodología para la Gestión de Conocimiento," reference TIC-980741, by the IST project MKBEEM (1999-10589) and by a FPI grant, funded by UPM.

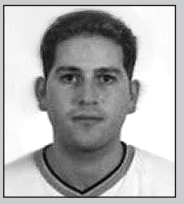
This article would not be possible without comments and feedback from developers and users of a wide range of ontology specification languages, who verified our tables: V.K. Chaudhri (XOL), Jeff Hefflin (SHOE), Ian Horrocks (OIL), Stefan Decker (FLogic), Belén Díaz (LOOM), Yolanda Gil (LOOM), Enrico Motta (OCML), James Rice (Ontolingua and OKBC), and Tom Russ (LOOM).

References

1. T. Berners-Lee and M. Fischetti, *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*, Harper, San Francisco, 1999.
2. R. Studer, R. Benjamins, and D. Fensel, "Knowledge Engineering: Principles and Methods," *IEEE Trans. on Data and Knowledge Eng.*, vol. 25, nos. 1–2, 1998, pp. 161–197.
3. R. Karp, V. Chaudhri, and J. Thomere, "XOL: An XML-Based Ontology Exchange Language (version 0.4)," Aug. 1999, www.ai.sri.com/~pkarp/xol (current Jan. 2002).
4. S. Luke and J. Heflin, *SHOE 1.01 Proposed Specification,* SHOE Project, Feb. 2000. www.cs.umd.edu/projects/plus/SHOE/spec1.01.htm (current Jan. 2002).
5. R. Kent, Conceptual Knowledge Markup Language (version 0.2). 1998. www.ontologos.org/CKML/CKML%200.2.html (current Jan. 2002).
6. O. Lassila and R. Webick, "Resource Description Framework (RDF) Model and Syntax Specification." W3C Recommendation, Jan. 1999, www.w3.org/TR/PR-rdf-syntax (current Jan. 2002).
7. D. Brickley and R.V. Guha, "Resource Description Framework (RDF) Schema Specification," W3C Proposed Recommendation, Mar. 1999, www.w3.org/TR/PR-rdf-schema (current Jan. 2002).
8. I. Horrocks et al., "OIL in a Nutshell," *Proc. ECAI '00 Workshop on Application of Ontologies and PSMs*, Berlin, Germany, 2000, pp. 4.1–4.12.
9. I. Horrocks and F. van Harmelen, *Reference Description of the DAML+OIL Ontology Markup Language*, draft report, 2001, www.daml.org/2000/12/reference.html (current Jan. 2002).
10. O. Corcho and A. Gómez-Pérez, "A Roadmap to Ontology Specification Languages," *12th Int'l Conf. Knowledge Eng. and Knowledge Management*, Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin, Oct. 2000, pp. 80–96.
11. J. Barwise, "An Introduction to First-Order Logic," *Handbook of Mathematical Logic*, J. Barwise, Amsterdam, 1977, pp. 5–46.

The Authors

Asunción Gómez-Pérez is an associate professor in the Computer Science School at Universidad Politécnica de Madrid, Spain. Her research interests include theoretical ontological foundations, methodologies for building and merging ontologies, ontological reengineering, uses of ontologies in applications related with e-commerce and knowledge management, and the Semantic Web. She received a BA in computer science, an MSc in knowledge engineering, and a PhD in computer sciences from the Universidad Politécnica de Madrid, Spain. She also received an MSc in business administration from the Universidad Pontificia de Comillas, Spain. Asunción is author of *Ontological Engineering* and is chairing the EKAW-2002 conference. Contact her at Facultad de Informática, Universidad Politécnica de Madrid, Campus de Monteguancedo s/n, Boadilla del Monte, 28660, Madrid, Spain; asun@fi.upm.es.



Oscar Corcho is an invited professor at the Universidad Pontificia de Salamanca. His research interests include ontology languages, the ontology translation problem, and the Semantic Web. He received his BA in computer science and his MSc in Software Engineering from the Polytechnic University of Madrid, and is a PhD student in Artificial Intelligence there. He belongs to the Ontology Group of the Artificial Intelligence Laboratory at the Computer Science School. Contact him at Facultad de Informática, Universidad Politécnica de Madrid, Campus de Monteguancedo s/n, Boadilla del Monte, 28660, Madrid, Spain; ocorcho@fi.upm.es.