

Chapter 2

The Most Outstanding Ontologies

This chapter is devoted to presenting the most outstanding ontologies. In this survey, we have considered different types of ontologies: knowledge representation ontologies (Section 2.1), top-level ontologies (Section 2.2), linguistic ontologies (Section 2.3) and domain ontologies (Section 2.4). In this last section, we will deal with ontologies from the following domains: e-commerce, medicine, engineering, enterprise, chemistry, and knowledge management.

At present, there is a huge number of ontologies; we have chosen those that are outstanding because of their use in important projects, their theoretical contributions, or their use as experimental bases to establish design criteria, to elaborate methodologies, etc.

After reading this chapter, you will be able to decide whether the ontologies presented here can be reused in your application, and you will know which applications are already using them.

2.1 Knowledge Representation Ontologies

A knowledge representation (KR) ontology (van Heijst et al., 1997) gathers the modeling primitives used to formalize knowledge in a KR paradigm. Examples of such primitives are *classes*, *relations*, *attributes*, etc.

The most representative KR ontology is the Frame Ontology (Gruber, 1993a), built for capturing KR conventions under a frame-based approach in Ontolingua. The Frame Ontology (FO) was modified in 1997 and some of its primitives were moved to the OKBC Ontology. The reason behind this change was the creation of OKBC (Chaudhri et al., 1998), a frame-based protocol for accessing knowledge bases stored in different languages: Ontolingua (Farquhar et al., 1997), LOOM (MacGregor, 1991), CycL (Lenat and Guha, 1990), etc.

Other ontology languages such as CycL (Lenat and Guha, 1990) and OCML (Motta, 1999) have also their own KR ontologies. In both cases, the foundations of such KR ontologies are similar to the Frame Ontology, since both languages are based on a combination of frames and first order logic. These and other languages will be described in depth in Chapter 4.

More recently, ontology markup languages have been created in the context of the Semantic Web: RDF (Lassila and Swick, 1999) and RDF Schema (Brickley and Guha, 2003), OIL (Horrocks et al., 2000), DAML+OIL (Horrocks and van Harmelen, 2001) and OWL (Dean and Schreiber, 2003). All these languages have also their corresponding KR ontologies. In this section we present these KR ontologies with their current primitives, as in April 2003¹.

2.1.1 The Frame Ontology and the OKBC Ontology

The Frame Ontology (Gruber, 1993a) was developed in KIF (Genesereth and Fikes, 1992) by the Knowledge Systems Laboratory at Stanford University. The Frame Ontology (FO) collects common knowledge-organization conventions used in frame-based representations. Its goal is to unify the semantics of the primitives most commonly used in the frame paradigm and to enable ontology developers to build ontologies with a frame-based approach.

The first version of the FO contained an axiomatization of classes and instances, slots and slot constraints, class and relation specialization primitives, relation inverses, relation composition, and class partitions. The FO was described by a set of ontological commitments that restricted the semantics of the FO primitives. Some examples of these ontological commitments are: relations are sets of tuples, functions are a special case of relations, classes are unary relations, etc.

The FO was modified in 1997 and some of its primitives were shifted to the OKBC Ontology. The reason for this change was the creation of a frame-based protocol to access knowledge bases stored in different languages: Ontolingua (Gruber, 1993a), LOOM (MacGregor, 1991), CycL (Lenat and Guha, 1990), etc., and the result was that the OKBC Ontology (Chaudhri et al., 1998) replaced some of the fundamental definitions of the original FO. At present the FO includes the OKBC Ontology and only provides formal definitions of the primitives not included in the latter. Such inclusion reflects that ontologies built with the FO primitives are more expressive than those built with the OKBC primitives. For instance, concept taxonomies built using OKBC primitives are based only on the *Subclass-Of* relation. However, concept taxonomies built using the FO may contain knowledge with exhaustive and disjoint partitions. Besides, the OKBC primitives are mainly concerned with frames, classes, and slots, while the FO includes more complex primitives for representing functions, relations, and axioms.

Both, the FO and the OKBC ontologies are available in the Ontolingua Server's ontology library². Figures 1.11 and 1.12 presented the vocabulary provided by both

¹ Some ontology markup languages, such as RDF(S) and OWL, are not fully stable yet. Their specification together with their KR primitives may undergo small changes in the future.

² <http://ontolingua.stanford.edu>

KR ontologies. The FO contains 23 classes, 31 relations, and 13 functions; the OKBC Ontology contains eight classes, 36 relations, and three functions. We will not present all the primitives contained in these two ontologies, but only the most representative.

When building a concept taxonomy using the FO and the OKBC Ontology, the following primitives can be used:

Classes, class partitions and instances

In the frame-based KR paradigm, two types of frames can be represented: classes and instances. On the one hand, classes (aka concepts) represent collections or stereotypes of objects. On the other hand, instances represent individuals belonging to one or to several of those classes. The latter are called *individuals* in the OKBC Ontology. Two of the primitives, related to classes and instances, identified in the FO and the OKBC Ontology are:

- ⌘# *Class* (?Class). This primitive defines the class ?Class as a collection of individuals. It is the only primitive appearing in both ontologies.
- ⌘# *Individual* (?Individual). This primitive defines an individual or instance.

Class taxonomies

Taxonomies are used to organize classes and instances in the ontology. The most important relations here are *Subclass-Of* (which means that a class is a specialization of another class) and *Instance-Of* (which states that an individual is an element of a class). Both primitives and some more specific ones for creating taxonomies are described below.

- ⌘# *Subclass-Of* (?Child-Class ?Parent-Class), which states that the class ?Child-Class is a subclass of the class ?Parent-Class.
- ⌘# *Superclass-Of* (?Parent-Class ?Child-Class), which states that the class ?Parent-Class is a superclass of the class ?Child-Class. This relation is the inverse relation of the *Subclass-Of* relation.
- ⌘# *Disjoint-Decomposition* (?Class ?Class-Set), which defines the set of disjoint classes ?Class-Set as subclasses of the class ?Class. This classification does not necessarily have to be complete, that is, there may be instances of ?Class that are not instances of any of the classes of ?Class-Set.
- ⌘# *Exhaustive-Decomposition* (?Class ?Class-Set), which defines the set of classes ?Class-Set as subclasses of the class ?Class. This classification is complete, that is, there are no instances of ?Class that are not instances of any of the classes of ?Class-Set. However, the classes in the set ?Class-Set are not necessarily disjoint, as with the previous primitive.
- ⌘# *Partition* (?Class ?Class-Set), which defines the set of disjoint classes ?Class-Set as subclasses of the class ?Class. This classification is complete, that is, the class ?Class is the union of all the classes that belong to ?Class-Set.
- ⌘# *Instance-Of* (?Individual ?Class), where the instance ?Individual is an instance of the class ?Class.

Figure 2.1 shows examples where we use some of these primitives for creating class taxonomies. The class `AmericanAirlinesFlight` is a subclass of the class `Flight`. Hence, the class `Flight` is a superclass of the class

AmericanAirlinesFlight. The classes AA7462, AA2010 and AA0488 form a disjoint decomposition of the class AmericanAirlinesFlight (that is, there are no flights operated by American Airlines that have two flight numbers from the set AA7462, AA2010 and AA0488, but there are also other kinds of flights operated by American Airlines). The classes EuropeanLocation, AsianLocation, AfricanLocation, AustralianLocation, AntarcticLocation, NorthAmericanLocation, and SouthAmericanLocation form a partition of the class Location (any location belongs to one, and only one, of the seven continents). Finally, NewYorkCity is an instance of the class NorthAmericanLocation.

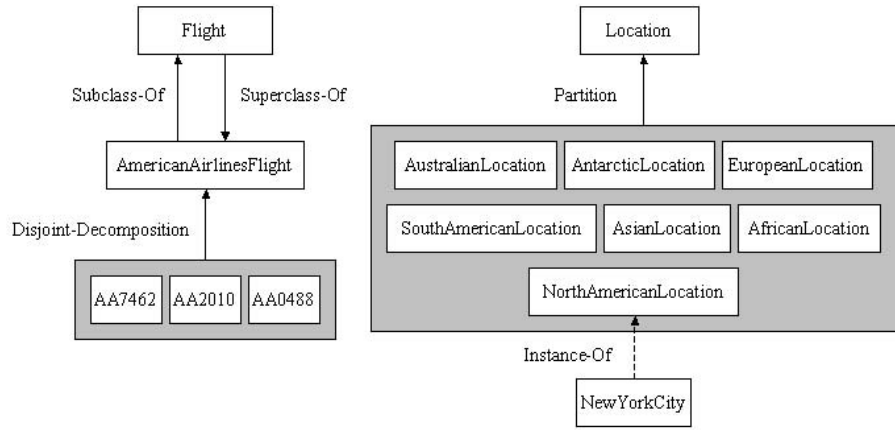


Figure 2.1: Examples of taxonomies with the FO and the OKBC Ontology primitives.

Relations and their properties

A relation represents the dependency between concepts in the domain. In Mathematics, relations are formally defined as sets of tuples of individuals. Relations in an ontology can be organized in relation taxonomies according to a specialization relationship, called *Subrelation-Of*. Several mathematical properties of a relation can also be determined: reflexive, irreflexive, symmetric, etc. Some of the primitives for defining relations identified in the FO are:

- ⊄# *Relation (?Rel)*, which defines a relation *?Rel* in the domain. The classes to which the relation applies are defined as the domain and range of the relation respectively.
- ⊄# *Subrelation-Of (?Child-Rel ?Parent-Rel)*. A relation *?Child-Rel* is a subrelation of the relation *?Parent-Rel* if, viewed as sets, *?Child-Rel* is a subset of *?Parent-Rel*. In other words, every tuple of *?Child-Rel* is also a tuple of *?Parent-Rel*, that is, if *?Child-Rel* holds for some arguments *arg_1*, *arg_2*, ..., *arg_n*, then *?Parent-Rel* holds for the same arguments. Thus a relation and its subrelation must have the same arity, which could be undefined.
- ⊄# *Reflexive-Relation (?Rel)*. Relation *?Rel* is reflexive if *?Rel(x,x)* holds for all *x* in the domain and range of *?Rel*.

- ⊥# *Irreflexive-Relation (?Rel)*. Relation *?Rel* is irreflexive if *?Rel(x,x)* never holds for all *x* in the domain and range of *?Rel*.
- ⊥# *Symmetric-Relation (?Rel)*. Relation *?Rel* is symmetric if *?Rel(x,y)* implies *?Rel(y,x)* for all *x* and *y* in the domain and range of *?Rel*.
- ⊥# *Antisymmetric-Relation (?Rel)*. Relation *?Rel* is antisymmetric if *?Rel(x,y)* implies *not ?Rel(y,x)* when *x* \neq *y*, for all *x* and *y* in the domain and range of *?Rel*.
- ⊥# *Asymmetric-Relation (?Rel)*. Relation *?Rel* is asymmetric if it is antisymmetric and irreflexive over its exact domain. The exact domain of *?Rel* is the set elements of the *?Rel* domain linked to some element of the *?Rel* range through this relation; that is, the exact domain only keeps the domain elements that participate in the relation.
- ⊥# *Transitive-Relation (?Rel)*. Relation *?Rel* is transitive if *?Rel(x,y)* and *?Rel(y,z)* implies *?Rel(x,z)*, for all *x* and *z* in the domain and range of *?Rel* respectively, and for all *y* in the domain and range of *?Rel*.
- ⊥# *Equivalence-Relation (?Rel)*. Relation *?Rel* is an equivalence relation if it is reflexive, symmetric, and transitive.
- ⊥# *Partial-Order-Relation (?Rel)*. Relation *?Rel* is a partial-order relation if it is reflexive, antisymmetric, and transitive.
- ⊥# *Total-Order-Relation (?Rel)*. Relation *?Rel* is a total-order relation if it is a partial-order relation for which either *?Rel(x,y)* or *?Rel(y,x)* holds for every *x* or *y* in its exact domain.

As shown in Figure 1.12, these primitives for defining relations have been represented in the FO as classes, and they are organized in a class taxonomy. For example, the class *Asymmetric-Relation*, which represents the collection of relations that are asymmetric, is a subclass of the classes *Antisymmetric-Relation* and *Irreflexive-Relation*. This specialization relationship can be extracted from the definition of what an asymmetric relation is, as described above.

Slots

A slot (aka attribute) defines a characteristic of a class, which is also inherited by its subclasses. Attributes can be defined with the following two primitives of the OKBC Ontology:

- ⊥# *Template-Slot-Of (?Slot ?Class)*, which states that *?Slot* is a slot of *?Class*. The slot *?Slot* can take different values in the different instances of *?Class*.
- ⊥# *Slot-Of (?Slot ?Frame)*, which states that *?Slot* is a slot of *?Frame*. *?Frame* can be either a *class* or an *individual*.

Facets and types of facets

A facet is a slot property. In the FO facets are defined as ternary relations that hold between a frame (which can be either a *class* or an *individual*), a slot, and the facet. Common facets in the frame-based KR paradigm are, for example, those that define the cardinality of a slot, the type of a slot, and default values. Some of the primitives related to facets that are identified in the OKBC Ontology are:

- ⊥# *Facet-Of (?Facet ?Slot ?Frame)*, where *?Facet* is facet of the slot *?Slot* in the frame *?Frame*.

- ⌘# *Minimum-Cardinality* (?Slot ?Frame ?Number), which expresses that ?Number is the minimum cardinality of the slot ?Slot in the frame ?Frame.
- ⌘# *Maximum-Cardinality* (?Slot ?Frame ?Number), which expresses that ?Number is the maximum cardinality of the slot ?Slot in the frame ?Frame.

Chapter 4 shows how to build ontologies with the FO and with the OKBC primitives in Ontolingua.

2.1.2 RDF and RDF Schema knowledge representation ontologies

RDF (Lassila and Swick, 1999) stands for *Resource Description Framework*. It is a recommendation of the W3C (the World Wide Web Consortium), developed for describing Web resources with metadata.

The RDF data model is equivalent to the semantic network KR paradigm, as explained by Staab and colleagues (2000), and by Conen and Klapsing (2001). A semantic network is a directed labeled graph composed of a set of nodes and a set of unidirectional edges, and each has a name. Nodes represent concepts, instances of concepts and property values. Edges represent properties of concepts or relationships between concepts. The semantics of the network depends on the node and edge names. The semantic network KR paradigm has less expressiveness than the frame-based KR paradigm, since it does not allow representing, for instance, default values and cardinality constraints on attributes.

The RDF data model consists of three components:

- ⌘# *Resources*, which are any type of data described by RDF. Resources are described with RDF expressions and are referred to as URIs (*Uniform Resource Identifiers*) plus optional anchor identifiers.
- ⌘# *Properties* (aka predicates), which define attributes or relations used to describe a resource.
- ⌘# *Statements*, which assign a value to a property in a specific resource. Just as an English sentence usually comprises a subject, a verb and objects, RDF statements consist of subjects, properties and objects. For instance, in the sentence “*John bought a ticket*”, John is the subject, bought is the verb, and ticket is the object. If we represent this sentence in RDF, John and ticket are resources, denoted graphically by nodes, while bought is a property, denoted graphically by an edge.

Not only can resources be the objects of a RDF statement, but RDF statements can also be objects themselves. For example, in the sentence “*John said that Peter bought a ticket*”, John is the subject, said is the property and Peter bought a ticket is the object, which can also be decomposed, as we did before. This is known as reification in RDF.

It is important to note that the RDF data model does not make any assumption about the structure of a document containing RDF information. That is, the statements can appear in any order in a RDF ontology.

The RDF KR ontology³ is written in RDFS (which will be presented later in this section) and contains the following modeling primitives⁴ (seven classes, seven properties, and one instance):

- ⌘ Class *rdf:Statement*. As we have commented, it defines the class of triples containing a subject, a property and an object.
- ⌘ Class *rdf:Property*. It defines the class of properties.
- ⌘ Classes *rdf:Bag*, *rdf:Seq* and *rdf:Alt*. They define the classes of collections (aka containers), and these can be unordered, ordered and alternative respectively. While it is clear what we mean by unordered and ordered collections, alternative collections may not be so. An alternative collection contains a set of resources from which we must select one for the single value of a property. For example, an alternative collection could be used to represent the values “single”, “double” or “triple” for the attribute occupancy of a RoomReservation.
- ⌘ Class *rdf:List*, properties *rdf:first* and *rdf:rest*, and instance *rdf:nil*. The class *rdf:List* defines the class of RDF lists. It is used with the properties *rdf:first* and *rdf:rest*, which represent the relationship between a list and its first item, and between the list and the rest of the list, respectively. The primitive *rdf:nil* is an instance of *rdf:List* that represents the empty list.
- ⌘ Class *rdf:XMLLiteral*. It is a datatype that defines the class of well-formed XML literal values.
- ⌘ Properties *rdf:predicate*, *rdf:subject* and *rdf:object*. They define the property, subject resource, and object resource of a statement respectively.
- ⌘ Property *rdf:type*. It defines the class to which a resource belongs.
- ⌘ Property *rdf:value*. It defines the value of a property, usually a string, when the value is a structured resource (another RDF statement).

The RDF data model does not provide modeling primitives for defining the relationships between properties and resources. For instance, in RDF we cannot define that the relation *arrivalPlace* can only hold between instances of the classes *Travel* and *Location*. This limitation is solved by the RDF Vocabulary Description Language (Brickley and Guha, 2003), also known as RDF Schema or RDFS. RDFS is a working draft of the W3C that extends RDF with frame-based primitives. The combination of RDF and RDF Schema is usually known as RDF(S).

The RDFS KR ontology⁵ is written in RDFS. It contains 16 new modeling primitives (six classes and nine properties) added to the RDF modeling primitives. Figure 2.2 shows the class taxonomy of the RDF(S) KR ontology. As we can see, there are 13 classes in this KR ontology. The top concept in the class taxonomy is

³ <http://www.w3.org/1999/02/22-rdf-syntax-ns>. At the time this description was written, the RDF KR ontology available at this URL was not yet compliant with the specification of RDF given by Lassila and Swick (1999) and extended by Brickley and Guha (2003). We have described this KR ontology based on the last document instead of the implemented KR ontology.

⁴ In this section, we will use the prefix *rdf* to refer to RDF primitives and *rdfs* to refer to RDF Schema primitives.

⁵ <http://www.w3.org/2000/01/rdf-schema>. As in the case of the RDF KR ontology, the ontology available at this URL is not yet compliant with the specification of RDF Schema given by Brickley and Guha (2003). We have described this KR ontology based on the document instead of the implemented KR ontology.

rdfs:Resource (which means that RDF statements, RDFS containers, RDFS classes, RDF properties, and RDFS literals are RDFS resources). The classes *rdf:Bag*, *rdf:Seq* and *rdf:Alt* are subclasses of *rdfs:Container*. The class *rdfs:Datatype* is a subclass of *rdfs:Class*, and the class *rdf:List* is defined apart in the class taxonomy.

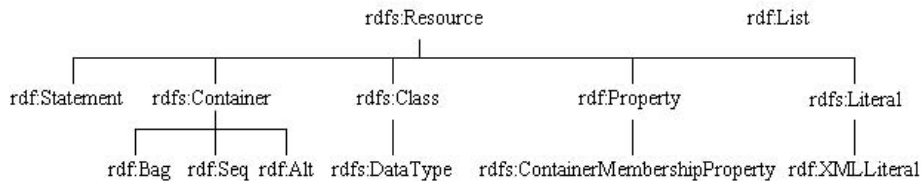


Figure 2.2: Class taxonomy of the RDF(S) KR ontology.

Table 2.1 summarizes the main features of the properties of the RDF(S) KR ontology. As we can see in this table, there are 16 properties defined in this KR ontology. In this table, we specify their domain and range, that is, the classes between which these properties can hold.

Table 2.1: Property descriptions of the RDF(S) KR ontology.

Property name	domain	range
<i>rdf:type</i>	<i>rdfs:Resource</i>	<i>rdfs:Class</i>
<i>rdf:subject</i>	<i>rdf:Statement</i>	<i>rdfs:Resource</i>
<i>rdf:predicate</i>	<i>rdf:Statement</i>	<i>rdf:Property</i>
<i>rdf:object</i>	<i>rdf:Statement</i>	<i>rdfs:Resource</i>
<i>rdf:value</i>	<i>rdfs:Resource</i>	<i>rdfs:Resource</i>
<i>rdf:first</i>	<i>rdf:List</i>	<i>rdfs:Resource</i>
<i>rdf:rest</i>	<i>rdf:List</i>	<i>rdf:List</i>
<i>rdfs:subClassOf</i>	<i>rdfs:Class</i>	<i>rdfs:Class</i>
<i>rdfs:subPropertyOf</i>	<i>rdf:Property</i>	<i>rdf:Property</i>
<i>rdfs:comment</i>	<i>rdfs:Resource</i>	<i>rdfs:Literal</i>
<i>rdfs:label</i>	<i>rdfs:Resource</i>	<i>rdfs:Literal</i>
<i>rdfs:seeAlso</i>	<i>rdfs:Resource</i>	<i>rdfs:Resource</i>
<i>rdfs:isDefinedBy</i>	<i>rdfs:Resource</i>	<i>rdfs:Resource</i>
<i>rdfs:member</i>	<i>rdfs:Resource</i>	<i>rdfs:Resource</i>
<i>rdfs:domain</i>	<i>rdf:Property</i>	<i>rdfs:Class</i>
<i>rdfs:range</i>	<i>rdf:Property</i>	<i>rdfs:Class</i>

In addition to these classes and properties, RDF also uses the properties *rdf:_1*, *rdf:_2*, *rdf:_3*, etc., each of which is both a subproperty of the property *rdfs:member* and an instance of the class *rdfs:ContainerMembershipProperty*. These properties (*rdf:_1*, *rdf:_2*, *rdf:_3*, etc.) are used to specify the members of collections such as sequences, bags and alternatives, which were previously mentioned. A more simple syntax for these properties consists in using *rdf:li* instead, which is equivalent to them. All these properties are not included in the RDF nor in the RDFS KR ontologies.

The RDFS primitives are grouped into core classes and properties, container classes and properties, collections, reification vocabulary, and utility properties.

- €# **Core classes** (*rdfs:Resource*, *rdfs:Literal*, *rdf:XMLLiteral*, *rdfs:Class*, *rdf:Property*, and *rdfs:Datatype*). The class *rdfs:Resource* is the most general class and defines any Web resource that can be described by RDF. The classes *rdfs:Literal* and *rdf:XMLLiteral* represent the class of untyped literal values (such as strings and integers) and well-formed XML string values respectively. The class *rdfs:Class* defines the class of all classes. The class *rdf:Property* defines the class of properties. The class *rdfs:Datatype* represents resources that are RDF datatypes.
- €# **Core properties** (*rdf:type*, *rdfs:subClassOf*, *rdfs:subPropertyOf*, *rdfs:domain*, *rdfs:range*, *rdfs:label*, and *rdfs:comment*). The property *rdf:type* states that a resource is an instance of a class. The properties *rdfs:subClassOf* and *rdfs:subPropertyOf* are used to define class taxonomies and property taxonomies respectively. The properties *rdfs:domain* and *rdfs:range* define the domain and range of the property they are applied to. Finally, the properties *rdfs:label* and *rdfs:comment*, which were previously classified as documentation primitives, are used for describing resources in natural language. The property *rdfs:comment* is mainly for long descriptions while *rdfs:label* is for defining alternative short labels of the resource to which it is applied.
- €# **Container classes and properties** (*rdfs:Container*, *rdf:Bag*, *rdf:Seq*, *rdf:Alt*, *rdfs:ContainerMembershipProperty*, and *rdfs:member*). The class *rdfs:Container* defines the class of resource collections, which can be a bag (*rdf:Bag*), a sequence (*rdf:Seq*), or an alternative (*rdf:Alt*). These containers were described above. The class *rdfs:ContainerMembershipProperty* defines the relationship between a resource and a container. The property *rdfs:member* is used to specify the members of a container. As we explained above, the properties *rdf:_1*, *rdf:_2*, *rdf:_3*, etc., are subproperties of this property, and *rdf:li* can also be used to express them.
- €# **Collections** (*rdf:List*, *rdf:first*, *rdf:rest*, and *rdf:nil*). The class *rdf:List* is used to describe lists. The properties *rdf:first* and *rdf:rest* are used to manage lists, and *rdf:nil* is an instance of *rdf:List* that represents the empty list.
- €# **Reification vocabulary** (*rdf:Statement*, *rdf:predicate*, *rdf:subject*, and *rdf:object*). This class and these properties were described when we referred to the RDF KR ontology. As we said, the class *rdf:Statement* defines the class of triples that can be described in RDF(S), and the properties *rdf:predicate*, *rdf:subject*, and *rdf:object* define the property, subject resource, and object resource of a statement, respectively.
- €# **Utility properties** (*rdfs:seeAlso*, *rdfs:isDefinedBy*, and *rdf:value*). The property *rdfs:seeAlso* defines a resource that might give additional information about the resource being described. The property *rdfs:isDefinedBy* provides the namespace where the resource is defined and is a subproperty of *rdfs:seeAlso*. The property *rdf:value* was described when we referred to the RDF KR ontology. It defines the value of a property when that value is a structured resource.

In Chapter 4 we will describe in detail how to use all these KR primitives to implement our ontologies in RDF(S), but now we want to show an example of how to use primitives of the RDF and RDFS KR ontologies. Below we present the

definitions of the class `Flight` and of the relation `arrivalPlace`. In these definitions, primitives of the RDFS KR ontology (such as `rdfs:Class`, `rdfs:comment`, `rdfs:subClassOf`, `rdfs:domain` and `rdfs:range`) are combined with primitives of the RDF KR ontology (such as `rdf:Property`). The properties `rdf:ID` and `rdf:resource` are also used. However, they should not be considered as KR primitives since they are only used to identify RDF resources. We will describe their differences in Chapter 4. Please note that `rdf:resource`, which is used to refer to a RDF resource, should not be mistaken for `rdfs:Resource`, which is the class of RDF resources.

```
<rdfs:Class rdf:ID="Flight">
  <rdfs:comment>A journey by plane</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Travel"/>
</rdfs:Class>

<rdf:Property rdf:ID="arrivalPlace">
  <rdfs:domain rdf:resource="#Travel"/>
  <rdfs:range rdf:resource="#Location"/>
</rdf:Property>
```

2.1.3 OIL knowledge representation ontology

OIL (Horrocks et al., 2000) stands for *Ontology Inference Layer*. This language has been built as an extension of RDF(S) by adding it more frame-based KR primitives and avoiding the RDF reification mechanism. OIL uses description logics to give clear semantics to its modeling primitives.

OIL was developed using a layered approach, as shown in Figure 2.3. Each new layer is built on top of the existing ones and adds new functionality and complexity to the lower layer. *Core OIL* groups the OIL primitives that have a direct mapping to RDF(S) primitives, though it does not allow RDF(S) reification, as shown in the figure. *Standard OIL* adds frame-based primitives. Its relationship with *Core OIL*

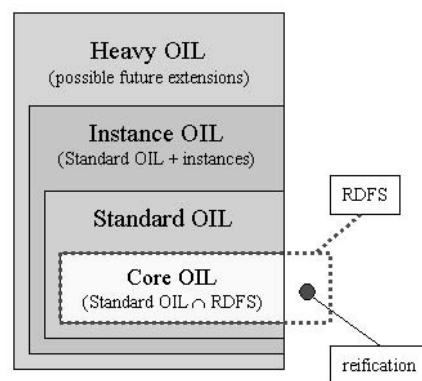


Figure 2.3: Layers of OIL.

was explained in depth by Broekstra and colleagues (2001). *Instance OIL* permits defining instances of concepts and roles and includes a full-fledged database capability. Finally, *Heavy OIL* is reserved for future extensions (such as rules, metaclasses, etc.). *Standard OIL* and *Instance OIL* share the same KR ontology. We must add that the *Heavy OIL* KR ontology has not been developed yet at the time this section was written.

Here we will only present the *Standard OIL* KR ontology⁶. This ontology consists of 37 classes and 19 properties. It is mostly written in RDF(S), except for two classes (*oil:Top* and *oil:Bottom*) written in OIL.

Figure 2.4 shows the class hierarchy of the OIL KR ontology in which we can see how this KR ontology extends the RDF(S) KR ontology. There are six groups of primitives that are classes:

- €# **Classes for defining concrete type expressions** (*oil:Equal*, *oil:Min*, *oil:Max*, *oil:GreaterThan*, *oil:LessThan*, and *oil:Range*). These primitives are subclasses of the primitive *oil:ConcreteTypeExpression*, which in its turn is subclass of *oil:Expression*. They allow defining numeric expressions for the numbers that are equal, greater or equal than, less or equal than, greater than and less than a number, as well as numeric ranges, respectively.
- €# **Classes for defining class expressions**. These primitives are defined as subclasses of the class *oil:ClassExpression*, which in its turn is subclass of *oil:Expression*. In OIL, classes can be primitive (*oil:PrimitiveClass*) or defined (*oil:DefinedClass*), and they specialize *rdfs:Class*. The difference between them was explained in Section 1.3.2, where we described how to model ontologies with description logic. Class expressions can also be formed with boolean expressions, property restrictions and enumerated expressions.
- €# With regard to **Boolean expressions** (primitives that are subclasses of *oil:BooleanExpression*), we can use three primitives: *oil:And*, *oil:Or*, and *oil:Not*. They express conjunction, disjunction, and negation of classes respectively.
- €# In relation to **property restrictions** (primitives that are subclasses of *oil:PropertyRestriction*), we can express qualified number restrictions⁷ with the primitives that are subclasses of *oil:CardinalityRestriction* (*oil:MinCardinality*, *oil:Cardinality* and *oil:MaxCardinality*). We can also express value restriction⁸ (*oil:ValueType*), existential restriction⁹ (*oil:HasValue*) and role fillers to deal with individuals (*oil:HasFiller*).

⁶ <http://www.ontoknowledge.org/oil/rdf-schema/2000/11/10-oil-standard>

⁷ A qualified number restriction defines a cardinality restriction for a role when it is applied to instances of a specific class. For example, we know that a person always has two parents, of which one is a man and the other is a woman. This is represented as a qualified number restriction of the role *hasParent*, which has cardinality 1 when it is applied to *Man* and cardinality 1 when it is applied to *Woman*.

⁸ Value restrictions are used to express that a role may have any number of values, and that these values must always be instances of the class specified in the restriction. For instance, a person can be married or not to somebody, but must always be married to a person (not to an animal).

⁹ Existential restrictions are used to express that a role must have at least one value that is an instance of the class specified in the restriction. For instance, we can define a friendly person as a person who must have at least one friend, which is another person. However, he/she can also have other friends that are not persons (such as an animal).

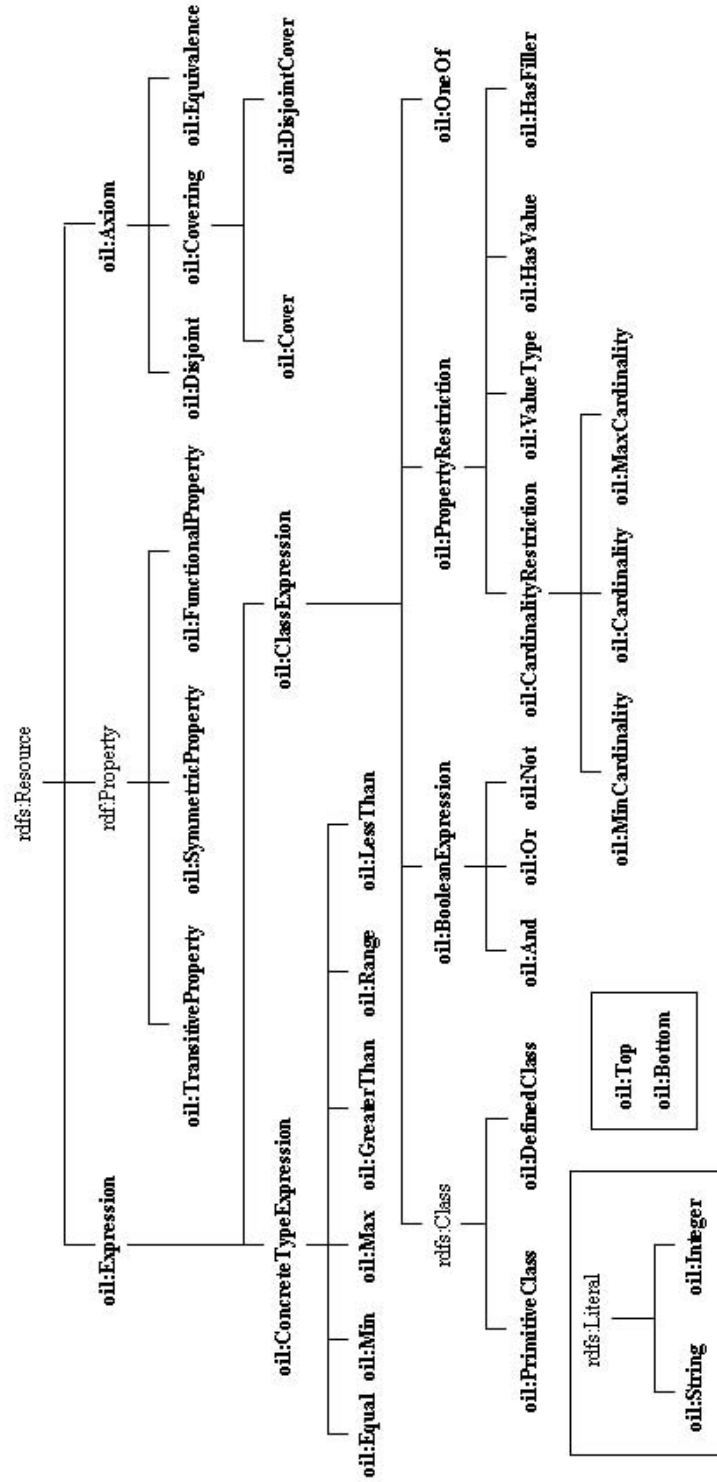


Figure 2.4: Class taxonomy of the Standard OIL KR ontology defined as an extension of RDF(S).

- €# Regarding **enumerated expressions**, we can use the class *oil:OneOf*. These primitives configure OIL as a SHIQ language¹⁰, according to the DL terminology explained in Section 1.3.2.
- €# **Classes for defining mathematical characteristics of properties** (*oil:TransitiveProperty*, *oil:FunctionalProperty* and *oil:SymmetricProperty*). They express that the property is transitive, that it can only have one value for each instance in its domain and that it is symmetric respectively. All of them specialize *rdf:Property*.
- €# **Classes for defining axioms** (primitives that are subclasses of *oil:Axiom*). These primitives are used to define disjoint and exhaustive knowledge in class taxonomies (*oil:Disjoint* and the subclasses of *oil:Covering*, which are *oil:Cover* and *oil:DisjointCover*, respectively), as well as equivalence between classes (*oil:Equivalence*). The primitive *oil:Disjoint* defines a set of classes that are disjoint, that is, that cannot have common instances. The primitive *oil:Cover* expresses that a class is the union of a set of classes, that is, that there are no instances of the class that are not instances of at least one of the classes in the set. The primitive *oil:DisjointCover* expresses that a class is the union of a disjoint set of classes.
- €# **Classes for defining datatypes** (*oil:String* and *oil:Integer*). They specialize the primitive *rdfs:Literal*, and define the datatype of strings and the datatype of integers respectively.
- €# **Predefined classes** (*oil:Top* and *oil:Bottom*). The class *oil:Top* is the most general class and subsumes every other class. The class *oil:Bottom* is the empty class and is subsumed by every other class.

There are also several primitives in the Standard OIL KR ontology that are properties. Table 2.2 summarizes the main features of these 19 properties, specifying their domain and range, that is, the classes between which these properties can hold. We will first describe these properties and later present the table.

- €# The properties *oil:subClassOf*, *oil:domain* and *oil:range*. They replace the corresponding primitives in RDF(S).
- €# The property *oil:hasOperand*. It connects a Boolean expression with the operands. It is used with the primitives *oil:And*, *oil:Or* and *oil:Not*, described above.
- €# The property *oil:individual*. It connects an *oil:OneOf* expression with its individuals.
- €# The properties *oil:hasPropertyRestriction*, *oil:onProperty*, *oil:toClass* and *oil:toConcreteType*. They are used to express the property restrictions of a class. The primitive *oil:toClass* is used with properties whose range is another class, and the primitive *oil:toConcreteType* are used with properties that are concrete types.
- €# The properties *oil:stringValue* and *oil:integerValue*. They connect a concrete type expression with a string value or an integer value respectively.

¹⁰ In Horrocks (2000) OIL appears as SHIQ(d), which means that it is a SHIQ language extended with concrete data types.

- ⊘# The properties *oil:individualFiller*, *oil:integerFiller* and *oil:stringFiller*. They represent property values.
- ⊘# The property *oil:number*. Used to express the number of a cardinality restriction.
- ⊘# The property *oil:inverseRelationOf*. Used to define the inverse of a property.
- ⊘# The properties *oil:hasObject*, *oil:hasSubject* and *oil:isCoveredBy*. They represent disjoint and exhaustive knowledge in class taxonomies.

Table 2.2: Property descriptions of the Standard OIL KR ontology.

Property name	domain	range
oil:subClassOf	rdfs:Class	oil:ClassExpression
oil:domain	rdf:Property	oil:ClassExpression
oil:range	rdf:Property	oil:ClassExpression
oil:hasOperand	oil:BooleanExpression	oil:Expression
oil:individual	oil:OneOf	rdfs:Resource
oil:hasPropertyRestriction	rdfs:Class	oil:PropertyRestriction
oil:onProperty	oil:PropertyRestriction	rdf:Property
oil:toClass	oil:PropertyRestriction	oil:ClassExpression
oil:toConcreteType	oil:PropertyRestriction	oil:ConcreteTypeExpression
oil:stringValue	oil:ConcreteTypeExpression	oil:String
oil:integerValue	oil:ConcreteTypeExpression	oil:Integer
oil:individualFiller	oil:HasFiller	rdfs:Resource
oil:stringFiller	oil:HasFiller	oil:String
oil:integerFiller	oil:HasFiller	oil:Integer
oil:number	oil:CardinalityRestriction	oil:Integer
oil:inverseRelationOf	rdf:Property	rdf:Property
oil:hasObject	oil:Axiom	oil:ClassExpression
oil:hasSubject	oil:Covering	oil:ClassExpression
oil:isCoveredBy	oil:Covering	oil:ClassExpression

In Chapter 4 we will describe in detail how to use these primitives to implement ontologies in OIL and we will use OILs plain text syntax. We now show a small example of how to use primitives of the OIL KR ontology with the XML syntax to get its flavor. Below we present the definition of the defined class `Flight`, which was described in Section 1.3.2. This class is a subclass of the class `Travel` that has exactly one value for the attribute `flightNumber`, whose type is `integer`, and that has a filler for the attribute `transportMeans` with value “plane”.

```

<oil:DefinedClass rdf:ID="Flight">
  <rdfs:comment>A journey by plane</rdfs:comment>
  <oil:subClassOf>
    <oil:And>
      <oil:hasOperand rdf:resource="#Travel"/>
      <oil:hasOperand>
        <oil:Cardinality oil:number="1">
          <oil:onProperty rdf:resource="#flightNumber"/>
          <oil:toClass rdf:resource="#oil:Integer"/>
        </oil:Cardinality>
      </oil:And>
    </oil:subClassOf>
  </oil:DefinedClass>

```

```

    </oil:hasOperand>
    <oil:hasOperand>
      <oil:HasFiller oil:stringFiller="plane">
        <oil:onProperty rdf:resource="#transportMeans"/>
      </oil:HasFiller>
    </oil:hasOperand>
  </oil:And>
</oil:subClassOf>
</oil:DefinedClass>

```

2.1.4 DAML+OIL knowledge representation ontology

Like OIL, DAML+OIL (Horrocks and van Harmelen, 2001) was developed as an extension of RDF(S). However, this language is not divided into different layers: it provides DL extensions of RDF(S) directly. DAML+OIL is a SHIQ language extended with datatypes and nominals¹¹.

The DAML+OIL KR ontology¹² is written in DAML+OIL and contains 53 modeling primitives (14 classes, 38 properties and one instance). Two of the classes (*daml:Literal* and *daml:Property*) and 10 of the properties (*daml:subPropertyOf*, *daml:type*, *daml:value*, *daml:subClassOf*, *daml:domain*, *daml:range*, *daml:label*, *daml:comment*, *daml:seeAlso* and *daml:isDefinedBy*) are equivalent to their corresponding classes and properties in RDF(S).

Figure 2.5 shows the class taxonomy of the DAML+OIL KR ontology, and how this KR ontology extends the RDF(S) KR ontology. The following groups of primitives that are classes are defined in the DAML+OIL KR ontology:

⌘ **Classes for defining classes, restrictions and datatypes** (*daml:Class*, *daml:Restriction* and *daml:DataType*). All these primitives specialize *rdfs:Class*. The primitive *daml:Class* is used to define classes. The primitive *daml:Restriction* is used to define property restrictions for classes (number restrictions, existential restrictions, qualified number restrictions, etc.). And the primitive *daml:DataType* is used to create datatypes. XML Schema datatypes (Biron and Malhotra, 2001) are permitted in DAML+OIL, and are considered subclasses of *daml:DataType*.

⌘ **Classes for defining properties** (*daml:UnambiguousProperty*, *daml:TransitiveProperty*, *daml:ObjectProperty*, and *daml:DatatypeProperty*). They are used to define properties, so they specialize the class *daml:Property* (which is equivalent to *rdf:Property*, as stated above). The primitive *daml:ObjectProperty* is used to define properties that connect a class with another class. It is specialized in the primitives *daml:TransitiveProperty* and *daml:UnambiguousProperty*, which refer to properties that are transitive and injective¹³ respectively. The primitive *daml:DatatypeProperty* is used to define properties that connect a class with a datatype. Finally, the primitive

¹¹ Also known as SHOIQ(d).

¹² <http://www.daml.org/2001/03/daml+oil>. There is another version of this ontology available at <http://www.w3.org/2001/10/daml+oil>, but the DAML+OIL developers recommend using the first one.

¹³ If the relation *R* is injective, and *R*(*x*,*y*) and *R*(*z*,*y*) hold, then *x*=*z*.

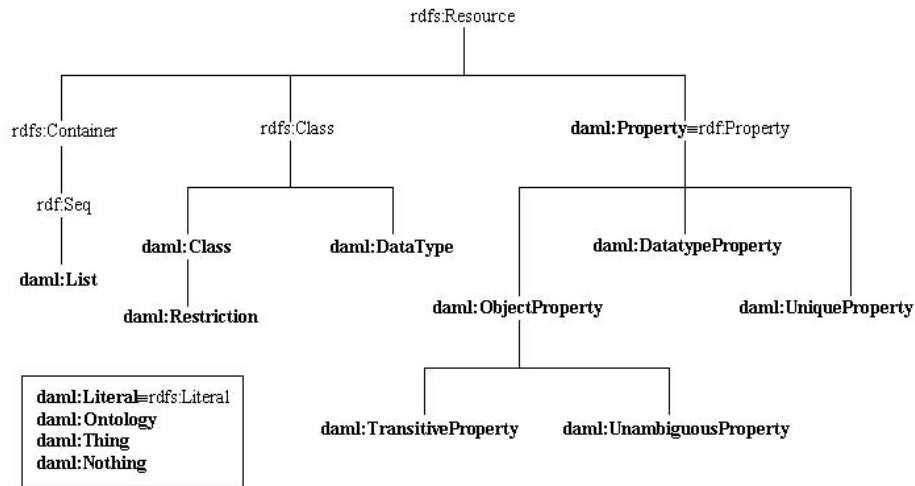


Figure 2.5: Class taxonomy of the DAML+OIL KR ontology defined as an extension of RDF(S).

daml:UniqueProperty can be used to define both kinds of relations (between classes and between a class and a datatype) provided that they are functional¹⁴.

€# **Classes for defining containers** (*daml:List*). DAML+OIL lists are special types of RDF sequences; hence *daml:List* is a subclass of *rdf:Seq*. Although lists are now defined in the RDF(S) KR ontology, they were not when the DAML+OIL KR ontology was created. This is why this primitive is included here.

€# **Predefined classes** (*daml:Thing* and *daml:Nothing*). They represent the most and the least general class respectively.

€# **Classes for defining literal values** (*daml:Literal*). This class represents untyped literal values (that is, strings and integers). It is equivalent to *rdfs:Literal*.

€# **Classes for describing ontologies** (*daml:Ontology*). This primitive is used as the root element of a DAML+OIL ontology, containing all its definitions.

DAML+OIL class expressions are built with KR primitives that are properties¹⁵. These DAML+OIL primitives allow expressing:

€# Conjunction (*daml:intersectionOf*), disjunction (*daml:unionOf*), and negation (*daml:complementOf*).

€# Collection of individuals (*daml:oneOf*).

€# Property restrictions. They are created with the class *daml:Restriction*, as described above. These restrictions are defined with two elements:

¹⁴ This primitive is equivalent to the primitive *oil:FunctionalProperty*. If the relation *R* is functional, and *R*(*x*,*y*) and *R*(*x*,*z*) hold, then *y*=*z*.

¹⁵ Let us remember that OIL class expressions were built with KR primitives that are classes, such as *oil:And*, *oil:Or*, *oil:MinCardinality*, etc.

daml:onProperty (which refers to the property name) and another element that expresses:

- ⊥# Value restriction (*daml:toClass*).
- ⊥# Role fillers (*daml:hasValue*).
- ⊥# Existential restriction (*daml:hasClass*) and number restriction (*daml:cardinality*, *daml:maxCardinality*, and *daml:minCardinality*).
- ⊥# Qualified number restriction (with the primitive *daml:hasClassQ* plus one of the following primitives: *daml:maxCardinalityQ*, *daml:minCardinalityQ* and *daml:cardinalityQ*).

Properties are not only used to create class expressions, but also to define other relationships between ontology components. The following properties are also defined in the DAML+OIL KR ontology:

- ⊥# The primitive *daml:inverseOf*. It defines the inverse of a role.
- ⊥# The primitives *daml:equivalentTo*, *daml:sameClassAs*, *daml:samePropertyAs*, and *daml:sameIndividualAs*. They define equivalences between resources, classes, properties and instances, respectively.
- ⊥# The primitive *daml:differentIndividualFrom*. It defines that two instances are different.
- ⊥# The primitives *daml:disjointWith* and *daml:disjointUnionOf*. They express disjoint and exhaustive knowledge between classes in the class taxonomy respectively.
- ⊥# The primitives *daml:versionInfo* and *daml:imports*. They give information about the ontology version and the ontologies imported by the current ontology. There are no restrictions on the contents of the *daml:versionInfo* primitive.
- ⊥# The primitives *daml:first*, *daml:rest* and *daml:item*. They are used for managing lists.

Finally, the primitive *daml:nil* is an instance of the class *daml:List*. It represents the empty list.

Table 2.3 summarizes the main features of the DAML+OIL KR ontology properties that are not a redefinition of the RDF(S) KR ontology primitives. As we can see in the table, there are 28 properties defined in this KR ontology, apart from 10 properties that are equivalent to the corresponding RDF(S) properties, as described at the beginning of this section. In this table we specify their domain and range, that is, the classes between which these properties can hold. If the value for the range is “not specified”, then the property can take any value which is not restricted to a specific class of the DAML+OIL KR ontology. The *xsd* prefix used in the range of the cardinality restriction properties (*xsd:nonNegativeInteger*) refers to the XML Schema datatype namespace¹⁶.

¹⁶ <http://www.w3.org/2000/10/XMLSchema>. As we will see later, this namespace is not used any more to refer to XML Schema. However, DAML+OIL still uses it.

Table 2.3: Property descriptions of the DAML+OIL KR ontology.

Property name	domain	range
daml:intersectionOf	daml:Class	daml:List
daml:unionOf	daml:Class	daml:List
daml:complementOf	daml:Class	daml:Class
daml:oneOf	daml:Class	daml:List
daml:onProperty	daml:Restriction	rdf:Property
daml:toClass	daml:Restriction	rdfs:Class
daml:hasValue	daml:Restriction	<i>not specified</i>
daml:hasClass	daml:Restriction	rdfs:Class
daml:minCardinality	daml:Restriction	xsd:nonNegativeInteger
daml:maxCardinality	daml:Restriction	xsd:nonNegativeInteger
daml:cardinality	daml:Restriction	xsd:nonNegativeInteger
daml:hasClassQ	daml:Restriction	rdfs:Class
daml:minCardinalityQ	daml:Restriction	xsd:nonNegativeInteger
daml:maxCardinalityQ	daml:Restriction	xsd:nonNegativeInteger
daml:cardinalityQ	daml:Restriction	xsd:nonNegativeInteger
daml:inverseOf	daml:ObjectProperty	daml:ObjectProperty
daml:equivalentTo	<i>not specified</i>	<i>not specified</i>
daml:sameClassAs	daml:Class	daml:Class
daml:samePropertyAs	rdf:Property	rdf:Property
daml:sameIndividualAs	daml:Thing	daml:Thing
daml:differentIndividualFrom	daml:Thing	daml:Thing
daml:disjointWith	daml:Class	daml:Class
daml:disjointUnionOf	daml:Class	daml:List
daml:versionInfo	<i>not specified</i>	<i>not specified</i>
daml:imports	<i>not specified</i>	<i>not specified</i>
daml:first	daml:List	<i>not specified</i>
daml:rest	daml:List	daml:List
daml:item	daml:List	<i>not specified</i>

In Chapter 4 we will describe in detail how to use these primitives to implement ontologies in DAML+OIL. We will now show a small example of how to use them to define the class `Flight` exactly as the class `Travel` that has exactly one value for the attribute `flightNumber`, whose type is `integer`, and that has a filler for the attribute `transportMeans` with value “plane”.

```

<daml:Class rdf:ID="Flight">
  <rdfs:comment>A journey by plane</rdfs:comment>
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Travel"/>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#flightNumber"/>
      <daml:toClass rdf:resource="xsd:integer"/>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#transportMeans"/>
      <daml:hasValue>

```

```

        <xsd:string rdf:value="plane"/>
    </daml:hasValue>
</daml:Restriction>
</daml:intersectionOf>
</daml:Class>

```

2.1.5 OWL knowledge representation ontology

The OWL language (Dean and Schreiber, 2003) has been created by the W3C Web Ontology (WebOnt) Working Group. It is derived from the DAML+OIL language, and it builds upon RDF(S). At the time of writing this section, the OWL specification is a W3C Working Draft, though both the language and its KR ontology¹⁷ (which is implemented in OWL) are already in a stable state.

Like OIL, OWL is divided in layers: OWL Lite, OWL DL, and OWL Full. OWL Lite extends RDF(S) and gathers the most common features of OWL, so it is intended for users that only need to create class taxonomies and simple constraints. OWL DL includes the complete OWL vocabulary, which is described in this section. Finally, OWL Full provides more flexibility to represent ontologies than OWL DL does. We refer to Dean and Schreiber (2003) for a detailed description of this layer.

There are 40 primitives in the OWL DL KR ontology (16 classes and 24 properties). Figure 2.6 shows the KR primitives used in OWL Lite and OWL DL. In the figure, we can see that some RDF(S) primitives can be used in all the versions of OWL (OWL Lite and OWL DL), and that OWL Lite primitives can be used in OWL DL. OWL Full KR primitives are the same as the OWL DL ones, as explained above.

In the figure we also present in parentheses the corresponding primitive in the DAML+OIL KR ontology, in case there is a correspondance with a DAML+OIL KR primitive. For instance, *owl:allValuesFrom* (*daml:toClass*) means that the primitive *owl:allValuesFrom* corresponds to the primitive *daml:toClass* from the DAML+OIL KR ontology.

Figure 2.7 shows the class taxonomy of the primitives that are classes in the OWL KR ontology. All belong to OWL Lite. Hence, they also belong to OWL DL and OWL Full. These primitives can be grouped as follows:

- ## **Classes for defining classes and restrictions** (*owl:Class* and *owl:Restriction*).
The primitive *owl:Class* specializes *rdfs:Class* and is used to define classes. The primitive *owl:Restriction* specializes *owl:Class* and is used to define property restrictions for classes (number restrictions, existential restrictions, universal restrictions, etc.).
- ## **Classes for defining properties** (*owl:ObjectProperty*, *owl:DatatypeProperty*, *owl:TransitiveProperty*, *owl:SymmetricProperty*, *owl:FunctionalProperty*, *owl:InverseFunctionalProperty*, and *owl:AnnotationProperty*). They are used to define properties, hence they specialize the class *rdf:Property*. The primitive

¹⁷ <http://www.w3.org/2002/07/owl>

OWL DL	
Class expressions allowed in:	rdfs:domain, rdfs:range, rdfs:subClassOf owl:intersectionOf, owl:equivalentClass, owl:allValuesFrom, owl:someValuesFrom
Values are not restricted (0..N) in:	owl:minCardinality, owl:maxCardinality, owl:cardinality
owl:DataRange, rdf:List, rdf:first, rdf:rest, rdf:nil	
owl:hasValue (<i>daml:hasValue</i>)	
owl:oneOf (<i>daml:oneOf</i>)	
owl:unionOf (<i>daml:unionOf</i>), owl:complementOf (<i>daml:complementOf</i>)	
owl:disjointWith (<i>daml:disjointWith</i>)	
OWL Lite	
owl:Ontology (<i>daml:Ontology</i>),	
owl:versionInfo (<i>daml:versionInfo</i>),	
owl:imports (<i>daml:imports</i>),	
owl:backwardCompatibleWith,	
owl:incompatibleWith, owl:priorVersion,	
owl:DeprecatedClass,	
owl:DeprecatedProperty	
owl:Class (<i>daml:Class</i>),	
owl:Restriction (<i>daml:Restriction</i>),	
owl:onProperty (<i>daml:onProperty</i>),	
owl:allValuesFrom (<i>daml:toClass</i>) (only with class identifiers and named datatypes),	
owl:someValuesFrom (<i>daml:hasClass</i>) (only with class identifiers and named datatypes),	
owl:minCardinality (<i>daml:minCardinality</i> ; restricted to {0,1}),	
owl:maxCardinality (<i>daml:maxCardinality</i> ; restricted to {0,1}),	
owl:cardinality (<i>daml:cardinality</i> ; restricted to {0,1})	
owl:intersectionOf (only with class identifiers and property restrictions)	
owl:ObjectProperty (<i>daml:ObjectProperty</i>),	
owl:DatatypeProperty (<i>daml:DatatypeProperty</i>),	
owl:TransitiveProperty (<i>daml:TransitiveProperty</i>),	
owl:SymmetricProperty,	
owl:FunctionalProperty (<i>daml:UniqueProperty</i>),	
owl:InverseFunctionalProperty (<i>daml:UnambiguousProperty</i>),	
owl:AnnotationProperty	
owl:Thing (<i>daml:Thing</i>)	
owl:Nothing (<i>daml:Nothing</i>)	
owl:inverseOf (<i>daml:inverseOf</i>),	
owl:equivalentClass (<i>daml:sameClassAs</i>) (only with class identifiers and property restrictions),	
owl:equivalentProperty (<i>daml:samePropertyAs</i>),	
owl:sameAs (<i>daml:equivalentTo</i>),	
owl:sameIndividualAs,	
owl:differentFrom (<i>daml:differentIndividualFrom</i>),	
owl:AllDifferent, owl:distinctMembers	
RDF(S)	
rdf:Property	
rdfs:subPropertyOf	
rdfs:domain	
rdfs:range (only with class identifiers and named datatypes)	
rdfs:comment, rdfs:label, rdfs:seeAlso, rdfs:isDefinedBy	
rdfs:subClassOf (only with class identifiers and property restrictions)	

Figure 2.6: OWL Lite and OWL DL KR primitives.

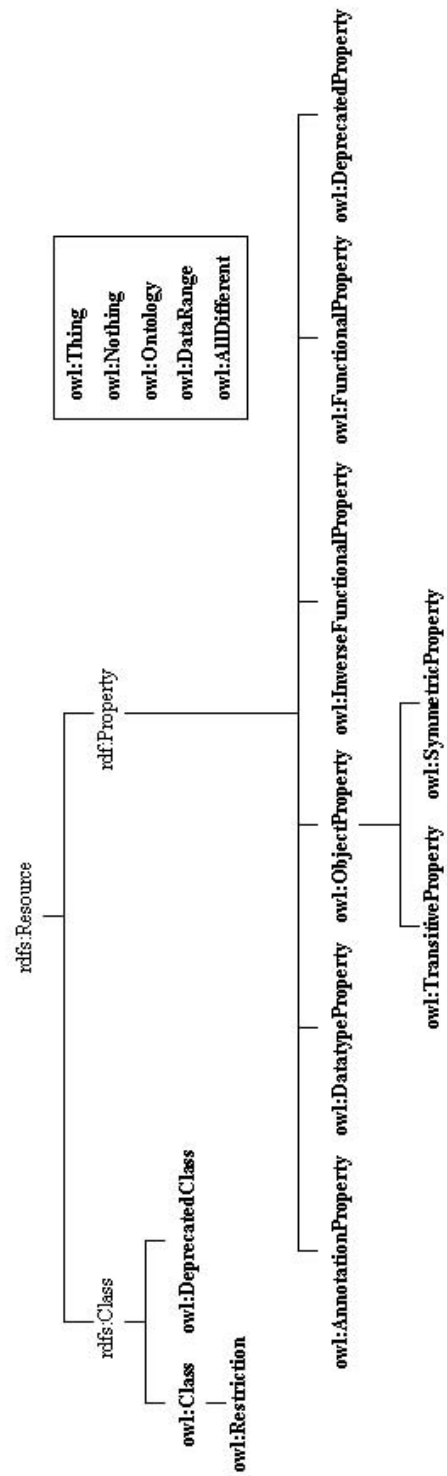


Figure 2.7: Class taxonomy of the OWL KR ontology defined as an extension of RDF(S).

owl:ObjectProperty serves to define properties that connect a class with another class, and the primitive *owl:DatatypeProperty* is used to define properties that connect a class with a datatype. The primitives *owl:TransitiveProperty* and *owl:SymmetricProperty* serve to define logical characteristics of properties. The primitives *owl:FunctionalProperty* and *owl:InverseFunctionalProperty* are used to define global cardinality restrictions of properties. In OWL Lite and OWL DL, *owl:InverseFunctionalProperty* is defined as a subclass of *owl:ObjectProperty*. In OWL Full, *owl:InverseFunctionalProperty* can be also an *owl:DatatypeProperty*. Hence, this primitive is defined as a subclass of *rdf:Property*, as shown in Figure 2.7. The primitive *owl:AnnotationProperty* is used to define properties that have no logical consequences on an OWL ontology, but just give information about its classes, properties, individuals or the whole ontology.

- €# **Classes for stating inequality among individuals** (*owl:AllDifferent*). This is to specify that several instances are different to each other. This is sometimes needed because OWL does not assume the unique names assumption in its ontologies. This means that two individual definitions with different identifiers could refer to the same individual.
- €# **Classes for describing enumerations of datatypes** (*owl:DataRange*). This is to create enumerated datatypes, that is, datatypes with a set of predefined values.
- €# **Predefined classes** (*owl:Thing* and *owl:Nothing*). They represent the most general and the least general class respectively.
- €# **Classes for describing ontologies** (*owl:Ontology*). This primitive is used as the root element of an OWL ontology, containing all its definitions.
- €# **Classes for describing ontology versioning** (*owl:DeprecatedClass* and *owl:DeprecatedProperty*). They specify, respectively, that a class or a property have been deprecated in the current version of the ontology. As occurs with *owl:AnnotationProperty*, these primitives have no logical consequences on an OWL ontology. They are only used for ontology versioning purposes.

Like in DAML+OIL, OWL class expressions are built with KR primitives that are properties. These will be organized in two groups: the one with primitives defined for OWL Lite (which can be used in OWL Lite, OWL DL and OWL Full) and the one with primitives defined for OWL DL (which can be used in OWL DL and OWL Full).

€# **Properties for defining class expressions in OWL Lite:**

- €# **Conjunction** (*owl:intersectionOf*). The range of this property is restricted in OWL Lite to class identifiers and property restrictions.
- €# **Property restrictions**. They are defined with the class *owl:Restriction*, described above. Restrictions are defined with two elements: *owl:onProperty* (which refers to the property name) and another element that expresses:
 - €# Value restriction (*owl:allValuesFrom*).
 - €# Existential restriction (*owl:someValuesFrom*).

- Other properties defined in OWL Lite and OWL DL are the following:

- Table 2.4 summarizes the main features of the properties of the OWL KR ontology, specifying their domain and range. If the value for the range is “not specified”, we mean that the property can take any value which is not restricted to a specific class of the OWL KR ontology. As we can see in the table, there are 24 properties defined in this KR ontology. Besides, in OWL we can use the properties *rdfs:subClassOf*,

rdfs:subPropertyOf, *rdfs:domain*, *rdfs:range*, *rdfs:comment*, *rdfs:label*, *rdfs:seeAlso*, and *rdfs:isDefinedBy* from the RDF(S) KR ontology.

As described in the table, the OWL KR ontology does not specify any domain or range for *owl:versionInfo*.

Table 2.4: Property descriptions of the OWL KR ontology.

Property name	domain	range
<i>owl:intersectionOf</i>	<i>owl:Class</i>	<i>rdf:List</i>
<i>owl:unionOf</i>	<i>owl:Class</i>	<i>rdf:List</i>
<i>owl:complementOf</i>	<i>owl:Class</i>	<i>owl:Class</i>
<i>owl:oneOf</i>	<i>owl:Class</i>	<i>rdf:List</i>
<i>owl:onProperty</i>	<i>owl:Restriction</i>	<i>rdf:Property</i>
<i>owl:allValuesFrom</i>	<i>owl:Restriction</i>	<i>rdfs:Class</i>
<i>owl:hasValue</i>	<i>owl:Restriction</i>	<i>not specified</i>
<i>owl:someValuesFrom</i>	<i>owl:Restriction</i>	<i>rdfs:Class</i>
<i>owl:minCardinality</i>	<i>owl:Restriction</i>	<i>xsd:nonNegativeInteger</i> <i>OWL Lite: {0,1}</i> <i>OWL DL/Full: {0,...,N}</i>
<i>owl:maxCardinality</i>	<i>owl:Restriction</i>	<i>xsd:nonNegativeInteger</i> <i>OWL Lite: {0,1}</i> <i>OWL DL/Full: {0,...,N}</i>
<i>owl:cardinality</i>	<i>owl:Restriction</i>	<i>xsd:nonNegativeInteger</i> <i>OWL Lite: {0,1}</i> <i>OWL DL/Full: {0,...,N}</i>
<i>owl:inverseOf</i>	<i>owl:ObjectProperty</i>	<i>owl:ObjectProperty</i>
<i>owl:sameAs</i>	<i>owl:Thing</i>	<i>owl:Thing</i>
<i>owl:equivalentClass</i>	<i>owl:Class</i>	<i>owl:Class</i>
<i>owl:equivalentProperty</i>	<i>rdf:Property</i>	<i>rdf:Property</i>
<i>owl:sameIndividualAs</i>	<i>owl:Thing</i>	<i>owl:Thing</i>
<i>owl:differentFrom</i>	<i>owl:Thing</i>	<i>owl:Thing</i>
<i>owl:disjointWith</i>	<i>owl:Class</i>	<i>owl:Class</i>
<i>owl:distinctMembers</i>	<i>owl:AllDifferent</i>	<i>rdf:List</i>
<i>owl:versionInfo</i>	<i>not specified</i>	<i>not specified</i>
<i>owl:priorVersion</i>	<i>owl:Ontology</i>	<i>owl:Ontology</i>
<i>owl:incompatibleWith</i>	<i>owl:Ontology</i>	<i>owl:Ontology</i>
<i>owl:backwardCompatibleWith</i>	<i>owl:Ontology</i>	<i>owl:Ontology</i>
<i>owl:imports</i>	<i>owl:Ontology</i>	<i>owl:Ontology</i>

To sum up, we can say that there are not many differences between the OWL KR ontology and the DAML+OIL one. In fact, most of the changes imply changing the names of the original DAML+OIL KR primitives, since they were not always easy to understand by non-experts. Two other important changes are the removal of qualified number restrictions (OWL is a SHIN language, according to the DL terminology) and the inclusion of symmetry as a characteristic of properties. The primitives for managing lists that were defined in the DAML+OIL KR ontology have not been included in this ontology, since OWL allows the use of the recent RDF(S) primitives for managing lists.

In Chapter 4 we will describe in detail how to use these primitives to implement ontologies in OWL. We now present a small example of how to use them to define

the class `Flight` as the class `Travel` that has exactly one value for the attribute `flightNumber`, whose type is `integer`, and that has a filler for the attribute `transportMeans` with value “plane”.

```
<owl:Class rdf:ID="Flight">
  <rdfs:comment>A journey by plane</rdfs:comment>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Travel"/>
    <owl:Restriction owl:cardinality="1">
      <owl:onProperty rdf:resource="#flightNumber"/>
      <owl:allValuesFrom rdf:resource="&xsd;integer"/>
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#transportMeans"/>
      <owl:hasValue rdf:datatype="&xsd:string">
        plane
      </owl:hasValue>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

2.2 Top-level Ontologies

Top-level ontologies (aka upper-level ontologies) describe very general concepts that are common across the domains and give general notions under which all the terms in existing ontologies should be linked to. Sometimes top-level ontologies are used to build domain ontologies, but often these are built first and then linked to upper-level ontologies.

On the framework of the Cyc project, the following characteristics are identified as desirable in a top-level ontology¹⁸:

- a) It should be “universal”: every concept imagined in a specific ontology can be correctly linked to the upper-level ontology in appropriate places, no matter how general or specific the concept is, and no matter what the background of the ontology builder is (nationality, age, native language, epoch, childhood experiences, current goals, etc.). For example, a top-level ontology that just classifies entities in *physical objects* and *mental objects* is not universal, since *processes* and *situations* are not considered.
- b) It should be “articulate”: on the one hand, there is a justification for every concept of the top-level ontology. On the other hand, there are enough concepts to enable and support knowledge sharing, natural language disambiguation, database cleaning and integration, and other kinds of applications. For example, a top-level ontology that classifies entities in *mortal entities* and *immortal entities* could be (in the best case) useful for theology, but not for other fields.

¹⁸ <http://www.cyc.com/cyc-2-1/cover.html>

In the next subsections, we will present the following top-level ontologies: the top-level ontologies of universals and particulars, built by Guarino and colleagues, Sowa's top-level ontology, Cyc's Upper Ontology, and one of the Standard Upper Ontology working group. The top-level ontologies of universals and particulars are available in WebODE, Cyc's Upper Ontology is available in CycL and DAML+OIL, the Standard Upper Ontology is available in KIF and DAML+OIL. We do not know of any implementation of Sowa's top-level ontology.

2.2.1 Top-level ontologies of universals and particulars

Guarino and colleagues have built two top-level ontologies, as shown in Figure 2.8: one of universals, and another of particulars. A universal is a concept¹⁹, like *car* or *traveler*, while a particular is an individual like *my car* or *John Smith*. Therefore, the terms *car* and *traveler* in a domain ontology can be linked to the top-level of particulars through the relation *Subclass-Of*, and they can be linked to the top-level of universals through the relation *Instance-Of*. Both top-level ontologies are presented in this section.

The *top-level ontology of universals* (Guarino and Welty, 2000) contains concepts whose instances are universals. This ontology has been obtained

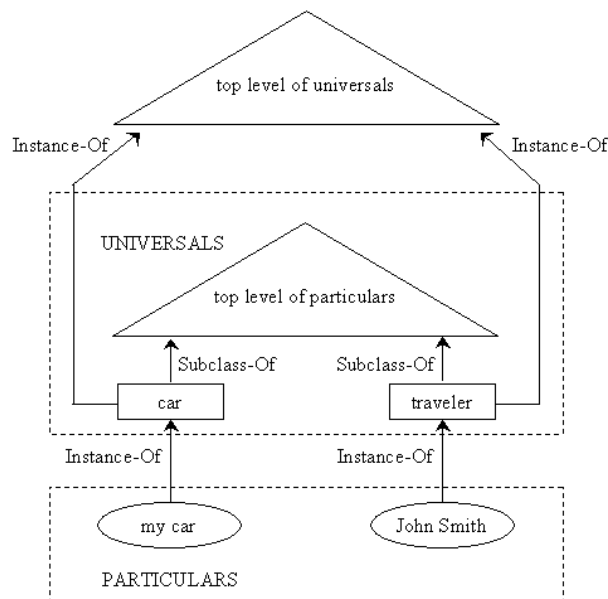


Figure 2.8: Relationship between particulars and universals.

¹⁹ This is a simplified definition, but it is enough for the purpose of this book. For a complete definition consult Guarino and Welty (2000).

considering the philosophical notions of *rigidity*, *identity* and *dependency*²⁰. Let us examine these notions (Gangemi et al., 2001), which will be explained in the context of the Ontoclean method in Chapter 3:

- ⌘# *Rigidity*. This notion is defined according to the idea of essence. A property is essential to an instance if and only if it is necessary for this instance. Thus, a property is rigid (+R) if and only if it is necessarily essential to all its instances; a property is anti-rigid (~R) if and only if it is not essential for all its instances; and a property is non-rigid (-R) if and only if it is not essential for some of its instances. For example, the concept `person` is usually considered rigid, since every person is essentially such. The concept `traveler` is considered anti-rigid, since every traveler can possibly be a non-traveler once the journey has finished. Finally, the concept `red` is non-rigid, since there are instances that are essentially red (e.g., `drop of blood`), and instances that are not essentially red (`my pullover`).
- ⌘# *Identity*. A property *carries* an identity criterion (+I) if and only if all its instances can be (re)identified by means of a suitable “sameness” relation. A property *supplies* an identity criterion (+O) if and only if such criterion is not inherited by any subsuming property. For example, if we take the DNA as an identity criterion, we can say that `person` not only carries the identity criterion, but also supplies it. Besides, if `traveler` is a subclass of `person`, then `traveler` only inherits the identity criterion of `person`, without supplying any further identity criteria.
- ⌘# *Dependency*. An individual *x* is constantly dependent on the individual *y* if and only if, at any time, *x* cannot be present unless *y* is fully present, and *y* is not part of *x*. For example, a hole in a wall is constantly dependent on the wall. The hole cannot be present if the wall is not present. A property *P* is constantly dependent (+D) if and only if, for all its instances, there exists something on which the instances are constantly dependent. Otherwise, the property *P* is not constantly dependent (-D). For instance, the concept `hole` is constantly dependent because every instance of `hole` is constantly dependent. Note that the constant dependence of a property is defined according to the constant dependence of individuals.

Every concept of the top-level ontology of universals has four attributes: *rigidity*, *supplies identity*, *carries identity*, and *dependency*. This ontology has been built considering several combinations of values of these attributes (Welty and Guarino, 2001). For instance, the concept *type* is rigid, supplies identity, and carries identity (nothing is explicitly stated about its dependency). This means that the concepts that are instances of the concept *type* will be rigid and will supply and carry identity. Every concept of a specific domain will be an instance of at least one of the leaves of the top-level of universals. Figure 2.9 presents the class taxonomy of this ontology.

²⁰ Another important notion is *unity*. However, it has not been used to classify the properties of this top-level ontology of universals. Therefore, such a notion will not be presented until Chapter 3, where the OntoClean method is described.

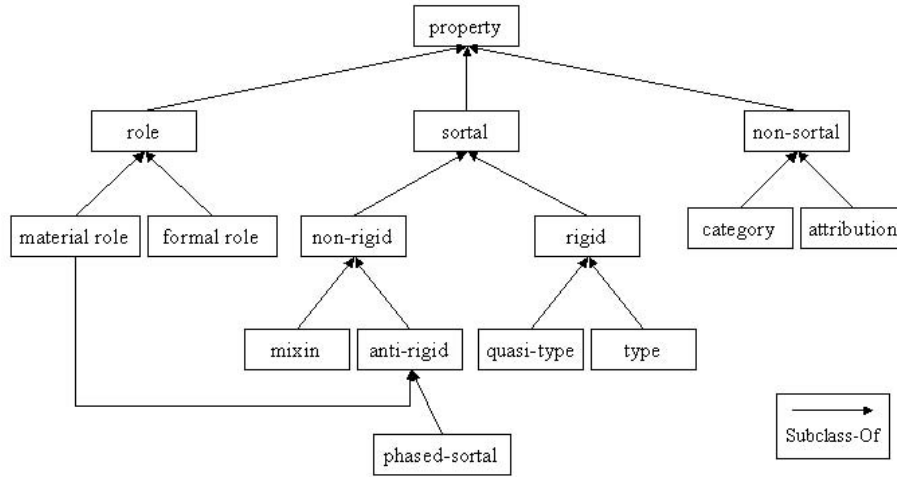


Figure 2.9: Class taxonomy of the top-level ontology of universals.

The *top-level ontology of particulars* (Gangemi et al., 2001) holds general concepts (for example, *object*) to which domain concepts can be linked with the relation *Subclass-Of*. Figure 2.10 shows part of the class taxonomy of this ontology. As we can see, the ontology contains three roots (*abstract*, *concrete* and *relation*). It is being developed following the principles established in the OntoClean method (Welly and Guarino, 2001) for cleaning ontologies, described in Section 3.8.3.

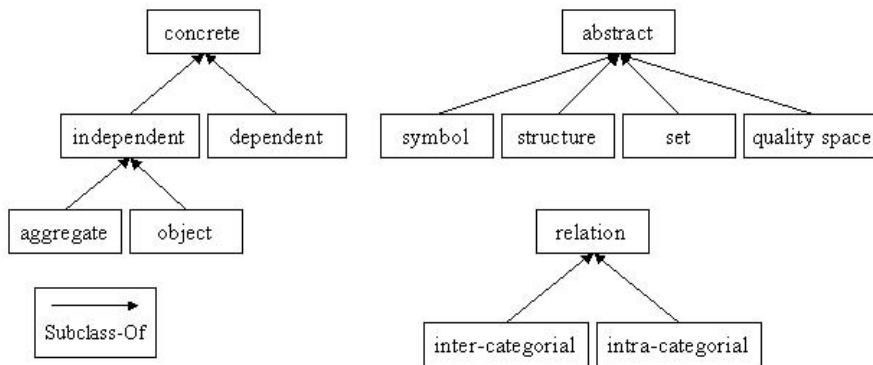


Figure 2.10: Partial view of the class taxonomy of top-level ontology of particulars.

Let us suppose that we want to link concepts of our travel domain to the top-level ontologies of universals and particulars. In this case, the domain concepts (i.e., *car*, *traveler*, etc.) will be subclasses of concepts of the top-level ontology of particulars, and they will also be instances of concepts of the top-level ontology of

universals. Instances like *my car* or *John Smith* are instances of classes linked to the top-level of particulars.

Both the top-level ontology of universals and the top-level ontology of particulars are available in the ontology engineering workbench WebODE, which will be presented in Chapter 5. At the end of the year 2002, the former had 15 concepts while the latter had over 30 concepts.

2.2.2 Sowa's top-level ontology

Sowa's top-level ontology includes the basic categories and distinctions that have been derived from a variety of sources in logic, linguistics, philosophy, and artificial intelligence (Sowa, 1999). Sowa's top-level ontology has 27 concepts, all of them identified in Figure 2.11.

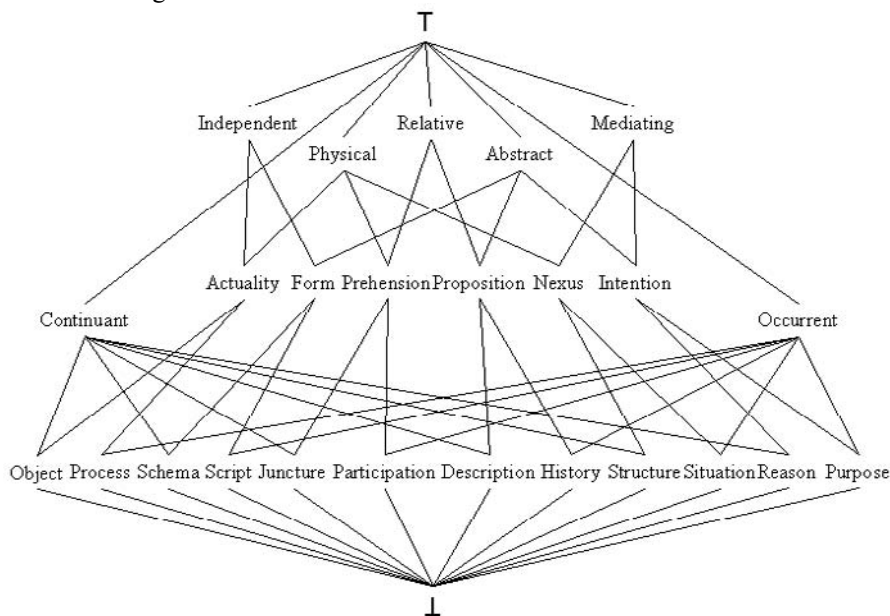


Figure 2.11: Sowa's top-level ontology.

This ontology has a lattice structure where the top concept is the *universal type* (represented as \top in Figure 2.11), and the bottom concept is the *absurd type* (γ). The *universal type* contains all the possible instances of the ontology. The *absurd type* does not have instances and is a subclass of every concept of the taxonomy. The direct subclasses of the *universal type* are the following: *independent*, *relative*, *mediating*, *continuant*, *physical*, *abstract*, and *occurrent*. By combining these primitive concepts more concepts of the lattice are obtained, for example,

$$\text{history} = \text{proposition} \sim \text{occurrent}$$

The structure of this top-level ontology is a lattice (see a definition in Section 3.6.2) because every pair of concepts of the taxonomy has, at least, a common direct or indirect superclass, and each pair of concepts has, at least, a common direct or indirect subclass. Let us take as an example the pair $\{proposition, occurrent\}$, where the common superclass is of the *universal type*, and a common subclass is *history*. In this example, the class *history* could be represented as the intersection of *proposition* and *occurrent*. In this top-level ontology, concepts can be obtained by combining concepts from the upper levels.

2.2.3 Cyc's upper ontology

Cyc's Upper Ontology is contained in the Cyc Knowledge Base (Lenat and Guha, 1990), which holds a huge amount of common sense knowledge. The Cyc KB is being built upon a core of over 1,000,000 assertions hand-entered and designed to gather a large portion of what people normally consider consensus knowledge of the world. It is divided into hundreds of microtheories (bundles of assertions in the same domain) and is implemented in the CycL language.

Cyc's Upper Ontology²¹ contains about 3,000 terms arranged in 43 topical groups (fundamentals, time and dates, spatial relations, etc.). The class *Thing* is the root of the ontology, and it is also the universal set. This means that when we link terms from a domain ontology to Cyc's Upper Ontology through the *genls* relation (which is the *Subclass-Of* relation in CycL), every concept of the domain ontology is a subclass of *Thing*, therefore, every instance of the domain ontology is an instance of *Thing*. Cyc's Upper Ontology has been built by performing the following steps: (1) dividing the universal set into tangible and intangible, into the static thing versus the dynamic process, into collection versus individual, etc.; and (2) refining the result when new knowledge is introduced (such as new concepts, new *Subclass-Of* relations, etc.). During the refining process, some of these categories might disappear.

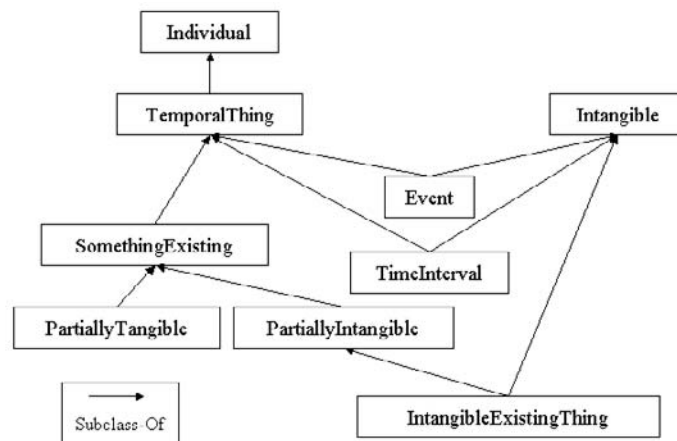


Figure 2.12: Fragment of the class taxonomy of Cyc's Upper Ontology.

²¹ <http://www.cyc.com/cyc-2-1/cover.html>

Figure 2.12 shows a fragment of Cyc’s Upper Ontology. As we can see, the concepts *TimeInterval* and *Event* are subclasses of the concepts *TemporalThing* and *Intangible*. A *TemporalThing* is an *Individual*. *SomethingExisting*, which is a subclass of the concept *TemporalThing*, is partitioned in the concepts *PartiallyTangible* and *PartiallyIntangible*. An *IntangibleExistingThing* is something *PartiallyIntangible* and *Intangible*.

Cycorp²², the company supplier of Cyc’s Upper Ontology, provides several tools to assist users in the handling of this ontology. Such tools include hypertext links that permit browsing directly some of the taxonomies and navigating among the references (a topical listing of the upper ontology divided into subject-areas to facilitate systematic study, etc.).

2.2.4 The Standard Upper Ontology (SUO)

The Standard Upper Ontology²³ is the result of a joint effort to create a large, general-purpose, formal ontology (Pease and Niles, 2002). It is promoted by the IEEE Standard Upper Ontology working group, and its development began in May 2000. The participants were representatives of government, academia, and industry from several countries. The effort was officially approved as an IEEE standard project in December 2000.

There are currently two “starter documents” agreed by the working group and that may be developed into a draft standard. One of the documents is known as the IFF (Information Flow Framework) Foundation Ontology, a meta-ontology based on Mathematics and viewed from the set-theoretic perspective. The other one is known as SUMO (Suggested Upper Merged Ontology). Part of its current structure is shown in Figure 2.13.

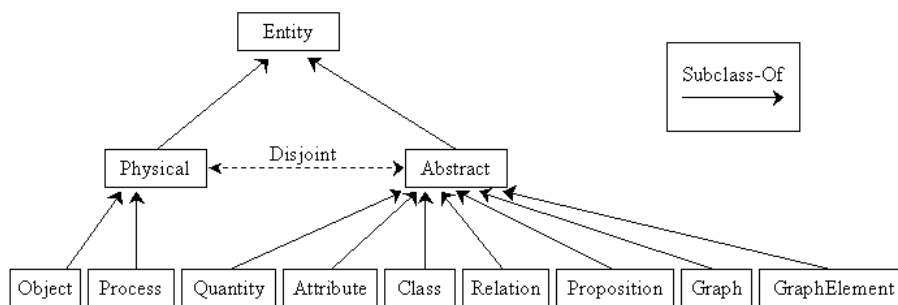


Figure 2.13: Structure of the first levels of SUMO (May 2002).

²² <http://www.cyc.org/>

²³ <http://suo.ieee.org/>

The goal of SUMO is to create a comprehensive and consistent top-level ontology from some of the best public sources, such as:

- €# CNR's group mereotopology (Borgo et al., 1996; Borgo et al., 1997).
- €# Upper-level ontologies; e.g., Sowa's upper ontology and Russell and Norvig's upper-level ontology (1995).
- €# Time theories; e.g., James Allen's temporal axioms (Allen, 1984).
- €# Plan and process theories (Pease and Carrico, 1997); etc.

Therefore, SUMO considers some high level distinctions, and contains temporal concepts and processes. It is a modular ontology, that is, the ontology is divided into sub-ontologies. The dependencies between the various sub-ontologies can be outlined as Figure 2.14 shows.

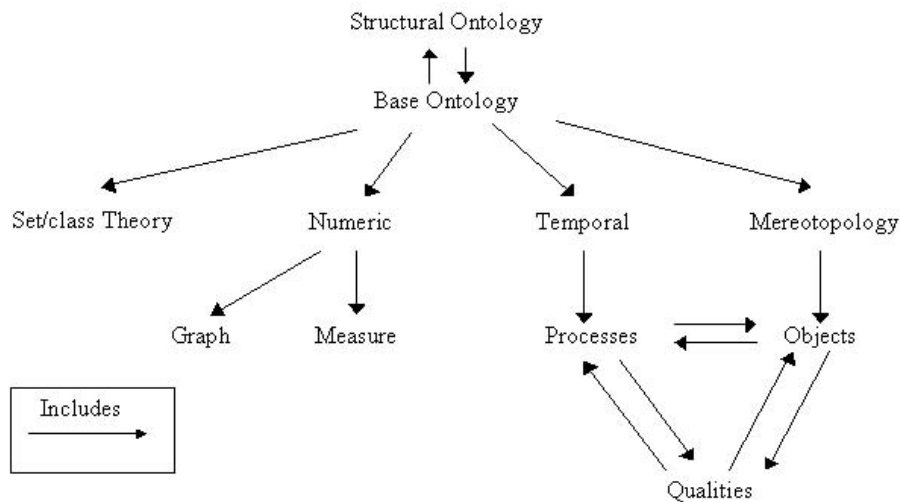


Figure 2.14: Modular structure of SUMO.

To decide which concepts should be removed, added or preserved during the evolution of this top-level ontology, a series of steps must be performed (Niles and Pease, 2001):

- 1) Take the current version of the top-level ontology as the base and add, one by one, lower-level ontologies to this base.
- 2) Eliminate from the top-level ontology the concepts not related to any concept of the domain ontologies, and add other top-level concepts suitable to model the lower level ontologies.

The people involved in the SUMO development are currently in the process of augmenting each of the records in the noun database of WordNet with pointers to SUMO concepts.

2.3 Linguistic Ontologies

This section collects information on linguistic ontologies. The purpose of this type of ontology is to describe semantic constructs rather than to model a specific domain. They offer quite a heterogeneous amount of resources, used mostly in natural language processing. The main characteristic of these ontologies is that they are bound to the semantics of grammatical units (words, nominal groups, adjectives, etc.).

Most linguistic ontologies use words as grammatical units. In fact, of the ontologies reviewed in this section, only the Generalized Upper Model (GUM) and SENSUS gather information on grammatical units that are bigger than words. Other ontologies focus on the word meaning (e.g., WordNet). Moreover, in some of the ontologies there is a one-to-one mapping between concepts and words in a natural language (e.g., wordnets of EuroWordNet), while in others many concepts may not map to any word in a language or may map to more than one in the same language (e.g., Mikrokosmos).

There are also differences with respect to their degree of language dependency; some linguistic ontologies depend totally on a single language (e.g., WordNet); others are multilingual – i.e., are valid for several languages – (e.g., GUM); some others contain a language-dependent part and a language-independent part (e.g., EuroWordNet); and others are language independent (e.g., Mikrokosmos).

The origin and motivations of these ontologies are varied and thus we have: on-line lexical databases (e.g., WordNet), ontologies for machine translation (e.g., Sensus), ontologies for natural language generation (e.g., GUM), etc.

In the next sections we present the following ontologies: WordNet, EuroWordNet, GUM, Mikrokosmos, and SENSUS. Some of these, as for example SENSUS and GUM, are also considered top-level ontologies since they chiefly contain very abstract concepts.

2.3.1 WordNet

WordNet (Miller et al., 1990; Miller, 1995) is a very large lexical database for English created at Princeton University and based on psycholinguistic theories. Psycholinguistics is an interdisciplinary field of research concerned about the cognitive bases of linguistic competence (Fellbaum and Miller, 1990). WordNet attempts to organize lexical information in terms of word meanings rather than word forms, though inflectional morphology is also considered. For example, if you search for *trees* in WordNet, you will have the same access as if you search for *tree*.

WordNet 1.7 contains 121,962 words and 99,642 concepts. It is organized into 70,000 sets of synonyms (“synsets”), each representing one underlying lexical concept. Synsets are interlinked via relationships such as synonymy and antonymy, hypernymy and hyponymy (*Subclass-Of* and *Superclass-Of*), meronymy and holonymy (*Part-Of* and *Has-a*). Approximately one half of the synsets include brief explanations of their intuitive sense in English. WordNet divides the lexicon into five categories: nouns, verbs, adjectives, adverbs, and function words. Nouns are

organized as topical hierarchies. Figure 2.15 shows part of the noun hierarchy where terms concerning a person, his (her) components, his (her) substances, and his (her) family organization, appear related. The only relations that we can see in the figure are *meronymy*, *antonymy*, and *hyponymy*, since it is a very reduced view of the noun hierarchy.

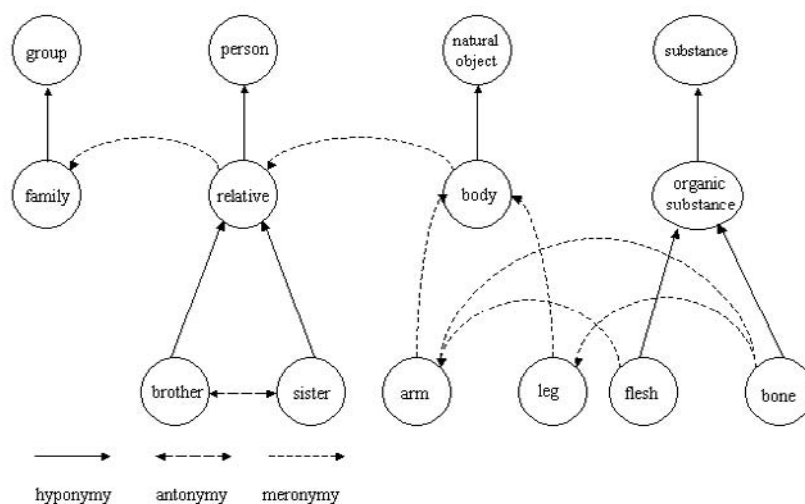


Figure 2.15: A partial view of the category of nouns of WordNet.

Verbs are organized according to a variety of entailment relations. For example, the verbs *succeed* and *try* are related through a backward implication, and *buy* and *pay* are related through a temporal inclusion. With adjectives and adverbs the relations of similarity and antonymy play an important role. For instance, *dry* is related to *sere*, *anhydrous*, *arid*, etc., through the relation of similarity. *Wet* is also related to *humid*, *watery*, or *damp* through the relation of similarity. Besides, *dry* and *wet* are related by means of the relation of antonymy.

2.3.2 EuroWordNet

EuroWordNet (Vossen, 1998; 1999)²⁴ is a multilingual database with wordnets for several European languages (Dutch, Italian, Spanish, German, French, Czech, Estonian). Some of the institutions involved in this project are: University of Amsterdam (The Netherlands), UNED (Spain), and University of Sheffield (United Kingdom).

The wordnets are structured in EuroWordNet in the same way as WordNet is for English, with interrelated synsets. The wordnets are linked to an Inter-Lingual-Index. Through this index the languages are interconnected so that it is possible to go from the words in one language to similar words in any other language, and to compare synsets and their relations across languages. The index also gives access to

²⁴ <http://www.hum.uva.nl/~ewn/>

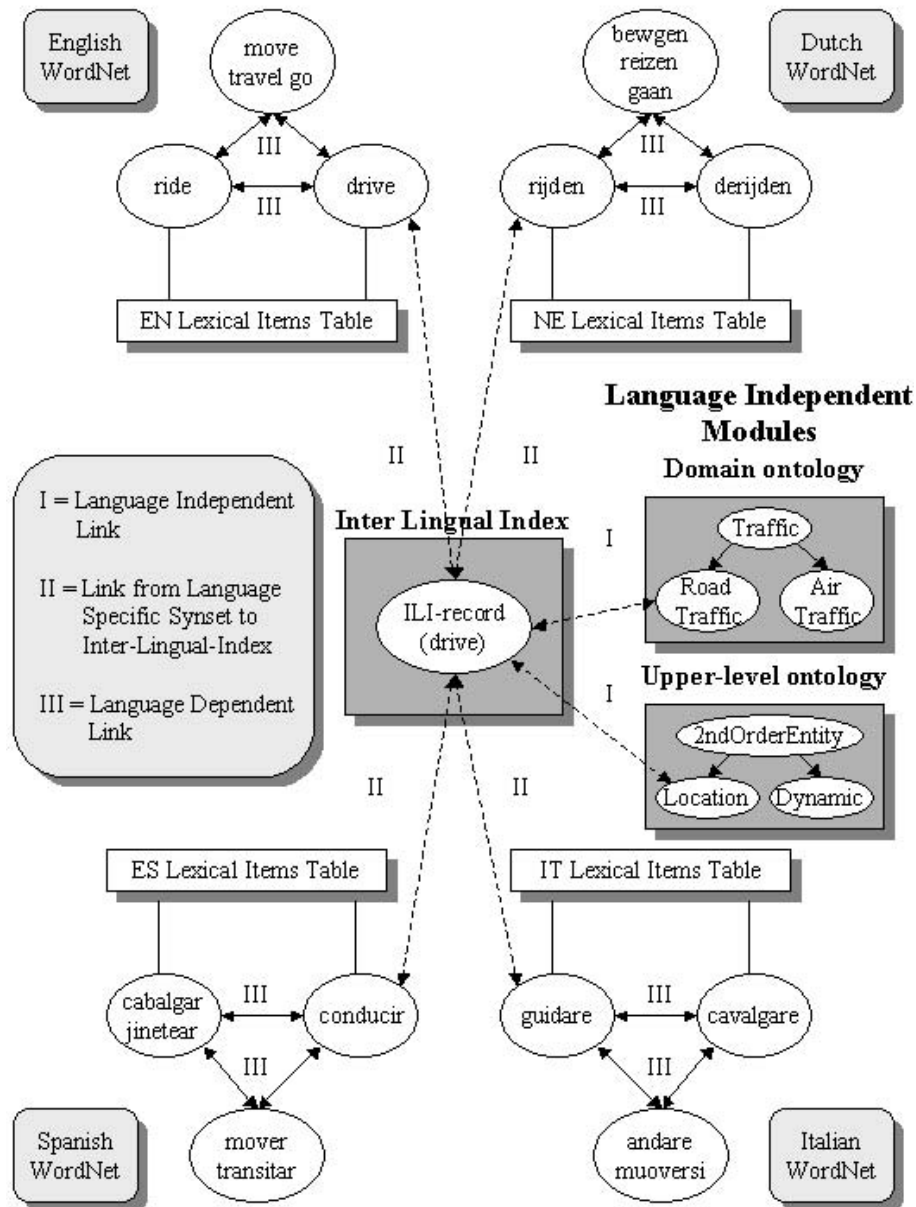


Figure 2.16: Links between different elements of EuroWordNet.

a shared upper-level ontology of 63 semantic distinctions. This upper-level ontology provides a common semantic framework for all the languages, while language specific properties are maintained in individual wordnets. This index can be used for monolingual and cross-lingual information retrieval.

Figure 2.16 shows a small section of EuroWordNet. We can see that the English words: *move*, *ride*, *drive*, etc., are related to words in other languages with similar meaning through the Inter-Lingual-Index. Therefore, we can find language dependent links inside words in the same language (represented by III), independent language links that connect the Inter-Lingual-Index with the domain ontologies and with the upper-level ontology (represented by I), and links that connect the Inter-Lingual-Index with the synsets of other different languages (represented by II).

The EuroWordNet project was completed in the summer of 1999, and the design of the EuroWordNet database, the defined relations, the upper ontology and the Inter-Lingual-Index are now frozen. Nevertheless, many other institutes and research groups are developing similar wordnets in other languages (European and non-European) using the EuroWordNet specification. If compatible, these wordnets can be added to the database and, through the index, connected to any other wordnet. Wordnets are currently developed, at least, for the following languages: Swedish, Norwegian, Danish, Greek, Portuguese, Basque, Catalan, Romanian, Lithuanian, Russian, Bulgarian and Slovene.

The cooperative framework of EuroWordNet is continued through the Global WordNet Association²⁵, a free and public association created to stimulate the building of new wordnets in EuroWordNet and WordNet.

2.3.3 The Generalized Upper Model

The Generalized Upper Model (GUM)²⁶ (Bateman et al., 1995) is the result of a continuous evolution that began with the Penman Upper Model, used in the Penman text generation system (Bateman et al., 1990). Three organizations were involved in the development of GUM: the Information Sciences Institute (ISI, USA), GMD/IPSI (Germany), and the Institute for the Technological and Scientific Research (CNR, Italy).

GUM is a linguistic ontology bound to the semantics of language grammar constituents. Unlike other linguistic ontologies, such as WordNet, it does not describe the semantics of words but the semantics that can be expressed in bigger grammatical units such as nominal groups, prepositional phrases, etc.

This ontology has two hierarchies, one of concepts and another of relations. Figure 2.17 shows the first levels of these hierarchies.

²⁵ <http://www.hum.uva.nl/~ewn/gwa.htm>

²⁶ <http://www.darmstadt.gmd.de/publish/komet/gen-um/newUM.html>

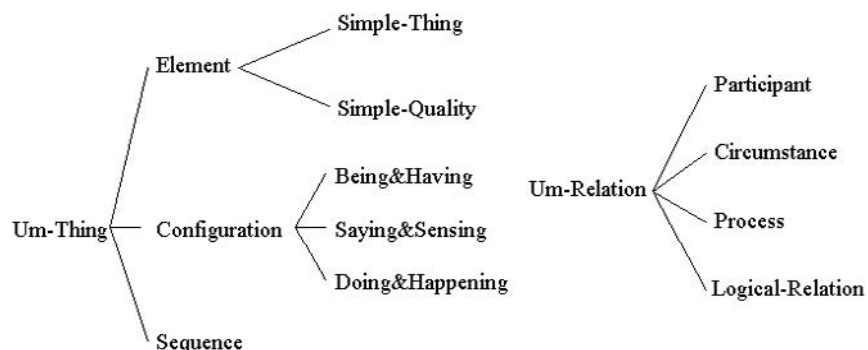


Figure 2.17: First levels of GUM hierarchies.

These taxonomies have their origin in Halliday's (1985) functional grammar, but can be applied to any theory. The concept hierarchy represents the basic semantic entities and includes *configurations* of the processes, and the different kinds of *objects* and *qualities*. A *configuration* is a set of objects that participate in some activity or that are in some state. An example of *configuration* is *being&having*, which indicates the existence of something or a relation of identity, possession, attribution, etc. The relation hierarchy represents the *participants* and the *circumstances* involved in the *processes*, and the *logical combinations* between them. The *actor*, the *message*, or the *attribute* are examples of *participants*. *Company*, *comparison*, *cause*, *mode*, *time*, *space*, etc., express *circumstances*.

2.3.4 The Mikrokosmos ontology

The Mikrokosmos Ontology²⁷ (Mahesh and Nirenburg, 1995; Mahesh, 1996) is a language-independent ontology that is part of the Mikrokosmos machine translation project on the domain of mergers and acquisitions of companies. The New Mexico State University, Carnegie Mellon University and some other organizations of the US government have participated in this project.

Mikrokosmos is not committed to any particular ontological theory, it is built on more practical considerations (Mahesh, 1996). Its main design principle is a careful distinction between language-specific knowledge represented in the lexicon, and language-neutral knowledge represented in the ontology. Lexicon entries represent word or phrase meanings by mapping these entries to concepts of the ontology.

Figure 2.18 shows the first levels of the ontology. Currently, this ontology has several thousands of concepts, most of which have been generated by retrieving objects, events, and their properties from different sources. Each concept is represented by a frame, which has a name in English, and the following attributes (Mahesh and Nirenburg, 1995): a definition that contains an English string used solely for human browsing purposes, a time-stamp for bookkeeping, taxonomy links

²⁷ <http://crl.nmsu.edu/mikro> (user and password are required)

(*Subclass-Of* and *Instance-Of*), etc. English terms are also used to refer to each concept in the ontology.

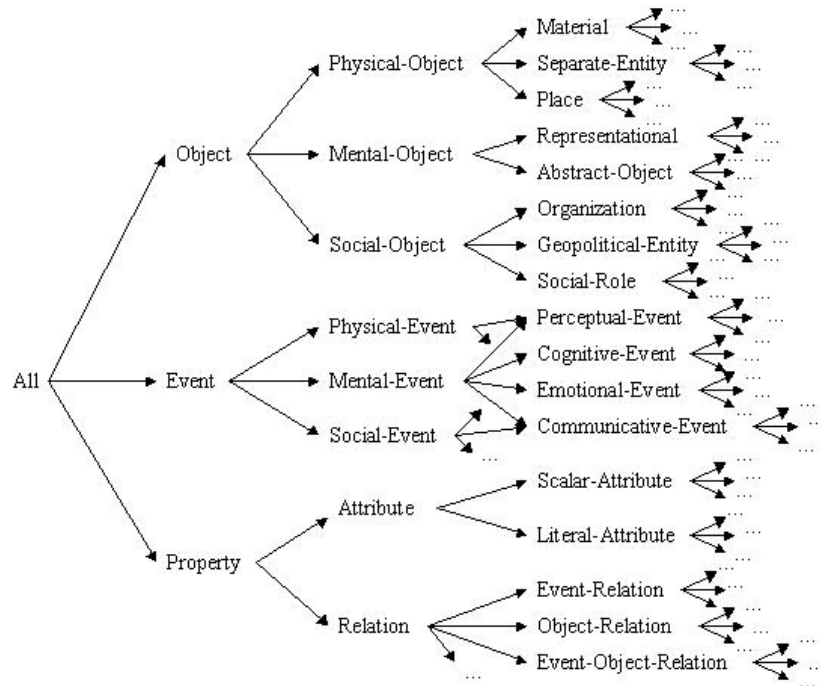


Figure 2.18: Mikrokosmos class taxonomy (from Mahesh and Nirenburg, 1995).

In parallel to the development of the Mikrokosmos ontology, a Spanish lexicon of several thousands words has been built. These words cover a wide variety of categories, though they put particular emphasis on the domain of mergers and acquisitions of companies.

2.3.5 SENSUS

SENSUS²⁸ (Swartout et al., 1997) is a natural language-based ontology developed by the Natural Language group at ISI to provide a broad conceptual structure for working in machine translation.

SENSUS contains more than 70,000 nodes representing commonly encountered objects, entities, qualities and relations. This ontology provides a hierarchically structured concept base (Knight and Luck, 1994). The upper (more abstract) region of the ontology is called the Ontology Base and consists of approximately 400 items that represent essential generalizations for the linguistic processing during translation. The middle region of the ontology provides a framework for a generic

²⁸ <http://www.isi.edu/natural-language/projects/ONTOLOGIES.html>

world model and contains items representing many word senses in English. The lower (more specific) regions of the ontology provide anchor points for different languages.

The current content of the SENSUS ontology was obtained by extracting and merging information from various electronic knowledge sources. This process, as shown in Figure 2.19, began by merging, manually, the PENMAN Upper Model, ONTOS (a very high-level linguistically-based ontology) and the semantic categories taken from a dictionary. As a result, the Ontology Base was produced. WordNet was then merged (again, by hand) with the Ontology Base, and a merging tool was used to merge WordNet with an English dictionary. Finally, and to support machine translation, the result of this merge was increased by Spanish and Japanese lexical entries from the Collins Spanish/English dictionary and the Kenkyusha Japanese/English dictionary.

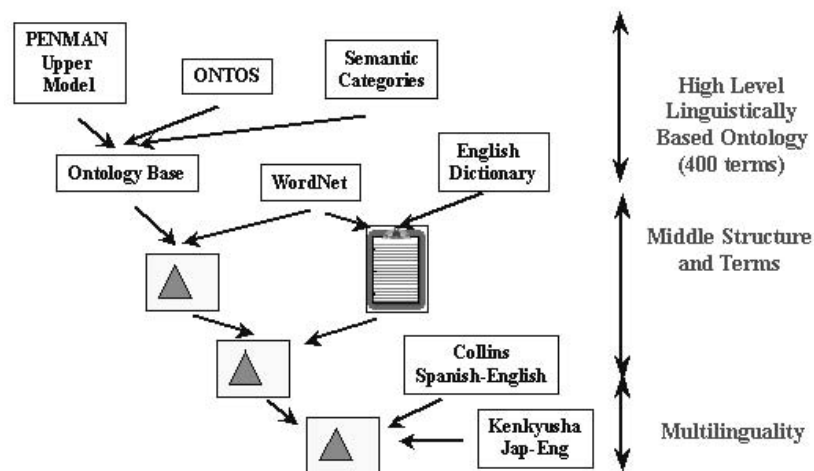


Figure 2.19: SENSUS ontology building process, by extracting and merging information from existing electronic resources (adapted from Swartout et al., 1997).

2.4 Domain Ontologies

As we explained in Chapter 1, domain ontologies (Mizoguchi et al., 1995; van Heijst et al., 1997) are reusable vocabularies of the concepts within a domain and their relationships, of the activities taking place in that domain, and of the theories and elementary principles governing that domain. In this section, we will deal with representative ontologies in the domains of e-commerce, medicine, engineering, enterprise, chemistry, and knowledge management.

2.4.1 E-commerce ontologies

The popularity of the Internet and the huge growth of new Internet technologies in recent years have brought about the creation of many e-commerce applications (Fensel, 2000; Berners-Lee, 1999). Technology is not the only key factor for the development of the current e-applications; the context of e-commerce, especially the context of B2B (Business to Business) applications, requires an effective communication between machines. As a consequence, several standards and initiatives started to ease the information exchange between customers and suppliers, and among different suppliers, by providing frameworks to identify products and services in global markets.

In this section, we present five different proposals to classify products in the e-commerce domain: UNSPSC²⁹, NAICS³⁰, SCTG³¹, e-cl@ss³² and RosettaNet³³. These proposals have been agreed by a wide group of people and organizations, and are codified using different computation languages and formats. Therefore, they provide consensus and also top-level terms that can be used to classify products and services in vertical domains³⁴. However, they cannot be considered heavyweight ontologies but simply lightweight ones, since they consist of concept taxonomies and some relations among them.

The United Nations Standard Products and Services Codes (UNSPSC) has been created by the United Nations Development Programme (UNDP) and Dun & Bradstreet. UNSPSC is a global commodity code standard that classifies general products and services and is designed to facilitate electronic commerce through the exchange of product descriptions.

Initially the UNDP managed the code of the Electronic Commerce Code Management Association (ECCMA)³⁵. This partnership finished, and as a result there are now two different versions of the UNSPSC: the United Nations Standard Products and Services Codes owned by the UNDP, and the Universal Standard Products and Services Classification managed by the ECCMA. In October 2002, both organizations signed an agreement in which they proposed to have one single version of the classification, which has marked the beginning of the UNSPSC unification project.

The UNSPSC coding system is organized as a five-level taxonomy of products, each level containing a two-character numerical value and a textual description. These levels are defined as follows:

²⁹ <http://www.unspsc.org/>

³⁰ <http://www.naics.com>

³¹ <http://www.bts.gov/programs/cfs/sctg/welcome.htm>

³² <http://www.eclasse.de/>

³³ <http://www.rosettanet.org/>

³⁴ Vertical portals usually serve a particular industry and provide deep domain expertise and content. They are normally related with traditional industry segments, such as Electronics, Automotive, Steel, etc. Horizontal portals are characterized by the large number of disperse suppliers and of distributors and resellers.

³⁵ <http://www.eccma.org>

- €# *Segment*. The logical aggregation of families for analytical purposes.
- €# *Family*. A commonly recognized group of inter-related commodity categories.
- €# *Class*. A group of commodities sharing a common use or function.
- €# *Commodity*. A group of products or services that can be substituted.
- €# *Business Function*. The function performed by an organization in support of the commodity. This level is seldom used.

UNSPSC version 6.0315 contains about 20,000 products organized in 55 segments. Segment 43, for instance, which deals with computer equipment, peripherals and components, contains about 300 kinds of products. Figure 2.20 shows part of this segment.

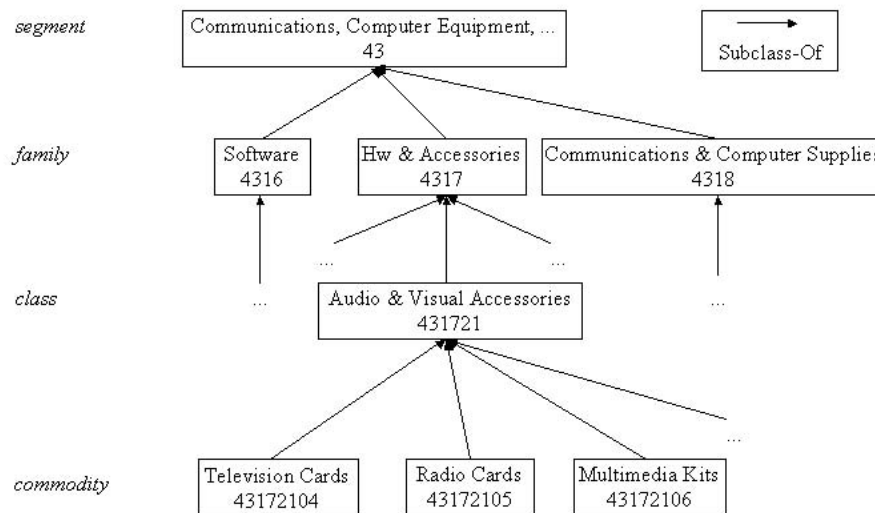


Figure 2.20: Part of the classification of UNSPSC for computer equipment.

NAICS (North American Industry Classification System) was created by the Census Office of USA in cooperation with the Economic National Classification Committee of USA, Statistics Canada, and Mexico's *Instituto Nacional de Estadística, Geografía e Informática* (INEGI). It classifies products and services in general, and is used in USA, Canada and Mexico. NAICS was developed after the Standard Industrial Classification (SIC) was revised. SIC was created in the 1930s to classify establishments according to the type of activity they were primarily engaged in and to promote the comparison of their data describing various facets of the US economy.

NAICS products are identified by means of a six-digit code, in contrast to the four-digit SIC code. The NAICS code includes a greater number of sectors and permits more flexibility to design subsectors. It also provides additional details not necessarily appropriate for all three NAICS countries. The international NAICS agreement fixes only the first five digits of the code. The sixth digit, when used,

identifies subdivisions of NAICS industries that consider the user's needs in individual countries. Thus, six-digit US codes may differ from counterparts in Canada or Mexico, but up to the five-digit level they are standardized. The general structure is:

XX	Industry Sector (20 broad sectors up from 10 SIC)
XXX	Industry Subsector
XXXX	Industry Group
XXXXX	Industry
XXXXXX	US, Canadian, or Mexican National specific

Table 2.5 presents the correspondence between some NAICS Sectors and SIC Divisions. Many of the new sectors reflect parts of SIC divisions, such as the *Utilities* and *Transportation* sectors, that are split from the SIC division *Transportation, Communications, and Public Utilities*.

Table 2.5: Correspondence between some NAICS Sector and SIC Divisions.

Code	NAICS Sectors	SIC Divisions
11	Agriculture, Forestry, Fishing, and Hunting	Agriculture, Forestry and Fishing
21	Mining	Mining
23	Construction	Construction
31-33	Manufacturing	Manufacturing
22	Utilities	Transportation, Communications, and Public Utilities
48-49	Transportation and Warehousing	Public Utilities
42	Wholesale Trade	Wholesale Trade
44-45	Retail Trade	Retail Trade
72	Accommodation and Food Services	
52	Finance and Insurance	Finance, Insurance, and Real Estate
53	Real Estate, Rental and Leasing	

SCTG (Standard Classification of Transported Goods) was sponsored by the Bureau of Transportation Statistics (BTS). It is a product classification for collecting and reporting Commodity Flow Survey (CFS) data. SCTG was developed by the US Department of Transportation's (DOT), Volpe National Transportation Systems Center (Volpe Center), Standards and Transportation Divisions of Statistics Canada, US Bureau of the Census (BOC), and the US Bureau of Economic Analysis (BEA).

This classification has four levels, each of which follows two important principles. First, each level covers the universe of transportable goods, and second, each category in each level is mutually exclusive. The general structure is:

XX	Product Category
XXX	Commodities or Commodity Groups (different in US and Canada)
XXXX	Domestic Freight Transportation Analyses
XXXXX	Freight Movement Data

The first level of the SCTG (two digits) consists of 43 product categories. These categories were designed to emphasize the link between industries and their outputs.

The second level (three digits) is designed to provide data for making comparisons between the Canadian goods and the US goods. Categories specified at this level consist of commodities or commodity groups for which very significant product movement levels have been recorded in both the United States (US) and Canada. The third level (four digits) is designed to provide data for domestic freight transportation analyses. Four-digit categories may be of major data significance to either the US or Canada, but not necessarily to both. The fourth level (five digits) is designed to provide categories for collecting (and potentially reporting) freight movement data. Product codes at this level have been designed to create statistically significant categories for transportation analysis.

Figure 2.21 presents a partial view of this classification. We can see that in this particular classification levels two and three do not contribute with additional classes to the root of the hierarchy. This is so because every branch of the tree has four levels.

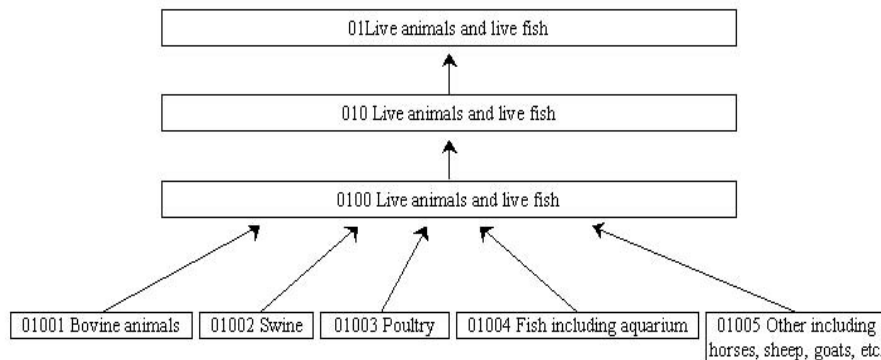


Figure 2.21: Partial view of the SCTG classification.

E-cl@ss is a German initiative to create a standard classification of material and services for information exchange between suppliers and their customers, and companies such as BASF, Bayer, Volkswagen-Audi, SAP, etc., make use of it.

The e-cl@ss classification consists of four levels of concepts (called *material classes*), with a numbering code similar to the one used in UNSPSC (each level adds two digits to its previous level). These four levels are: *Segment*, *Main Group*, *Group* and *Commodity Class*. Inside the same commodity class we can have several products (in this sense, several products can share the same code).

E-cl@ss contains about 12,000 products organized in 21 segments. Segment number 27³⁶, which deals with *Electrical Engineering*, contains about 2,000 products. The main group 27-23, which deals with *Process Control Systems* and with other computer devices, contains about 400 concepts. Figure 2.22 shows a partial view of this classification.

³⁶ Please note that the numbering of segments is not consecutive.

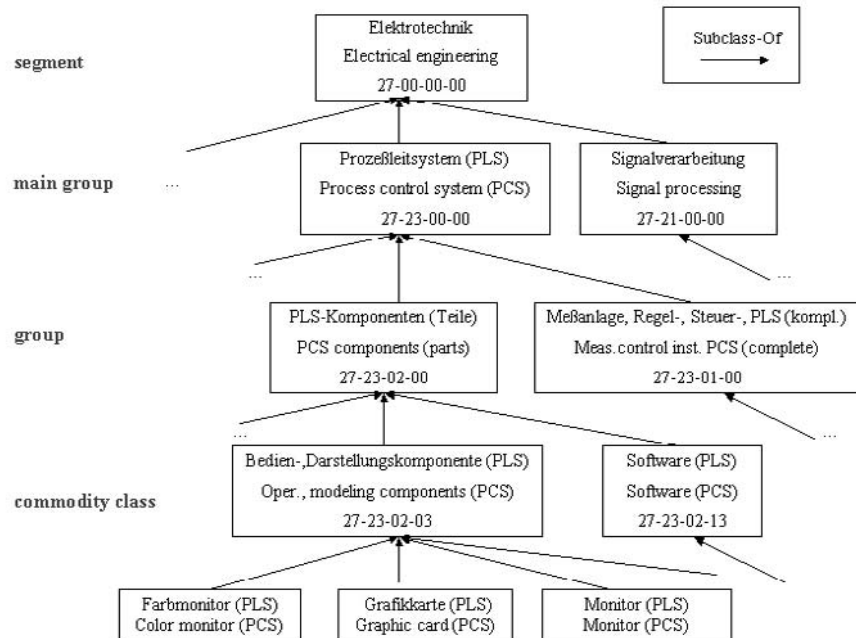


Figure 2.22: Part of the classification of e-cl@ss for electrical engineering products (German and English).

E-cl@ss provides a set of attributes for every product that is a leaf in the classification. The set of attributes is an addition of individual characteristics describing the related commodity. This set distinguishes e-cl@ss from UNSPSC and offers a solution to the shallowness of that. For example, *PC System* (with code 24-01-99-03) has attributes like *product type*, *product name*, etc., in e-cl@ss.

The e-cl@ss search tool, which is available on-line³⁷, allows finding terms with an interface in different languages (German, Spanish, English and Czech). In fact, the terms found are presented in any of these languages. The e-cl@ss classification can also be downloaded from the same URL.

The **RosettaNet** classification has been created by RosettaNet, which is a self-funded, non-profit consortium of about 400 companies of Electronic Components, Information Technology, Semiconductor Manufacturing and Solution Provider companies. Started in the IT industry, RosettaNet is currently being expanded to other vertical areas, notably the automotive, consumer electronics and telecommunications industries.

The RosettaNet classification does not use a numbering system, as UNSPSC does, but is based on the names of the products it defines. This classification is

³⁷ <http://www.eiclass.de/>

related to the UNSPSC classification and provides the UNSPSC code for each product defined in RosettaNet. This classification has only two levels in its product taxonomy:

€# *RN Category*. A group of products, such as *Video Products*.

€# *RN Product*. A specific product, such as *Television Card*, *Radio Card*, etc.

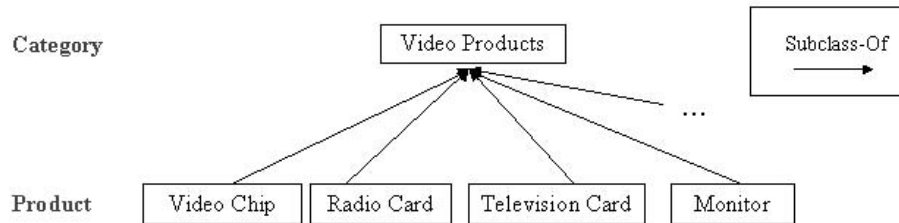


Figure 2.23: Partial view of the RosettaNet classification.

The RosettaNet classification consists of 14 categories and about 150 products. It should be added that RosettaNet is more specific than the UNSPSC classification. Figure 2.23 shows a small section of the RosettaNet classification related to video products for computer equipment, and table 2.6 presents the classification structured as in its original Microsoft Excel format. Unlike in the previous formats, the order of contents here is of great importance, since the relationship between products and the category they belong to are given by the order in which they appear. Hence *Monitor*, *RadioCard*, *TelevisionCard* and *VideoChip* are products from the category *Video Products* in the RosettaNet classification.

Table 2.6: Source format for the RosettaNet classification of video products.

RosettaNet Category Name	RosettaNet Product Name	UNSPSC Code	UNSPSC Code Name
Video Products			
	Monitor	43172401	Monitors
	Radio Card	43172105	Radio Cards
	Television Card	43172104	Television Cards
	Video Chip	321017	Hybrid Integrated Circuits

In this section, we have described five classifications of products and services (UNSPSC, NAICS, SCTG, e-cl@ss, and RosettaNet), which present a big overlap between them, so that a product or service could be classified in different places in each classification. The proliferation of initiatives reveals that B2B markets have not reached a consensus on coding systems, on level of detail, on granularity, etc., which is an obstacle for the interoperability of applications following different standards. For instance, an application that uses the UNSPSC code cannot interoperate with an application that follows the e-cl@ss coding system. To align such initiatives, some works have proposed to establish ontological mappings between existing standards (Bergamaschi et al., 2001; Corcho and Gómez-Pérez, 2001; Gordijn et al., 2001).

Figure 2.24 shows an example of *Equivalent-To* and *Subclass-Of* relations between concepts of the RosettaNet and the UNSPSC classifications. The concept *Video Chip* of the RosettaNet classification is a subclass of the concept *Hybrid Integrated Circuits* of the UNSPSC classification, the concept *Monitor* of RosettaNet is equivalent to the concept *Monitors* of UNSPSC, etc. As we can see, two sibling concepts in RosettaNet are classified in different UNSPSC classes: *Monitor* and *Radio Card* are subclasses of the same concept in RosettaNet (*Video Products*), while their equivalent concepts in UNSPSC (*Monitors* and *Radio Cards* respectively) are subclasses of different concepts in that classification (*Monitors & Displays* and *Radio Cards* respectively).

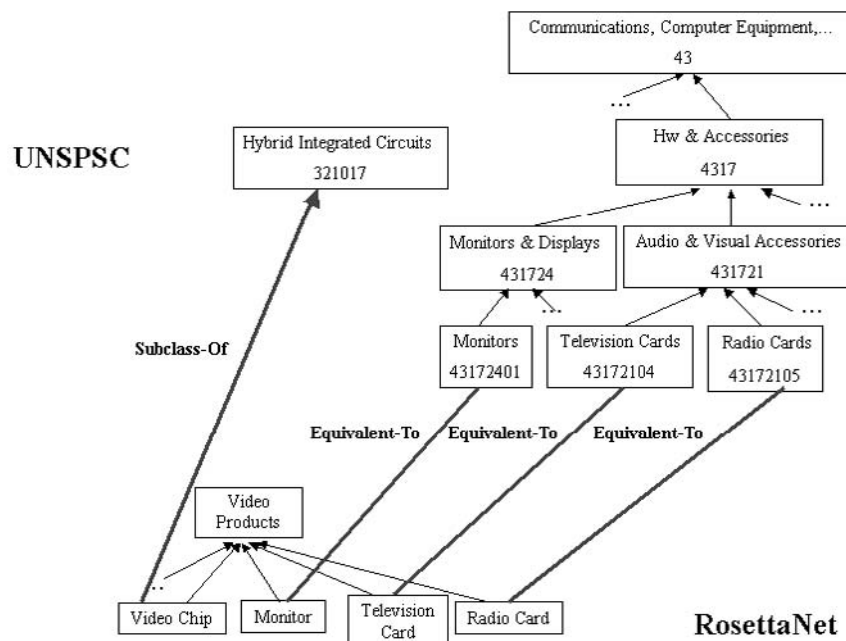


Figure 2.24: Equivalence relationships between the RosettaNet and UNSPSC classifications (Corcho and Gómez-Pérez, 2001).

2.4.2 Medical ontologies

Medical ontologies are developed to solve problems such as the demand for the reusing and sharing of patient data, the transmission of these data, or the need of semantic-based criteria for statistical purposes. The unambiguous communication of complex and detailed medical concepts is a crucial feature in current medical information systems. In these systems several agents must interact between them in order to share their results and, thus, they must use a medical terminology with a clear and non-confusing meaning.

GALEN³⁸ (Rector et al., 1995), developed by the non-profit organization OpenGALEN, is a clinical terminology represented in the formal and medical-oriented language GRAIL (Rector et al., 1997). This language was specially developed for specifying restrictions used in medical domains. GALEN was intended to be used with different natural languages and integrated with different coding schemata. It is based on a semantically sound model of clinical terminology known as the GALEN CODing REference (CORE) model. Figure 2.25 shows the GALEN CORE top-level ontology.

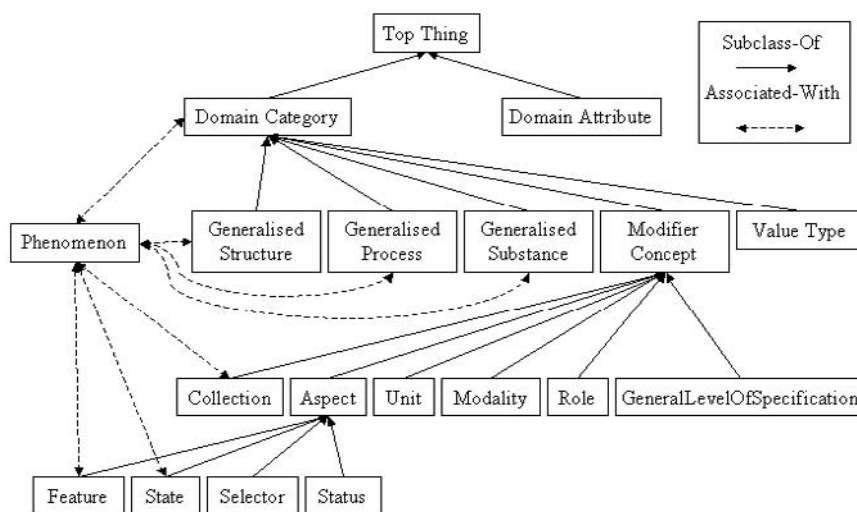


Figure 2.25: GALEN CORE top-level ontology.

The GALEN CORE top-level ontology establishes four general categories (which are subclasses of *DomainCategory*):

- €# Structures (*GeneralisedStructure*), which are abstract or physical things with parts that are time-independent (such as *microorganism*, *protocol* or *heart*).
- €# Substances (*GeneralisedSubstance*), which are continuous abstract or physical things that are time-independent, such as *bile*, *drugs* or *radiation*.
- €# Processes (*GeneralisedProcess*), which are changes that occur over time, such as *irradiation*, *clinical act* or *breathing*.
- €# Modifiers (*ModifierConcept*), which refine or modify the meaning of the other three categories, such as *severe diabetes*. In this ontology the following types of modifiers are considered: modifiers of *aspect* (classified in *feature*, *state*, *selector*, and *status*), *unit*, *modality*, *role*, *general level of specification*, and *collection*.

The category *ValueType* is also a subclass of *DomainCategory*. It defines value types such as *Integer*, *Ordinal*, etc. Besides, the category *Phenomenon* is included in this top-level ontology, as shown in figure 2.25. It gathers the medical intuitions of *Disease* and *Disorder* and it is associated with domain categories, more specifically

³⁸ <http://www.opengalen.org/>

with structures, processes, and substances, and with the modifiers of feature, state, and collection.

Finally, the GALEN CORE top-level ontology defines relationships between concepts that belong to general categories. These relationships are called attributes (*DomainAttribute*) and are divided into two types: *constructive attributes*, which link processes, structures and substances together; and *modifier attributes*, which link processes, structures and substances to modifiers.

UMLS³⁹ (Unified Medical Language System), developed by the United States National Library of Medicine, is a large database designed to integrate a great number of biomedical terms collected from various sources (over 60 sources in the 2002 edition) such as clinical vocabularies or classifications (MeSH, SNOMED, RCD, etc.).

UMLS is structured in three parts: *Metathesaurus*, *Semantic Network* and *Specialist Lexicon*.

The *Metathesaurus* contains biomedical information about each of the terms included in UMLS. If a term appears in several sources, which is usual, a concept will be created in UMLS with a preferred term name associated to it. The original source information about the terms (such as, definition, source, etc.) is attached to the concept and some semantic properties are also specified, such as concept synonyms, siblings and parents, or the relationships between terms. In the UMLS edition of the year 2002, the *Metathesaurus* contained about 1,5 million terms.

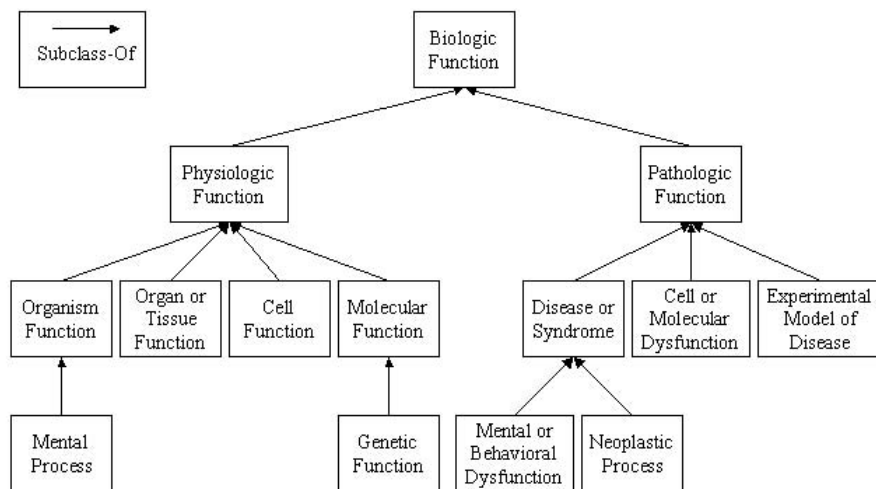


Figure 2.26: Part of the Semantic Network of the UMLS ontology.

³⁹ <http://www.nih.gov/research/umls/>

- €# The *Semantic Network* is a top-level ontology of biomedical concepts and relations among these concepts. Figure 2.26 shows a partial view of this top-level ontology. The *Semantic Network* was not derived from the biomedical sources integrated in UMLS but created as a part of UMLS with the aim of providing a consistent structure or categorization in which the *Metathesaurus* concepts are included. Each Metathesaurus concept is attached to a concept or concepts of the *Semantic Network*. Thus, the *Semantic Network* was introduced in order to solve the heterogeneity among the UMLS sources, and it could be considered the result of the integration of the UMLS sources. In the 2002 edition, the *Semantic Network* contained 134 top-level concepts and 54 relationships among them.
- €# The *Specialist Lexicon* contains syntactic information about biomedical terms to be used in natural language processing applications.

ON9⁴⁰ (Gangemi et al., 1998) is a medical set of ontologies that includes some terminology systems, like UMLS. Figure 2.27 shows an inclusion network of some ON9 ontologies. Here, ontologies are represented with boxes. Thick dashed boxes are sets of ontologies (some show the elements explicitly). Continuous arrows mean *included in*, and dashed arrows mean *integrated in*. The ontologies at the top of the hierarchy are the *Frame Ontology* and the set of *KIF ontologies* (Gangemi et al., 1998).

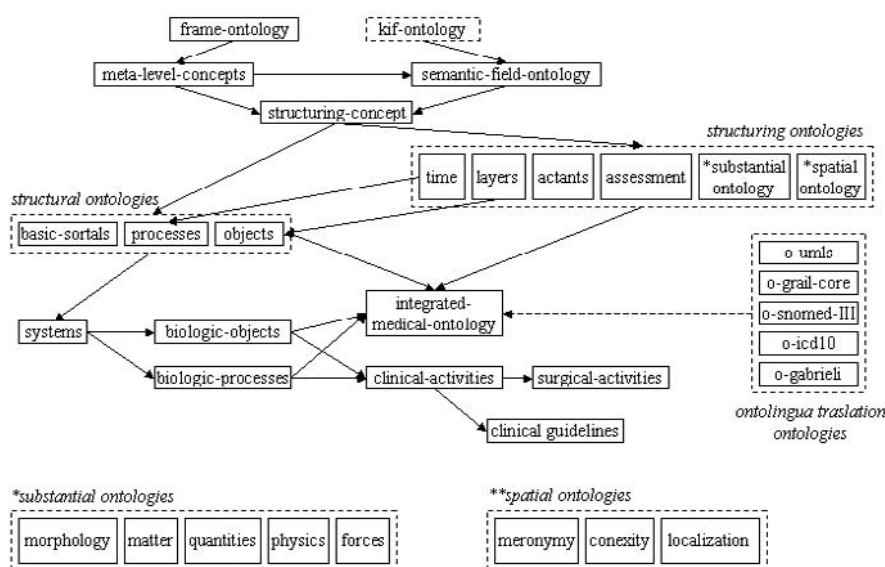


Figure 2.27: A significant subset of the inclusion network of the ON9 library of ontologies (from Gangemi et al., 1998).

⁴⁰ <http://saussure.irmkant.rm.cnr.it/ON9/index.html>

To link these ontologies with the generic ontology library several ontologies have been defined: *Structuring-Concepts*, *Meta-Level-Concepts* and the *Semantic-Field-Ontology*. The sets of *Structural ontologies* and of *Structuring ontologies* contain generic ontologies. Particularly, the *Integrated-Medical-Ontology* includes all the generic ontologies used to gather the terminological ontologies of the five terminological systems.

2.4.3 Engineering ontologies

Engineering ontologies contain mathematical models that engineers use to analyze the behavior of physical systems (Gruber and Olsen, 1994). These ontologies are created to enable the sharing and reuse of engineering models among engineering tools and their users. Among the various engineering ontologies, the EngMath ontologies and PhysSys deserve special mention.

EngMath (Gruber and Olsen, 1994) is a set of Ontolingua ontologies developed for mathematical modeling in engineering. These ontologies include conceptual foundations for scalar, vector, and tensor quantities as well as functions of quantities, and units of measure.

When the EngMath ontologies were designed, the developers had three kinds of uses in mind. First, these ontologies should provide a machine and human-readable notation for representing the models and domain theories found in the engineering literature. Second, they should provide a formal specification of a shared conceptualization and a vocabulary for a community of interoperating software agents in engineering domains. And third, they should put the base for other formalization efforts including more comprehensive ontologies for engineering and domain-specific languages.

In Figure 2.28, we can see some of the ontologies that make up EngMath:

- €# *Abstract-Algebra*. It defines the basic vocabulary for describing algebraic operators, domains, and structures such as *fields*, *rings*, and *groups*.
- €# *Physical-Quantities*. This ontology models the concept of physical quantity. A physical quantity is a measure of quantifiable aspect. The ontology *Physical-Quantities* has 11 classes, three relations, 12 functions (*addition*, *multiplication*, *division*, etc.), and two instances.
- €# *Standard-Dimensions*. It models the physical quantities most commonly used. It has 18 classes (*mass quantity*, *length quantity*, *temperature quantity*, etc.), and 27 instances.
- €# *Standard-Units*. This ontology defines a set of basic units of measure. It has one class (*Si-Unit*, that is, unit of the International System), and 60 instances (*kilogram*, *meter*, *Celsius-degree*, etc.).
- €# *Scalar-Quantities*. This ontology permits modeling quantities whose magnitude is a real number, for example, “6 kilograms”. It has one class (*scalar quantity*), six functions (which specialize functions of the ontology *Physical-Quantities* for scalar quantities), and one instance.

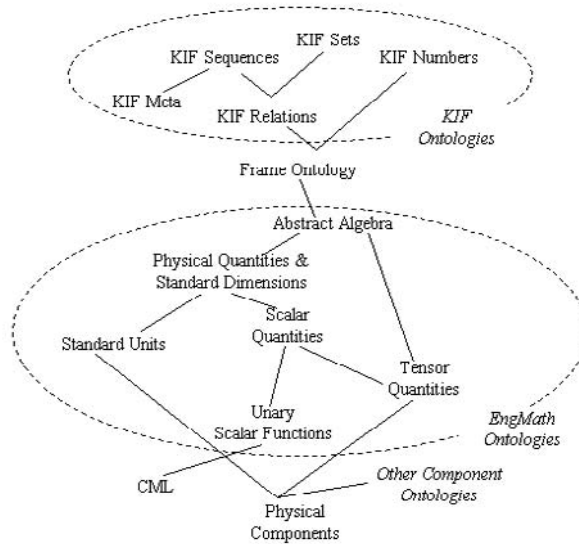


Figure 2.28: Structure of the EngMath ontologies (from Gruber and Olsen, 1994).

EngMath has been an important experimental base to establish the design criteria for building ontologies described in Section 1.6.

PhysSys (Borst, 1997) is an engineering ontology for modeling, simulating and designing physical systems. It forms the basis of the OLMECO library⁴¹, a model component library for physical systems such as heating systems, automotive systems and machine tools. PhysSys formalizes the three viewpoints of physical devices: system layout, physical process behavior and descriptive mathematical relations.

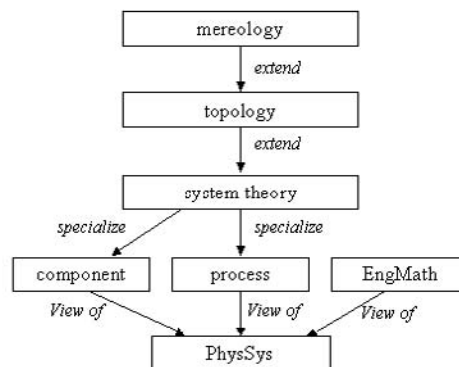


Figure 2.29: Structure of the PhysSys ontologies (from Borst, 1997).

⁴¹ <http://www.rt.el.utwente.nl/bnk/olmeco.htm>

Figure 2.29 gives a general view of the structure of PhysSys ontology. As we can observe, this structure is a pyramid where the most specific ontologies (the nearest to the base) import the most general ones. Next, we describe briefly each of the PhysSys ontologies:

- €# *Mereology Ontology*. It defines the relation *Part-Of* and its properties. This relation permits stating that devices are formed by components, which in their turn, can be made up of smaller components. The *Mereology Ontology* is an Ontolingua implementation of the Classical Extensional Mereology described in Simons (1987).
- €# *Topology Ontology*. It defines the relation *is-connected-to* and its properties. This ontology is useful to describe the physical behavior of devices since it represents how the components interact inside the system.
- €# *System Theory Ontology*. It defines standard system-theoretic notions such as *system*, *sub-system*, *system boundary*, *environment*, etc.
- €# *Component Ontology*. It is focused on the structural aspects of devices and is useful for representing what kind of dynamic processes occur in the system. The *Component Ontology* is constructed with the *Mereology Ontology*, the *Topology Ontology*, and the *System Theory Ontology*.
- €# *Physical Process Ontology*. It specifies the behavioral view of physical systems.
- €# *Mathematical Ontology*. It defines the mathematics required to describe physical processes.

2.4.4 Enterprise ontologies

Enterprise ontologies are usually created to define and organize relevant knowledge about activities, processes, organizations, strategies, marketing, etc. All this knowledge is meant to be used by enterprises. Here, we will present the Enterprise Ontology and TOVE. Both have been essentially the experimental basis of some methodological approaches of ontology engineering presented in Chapter 3.

The **Enterprise Ontology**⁴² (Uschold et al., 1998) was developed within the Enterprise Project by the Artificial Intelligence Applications Institute at the University of Edinburgh with its partners IBM, Lloyd's Register, Logica UK Limited, and Unilever. The project was supported by the UK's Department of Trade and Industry under the Intelligent Systems Integration Programme (project IED4/1/8032). This ontology contains terms and definitions relevant to businesses. It is implemented in Ontolingua and it has 92 classes, 68 relations, seven functions and 10 individuals. Figure 2.30 shows a partial view of the class taxonomy.

Conceptually, the Enterprise Ontology is divided into four main sections:

- €# Activities and processes. The central term here is *Activity*, which is intended to capture the notion of anything that involves doing, particularly when this indicates action. The concept of *Activity* is closely linked to the idea of the *Doer*, which may be a *Person*, *Organizational-Unit* or *Machine*.

⁴² <http://www.aiai.ed.ac.uk/~enterprise/enterprise/ontology.html>

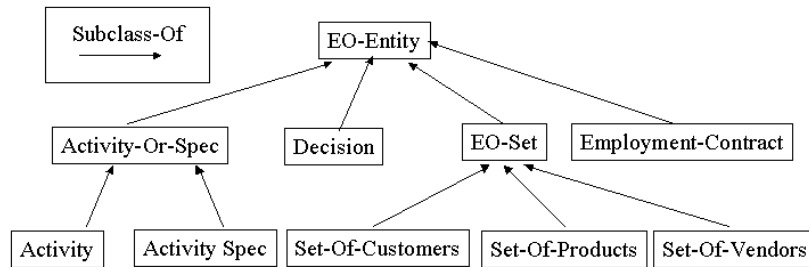


Figure 2.30: Partial view of the taxonomy of the Enterprise Ontology.

- £# Organization. The central concepts of the Organization section are: *Legal-Entity* and *Organizational-Unit*.
- £# Strategy. The central concept of this section is *Purpose*. *Purpose* captures the idea of something that a *Plan* can *help achieve* or the idea that an *Organizational-Unit* can be responsible for. In fact, this section includes any kind of *purpose*, whether in a level of organization and time scale (normally called strategic), or in a detailed and short term.
- £# Marketing. The central concept of this section is *Sale*. A *Sale* is an agreement between two *Legal-Entities* for the exchange of a *Product* for a *Sale-Price*. Normally the *Products* are goods or services and the *Sale-Price* is monetary, although other possibilities are included. The *Legal-Entities* play the (usually distinct) roles of *Vendor* and *Customer*. A *Sale* may have been agreed on in the past, and a future *Potential-Sale* can be envisaged, whether the actual *Product* can or cannot be identified and whether it exists or not.

This ontology has been a relevant experimental foundation for the Uschold and King's (1995) approach to develop ontologies, which is described in Chapter 3.

The **TOVE**⁴³ (TOronto Virtual Enterprise) (Fox, 1992) project is being carried out by the Enterprise Integration Laboratory (EIL) at the University of Toronto. Its goal is to create a data model able (1) to provide a shared terminology for the enterprise that agents can both understand and use, (2) to define the meaning of each term in a precise and unambiguous manner, (3) to implement the semantics in a set of axioms that will enable TOVE to deduce automatically the answer to many "common sense" questions about the enterprise, and (4) to define a symbology for depicting a term, or the concept constructed thereof in a graphical context.

TOVE ontologies are implemented with two different languages: C++ for the static part, and Prolog for the axioms.

Figure 2.31 shows the structure of the TOVE ontologies. Up to now, the existing ontologies developed to model Enterprises are: Foundational Ontologies (Activity and Resource) and Business Ontologies (Organization, Quality, Products and Requirements). These ontologies cover activities, state, causality, time, resources, inventory, order requirements, and parts. They have also axiomatized the definitions for portions of the knowledge of activity, state, time, and resources.

⁴³ <http://www.eil.utoronto.ca/tove/toveont.html>

Axioms are implemented in Prolog and provide answers for common-sense questions via deductive query processing.

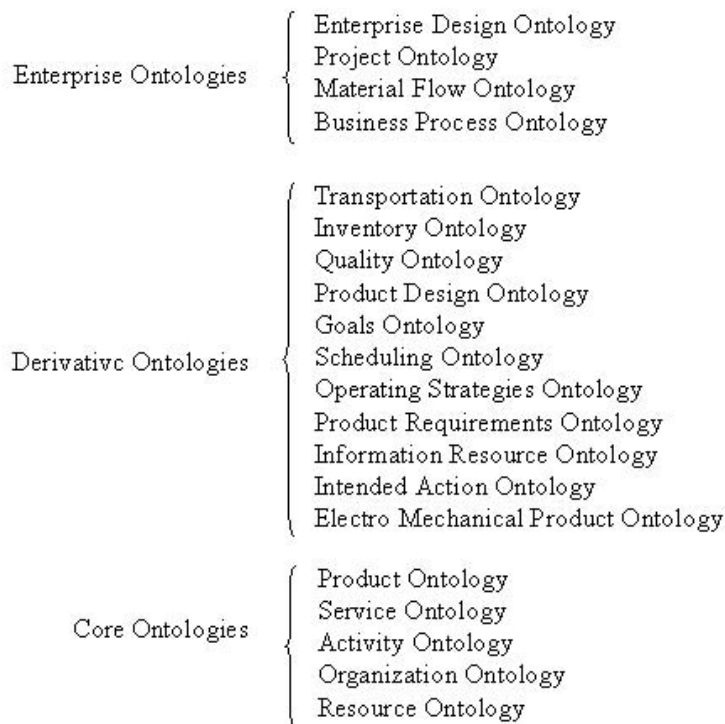


Figure 2.31: Structure of the TOVE ontologies.

According to TOVE developers, their future work will be confined to the development of ontologies and axioms for quality, activity-based costing, and organization structures.

This ontology has been an important experimental basis for the Grüninger and Fox's (1995) method to develop ontologies, described in Chapter 3.

2.4.5 Chemistry ontologies

Chemistry ontologies model the composition, structure, and properties of substances, processes and phenomena. They can be used for many purposes: for education (to teach students the periodic table of elements, the rules for molecule composition, etc.), for environmental science (to detect environmental pollutants), for scientific discovery (to analyze publications to learn about new molecules' composition or to synthesize chemicals), etc. We will describe the set of chemistry ontologies developed by the Ontology Group of the Artificial Intelligence Laboratory at UPM: *Chemicals* (composed of *Chemical Elements* and *Chemical*

Crystals), *Ions* (composed of *Monatomic Ions* and *Polyatomic Ions*) and *Environmental Pollutants*. All of them are available in WebODE, and both the *Chemical Elements* and the *Chemical Crystals* ontologies are also implemented in Ontolingua and available in the Ontolingua Server.

Figure 2.32 shows how all these ontologies are integrated in a hierarchical architecture. It should be interpreted that the ontologies on top of this hierarchy include the lower-level ontologies. Note that this hierarchical architecture facilitates future users the comprehension, design and maintenance of ontologies. As we can see, *Chemical Elements* is a key point in this ontology structure. It imports the *Standard-Units* ontology available in the Ontolingua Server, which, in its turn, imports other ontologies in the same ontology server, such as *Standard-Dimensions*, *Physical-Quantities*, *KIF-Numbers* and the *Frame-Ontology*.

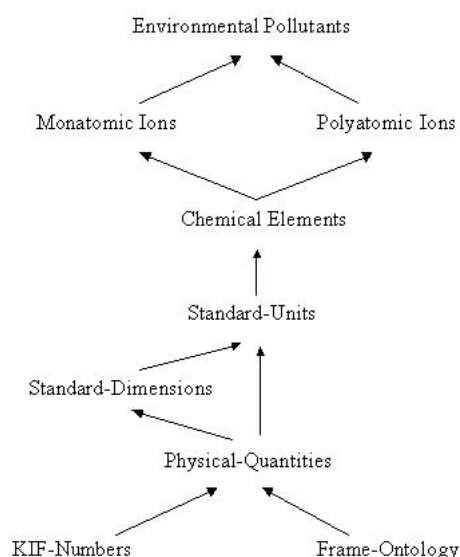


Figure 2.32: Relationship between the chemistry ontologies described in this section and other ontologies in the Ontolingua Server.

Chemicals is composed of two ontologies: *Chemical Elements* and *Chemical Crystals*. These ontologies were used to elaborate METHONTOLOGY (Fernández-López et al., 1999), an ontology development methodology that will be described in Chapter 3.

The *Chemical Elements* ontology models knowledge of the chemical elements of the periodic table, such as what elements these are (*Oxygen*, *Hydrogen*, *Iron*, *Gold*, etc.), what properties they have (*atomic number*, *atomic weight*, *electronegativity*, etc.), and what combination constraints of the attribute values they have. *Chemical Elements* contains 16 classes, 20 instance attributes, one function, 103 instances and 27 formal axioms.

Chemical Crystals was built to model the crystalline structure of the chemical elements. Therefore, *Chemical Elements* imports this ontology. The ontology contains 19 classes, eight relations, 66 instances and 26 axioms.

Ions is built on top of *Chemical Elements* and is also composed of two ontologies: *Monatomic Ions* (which model ions composed of one atom only) and *Polyatomic Ions* (which model ions composed of two or more atoms). Ions contains 62 concepts, 11 class attributes, three relations and six formal axioms.

Finally, the **environmental pollutants** ontology (Gómez-Pérez and Rojas, 1999) imports *Monatomic Ions* and *Polyatomic Ions* and is composed of three ontologies: *Environmental Parameters*, *Water* and *Soil*. The first ontology defines parameters that might cause environmental pollution or degradation in the physical environment (air, water, ground) and in humans, or more explicitly, in their health. The second and third ontologies define water and soil pollutants respectively. These ontologies define the methods for detecting pollutant components of various environments, and the maximum concentrations of these components permitted according to the legislation in force.

2.4.6 Knowledge management ontologies

The objectives of knowledge management (KM) in an organization are to promote knowledge growth, knowledge communication and knowledge preservation in the organization (Steels, 1993). To achieve these objectives corporate memories can be created. A corporate memory is an explicit, disembodied, persistent representation of knowledge and information in an organization (van Heijst et al., 1996). According to Dieng-Kuntz and colleagues (1998, 2001), a corporate memory can be built following different techniques that can be combined: document-based, knowledge-based, case-based, groupware-based, workflow-based and distributed. Ontologies are included among the knowledge-based techniques for building corporate memories.

Basically, Abecker and colleagues (1998) distinguish three types of KM ontologies:

- €# *Information ontologies*. They describe the different kinds of information sources, their structure, access permissions, and format properties.
- €# *Domain ontologies*. They model the content of the information sources.
- €# *Enterprise ontologies*. They model the context of an organization, business process, organization of the enterprise, etc., as described in Section 2.4.4.

Figure 2.33 shows an example of the existing relations between the aforementioned ontologies. The ontologies presented in the figure have been built for the corporate memory of the Artificial Intelligence Laboratory at UPM in Madrid. There are two domain ontologies: *Hardware&Software* (which models the hardware equipment of the laboratory and the software installed in it) and *Documentation* (which models all the documents generated in the laboratory: publications, theses, faxes, etc.). There are four enterprise ontologies: *Organization* and *Groups* (which model the structure

of the laboratory and its organization in different research groups), *Projects* (which models the projects that are developed in the laboratory), and *Persons* (which models the members of the laboratory: research staff, administrative staff, students, etc.). Three of these ontologies are also information ontologies: *Documentation*, *Projects* and *Persons* since they describe the information sources of the corporate memory and store information.

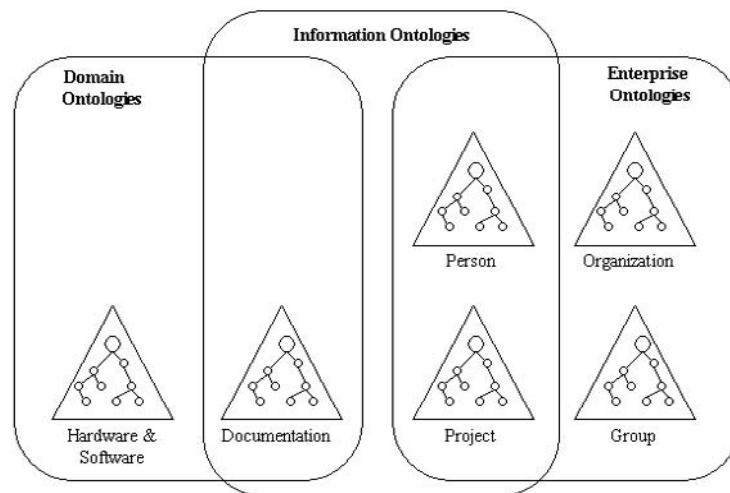


Figure 2.33: Information, domain and enterprise ontologies in a corporate memory for a research and development laboratory.

The **(KA)² ontologies** (Decker et al., 1999) are also good examples of KM ontologies. They were built inside the Knowledge Annotation Initiative of the Knowledge Acquisition community (Benjamins et al., 1999), also known as the (KA)² initiative. Its goal was to model the knowledge-acquisition community with the ontologies built by 15 groups of people at different locations. Each group focused on a particular topic of the (KA)² ontologies (problem solving methods, ontologies, etc.). The result was seven related ontologies: an *organization ontology*, a *project ontology*, a *person ontology*, a *publication ontology*, an *event ontology*, a *research-topic ontology* and a *research-product ontology*. They formed the basis to annotate WWW documents of the knowledge acquisition community, and thus to enable intelligent access to these Web documents.

The first release of the (KA)² ontologies was built in the FLogic language (Kifer et al., 1995), which is described in Chapter 4. These FLogic ontologies were translated into Ontolingua with ODE translators (Blázquez et al., 1998) to make them accessible to the entire community through the European mirror of the Ontolingua Server in Madrid⁴⁴. The updated version of (KA)² is not in Ontolingua

⁴⁴ <http://granvia.dia.fi.upm.es:5915/> Log in as “ontologias-ka2” with password “adieu007”

but in DAML+OIL and is maintained at the AIFB (Institute for Informatics and Formal Description Methods) of the University of Karlsruhe⁴⁵.

In the context of the European IST project Esperonto⁴⁶, five KM ontologies have been developed in WebODE to describe **R&D projects**: *Project*, *Documentation*, *Person*, *Organization*, and *Meeting*. These ontologies describe R&D projects and their structure, documents that are generated in a project, people and organizations participating in it, and meetings (administrative, technical, etc.) held during a project lifecycle. Figure 2.34 shows the relationships between all these ontologies (a project has associated meetings, a document belongs to a project, a document summarizes a meeting, people have a role in a project, etc.).

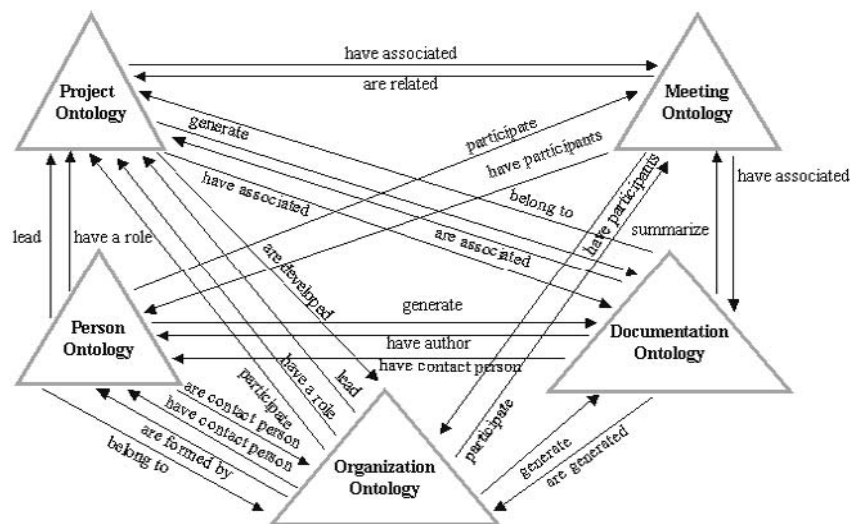


Figure 2.34: Main ad hoc relationships between KM ontologies for R&D projects.

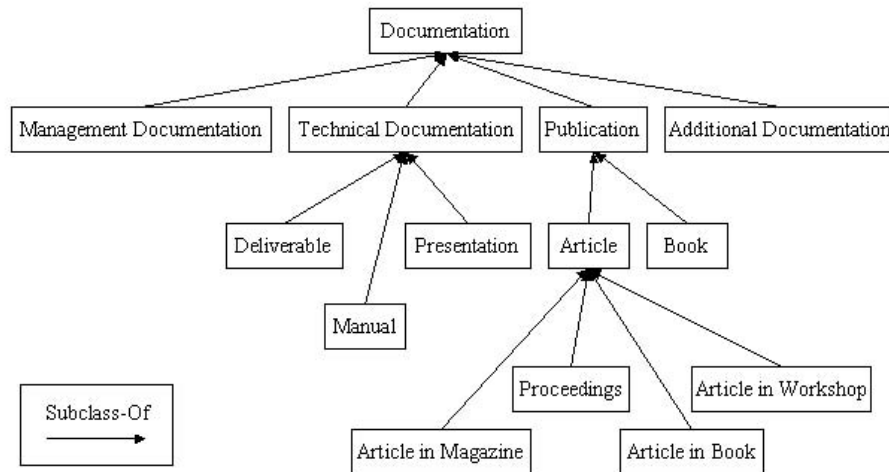
Part of the concept taxonomy of the *Documentation* ontology is presented in Figure 2.35. As the figure shows, we have distinguished four types of documents in a project: management documents, technical documents, publications and additional documents. Some of the technical documents generated in a project are deliverables, manuals and presentations. Publications can be done as books or articles, and there are different types of articles, depending on where they are published: in workshops, as part of the proceedings of a conference, in books or in magazines and journals.

These ontologies can be accessed at the Esperonto Web site, powered by the knowledge portal ODESeW⁴⁷.

⁴⁵ <http://ka2portal.aifb.uni-karlsruhe.de>

⁴⁶ <http://www.esperonto.net/>

⁴⁷ <http://webode.dia.fi.upm.es/sew/index.html>

Figure 2.35: Fragment of the *documentation ontology* of R&D projects.

2.5 Bibliographical Notes and Further Reading

To those readers who want to have a thorough grounding in the contents of this chapter, we recommend the following readings, grouped by topics:

- £# *Ontologies in general.* The deliverable D.1.1 “Technical Roadmap” of the OntoWeb thematic network, funded by the European Commission, contains brief descriptions of and references to the most outstanding ontologies. The OntoRoadMap application, which has been created inside the project OntoWeb (<http://babage.dia.fi.upm.es/ontoweb/wp1/OntoRoadMap/index.html>), lets researchers consult and update the information on existing ontologies.
- £# *Knowledge representation ontologies.* We recommend to download the KR ontologies that have been presented in Section 2.1: the Frame Ontology and the OKBC Ontology (available at the Ontolingua Server: <http://ontolingua.stanford.edu/>), the RDF and RDF Schema KR ontologies (available at <http://www.w3.org/RDF/>), the OIL KR ontology (available at <http://www.ontoknowledge.org/oil/>), the DAML+OIL KR ontology (available at <http://www.daml.org/language/>), and the OWL KR ontology (available at <http://www.w3.org/2001/sw/WebOnt/>).
- £# *Top-level ontologies.* More information about top-level ontologies can be found in <http://www.sop.inria.fr/acacia/personnel/phmartin/RDF/phOntology.html>. We also advise to consult the SUO Web page at <http://suo.ieee.org/>.
- £# *Linguistic ontologies.* For linguistic ontologies, we recommend periodically access to the Web page of the OntoWeb SIG on Language Technology in Ontology Development and Use (<http://dfki.de/~paulb/ontoweb-lt.html>). Other linguistic ontologies different to the ones dealt with in this chapter are: Corelex, which is presented in <http://www.cs.brandeis.edu/~paulb/CoreLex/corelex.html>

and (Buitelaar, 2001); EDR, presented in <http://www.iiijnet.or.jp/edr/> and (Miyoshi et al., 1996); and the Goi-Takei's ontology, presented in (Ikehara et al., 1997) and <http://www.kecl.ntt.co.jp/icl/mtg/topics/lexicon-index.html>.

€# *E-commerce ontologies.* You can read about e-commerce ontologies in the deliverable D3.1 of OntoWeb (IST-2000-29243), which deals with e-commerce content standards. This deliverable is available in the following URL: <http://www.ontoweb.org/download/deliverables/D3.1.pdf>. We also recommend to read about the initiatives for aligning e-commerce classifications (Bergamaschi et al., 2001; Corcho and Gómez-Pérez, 2001; Gordijn et al., 2000).

€# *Medical ontologies.* More information about the GALEN ontology can be found in <http://www.opengalen.org/resources.html>. We recommend to download the open source tool OpenKnoME (<http://www.topthing.com>) to compile and to browse the GALEN sources. Access to UMLS sources and resources (applications and documentation) is free, though it is necessary to sign the UMLS license agreement at <http://www.nlm.nih.gov/research/umls/license.html>. A general overview of medical ontologies is presented by Bodenreider (2001).

€# *Chemistry ontologies.* One of the data sets included in the DAML data sources wishlist (<http://www.daml.org/data/>) is the periodic table of the chemical elements, which would be considered as a reference data set for chemical and related industries, probably combined with other chemistry ontologies including elements, compounds, etc. So we recommend to take a close look at this effort.

€# *Content standards.* We recommend to have a look at the Special Interest Group on Content Standards of OntoWeb (IST-2000-29243), whose URL is: <http://www.ladseb.pd.cnr.it/infor/ontology/OntoWeb/SIGContentStandards.htm>. The goal of this SIG is to coordinate cooperation and participation with current initiatives related to ontology-based content standardization and content harmonization across different standards.

€# *Legal ontologies.* We recommend to have a look at the Legal Ontologies Working Group within the OntoWeb Content Standards SIG, whose URL is: <http://ontology.ip.rm.cnr.it/legontoweb.html>. The objective of this working group is (1) to collect the (possibly formalized) ontologies proposed so far in the literature and still maintained or 'alive'; (2) to contact the reference persons; (3) to create a Web page in which a general outline of Legal Ontologies is presented; and (4) to provide a preliminary description of the steps to reach a "Common Core Legal Ontology Library". Several events have dealt with legal ontologies, such as <http://lri.jur.uva.nl/jurix2001/legont2001.htm>, <http://www.cs.wustl.edu/icail2001/>, and <http://www.cfslr.ed.ac.uk/icail03/>. Other URLs related to legal ontologies are: <http://www.csc.liv.ac.uk/~lial/>, <http://www.lri.jur.uva.nl>, <http://www.idg.fi.cnr.it/researches/researches.htm>, and <http://www.austlii.edu.au/au/other/col/1999/35/>.

Ontological Engineering
with examples from the areas of Knowledge
Management, e-Commerce and the Semantic Web. First
Edition

Gómez-Pérez, A.; Fernandez-Lopez, M.; Corcho, O.
2004, XII, 404 p., Hardcover
ISBN: 978-1-85233-551-9