Aggregation in Ontologies: Practical Implementations in OWL

Csaba Veres

Dept. of Computer and Information Science (IDI)
Norwegian University of Science and Technology (NTNU)
N-7491 Trondheim-NTNU
Csaba.Veres@idi.ntnu.no

Abstract. Data modeling for Web Applications needs to be guided not only by the specific requirements of a particular application, but also by the goal of maximizing interoperability between systems. This necessitates the adoption of widely accepted design methods and a set of rich, theoretically motivated principles for organizing data in ontologies. This paper presents one set of such principles. It is based on the observation that current ontologies emphasize the abstraction mechanism of generalization but ignore the various forms of aggregation. We explore possible techniques for modeling aggregation with OWL, investigate the semantics of aggregation, and consider the merits of aggregation over generalization for modeling knowledge in particular situations.

1 Introduction

Aggregation is variously defined as[1]

- The act of aggregating, or the state of being aggregated; collection into a mass or sum; a collection of particulars; an aggregate. (Webster's Revised Unabridged Dictionary)
- several things grouped together or considered as a whole (WordNet 2.0)
- <programming> A composition technique for building a new object from one or more existing objects that support some or all of the new object's required interfaces. (The Free On-line Dictionary of Computing)

These definitions show that some form of *aggregation* is useful when thinking about conceptual entities that are somehow constituted from smaller, self contained entities.

In data modeling, *aggregation* and *generalization* are two major forms of *abstraction* that can help organize data [11]. Similarly, object oriented analysis and design as implemented in the Unified Modeling Language (UML)[2] includes these abstraction mechanisms. However in spite of its widespread use, *aggregation* appears to be a troublesome concept. In data modeling, the term is used in several distinct senses. While [10] considers an attribute which includes a numeric quantity and the unit of measure (e.g. "12 cases") a form of *aggregation*,

[1] All definitions from http://www.dict.org
[2] http://www.uml.org

D. Lowe and M. Gaedke (Eds.): ICWE 2005, LNCS 3579, pp. 285–295, 2005.

[9] defines an *aggregate object* as a "named collection of other objects and their connections" which is useful for describing the model, but is not itself implemented. In UML the semantics of *aggregation* has not been made clear. Martin Fowler laments that "... the difficult thing is considering what the difference is between aggregation and association" [3]. Aggregation is sometimes treated as *part-of*, as in "a wheel is part of a car". Indeed this *part-whole aggregation* seems to be the sense that is often illustrated in connection with UML *aggregation*. However, the *part-of* relation is only one of many possible aggregation mechanisms, and is itself problematical partly because of its diverse semantics [2]. Since the representation of part-whole hierarchies is of primary importance in medical informatics, its use in ontologies has been widely investigated [8], and will not be further considered here.

The literature on *semantic data modeling* also considers the role of various sorts of aggregates and composites in data modeling. A primary aim of semantic data modeling is to model objects and the relationships between them directly [1], for the purpose of constructing databases whose organisation mirrors naturally occurring structures, under the assumption that this will make it easier to represent the primitives of a problem domain than if they first had to be translated into some sort of artificial specification constructs [4]. In defining the Semantic Data Model (SDM), [4] specifies a number of possible groupings:

1. grouping class - based on shared attribute (e.g. SHIP_TYPES contains groups of similar types of SHIP)
2. enumerated grouping class - useful when we wish to group similar types, but there is no clear attribute to define the type. This is a class whose members are other classes. (e.g. TYPES_OF_HAZARDOUS_SHIPS)
3. user-controllable grouping class - simple *aggregates* of arbitrary elements in a class. This is a class whose members are themselves members of a different class. Once again the member elements are of the same type (e.g. CONVOY)

Along similar lines [6] introduces *composition* and *collection* as methods for aggregation in the Format Model. The currently relevant constructor is *collection* which " ... allows one to specify the formation of the sets of objects, all of a given type" [6] (p. 522). For example a CONVOY is a set of SHIPS and STAFF is a collection of STAFF-MEMBER, which is in turn a *classification* (a sort of generalization) of FACULTY-MEMBER and SUPPORT-EMPLOYEE.

Given the somewhat conflicting views of aggregates, there is sometimes a confusion in modeling about whether or not aggregation or generalization is a more useful way to model a particular relationship. For example [13] considers the "EAGLE/SPECIES problem" [14] in which it seems that an entity (EAGLE) has to be represented as both an individual (instance of SPECIES) and a class (set of all EAGLES), in order to accommodate the necessary domain facts. But [13] argues that replacing some of the generalizations with an aggregation helps solve the problem. Briefly, the main problem with the model was that EAGLE was modeled as a subclass of SPECIES. But considering the Linnaean taxonomy suggests that the relationship should be modeled as a form of aggregation in which

the class SPECIES (more correctly, FAMILY ACCIPITRIDAE) is not a supertype, but an aggregate of the classes EAGLE, HAWK, etc. Replacing the *generalization* with *aggregation* is part of the solution to the problem.

Turning now to a novel but we think natural source of information about semantics, the linguist Anna Wierzbicka claims that concepts involving collections are even more common than the ones we have considered thus far [15]. In fact, many of the concepts that might be commonly modeled as types are better describes as collections. For example, "a gun is a subtype of weapon", or "knife is a subtype of cutlery" are erroneous modeling decisions because the categories WEAPON and CUTLERY are not taxonomic. That is, the categories do not consist of entities of a single type. Instead, they aggregate entities of different types into one supercategory, with the aggregation based on one of several possible criteria. These aggregation classes relate to particular human needs, which tend not to be about the typology of things, but about their utility and origin. Thus WEAPON is a concept that aggregates other concepts like GUN or KNIFE or sometimes even FEATHER-PILLOW on the basis of their possible functions. Clearly, classifying FEATHER-PILLOW a as a subtype of WEAPON would be odd. [12] presents theoretical arguments for the usefulness of the various class types in constructing ontologies, which avoids odd classifications such as the preceding example. The different categories of such aggregates are, briefly:

- **purely functional** (e.g. WEAPON). Artifacts are often made to fulfill specific roles, so it is easy to think of a GUN as a *kind of weapon*. But really it is an artifact that can be used as a WEAPON. These categories are fuzzy and to some extent open, such that almost anything *could* be a weapon but nothing definitely is. Is a KNIFE a kind of WEAPON? Is it a WEAPON as much as a GUN is? Is a ROCK a kind of WEAPON? Is a FEATHER PILLOW a kind of WEAPON?
- **functional + origin** (Why is it there? "What for?" and "Where from?"). As an example, the term VEGETABLE means, roughly: "a thing of any kind that people cause to grow out of the ground for people to cook for food" [15], (p. 323). Similarly, MEDICINES have a function to cure disease, but must also be manufactured by people. This class classifies heterogeneous entities by their function, origin, and sometimes mode of use. But they are not collocations because the entities are not 'used together'.
  The terms for these concepts have an interesting syntax in English: they appear to be count nouns but their interpretation is not that of typical count nouns. If I say "I had two vegetables for dinner", I am likely to mean two different sorts of vegetable (e.g. carrot and broccoli) rather than two pieces of the same vegetable (e.g. two carrots). Compare this to "I have two birds in my cage", which could easily refer to two parrots.
- **collocations**. These have to be used/placed together in some sense.
  - **functional** (e.g. FURNITURE, TABLEWARE, NIGHTWEAR). These are defined by function but in addition, they have to be collected in a place (and/or time). That they differ from purely functional categories is demonstrated by the observation that a table that never made it out

of the manufacturer's warehouse is not FURNITURE, but a home made explosive device kept in the basement can still be considered a WEAPON.

- **reason** (why is it together?) (e.g. GARBAGE, CONTENTS, DISHES (as in "dirty dishes" after a meal)). These are collections that require a unity of place, but without reference to function. LEFTOVERS form a collection because they came from the same place/source: they have the same immediate origin. The CONTENTS of a bag have all been placed together by someone, for some reason.

While there appear to be some similarities between these linguistically hypothesized categories and those proposed for semantic data models, there are three important points of difference. First, the linguistic categories specify a set of conditions under which natural collections exist, which potentially limits the possible sorts of, for example, enumerated- and user controllable groupings in [4]. Secondly, Wierzbicka's non taxonomic supercategories are composed of heterogeneous types, whereas the semantic data model collections are of like types. For example the functional category WEAPON consists of many diverse types including HANDGUN, INTERCONTINENTAL-BALLISTIC-MISSILE and, sometimes, FEATHER-PILLOW, whereas a CONVOY consists entirely in SHIPS. But possibly the most important feature of these categories is that they correlate with grammatical properties such as singular/plural, presence of determiners and numerals, and so on. That is, it should be possible to determine if a category descriptor belongs to one of these categories based on grammatical cues alone. This is not only of considerable theoretical interest, but also enhances the process of automatic ontology generation. In summary these linguistically motivated categories are nontaxonomic supercategories that complement taxonomic supercategories and previously identified categories that are aggregations of like types, and can be automatically discovered on the basis of their grammatical status.

The theoretical basis for the usefulness of these categories for constructing ontologies appears in [12]. In the present paper, we explore different ways of capturing their semantics in OWL. The suggestions are in the spirit of the Semantic Web Best Practices and Deployment Working Group[3] in providing engineering guidelines for OWL developers, rather than a rigorous formal semantics. However, we will see that the subtle semantics inherent in these categories will require a novel way to represent meaning.

2 Implementation

There are two immediately obvious approaches to implement these collections in OWL, both involving their own drawbacks. Both approaches involve using *enumerated* classes, where the class is defined simply by enumerating its members, and implemented with *rdf:parseType="Collection"*. In what follows, we investigate the relative merits of each proposal. We will also see that a novel technique is required for expressing all four distinct semantic relationships with the same formal representational device.

[3] http://www.w3.org/2001/sw/BestPractices/

2.1 Sets of Sets (SOS Solution)

One possibility is that the different sorts of heterogeneous collections be treated as set of sets. Thus, for example, VEGETABLE is considered as a set that acts as a container for a collection other sets. The members of the set are other sets like CARROT, TOMATO, and so on. Consider the difference between the subclass view of VEGETABLE as (1) a set which has subsets CARROT and TOMATO versus (2) VEGETABLE as a set which has as elements the sets CARROT and TOMATO:

1. $S_{vegetable}$ == {$carrot_1$, $carrot_2$, $tomato_1$, $tomato_2$}
   (a) S_{carrot} == {$carrot_1$, $carrot_2$}
   (b) S_{tomato} == {$tomato_1$, $tomato_2$}
2. $S_{vegetable}$ == {S_{carrot}, S_{tomato}}

The subset relationship in 1. expresses the semantics of the subtype interpretation of *is-a*. $Carrot_1$ is an element of the set *carrot* and it is also a member of the set *vegetable*. It is a subtype because it can be picked out with other "like" elements of *vegetable* to form a subset. But once again it is literally true that a carrot (e.g. $carrot_1$) is a *vegetable* because it is also an element of *vegetable*. Conversely, any defining property that is true of the members of *vegetable* must also be true of the members of *carrot*.

But the suggested representation in 2. is quite different. Here, the elements of the set *vegetable* are other sets. The set *vegetable* is an abstract entity that has other sets as members, not the elements of those sets. All we can say about $carrot_1$ is that it is an element of *carrot*, and not that it is an element of *vegetable*. Since an instance of the set *carrot* is not also an instance of the set *vegetable*, it follows that "two carrots" are not "two vegetables", as the linguistic intuition requires.

This solution implements the set of sets idea by defining the class *vegetable* as consisting of an enumerated collection of the sets comprising the class, as shown below.

```
<owl:Class rdf:ID="Vegetable">
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Carrot"/>
    <owl:Thing rdf:about="#Tomato"/>
  </owl:oneOf>
</owl:Class>
```

This solution retains important components of the meaning. For example 'two vegetables' can be interpreted as referring to two elements of the set *vegetable*, which in this example are the sets *carrot* and *tomato*. So if I had 'two vegetables' for dinner then I would have had two 'types' of *vegetable* as required. There are two drawbacks of this solution. The first is that it can only be implemented in OWL-Full because it requires classes to be treated as individuals. In this case inference becomes undecidable [5], so that automatic classification and consistency checking becomes impossible. Second, properties cannot be inherited from

the collection class to its members, which are individuals. In OWL, property restrictions are only inherited by subclasses. We will explore this problem in more depth after we consider the second possible implementation.

2.2 Union of Sets (UOS Solution)

The following solution has the advantage that it is compatible with OWL-DL. Once again we use a collection class, but this time it is defined as the union of the component sets:

```
<owl:Class rdf:ID="Vegetable">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Carrot"/>
    <owl:Class rdf:about="#Tomato"/>
  </owl:unionOf>
</owl:Class>
```

This class axiom states that the class extension of *Vegetable* exactly corresponds to the union of the class extensions of *Carrot* and *Tomato*. This results in a set that is identical to definition 1., but it specifically disables the inheritance of properties that is made available through the *rdfs:subClassOf* property. The definition also loses the desirable property of the previous solution that the types of vegetable can be treated as pseudo-substances. In the present definition "two vegetables" could refer to two individual carrots. In this sense the definition captures the semantics of collections less faithfully than the first proposal.

2.3 Unique Features

We have previously noted that one potential problem is that neither of the proposed solutions will result in the contained classes inheriting properties from the container classes. For example, it might be useful to have all VEGETABLES have a property *hasVitamin*. Then CARROT might have vitamin A and C, TOMATO C and D, and so on. If CARROT and TOMATO were (standardly) defined as subclasses of VEGETABLE and *hasVitamin* had VEGETABLE as its domain, then all subclasses of VEGETABLE would inherit the property. If the range were further restricted in each subclass to the union of the appropriate (member of the disjoint classes of) VITAMIN, this would serve as a constraint in assigning the vitamin contents of each instance of CARROT, for example. But this is not possible if the subclass relationship is not explicitly defined. To overcome this limitation we introduce a unique solution by introducing some further concepts in the ontology. We will demonstrate this with the SOS approach, since many of the steps are identical for UOS.

The first point is that, since CARROT and so on are no longer modeled as subclasses of VEGETABLE, we need to decide if they are subclasses of something else. One reference source that can help identify the superclasses (*hypernyms*) of

Sense 2
carrot, cultivated carrot, Daucus carota sativa
=> **plant, flora, plant life** – (a living organism lacking the power of locomotion)
=> **entity** – (that which is perceived or known or inferred to have its own distinct existence (living or nonliving))
Sense 3
carrot – (orange root; important source of carotene)
=> **root vegetable** – (any of various fleshy edible underground roots or tubers)
=> **food** – (any solid substance (as opposed to liquid) that is used as a source of nourishment; "food and drink")
=> **entity** – (that which is perceived or known or inferred to have its own distinct existence (living or nonliving))

Fig. 1. Relevant WordNet senses for carrot, showing a selection of their hypernyms

words in everyday English is WordNet[4] which classifies nouns according to the *is-a-kind-of* relation [7]. There is some evidence that WordNet classifications are useful in constructing ontologies (e.g. [13]). Figure 1 shows selected *hypernyms* for *carrot*.

The first point of note is that there are two distinct relevant senses of *carrot*, only one of them being the vegetable sense we are discussing. The second sense classifies *carrot* as a type of *plant* (among other things). Since we are suggesting that VEGETABLE is not a taxonomic concept we take only the hyponymy in sense 2 into consideration. Thus CARROT is modeled as a subclass of PLANT. In order to represent that VEGETABLE is an aggregate concept, we create a separate hierarchy of concepts to represent the four possible kinds of non taxonomic aggregations that the linguistic descriptor *is-a* can indicate. Figure 2. is a screen shot of the Protege ontology editor displaying these classes[5].

Interestingly *aggregation* (*group, grouping*) appears in WordNet as one of the unique beginners in the noun hierarchy. It is of considerable theoretical interest that WordNet independently assigns such priority to this concept, strengthening arguments both for the utility of WordNet in structuring domain ontologies, and for the present use of aggregation in modeling real world domains.

But now we have a way to tackle the problem of property inheritance. In figure 2 we show a property *hasVitamin* whose domain is defined as VEGETABLE. We can then create an instance of VEGETABLE, say *carrot1*, which gives the desired result of instantiating the *hasVitamin* property. In OWL it is possible to define individuals with multiple types, so we add the type definition for CARROT to *carrot1* so that the individual can instantiate the properties of CARROTS as well. The net effect is that *carrot1* now has all properties from PLANT and VEGETABLE.

[4] http://wordnet.princeton.edu
[5] Notice that VEGETABLE is a subclass of FUNCTION_ORIGIN. This is not strictly speaking correct. A cleaner approach might be to use the metaclass facility. However, this is not possible with OWL-DL

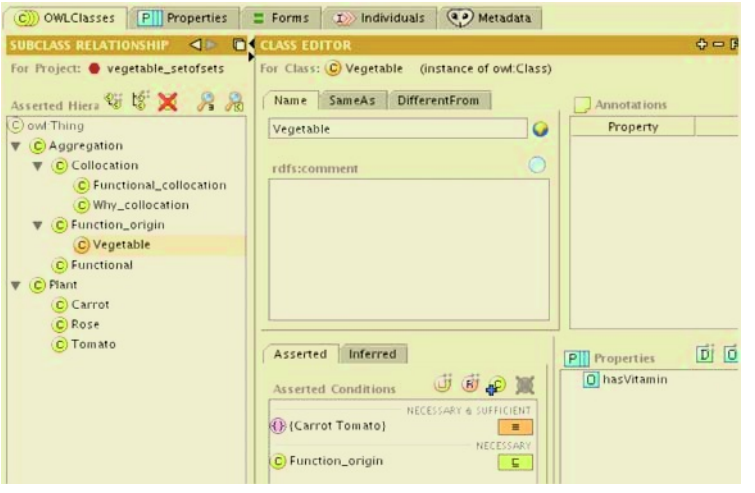

Fig. 2. Protege showing definition of collection classes

Unfortunately this solution as implemented with SOS introduces a severe problem. In the definition of VEGETABLE we stipulate that its only instances are the classes CARROT and TOMATO ("asserted conditions" in figure 2). But this will prevent the creation of an ordinary instance VEGETABLE:*carrot1*, unless it was inferred that *carrot1* was the same as either one of the defined members CARROT or TOMATO.

Let us now turn to the UOS solution. It is obvious that the only difference is that we define VEGETABLE as:

$$Vegetable \equiv Broccoli \sqcup Carrot \sqcup Lettuce \sqcup Spinach \sqcup Tomato$$

One immediate negative outcome is that some of the desirable semantics is lost: now, "two carrots" are also "two vegetables". But the opposite result that instances of VEGETABLE are also instances of CARROT is exactly what is required for solving the property inheritance problem as suggested above. The solution will therefore work in UOS. An individual *carrot1* can be defined with multiple types CARROT and VEGETABLE. Alternatively two differently named individual instances of the two classes can be equated with *owl:sameAs* as shown below.

```
<Vegetable rdf:ID="carrot">
  <owl:sameAs>
    <Carrot rdf:ID="carrot1">
      <owl:sameAs rdf:resource="#carrot"/>
    </Carrot>
  </owl:sameAs>
</Vegetable>
```

Either method will allow inheritance of properties from both classes. The other consequence of the change in definition is that the taxonomy is OWL - DL,

allowing us to take advantage of available description logic reasoners. Classifying the taxonomy with Racer then reveals the result shown in figure 3. The inferred hierarchy is identical to an ontology that might have been built with more "traditional" approaches in which the subclass relationships were asserted. Retaining the originally asserted hierarchy as the foundational ontology maintains the benefit of explicit semantics while computing subsumption relations is beneficial for downward compatibility with ontologies in which the subclass relations are asserted.

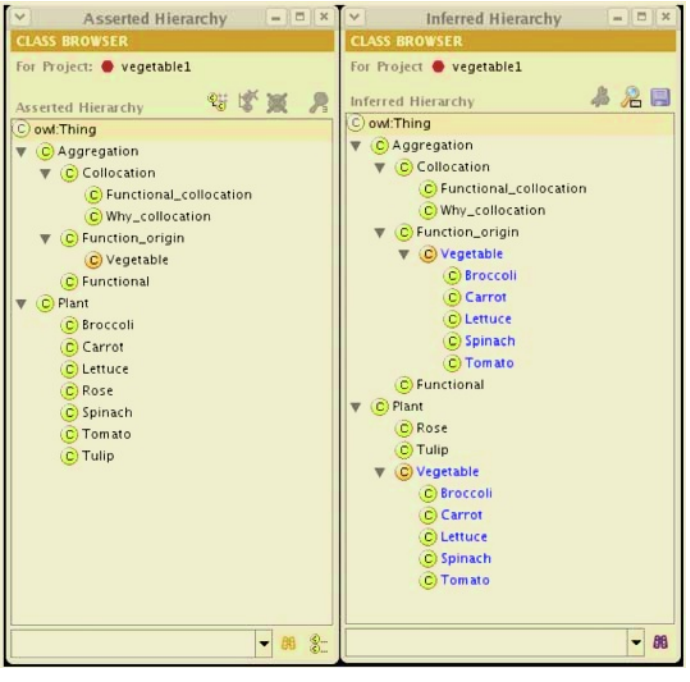

Fig. 3. Asserted and inferred hierarchies for the UOS solution

TO summarize thus far, it seems that SOS will not work, but UOS will work with one possible exception. But an important novel feature of the solutions is that the semantics of the categories is explicitly represented because the nature of the aggregation can be "read off" the ontology. In this example the reason for the category can be defined in that it represents a collection of entities that serve the same purpose or function (to provide a certain kind of nutrition) and have the same origin (growing in the ground). The specific function and origin are therefore facts that can be represented in the ontology. Perhaps the usefulness of this is less obvious for VEGETABLE than a concept like WEAPON. Here, division into subclasses would appear odd if the subclasses of WEAPON included GUN, KNIFE, ICE-PICK, FEATHER PILLOW, and DECK CHAIR. Collecting them in an aggregation whose intent was defined would clear up the mystery.

As a result of creating a separate hierarchy for aggregations, a third possible implementation solution presents itself. Called this the Instance Solution (IS), it involves modeling VEGETABLE as an instance of FUNCTION_ORIGIN, rather than as a subclass. The motivation for this solution is that it overcomes a possibly awkward feature of the previous solutions, that the subclass structure of, say, vegetable, is defined through the collection feature of OWL. If a new kind of vegetable were to be added, VEGETABLE would be redefined. It is necessary under this proposal to create two more properties *hasMember* (domain: AGGREGATION, range: PLANT in the current example) and its inverse *memberOf* to describe the relationship between the various subclasses of PLANT and the instances of the AGGREGATION classes. These properties can be used to infer the instances (and therefore the classes) that are associated with the different kinds of aggregation.

Unfortunately there are drawbacks to this solution also. First, there is now no definition of the aggregate classes in the ontology. That is, there is no reason why TULIP, GUN, or anything else could not be made a *memberOf* VEGETABLE. Second, direct inheritance of properties from VEGETABLE is not possible because VEGETABLE is now an instance. One possible workaround for this is to define datatype properties for the aggregate classes that can be used to describe the purpose of their instances. In our implementation we have three such properties (*function*, *origin*, and *purpose*) which can be used in various combinations to describe each instance of the subclasses of AGGREGATION. For example VEGETABLE has *function:eating, origin: grown_in_ground*. This can be used by an application to infer additional properties for an instance of CARROT through its *memberOf:vegetable* property.

3 Conclusion

We have suggested that aggregation is a useful alternative to generalization for some concepts. We investigated three approaches to modeling aggregation in OWL-DL. Each has their advantages and drawbacks. The second approach, UOS, seems most useful, but the last approach, IS, also has some merit. We have also noted that none of the approaches can fully capture the intended semantics, and suggested some possible ways in which a more complete semantics can be captured in more expressive logics. However, the research agenda involves an evaluation of the usefulness of these slightly imperfect implementations in developing workable ontologies in a tractable logic like OWL-DL.

Acknowledgements

This work was partly sponsored by the Norwegian Research Council through the WISEMOD project, number 160126V30 in the IKT-2010 program.

References

1. Abiteboul, S. and Hull, R. IFO: A Formal Semantic Database Model. ACM Transactions on Database Systems, Vol. 12, No. 4, December 1987, Pages 525-565. (1987).

2. Artale, A., Franconi, E., Guarino, N., and Pazzi, L. Part-whole relations in object-centered systems: An overview. Data & Knowledge Engineering, 20(3):347-383. (1996).
3. Fowler, M. and Scott, K. UML Distilled: Applying the Standard Object Modeling Language. The Addison-Wesley Object Technology Series (1997).
4. Hammer, M. and McLeod, D. Database Description with SDM: A Semantic Database Model. ACM transactions on Database Systems, Vol. 6, No. 3, September 1981, Pages 351-386. (1981).
5. Horrocks, I., Patel-Schneider, P. and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language". Journal of Web Semantics, 1(1):7–26, (2003).
6. Hull, R. and Yap, C. K. The Format Model: A Theory of Database Organization. Journal of the Association for Computing Machinery, Vol 31, No 3, pp. 518-537. (1984).
7. Miller, G. Nouns in WordNet. In C. Fellbaum (Ed.) WordNet: An Electronic Lexical Database. MIT Press, Cambridge, MA. (1998).
8. Rector, A. Medical Informatics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider (Eds.) The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press (2003).
9. Ross, R. G. Entity Modeling: Techniques and Application. Database Research Group, Boston, Massachusetts, (1987).
10. Simsion, G. C. Data Modeling Essentials: Analysis, Design, and Innovation, 2nd edition. Coriolis, Scottsdale, Arizona, (2001).
11. Smith, J. M. and Smith, D. C. P. Database Abstrctions: Aggregation and Generalisation. ACM Transactions on Database Systems, Vol. 2, No. 2, (1977).
12. Verec, C., Ontology and Taxonomy: why "is-a" still isn't just "is-a". In Proceedings of The 2005 International Conference on e-Business, Enterprise Information Systems, e-Government, and Outsourcing, EEE'05 Las Vegas, Nevada, (2005).
13. Veres, C. Refining our intuitions with WordNet: a tool for building principled ontologies. In Proceedings of ISoneWORLD, Las Vegas, Nevada, (2004).
14. Welty, C., and Ferrucci, D. What's in an Instance? RPI Computer Science Technical Report. 1994.
15. Wierzbicka, A. Apples are not a 'kind of fruit': the semantics of human categorization. American Ethnologist, 313–328 (1984).