# Applying Ontology Design Patterns in Bio-ontologies

Mikel Egaña[1], Alan Rector[1], Robert Stevens[1], and Erick Antezana[2,3]

[1] School of Computer Science, University of Manchester, UK
[2] Department of Plant Systems Biology, VIB, Gent, Belgium
[3] Department of Molecular Genetics, Gent University, Belgium
{eganaarm,rector,robert.stevens}@manchester.ac.uk,
erant@psb.ugent.be

**Abstract.** Biological knowledge has been, to date, coded by biologists in axiomatically lean bio-ontologies. To facilitate axiomatic enrichment, complex semantics can be encapsulated as Ontology Design Patterns (ODPs). These can be applied across an ontology to make the domain knowledge explicit and therefore available for computational inference. The same ODP is often required in many different parts of the same ontology and the manual construction of often complex ODP semantics is loaded with the possibility of slips, inconsistencies and other errors. To address this issue we present the Ontology PreProcessor Language (OPPL), an axiom-based language for selecting and transforming portions of OWL ontologies, offering a means for applying ODPs. Example ODPs for the common need to represent "modifiers" of independent entities are presented and one of them is used as a demonstration of how to use OPPL to apply it.

## 1 Introduction

Many bio-ontologies have been created to represent biological knowledge [1]. Biology is an interesting test-bed for knowledge management, due to the volatility, breadth and complexity of the knowledge that needs to be represented in bio-ontologies. Such representation is usually undertaken by biologists, which has both advantages and disadvantages. It is positive because biologists as domain experts are the ones who perceive the subtleties of the knowledge that, if well represented, can make a difference in the usefulness of the ontology being built. A negative aspect is that biologists often lack training with Knowledge Representation (KR) languages with strict semantics and, therefore, do not use many of the features of those languages. As a result, there are difficulties with maintenance and computational use in many bio-ontologies that could be helped by richer axiomatic content. With increasing demands for re-use and the increasing scale of ontologies, these problems are becoming more severe.

Ontology Design Patterns (ODPs) are one solution that can help address the problems mentioned above. ODPs encapsulate in a single named representation the semantics that require several statements in low level ontology languages. ODPs instantiate high-level metamodels (such as the *logical ODPs* described in

© Springer-Verlag Berlin Heidelberg 2008

[2]) in concrete languages such as OWL: Therefore ODPs are equivalent to the notion of *content ODPs* described in [2].

Providing predefined ODPs can help biologists overcome the difficulty of using a logic-based language, that most biologists (and other end-users) often find difficult and counter-intuitive. ODPs also provide a vocabulary for discussing alternative representations for similar notions. An example of such ODPs are those that enable the "modification" of independent entities. We describe three such ODPs in Section 2; their pros and cons; and the features each supports.

Having selected a suitable ODP for a representation requirement, a further issue is the application of that ODP. ODPs often encapsulate complex semantics and are repeatedly applied across an ontology. Such activities, when carried out by humans, are often error prone. To ease the application of ODPs, as well as the general application of axioms during enrichment, we present the Ontology PreProcessor Language (OPPL) [3], a high-level macro language for adding axioms to an ontology, in Section 3. The rapid and consistent application of transformations to an ontology can ease the "experimentation" of using different ODPs, by allowing alternative modelling options to be applied rapidly and tested. Once a final choice is made, OPPL scripts can be re-applied and/or modified as necessary.

## 2 Ontology Design Patterns for Modelling Biological Knowledge

Although ODPs have already been explored as a KR technique [4], they have not been widely used in bio-ontologies, except in a few cases such as the development and axiomatic enrichment of the Cell Cycle Ontology (CCO) [5]. The applicability of ODPs is, however, much wider and should be a significant component of the migration of axiomatically lean bio-ontologies to ones that are axiomatically rich. In this section we briefly present some ODPs to show their benefits in modelling biological knowledge.

ODPs are presented in OWL as instantiations of more abstract models, and therefore they are simply OWL fragments, but they exemplify more general structures. The OWL to UML mapping used for representing ODPs [6] is shown in Figure 1.

The three ODPs presented tackle the same problem: how to represent "modifiers" and "values". Modifiers and values are a subset of the constructs that refine the status of independent entities using dependent entities, variously called "qualities", "attributes", "features", etc. by different authors. The terms "modifier" and "value" are used in this paper as being neutral amongst various proposed upper ontologies and familiar to our users.

There are three mechanisms advocated for representing modifiers by different authors. In BFO [7], the authors advocate the use of what we here call the Entity-Quality ODP; In DOLCE [8], the authors advocate the use of what corresponds to what we here call the Entity-Feature-Value ODP, although they use the word "Quality" for what we here call "Feature" (we have used two different terms, "Quality" and "Feature", to avoid confusion between the two). Finally,

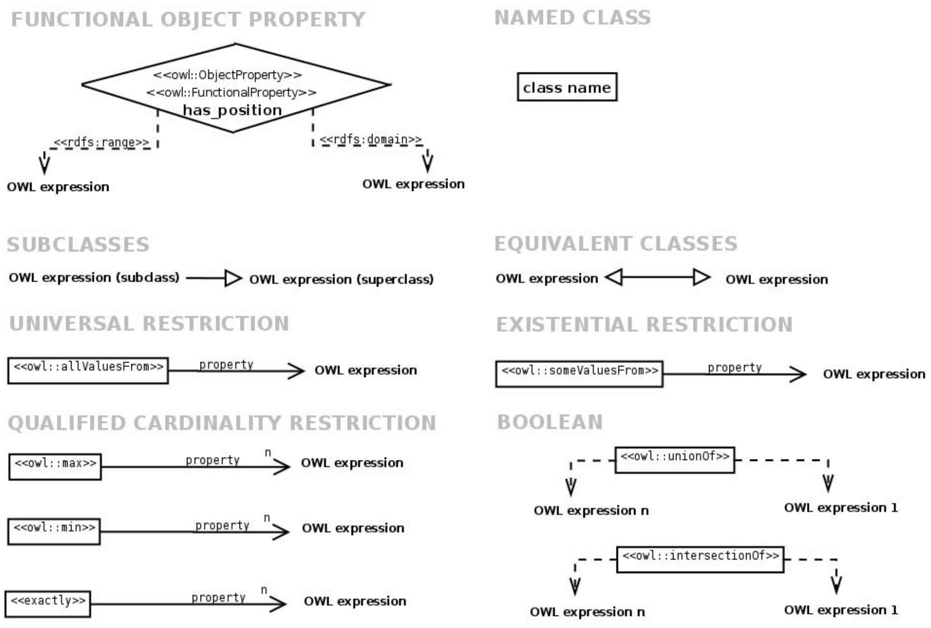

**Fig. 1.** Partial OWL to UML mapping for representing ODPs. Names of some OWL constructs are presented in grey, and under them, their representation in UML, in bold. Named entities are represented by boxes (classes) and diamonds (properties); an OWL expression can have any complexity

there is the "naive" Entity-Property-Quality ODP, which is the approach that corresponds most closely to what users often expect.

The position of structures in a cellular context is used as a running example for each of the three ODPs. Certain cellular components have a position within a cell as part of the overall processes in which they participate. Thus, "position" is the modifier and the "cellular component" is the independent entity modified. The possible "values" for the "modifier" are "apical" and "basal". This modifier applies only to "cell parts", *e.g.* mitochondria; the "values" are mutually exclusive and only one may apply to any one "cell component". This scenario is a real problem faced, for example, by the Gene Ontology (GO) [9], where we can find terms like `basal labyrinth` (GO:0033774) without any further axiomisation in terms of position (only `is_a` and `part_of` relationships).

The requirements are: (1) to represent which modifiers apply to which independent entities and *vice versa*; (2) to represent which values apply to each modifier (possibly according to which independent entity it applies—*e.g.*, "position" does not apply to "cytoplasm"); (3) the mutual constraints amongst the values—whether there can be only one or more than one values and whether the values are mutually exclusive (the usual case); and (4) whether the modifiers apply to all or only some of a given category of independent entity—*e.g.* position applies in principle to any cell part but it is only relevant for mitochondria in relation to stomach epithelial cells where the distinction is vital for their biological function. A different ODP may be chosen in each implementation depending on the requirements of the user or the system; therefore there is no one "best" ODP.

## 2.1 Entity-Quality ODP

The application of this ODP is shown in Figure 2. The use of "position" corresponds roughly to the use of "quality" in BFO. The entities are linked to the qualities by Qualified Cardinality Restrictions (QCRs) (`max 1` if the quality is

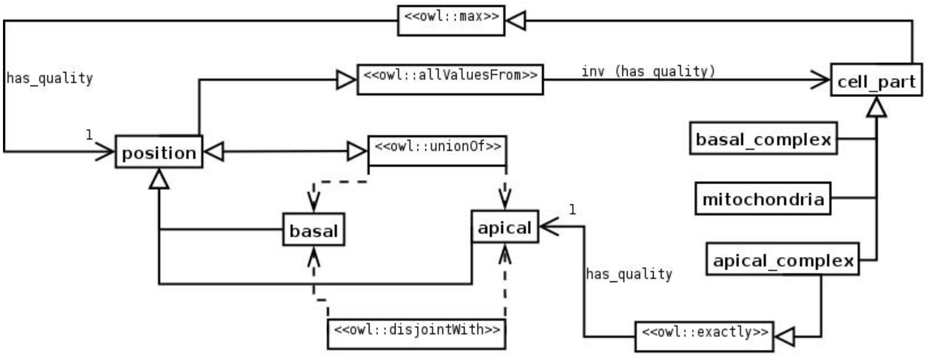

**Fig. 2.** Structure of the application of the Entity-Quality ODP

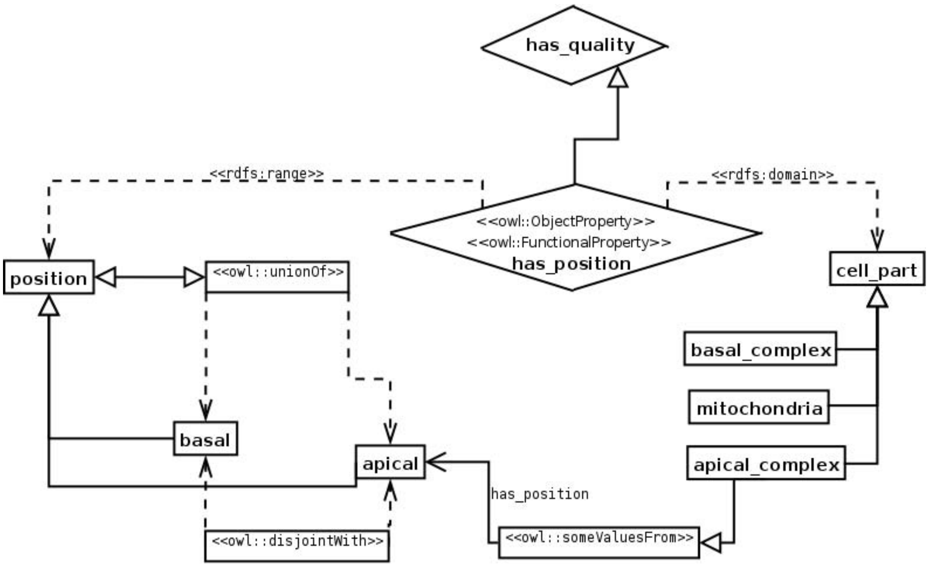

**Fig. 3.** Structure of the application of the Entity-Property-Quality ODP

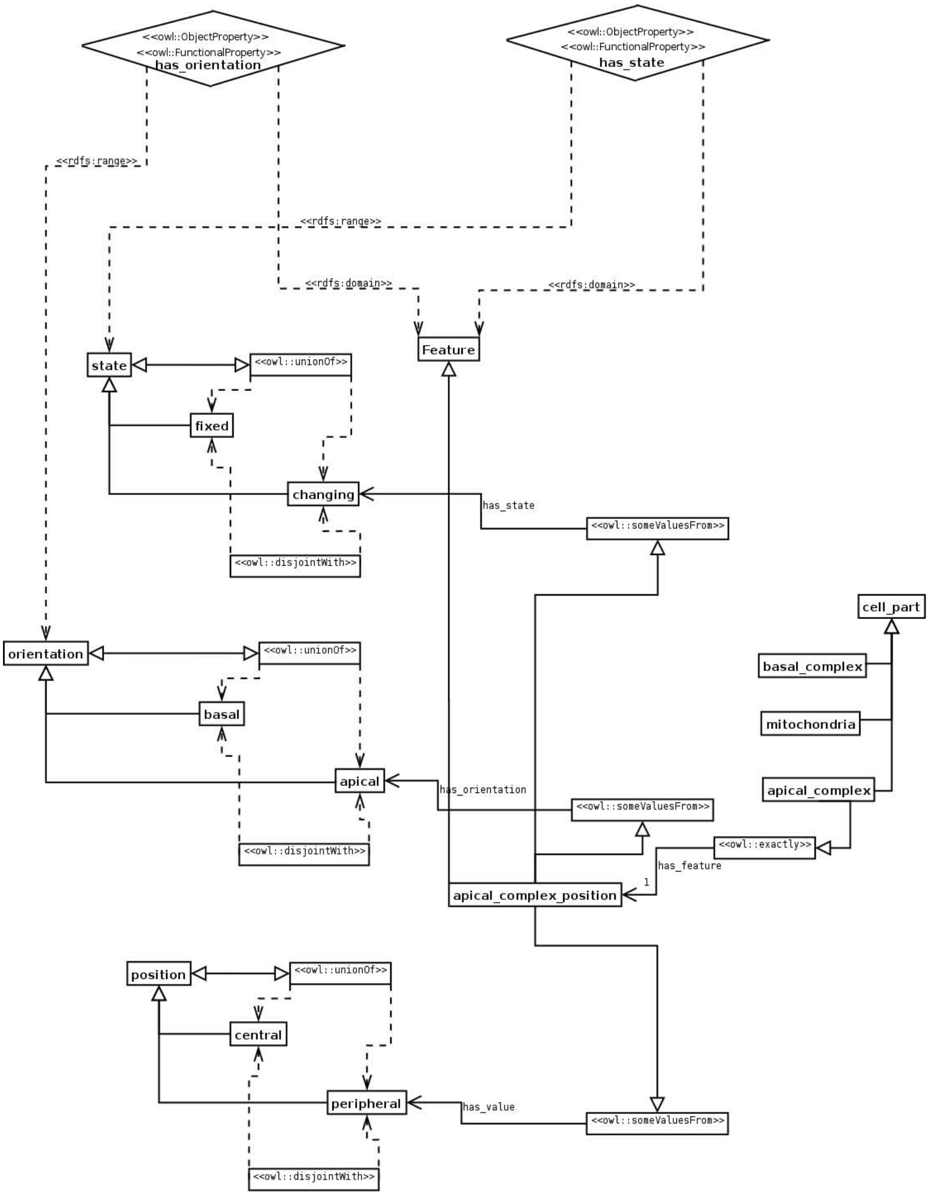

**Fig. 4.** Structure of the application of the Entity-Feature-Value ODP

accidental or `exactly 1` if the quality is intrinsic). This ODP, compared with the following two, offers simplicity in authoring as only one general object property is needed (`has_quality`). The disadvantages of this ODP are: it cannot handle multi-aspect qualities (such as colour's saturation and intensity aspects);

it may be more difficult to use in software; it requires the maximum cardinality to be specified (a step that users often omit).

### 2.2 Entity-Property-Quality ODP

Qualities can also be modelled using the Entity-Property-Quality ODP (Figure 3), in which (optionally functional) object properties are used to represent different types of qualities: the domain and range are the entity and the quality values, respectively, to limit the modifiers to the required entities. The entities are linked to quality values by simple existential restrictions. Therefore this ODP results in a proliferation of object properties (which is more difficult to maintain than the class hierarchy), but overall it is easier to author as it is closer to user intuition. Again, this ODP can not handle multi-aspect qualities.

The difference between the Entity-Quality ODP and the Entity-Property-Quality ODP lies mainly in two points: (1) how to limit the entities being modified (Entity-Quality ODP uses the universal restriction and the `max 1` QCR, whereas the Entity-Property-Quality ODP uses domain and ranges), (2) how to limit cardinality (Entity-Quality ODP uses the QCR `exactly 1` and Entity-Property-Quality ODP uses the fact that the object property is functional).

### 2.3 Entity-Feature-Value ODP

This is the most general ODP for representing modifiers (Figure 4). This ODP is the only one that allows for modifiers with multiple aspects (*e.g.* a position with an orientation and a state), which is its main advantage over the other two ODPs. Entities are linked to features with QCRs (`exactly 1` for intrinsic or `max 1` for accidental features). The feature is linked to different aspects via existential restrictions. Another advantage is that it requires only a few object properties. It is, however, the ODP that needs most entities and therefore is the most difficult to author. “Feature” is the equivalent of DOLCE’s “Quality”.

## 3 Applying Ontology Design Patterns with the Ontology PreProcessor Language

The Ontology PreProcessor Language[1] (OPPL) is a high-level language for applying ODPs in ontologies. OPPL offers an API-like access to the axioms and annotations of any OWL ontology. The OPPL syntax is a modified version of the Manchester OWL Syntax [10], with some added keywords such as `ADD`, `SELECT` and `REMOVE`. OPPL is capable of querying an ontology and adding/removing axioms of arbitrary complexity to/from the obtained entities (axioms can be added or removed also without selecting any entity).

The core of the OPPL syntax is the “OPPL instruction” (Figure 5): the OPPL instructions are written in a flat file and the OPPL software interprets them, applying the changes to an ontology and generating a new ontology (comments, starting with `#`, are ignored by the OPPL software).

[1] `http://oppl.sourceforge.net/`

```
SELECT equivalentTo part_of only (mitochondria or chloroplast);
ADD subClassOf has_function some energy_production;
```

**Fig. 5.** OPPL instruction

The instruction from Figure 5, when interpreted by the OPPL software, will query the reasoner to select (`SELECT` keyword) any class that is equivalent to the anonymous class `part_of only (mitochondria or chloroplast)` and will add (`ADD` keyword) the axiom `has_function some energy_production` to it as a necessary condition. Named entities (classes, individuals or object properties) can also be added or removed, not only selected, and many axioms can be added/removed to/from the same entity.

OPPL, compared to the macros implementation described in [11], works at a much more abstract level (axioms instead of RDF/XML), is able to exploit reasoning, is able to query the model, and is able to remove axioms (not only add). In comparison with SPARQL[2] and SPARQL DL [12], OPPL offers the possibility of adding and removing axioms (not only querying). In terms of querying, OPPL allows for a greater expressivity than SPARQL DL, at the price of not allowing variables (i.e. the condition that an entity must fulfill to be selected by the reasoner is formed by expressions where only named entities can be found). The impossibility of using variables within query expressions makes OPPL rather "local" to the ontology being modified, as the user must be familiarized with the entities from the ontology.

OPPL is also well suited to the application of ODPs. Figure 6 shows an extract from the OPPL flat file used to apply the Entity-Quality ODP in CCO[3].

A selection criterion is needed to retrieve only the intended entities that form the target of the ODP. OPPL offers the `SELECT` instruction, allowing the definition of a condition to retrieve all the entities that match the condition. Such condition can be stated either via logic axioms or annotations. For the conditions based on logic axioms, a reasoner can be used to retrieve the entities (*e.g.* `equivalentTo part_of only (mitochondria or chloroplast)`) from the inferred model.

A condition based on annotation values (*e.g.* `SELECT label "(basal|apical) (.+?)"`) is defined based on strings: any entity whose annotation matches a regular expression will be selected. Apart from being selected, the content of the matched string will be available, via the < > constructor (*e.g.* `has_position exactly 1 <1>`), to the later instructions, which can resolve that content against the OWL ontology and exploit it for new axioms (in this case, the first group of the label of whatever class matches the regular expression). This annotation processing feature of OPPL is especially useful when dealing with bio-ontologies, since most of them have axioms "buried" in annotation

[2] `http://www.w3.org/TR/rdf-sparql-query/`

[3] The original OPPL file with the ontologies and execution logs can be downloaded from: `http://www.gong.manchester.ac.uk/OPPL_EKAW2008.tar.gz`

```
######### Applying the Entity-Quality ODP in CCO #########

# Quality values

ADD Class: modifier;

ADD ObjectProperty: has_position;

ADD Class: position;ADD subClassOf modifier;REMOVE subClassOf Thing;

ADD Class: apical;ADD subClassOf position;REMOVE subClassOf Thing;

ADD Class: basal;ADD subClassOf position;ADD disjointWith apical;

# constrain the quality values to the entities (CCO_C0001882 = cell part)

SELECT Class: position;ADD equivalentTo apical or basal;
ADD subClassOf inv (has_position) only CCO_C0001882;

# not having a position is legal

SELECT Class: CCO_C0001882;
ADD subClassOf has_position max 1 position;

# In order to apply the ODP in different places of the ontology, we need
# a general condition that will catch different target classes (doing it
# by hand would be tedious, inefficient and would betray the aim of ODPs).
# We will define a regexp "(basal|apical) (.+?)": <1> refers to the first
# group from the string  that matches the regexp

SELECT label "(basal|apical) (.+?)";ADD subClassOf has_position exactly 1 <1>;
```

**Fig. 6.** An extract of an OPPL flat file

values, GO being an example of such a tendency [13]. GO has approximately 20 000 classes and the procedure of executing the file of Figure 6 catches 24 classes to which to apply the Entity-Quality ODP, which saves a lot of time as it would be very inefficient to apply the ODP, one by one, in those 24 classes[4].

OPPL offers a straightforward, flexible and reusable way of "programmatically" interacting with the ontology. OPPL instructions can be re-used in different parts of an ontology, in separate stages of development or by different users. Using OPPL, complex ODPs can be applied or rejected in one go (just by uncommenting or commenting the OPPL instructions); ODPs can be stored for application at any time; ODPs can be shared (sharing the flat files with the OPPL instructions) and the design decisions can be made explicit using the comments.

In a large ontology where an ODP is repeated many times application of that ODP via OPPL avoids tedium and slips and provides consistency. For example OPPL can be used to easily try and compare the different ODPs for modifiers reviewed in this paper.

Although OPPL is an early attempt towards a flexible way of working with ontologies, it has been successfully used within the axiomatic enrichment and

[4] The execution was done using CCO, which incorporates big and representative parts of GO; it could be that the matched classes are even more.

maintenance of CCO: OPPL has been used to apply ODPs such as the Sequence ODP, to make corrections on the ontology, to store and execute OWL queries and to check consistency. As a consequence of applying the Sequence ODP, new queries could be performed against CCO about specific cell-cycle-related events taking into account their sequentiality [5].

## 4 Conclusion

ODPs encapsulate the complex semantics needed for rich modelling in concrete models. Encapsulation in the form of ODPs, on its own, is not, however, enough; the encapsulation must be usable, and hence ODPs need to be easy to apply. Therefore we have developed OPPL, to be able to consistently and efficiently apply ODPs in bio-ontologies. OPPL will be further extended with capability for variables, enabling more subtle transformations of material already encoded in an OWL ontology. Using variables means that the user can work with the "pure" structure of the ontology in an ontology-independent manner. We have also demonstrated how OPPL can be used (and has been used) to apply ODPs in actual bio-ontologies. OPPL, combined with ODPs public repositories[5], composes a basic infrastructure for exploring, choosing and applying ODPs. Using such infrastructure, ODPs offer a route for an enhanced knowledge management in biology.

The use of ODPs in biology can be regarded as a microcosm of the challenges that knowledge management will have to face as it becomes more widespread, especially through the Semantic Web. Although ODPs' usage in bio-ontologies is still limited, they have already brought benefits in terms of axiomatic richness and maintainability [13,5]. We therefore envisage that they will be of similar benefit for the wider Semantic Web.

## Acknowledgements

Mikel Egaña Aranguren is funded by the University of Manchester and EPSRC. Erick Antezana is funded by the European Science Foundation (ESF) for the activity entitled Frontiers of Functional Genomics.

## References

1. Bodenreider, O., Stevens, R.: Bio-ontologies: current trends and future directions. Brief. Bioinformatics 7(3), 256–274 (2006)
2. Pressuti, V., Gangemi, A., David, S., de Cea, G.A., Suárez-Figueroa, M., Montiel-Ponsoda, E., Poveda, M.: A Library of Ontology Design Patterns. NeOn Deliverable 2.5.1 (2008)
3. Egaña, M., Antezana, E., Stevens, R.: Transforming the Axiomisation of Ontologies: The Ontology Pre-Processor Language. In: OWLed. (2008)

[5] `http://ontologydesignpatterns.org`, `http://odps.sf.net/`

4. Gangemi, A.: Ontology Design Patterns for Semantic Web Content. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 262–276. Springer, Heidelberg (2005)
5. Aranguren, M.E., Antezana, E., Kuiper, M., Stevens, R.: Ontology Design Patterns for bio-ontologies: a case study on the Cell Cycle Ontology. BMC bioinformatics 9 (suppl. 5), S1 (2008)
6. Brockmans, S., Volz, R., Eberhart, A., Löffler, P.: Visual Modelling of OWL DL Ontologies using UML. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 198–213. Springer, Heidelberg (2004)
7. Grenon, P., Smith, B., Goldberg, L.: Biodynamic Ontology: Applying BFO in the Biomedical Domain. In: Pisanelli, D.M. (ed.) Ontologies in Medicine, pp. 20–38. IOS Press, Amsterdam (2004)
8. Gangemi, A., Guarino, N., Masolo, C., Oltramari, A., Schneider, L.: Sweetening ontologies with DOLCE. In: Gómez-Pérez, A., Benjamins, V.R. (eds.) EKAW 2002. LNCS(LNAI), vol. 2473, pp. 166–182. Springer, Heidelberg (2002)
9. Gene Ontology Consortium: Gene Ontology: tool for the unification of biology. Nature Genetics 23, 25–29 (2000)
10. Horridge, M., Drummond, N., Goodwin, J., Rector, A., Stevens, R., Wang, H.: The Manchester OWL syntax. In: OWLed. (2006)
11. Vrandecić, D.: Explicit Knowledge Engineering Patterns with Macros. In: Welty, C., Gangemi, A. (eds.) Ontology Patterns for the Semantic Web Workshop (ISWC) (2005)
12. Sirin, E., Parsia, B.: SPARQL-DL: SPARQL Query for OWL-DL. In: OWLED (2007)
13. Aranguren, M.E., Wroe, C., Goble, C., Stevens, R.: In situ migration of handcrafted ontologies to reason-able forms. Data and Knowledge Engineering 66(1), 147–162 (2008)