A pattern-driven approach to biomedical ontology engineering

Jennifer D. Warrender and Phillip Lord

School of Computing Science, Newcastle University, Newcastle-upon-Tyne, UK

Abstract. Developing ontologies can be expensive, time-consuming, as well as difficult to develop and maintain. This is especially true for more expressive and/or larger ontologies. Some ontologies are, however, relatively repetitive, reusing design patterns; building these with both generic and bespoke patterns should reduce duplication and increase regularity which in turn should impact on the cost of development.
Here we report on the usage of patterns applied to two biomedical ontologies: firstly a novel ontology for karyotypes which has been built ground-up using a pattern based approach; and, secondly, our initial refactoring of the SIO ontology to make explicit use of patterns at development time. To enable this, we use the Tawny-OWL library which enables full-programmatic development of ontologies. We show how this approach can generate large numbers of classes from much simpler data structures which is highly beneficial within biomedical ontology engineering.

1 Introduction

Ontologies are used widely in biomedicine for many purposes, including instance classification, schema reconciliation, or as a controlled vocabulary [1]. The range of ontology purposes are reflected in their sizes. Systemized Nomenclature of Medicine (SNOMED)[1] has over 290,000 concepts which presents scalability challenges when querying the ontology. The Gene Ontology (GO)[2] is about 1/10th of this size, at 30,000 terms, while ontologies such as the Dublin Core (DC)[3] or Simple Knowledge Organization System (SKOS)[4] have less than 100 terms each.

In some cases, the formal semantics of ontologies has been used with computational reasoning systems. However, ontologies vary in the use of the expressiveness of their language. Perhaps the most common is to use just subsumption and existential restrictions (effectively the OWL2 EL profile) of which the GO is a well-known example. Whilst SKOS is an example of the OWL Full profile. Other ontologies use most of the constructs within the OWL language [2]. While there have historically been many ontology languages, in the life sciences the two most common are probably OBO format [3] and OWL [4]. Several mappings between these two exist [5].

[1] http://www.ihtsdo.org/snomed-ct/
[2] http://www.geneontology.org/
[3] http://dublincore.org/
[4] http://www.w3.org/TR/skos-primer/

There are a variety of methodologies and approaches that exist for ontology engineering, some of which include the use of patterns. Patterns are templates that encourage best practice; these were originally popularised in the context of software engineering [6], but there has also been considerable research on Ontology Design Patterns (ODP). ODPs are formal, reusable and successful modelling solutions to recurrent modelling problems that are used for creating and maintaining ontologies [7]. Two well-known ODPs are:

- `value partition` [8], a good practice ODP that is used to model attributes of objects that can only have a predefined set of values.
- `sequence` [9], a domain modelling ODP. This ODP is used to model a sequence of events, one after the other.

There are many tools available for ontology construction, one of which is Protégé[5]. Amongst others, Protégé had supported the value partition pattern through the use of graphical wizards[6].

As well as ODPs which address generic concerns cross-cutting a number of domains, a need has also been recognised for patterns within a single ontology. One mechanism for expressing these patterns is OPPL2 – a pre-processing language, also available as a Protégé plugin, which can be used to automate addition or transformation of ontology terms derived by a declarative rule language [10]. A similar idea is found with "Safe Macros", where patterns are expressed as annotation properties within the ontology, which may be expanded to logical axioms with a post-processor [11]. Other tools use patterns to leverage alternative data entry environments, generally spreadsheets. For example, RightField [12] and Populous [13] enable constrained data entry using an Excel spreadsheet, and then use OPPL to expand this data into OWL expressions. Quick term templates [14] similarly uses spreadsheets and the mapping language, M^2 [15].

However, ontology development using GUI based tools is time-consuming due to the necessity of GUI interaction when there are major modelling changes. Therefore, a number of text based/programming environment tools for ontology construction have been developed. Examples include Tawny-OWL [16] (Clojure), Thea-OWL [17] (Prolog) and InfixOWL [18] (Python), all of which can generate OWL. These tools enable the development of ontologies within a programmatic environment. These also provide a mechanism for the use of patterns; repetitive ontology construction tasks can be automated. Next, we give a brief introduction to Tawny-OWL, and introduce the idea of using a programmatic environment for localised pattern development, which are (predominately) useful within a single ontology.

2 Patternising the Pizza Ontology

Tawny-OWL [16] is a library written in the lisp dialect Clojure, wrapping the OWL API [19]. It is designed to be used more as a *textual user interface* for on-

[5] http://protege.stanford.edu/
[6] http://protegewiki.stanford.edu/wiki/Protege_Wizards

tology development, rather than an API for ontology manipulation. Clojure provides an evaluative environment, which means that Tawny-OWL can be used to add (or remove) entities to an ontology incrementally and interactively. Tawny-OWL has complete support for OWL2. including data types. It also provides direct access to reasoners, which combined with unit testing and a versioning system enables continuous integration of ontologies. Tawny-OWL has been designed to be convenient for building simple ontologies without in-depth knowledge of Clojure, however, fully programmatic use of Tawny-OWL will require this knowledge.

The syntax of Tawny-OWL is relatively straight-forward, having been modelled after Manchester Syntax [20]. Consider, for example, this definition (Listing 1) from the Pizza Ontology[7] recast into Tawny-OWL syntax (aka tawny-pizza)[8], which we use here as an exemplar of Tawny-OWL and patterns.

```
(defclass Pizza
   :label "Pizza"
   :subclass (owl-some hasTopping PizzaTopping)
   (owl-some hasBase PizzaBase))
```

Listing 1. A basic class definition.

As Tawny-OWL is built on a full programming language it is also capable of expressing arbitrarily complex patterns. It provides support for two design patterns which are so commonly used that it they are rarely recognised as a pattern, the first being the closure axiom [21]. In Listing 2, we show an expression which returns the two existential, and one universal restrictions necessary to describe the toppings for a Margherita pizza. Covering axioms are also supported.

```
;; A - Usage
(some-only hasTopping TomatoTopping MozzarellaTopping)

;; B - As Manchester Syntax
hasTopping some TomatoTopping
hasTopping some MozzarellaTopping
hasTopping only
   (MozzarellaTopping or TomatoTopping)
```

Listing 2. A closed restriction.

Currently, Tawny-OWL has preliminary support for other general purpose ontology design patterns, in the form of the value partition ODP. The usage of this Tawny-OWL pattern can be seen in Listing 3, which generates four classes, a disjoint axiom and an object property.

```
(value-partition
 Spiciness
 [Mild Medium Hot])
```

Listing 3. Example value partition usage in the Pizza Ontology.

[7] http://owl.cs.manchester.ac.uk/co-ode-files/ontologies/pizza.owl
[8] Available from https://github.com/phillord/tawny-pizza

Closure axioms and the value partition are currently the only generic patterns used in the Pizza ontology. We do use a localised pattern, which we called "named pizza" where a particular pizza is defined by an enumeration of its ingredients (Listing 4). This pattern also makes use of the closure pattern described earlier.

```
(defn generate-named-pizza [& pizzalist]
  (doseq [[named & toppings] pizzalist]
    (owl-class
     named
     :subclass NamedPizza
               (some-only hasTopping toppings))))
```

Listing 4. Example of localised patterning in the Pizza Ontology. In this example, `owl-class`, and `some-only` are parts of OWL, `hasTopping` is an object property, `pizzalist`, `named` and `toppings` are variables, while `defn` and `doseq` are parts of Clojure.

The syntactic concision of this pattern, as shown in Listing 5, is advantageous as a pizza can have many toppings. A secondary advantage is to ensure consistency of all the named pizza definitions, as well as supporting maintainability should we wish to change these definitions. Tawny-pizza is currently an example usage – in real usage, we would probably read this data from a spreadsheet, or a simpler flat file, similar to tools such as RightField.

```
[CajunPizza MozzarellaTopping OnionTopping PeperonataTopping
 PrawnsTopping TobascoPepperSauce TomatoTopping]

[CapricciosaPizza AnchoviesTopping MozzarellaTopping
 TomatoTopping PeperonataTopping HamTopping CaperTopping
 OliveTopping]
```

Listing 5. Example Named Pizza inputs.

In this section, we have introduced Tawny-OWL and shown how it enables the application of patterns; however, this has been in the context of the pizza ontology which is only an exemplar ontology, rather than one intended for real use. In the next section we consider the use of patterns in the Karyotype Ontology.

3 Patternising a novel ontology

Next we are going to describe the usage of patterns applied to a novel ontology for karyotypes which has been built ground-up. Here, we show that the ontology would be difficult to write by hand, and therefore that ontology construction is aided by Tawny-OWL and the use of patterns. The Clojure code for the Karyotype project is available at https://github.com/jaydchan/tawny-karyotype.

First, we introduce *karyotypes* which are a description of all the chromosomes in a cell. Humans have 46 chromosomes, in 23 pairs. Locations along the chromosome can be identified by their patterns of *chromosome bands* which are visible under light microscopy. A karyotype describes the total number of chromosomes, the sex chromosomes and any chromosomal abnormalities (if any).

Abnormalities are described by their kind (e.g. insertion, deletion) and the location they affect defined relative to the visible bands. Human karyotypes are normally represented in string format, as defined by the International System for human Cytogenetic Nomenclature 2009 (ISCN2009) [22]. For example, the karyotype `46,XY` is a normal male karyotype. However, current ISCN strings can be complicated, lack formal interpretation, and are not computationally amenable (trivially, they cannot even be represented in ASCII as they include meaningful underlining, used to distinguish homologous chromosomes). As a result, these ISCN strings can be hard to parse, validate and query. This is also true for the ISCN specification.

In order to overcome these problems, we have developed the Karyotype Ontology [23]. For this work, we made use of a pragmatic methodology to model karyotypic information, meaning we made no distinctions unless required by the use cases identified in the ISCN2009. The karyotype ontology demonstrates the requirement for a pattern driven approach, as it is highly repetitive; almost all of the entities are part of at least one pattern.

Our initial construction of the Karyotype Ontology involved the use of a partonomic structure, built in Manchester Syntax. A resulting definition of human chromosome 1 can be seen in Listing 6. This ontology, however rapidly became unmanageable simply because of the number of restrictions on each chromosome. This is made more complex still because banding patterns are visible at 5 different resolutions, each containing more bands than the last.

```
Class: NormalHumanChromosome1
   SubClassOf:
      HumanAutosome
      hasPart exactly 1 HumanChromosomeBand1pTer
      hasPart exactly 1 HumanChromosomeBand1p36.3
      hasPart exactly 1 HumanChromosomeBand1p36.2
      ... (23 other hasPart relations removed)
      hasPart exactly 1 HumanChromosomeBand1qTer
```

Listing 6. Incomplete chromosome definition for a normal Human Chromosome 1 using strict partonomy.

The use of Tawny-OWL enables these concepts to be generated to a standard pattern. Listing 7 shows an expression using the `humanbands` pattern, for a (small!) subset of the human chromosome bands. Additionally, we have specialised and inverted the `hasPart` relationship to `isBandOf` and `isSubBandOf`, as this produces smaller classes with fewer restrictions, which is simpler to work with[9]. The OWL ontology which is generated as a result is shown in Listing 8 as Manchester Syntax.

```
(humanbands HumanChromosome1
```

[9] At the current time, we are uncertain whether this small change in semantics will impact the reasoning we may wish to perform. If we need to invert, or add both the forward and inverse relationship at a later date, the programmatic nature of Tawny-OWL makes this easy to achieve

```
 ["p36.3" "p36.33" "p36.32" "p36.31"])
```

Listing 7. Incomplete chromosome definition for a Human Chromosome 1 showing the input format of the human bands pattern.

```
;; A - Generic chromosome 1 band definitions
Class: HumanChromosome1Band
   SubClassOf:
      HumanChromosomeBand
      isBandOf some HumanChromosome1

Class: HumanChromosome1Bandp
   SubClassOf:
      HumanChromosome1Band

;; B - Chromosome 1 band definition
Class: HumanChromosomeBand1p36.3
   SubClassOf:
      HumanChromosome1Bandp

;; C - Chromosome 1 sub-band definition
Class: HumanChromosomeBand1p36.31
   SubClassOf:
      HumanChromosome1Bandp
      isSubBandOf some HumanChromosomeBand1p36.3
```

Listing 8. Elided output from Listing 7.

The bulk of the karyotypes ontology consists of entities which are part of this pattern, the total size of which is shown in Table 1. In practice, building this ontology manually would have been impractical, particularly from a maintainability point of view. Therefore, to aid construction we tested tools such as OPPL and Populous. However, both had difficulties because of the split between different source files and forms for expressing the patternised and non-patternised sections of the ontology. This contrasts with Tawny-OWL which uses a single syntax, and also provides testing and reasoning services again within a single syntax.

Table 1. Human Chromosome model statistics.

Class Type	Biological Object	Number of Classes
Chromosome	24	27
Centromere	24	25
Telomere	48	25
Bands and Sub-bands	1213	1286
Total Number	1309	1363

The `humanbands` pattern is not the only pattern in KO. While canonical chromosomes are expressed as a partonomy, this is not sufficient to express all abnormal karyotypes. As a simple example, a karyotype can be defined by its loss of a chromosome, as opposed to its congenital absence: the karyotype `45,X,-Y` is a male karyotype where the Y chromosome has been lost; although partonomically identical to the karyotype `45,X`, it is considered different as the latter is a congenital absence.

Therefore, karyotypes are expressed as events, similar to the way ISCN strings are modelled. Using the event-based change approach, a `45,X` female is described as `46,XN`, with a deletion of a sex chromosome (see Listing 9), in this case N represents an unknown sex chromosome, as opposed to `45,X,-Y` which is derived from a `46,XY` male.

```
(defclass k45_X
  :label "The␣45,X␣karyotype"
  :subclass ISCNExampleKaryotype
  (owl-some derivedFrom k46_XN)
  (deletion 1 HumanSexChromosome))
```

Listing 9. The karyotypic definition for 45,X.

Within the Karyotype Ontology, we define an event as a concept, supported by a usage pattern and a function which generates a restriction according to this pattern. One simple example is the inverse event pattern and its usage is shown in Listing 10.

```
;; A - Inversion pattern
(defn inversion [n band1 band2]
    (exactly n hasEvent
        (owl-and Inversion
            (owl-some hasBreakPoint
                      band1 band2))))

;; B - Usage
(inversion 1 2p21 2q31)

;; C - As Manchester Syntax
hasEvent exactly 1
         (Inversion and (hasBreakPoint some 2p21 2q31))
```

Listing 10. The pattern, usage and resultant OWL used to define inverse events.

While the large, partonomic section of the karyotype is now complete, the ontology is still being developed. In time, we intend to extend the ontology and expect that the generators will be become the main "user interface" to the work, in order to present an end-user syntax. Patterns have, therefore, been useful in the development of the Karyotype Ontology, and will be so in downstream usage.

4 Patternised development of an existing ontology

While the development of the Karyotype Ontology shows that patterns have been useful in this one context, this does not demonstrate that it is generally useful; therefore, in this section, we apply a similar development methodology to an existing ontology, namely SIO. We show that, while in the Karyotype Ontology we see a few patterns generating many entities, in SIO we see the reverse; there are many patterns generating relatively few entities in the ontology construction. As with Karyotype Ontology, we can exploit patterns in downstream usage.

Semanticscience Integrated Ontology (SIO) [24] is a simple upper level OWL ontology for the integration of types and relations that provides rich descriptions of objects, processes and their attributes. It defines 1395 classes, 202 object properties, 1 data property and 8 annotation properties [25]. We chose SIO as it is explicit in promoting the use of ODPs to describe and associate numerous entities e.g. qualities and capabilities. In this paper, *SIO* is used to refer to the existing ontology, whilst *tawny-sio* refers to the Tawny-OWL recasting of SIO. The Clojure code for tawny-sio project is available at https://github.com/jaydchan/tawny-sio.

SIO has not been developed in Tawny-OWL and therefore is only available as an OWL file. To enable the development of a patternised form of SIO, first rendered this OWL file into Tawny-OWL syntax. SIO uses numeric IDs as the fragment of its URL. While there are good reasons for this, it means that the fragment is unsuitable at a code level as a memorable identifier for the entity. Therefore Tawny-OWL provides a Clojure-safe syntactic transformation of the rdfs:label. Hence, SIO_000395 becomes `to_regulate`, with a few specific replacements for tawny-sio concepts that transform to reserved words (e.g. "true" and "false").

We now describe how patterns have been applied to tawny-sio. A generic pattern was identified that is useful for most of the tawny-sio classes (Listing 11), which supports a SIO standard that (almost) all classes have a name, parent and textual description. The `sio-class` enforces this, as well as providing syntactic sugar.

```
(defn sio-class [name parent description & frames]
  (apply owl-class
         (list* (make-safe name)
                :subclass parent
                :label name
                :annotation (desc description)
                frames)))
```

Listing 11. A common pattern for tawny-sio classes. `name`, `parent`, `description` and `frames` are variables.

The `sio-class` function makes use of a second pattern, namely the description pattern which adds a standardized annotation using the Dublin Core ontology, as show in Listing 12.

```
(def dc-description (iri "http://purl.org/dc/terms/description"))
```

```
(defn desc [description]
   (annotation dc-description
       (literal description :lang "en")))
```

Listing 12. The description pattern for tawny-sio classes.

The exceptions to the `sio-class` pattern, are the concepts that model the chemical elements (atoms). An alternative pattern is used for these; the encoding of this pattern is shown in Listing 13. As this pattern is specialised for atoms, the superclass is "hard-coded" into the pattern.

```
;; A - See-also pattern
(defn see-also [value]
   (annotation seeAlso
       (literal value :type :RDF_PLAIN_LITERAL)))

;; B - Auxiliary function
(defn owl-atom-annotation-maybe [cls chebi]
  (if-not (nil? chebi)
   (add-annotation
    cls (see-also chebi))))

;; C - Atom pattern
(defn owl-atom [name chebi]
 (owl-atom-annotation-maybe
  (owl-class (make-safe name)
             :subclass atom
             :label name)
  chebi))
```

Listing 13. The atom generator function. `owl-atom` is used to distinguish from `atom` which is used by Clojure.

One interesting outcome of encoding this pattern concerns ChEBI [26] annotations. Most SIO elements have a `seeAlso` annotation that links the atom to its equivalent ChEBI ID. However, elements 112 to 118 lack this annotation, although 112 aka Copernicium aka ununubium can be found with ChEBI value CHEBI:33517. Encoding this pattern, forces us to deal with these exceptions explicitly.

5 Patterns for downstream usage

The SIO wiki pages includes many exemplar ODPs that can be used in conjunction with SIO. One such ODP is the biochemical pathway pattern. This ODP includes a variation of the sequence ODP [9]. The biochemical pathway ODP (see Listing 14), as well as other ODPs, has been encoded into tawny-sio and available for downstream usage.

```
;; A - Precedes pattern
(defn biochemical-pathway0 [reactions]
```

```
  (owl-and (first reactions)
           (owl-some precedes (rest reactions))))

;; B - Biological pathway pattern
(defn biochemical-pathway [name reactions]
  (owl-class name
             :equivalent
             (owl-and pathway
                      (owl-some has_proper_part
                                (biochemical-pathway0 reactions))
                      (owl-some has_proper_part reactions))))

;; C - Usage
(biochemical-pathway "glycosis"
                     [hexokinase_reaction
                      phosphoglucose_isomerase_reaction
                      ...])

;; D - As Manchester Syntax
Class: 'glycolysis'
   EquivalentTo:
      'pathway'
       and 'has proper part' some
           ('hexokinase reaction' and 'precedes' some
           ('phosphoglucose isomerase reaction' and 'precedes' ...)
       and 'has proper part' some 'hexokinase reaction'
       and 'has proper part' some 'phosphoglucose isomer reaction'
       ...
```

Listing 14. The Clojure function and auxiliary function for biochemical pathway pattern, its usage and resulting OWL class. `name` and `reactions` are variables.

The `biochemical-pathway` generator function is not actually used as part of tawny-sio; SIO is intended for use as an upper and middle ontology, and does not, therefore, model any pathways itself. Instead SIO documents the pattern for downstream users. However, there is no computational representation of this pattern. Within tawny-sio, we can provide such a representation which is not only descriptive but which can be used to computationally generate specific pathways. As we have also described with the Karyotype Ontology, this pattern becomes a part of the "user interface" of the ontology. As well as this pattern, we now have generators for molecule, enzyme and biochemical reaction patterns.

As with the atoms, encoding these exemplar ODPs has highlighted some interesting issues. For example, the `target_role` class associated with the biochemical pathway ODP[10] is missing from SIO. Further investigation of the missing `target_role` class, shows that the actual role class is `reactant_role` [27]. The process of the computable encoding of the patterns, is in itself a useful process for quality control and consistency of the ontology.

[10] http://code.google.com/p/semanticscience/wiki/ODPBiochemistry

In this section, we have shown how suggested usage patterns for an ontology can become part of the ontology.

6 Discussion

In this paper, we discuss the use of generalised and localised patterns in ontology engineering. We describe the use of patterns in three distinct ontologies: tawny-pizza, an *exemplar* recasting of the pizza ontology into Tawny-OWL; the Karyotype Ontology, a *novel* ontology built for describing karyotypes using an event-based approach; and tawny-sio, a fork of an *existing* upper ontology describing scientific objects, processes and their attributes. We demonstrate that for the Karyotype Ontology, most entities are part of one or more localised patterns. For SIO, the localised patterns encourage consistency and can be made available for downstream users.

There are other tools for ODPs such as OPPL2 and safe macros. However, Tawny-OWL has the advantage that a single syntax is used for both the patternised and non-patternised parts of the ontology. The basic syntax of Tawny-OWL is similar to Manchester Syntax, is "unprogrammatic" and should be usable by non-programmers, without significant, additional effort, especially when compared to the equivalent Java code.

Tawny-OWL currently provides support for three well-known existing patterns: value partition, closure and covering axioms. These patterns have been used in tawny-pizza (see Section 2) and are available for use in other ontologies. However, as it is fully programmatic, Tawny-OWL can encode patterns, localised to the scope of a single ontology. These localised patterns are useful as they enable a concise syntax, ensure consistency and support maintainability should the need for change arise. The Karyotype Ontology is predominantly made up of a variety of localised patterns – examples include the `humanbands` pattern and `inversion` pattern (see Section 3). Currently the ontology is still being developed and our patterns are incomplete. For example, karyotypic events such as deletions and insertions affect a sequence of bands, which are currently not modelled in our ontology. There are 3 ways we could achieve this: firstly to utilise a variant of the sequence design pattern [9]; secondly the assignation of ordinal numbers to the chromosome bands as a datatype; and, finally, we could use a pattern in Clojure which expands to all the affected bands. *A priori*, it is difficult to determine which of these will work best, particularly with respect to non-functional characteristics such as reasoning time. The use of Tawny-OWL will enable us to test this by generating multiple test versions of the ontology. With 800 classes this would otherwise be impractical.

We have demonstrated that patterns have benefited in both tawny-pizza and in the construction of the Karyotype Ontology. However, we wish to understand whether patterns are generally useful. Therefore, we have also applied this methodology to SIO, which we chose after the construction of Tawny-OWL, and which was built without knowledge of Tawny-OWL.

We find that patterns are less useful within tawny-sio than the Karyotype Ontology. However, a number of patterns were identified and their use could increase the consistency and concision of this ontology. Furthermore, there are additional patterns, which whilst not themselves used in tawny-sio, are potentially useful for downstream users of tawny-sio.

The utility of the pattern approach will depend on the nature of the ontology. For example, a structurally simple ontology may use few patterns. However, we note that patterns are not limited to the logical component of OWL; within tawny-sio we have used a number of annotation patterns.

While the use of general ontology design patterns is well documented, the use of localised patterns is less so. In this paper, we have described the application of Tawny-OWL to three ontologies which have allowed us to test the utility of this form of pattern. It appears to be a promising methodology which could substantially impact ontology engineering.

References

1. Stevens, R., Lord, P.: Application of Ontologies in Bioinformatics. In Staab, S., Studer, R., eds.: Handbook on Ontologies. International Handbooks on Information Systems. Springer Berlin Heidelberg (2009) 735–756
2. Warren, P.: Ontology Users Survey Summary of Results. Technical report, The Knowledge Media Institute (KMi), The Open University (2013)
3. Mungall, C., Ruttenberg, A., Horrocks, I., Osumi-Sutherland, D.: OBO Flat File Format 1.4 Syntax and Semantics [DRAFT]. http://oboformat.googlecode.com/svn/branches/2011-11-29/doc/obo-syntax.html
4. W3C OWL Working Group: OWL 2 Web Ontology Language Document Overview (Second Edition). http://www.w3.org/TR/owl2-overview/ (2012)
5. Tirmizi, S., Aitken, S., Moreira, D., Mungall, C., Sequeda, J., Shah, N., Miranker, D.: Mapping between the OBO and OWL ontology languages. Journal of Biomedical Semantics **2**(1) (2011) 1–16
6. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1995)
7. Egaa, M., Rector, A., Stevens, R., Antezana, E.: Applying Ontology Design Patterns in Bio-ontologies. In Gangemi, A., Euzenat, J., eds.: Knowledge Engineering: Practice and Patterns. Volume 5268 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2008) 7–16
8. Rector, A.: Representing Specified Values in OWL: value partitions and value sets. http://www.w3.org/TR/swbp-specified-values/ (2005)
9. Drummond, N., Rector, A., Stevens, R., Moulton, G., Horridge, M., Wang, H.H., Seidenberg, J.: Putting OWL in order: Patterns for sequences in OWL. Concrete (2006) 1–10
10. Egana Aranguren, M., Stevens, R., Antezana, E.: Transforming the Axiomisation of Ontologies: The Ontology Pre-Processor Language. Nature Precedings (Dec 2009)
11. Mungall, C., Ruttenberg, A., Osumi-Sutherland, D.: Taking shortcuts with OWL using safe macros. Nature Preceedings (2010)

12. Wolstencroft, K., Owen, S., Horridge, M., Krebs, O., Mueller, W., Snoep, J.L., du Preez, F., Goble, C.: RightField: embedding ontology annotation in spreadsheets. Bioinformatics **27**(14) (2011) 2021–2022
13. Jupp, S., Horridge, M., Iannone, L., Klein, J., Owen, S., Schanstra, J., Stevens, R., Wolstencroft, K.: Populous: A Tool for Populating Templates for OWL Ontologies. In Burger, A., Marshall, M.S., 0001, P.R., Paschke, A., Splendiani, A., eds.: SWAT4LS. Volume 698 of CEUR Workshop Proceedings., CEUR-WS.org (2010)
14. Rocca-Serra, P., Ruttenberg, A., O'Connor, M.J., Whetzel, P.L., Schober, D., Greenbaum, J., Courtot, M., Brinkman, R.R., Sansone, S.A., Scheuermann, R., Scheuermann, R., Peters, B.: Overcoming the ontology enrichment bottleneck with Quick Term Templates. Appl. Ontol. **6**(1) (January 2011) 13–22
15. OConnor, M., Halaschek-Wiener, C., Musen, M.: Mapping Master: A Flexible Approach for Mapping Spreadsheets to OWL. In Patel-Schneider, P., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J., Horrocks, I., Glimm, B., eds.: The Semantic Web ISWC 2010. Volume 6497 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2010) 194–208
16. Lord, P.: The Semantic Web takes Wing: Programming Ontologies with Tawny-OWL. OWLED 2013 (March 2013)
17. Vassiliadis, V., Wielemaker, J., Mungall, C.: Processing OWL2 ontologies using Thea: An application of logic programming. In: OWLED 2009. (2009)
18. Ogbuji, C.: InfixOWL: An Idiomatic Interface for OWL. In Dolbear, C., Ruttenberg, A., Sattler, U., eds.: OWLED. Volume 432 of CEUR Workshop Proceedings., CEUR-WS.org (2008)
19. Horridge, M., Bechhofer, S.: The OWL API: A Java API for OWL ontologies. Semant. web **2**(1) (January 2011) 11–21
20. Horridge, M., Patel-Schneider, P.F.: OWL 2 Web Ontology Language Manchester Syntax (Second Edition). http://www.w3.org/TR/owl2-manchester-syntax/ (2012)
21. Stevens, R.: Closing Down the Open World: Covering Axioms and Closure Axioms. http://ontogenesis.knowledgeblog.org/1001 (2011)
22. Shaffer, L., on Human Cytogenetic Nomenclature, I.S.C., Slovak, M., Campbell, L.: ISCN 2009: An International System for Human Cytogenetic Nomenclature (2009). Karger (2009)
23. Warrender, J.D., Lord, P.: The Karyotype Ontology: a computational representation for human cytogenetic patterns. BioOntologies 2013 (2013)
24. Dumontier, M.: SIO - semanticscience - The Semanticscience Integrated Ontology (SIO) - Scientific Knowledge Discovery - Google Project Hosting . http://code.google.com/p/semanticscience/wiki/SIO Accessed: 2013-09-27.
25. Dumontier, M., Baker, C.J.O., Baran, J., Callahan, A., Chepelev, L., Cruz-Toledo, J., Duck, G., Klassen, D., McCusker, J.P., Samwald, M., Villanueva-Rosales, N., Wilkinson, M., Hoehndorf, R.: The Semanticscience Integrated Ontology (SIO) for Biomedical Research and Knowledge Discovery. Draft available at http://code.google.com/p/semanticscience/wiki/SIO (2013)
26. Hastings, J., de Matos, P., Dekker, A., Ennis, M., Harsha, B., Kale, N., Muthukrishnan, V., Owen, G., Turner, S., Williams, M., Steinbeck, C.: The ChEBI reference database and ontology for biologically relevant chemistry: enhancements for 2013. Nucleic Acids Research **41**(D1) (2013) D456–D463
27. Boelling, C., Dumontier, M., Weidlich, M., Holzhtter, H.G.: Role-based representation and inference of biochemical processes. In Cornet, R., Stevens, R., eds.: ICBO. Volume 897 of CEUR Workshop Proceedings., CEUR-WS.org (2012)