Towards Diagrammatic Ontology Patterns

Gem Stapleton[1], John Howse[1], Kerry Taylor[2], Aidan Delaney[1], Jim Burton[1], and Peter Chapman[1]

[1] University of Brighton, UK, www.ontologyengineering.org
{g.e.stapleton,john.howse,a.j.delaney,j.burton,p.chapman}@brighton.ac.uk
[2] CSIRO Computational Informatics and Australian National University, Australia
kerry.taylor@csiro.au

Abstract. It has long been recognized that patterns can be a useful and important tool when building models. This is reflected by their adoption in the practice of ontology and software engineering. Similarly, visual representations of information are often seen as beneficial with, for example, software engineering making use of the suite of diagrammatic notations forming the UML. Likewise, ontology engineering has seen the development of a variety of different visualizations for classes and properties. This paper combines these two strands of work, making visual (diagrammatic) patterns available to ontology engineers.

1 Introduction

This paper ties together two strands of research: ontology engineering using patterns and ontology visualization. Major benefits of using patterns include the simplification of the modelling process and the provision of a consistent approach to specifying commonly occurring constructions. Patterns give ontology engineers convenient access to these constructions. Similarly to the design patterns of object-oriented software development [3], they provide a common language for analyzing and sharing reusable, composable design abstractions. The visual ontology engineering patterns we present in this paper are lower-level than a typical design pattern from software development but fulfil the same role and may serve as building blocks for more complex patterns. The aim of visualization is similar to that of patterns, in the sense that visualizations are intended to aid ontology engineering. Visualizations (such as OWLViz [5], OntoGraf [1] and CMap [4]) can bring about benefits by revealing information that could be unapparent when using traditional notations. The main contribution of this paper is a set of diagrammatic patterns for ontology engineering, defined using *concept diagrams* [7]. Section 2 an introduction to concept diagrams, focusing on the aspects needed for this paper. Section 3 defines patterns for commonly occurring constraints and section 4 applies the patterns.

2 Concept Diagrams for Ontology Modelling

This section provides an introduction to concept diagrams, designed as part of the Ontology Engineering with Diagrams project (`www.ontologyengineering.org`).

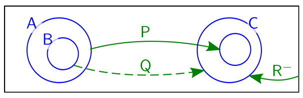

Fig. 1. A concept diagram

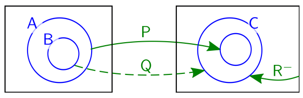

Fig. 2. Multiple boundary rectangles

Readers interested in the full notation and its formalization should see [7]. Concept diagrams represent classes using *closed curves* and properties using *arrows* which can be *solid* or *dashed*. The spatial relationships between the closed curves and the sources and targets of arrows convey semantic information.

Fig. 1 asserts that the class B is subsumed by A and both A and B are disjoint from C. The spatial properties of inclusion and exclusion correspond to the semantic properties of set inclusion and disjointness. The *solid* arrow P asserts that individuals in A are only related to elements in C: the target of the arrow is the set of things to which the elements in A are related and this set is subsumed by C. The solid arrow R^-, where R^- is the *inverse* of R, asserts that all things are, between them, related to exactly the individuals in C under R^-; the source of this arrow is the boundary rectangle which represents the set of all things, often denoted $\top$. Lastly, the *dashed* arrow provides *partial* information about property Q: under Q, the individuals in B are, between them, related to at least the elements in C. Under some circumstances, we may not wish to assert disjointness and subsumption relationships. Concept diagrams make this readily achievable, whilst avoiding clutter, by using multiple rectangles. Fig. 2 visualizes the same information as Fig. 1 except for the disjointness of C with A and B. Spatial relationships only convey information within a single rectangle.

3 Diagrammatic Patterns for Common Constructions

We now demonstrate how to express nine commonly occurring axioms using patterns, contrasting with Description Logic (DL). Common constraints that are imposed on classes are subsumption (subset), disjointness and equivalence. To express that one class is subsumed by another class, concept diagrams use curve containment, reflected in our first pattern; to express class subsumption in DL, one asserts $\mathsf{C}_2 \sqsubseteq \mathsf{C}_1$. Similarly, concept diagrams use curve disjointness (i.e. non-overlapping curves) to express class disjointness, captured in DL by $\mathsf{C}_1 \sqcap \mathsf{C}_2 \sqsubseteq \bot$.

Pattern 1: Class Subsumption Class C_1 subsumes class C_2, Fig. 3.

Pattern 2: Class Disjointness Classes C_1 and C_2 are disjoint, Fig. 4.

Pattern 2 has an obvious generalization to (concisely) assert that n classes are pairwise disjoint (that is, any pair of the n classes are disjoint). Using DL, one axiom is required for each pair of classes to capture this disjointness information: $\mathsf{C}_1 \sqcap \mathsf{C}_2 \sqsubseteq \bot$,...,$\mathsf{C}_1 \sqcap \mathsf{C}_n \sqsubseteq \bot$, ..., $\mathsf{C}_{n-1} \sqcap \mathsf{C}_n \sqsubseteq \bot$; OWL has a more succinct representation: DisjointClasses($\mathsf{C}_1, ..., \mathsf{C}_n$). In the diagram below, the

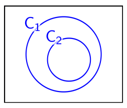

Fig. 3. Pattern 1: Class Subsumption

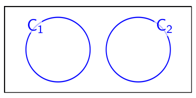

Fig. 4. Pattern 2: Class Disjointness

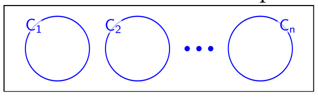

Fig. 5. Pattern 3: General Class Disjointness

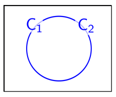

Fig. 6. Pattern 4: Class Equivalence

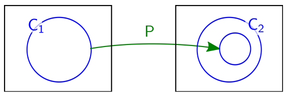

Fig. 7. Pattern 5: All Values From

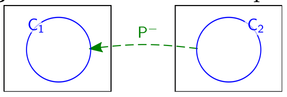

Fig. 8. Pattern 6: Some Values From

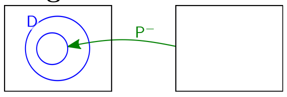

Fig. 9. Pattern 7: Domain

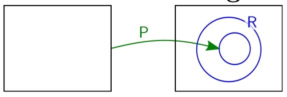

Fig. 10. Pattern 8: Range

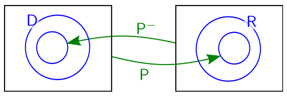

Fig. 11. Pattern 9: D & R

ellipsis indicates the presence of a further $n - 3$ circles labelled in the obvious fashion.

Pattern 3: General Class Disjointness $C_1, .., C_n$ are pairwise disjoint, Fig. 5.

Ontology engineers often want to express that two classes are equivalent. As with the other patterns, there are many semantically equivalent, but syntactically different, concept diagrams that express class equivalence. The following pattern employs two overlaying (completely concurrent) curves.

Pattern 4: Class Equivalence Classes C_1 and C_2 are equivalent, Fig. 6.

In DL, Class Equivalence can be expressed by $C_1 \equiv C_2$. Again, the Class Equivalence pattern has an obvious generalization to the n-class case: to express that n classes are equivalent draw n overlaying curves. The number of DL axioms to express many classes are all equivalent to each other increases rapidly, whereas only a single diagram is needed, omitted for space reasons. A common property restriction is to enforce 'All Values From' and 'Some Values From' constraints.

Pattern 5: All Values From All individuals in class C_1 have all values, under property P, from class C_2, Fig. 7

The arrow in the above diagram formally asserts that the image of the property P (considering P as a binary relation), when its domain is restricted to C_1, is a subset of C_2. In other words, the only things that individuals in C_1 are related to, under P, must be in C_2. The use of multiple bounding boxes ensures that no unintended disjointness information between classes is asserted. In DL, the All Values From pattern is captured by $C_1 \sqsubseteq \forall P.C_2$. We also present a pattern for 'Some Values From', expressed in DL by $C_1 \sqsubseteq \exists P.C_2$.

Pattern 6: Some Values From Individuals in class C_1 have at least one value, under property P, from class C_2, Fig. 8.

The above diagram makes use of the *inverse* of property P. To justify the correctness of the Some Values From pattern, consider an individual, c_1, in the class C_1. The pattern must ensure that c_1 has a value, c_2, from C_2 under property P. Well, c_1 has such a value, c_2, if and only if c_2 has the value c_1 under P^-. Equivalently, the image of P^- when its domain is restricted to C_2 includes at least all of the individuals in C_1, captured by the dashed arrow.

Our last three patterns concern domains and ranges of properties. Firstly, consider the domain, D, of property P. The domain of P is D if and only if the range of the inverse, P^-, of P is D.

Pattern 7: Domain of a Property The domain of property P is D, Fig. 9.

The corresponding DL formalization of this pattern is $\forall \mathsf{P}.\top \sqsubseteq \mathsf{D}$; the construction $\forall \mathsf{P}.\top$ builds the pre-image of the property P. The Domain of a Property pattern employs the same style of construction: the arrow builds the pre-image of P. In DL, the *range* is typically defined by $\top \sqsubseteq \forall \mathsf{P}.\mathsf{R}$. The range can also be defined in DL by constructing the pre-image of the inverse, P^-, of P and asserting that this pre-image is subsumed by R: $\forall \mathsf{P}^-.\top \sqsubseteq \mathsf{R}$. Our Range of a Property pattern constructs the image of P, using an arrow, and asserts that this image is subsumed by the range, R.

Pattern 8: Range of a Property The range of property P is R, Fig. 10.

Pattern 9: Domain and Range of a Property The domain and range of property P are D and R respectively, Fig. 11.

4 Applying the Patterns

We demonstrate the application of the patterns to the Semantic Sensor Networks (SSN) Ontology [2] as a case study. The examples represent just a small fragment of that ontology, but have been chosen to illustrate the application of the patterns above. The SSN ontology, the class MeasurementCapability is subsumed by Property and there are four pairwise disjoint classes: Sensor, Stimulus, Property, and Sensing. SensorInput is equivalent to Stimulus. The Class Subsumption, General Class Disjointness, and Class Equivalence patterns yield the diagrams in Figs 12, 13 and 14 respectively. Regarding property restrictions, the SSN ontology includes the constraint that sensors *detect* only stimuli. The property detects relates individuals in the Sensor class only to individuals in the Stimulus class. This property restriction is an All Values From constraint and the corresponding diagram is in Fig. 15. The SSN ontology also makes plentiful use of Some Values From property restrictions. One example is that sensors *implement* some sensing. The property implements relates individuals in the Sensor class to some individual(s) in the Sensing class. The Some Values From diagrammatic pattern thus gives rise to Fig. 16. Lastly, we demonstrate instances of the Domain of a Property, Range of a Property, and the Domain and Range of a Property patterns. To do so, we make use of a further two classes in the SSN ontology:

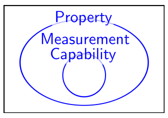

Fig. 12. Subsumption

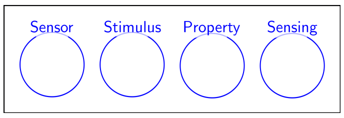

Fig. 13. General Disjointness

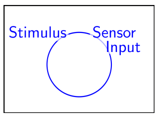

Fig. 14. Equivalence

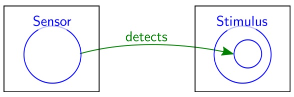

Fig. 15. All Val. From

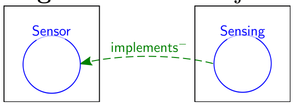

Fig. 16. Some Val. From

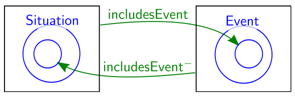

Fig. 17. D & R

Situation and Event. There is a property, includesEvent, with domain Situation and range Event shown in Fig. 17.

5 Conclusion

We have presented nine diagrammatic patterns for defining constraints that occur frequently in ontology engineering. These patterns are all formal, since concept diagrams have a fully defined syntax and semantics [7]. There are various avenues for significant future work. A particular goal is to provide tool support for ontology engineering using concept diagrams. This will treat concept diagrams as 'first-class' citizens in the model development process, rather than purely as a visualization of an ontology. We envisage producing a tool that allows the diagrammatic patterns to be accessed. A big challenge is to ensure that the resulting drawn (concrete) diagram has an effective layout. This will build on the now substantial body of work that solves Euler diagram layout problems [6].

References

1. OntoGraf. http://protegewiki.stanford.edu/wiki/OntoGraf, accessed July 2013.
2. M. Compton et al. The SSN ontology of the semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17(0):25–3, 2012.
3. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional, 1994.
4. P. Hayes, T. Eskridge, M. Mehrotra, D. Bobrovnikoff, T. Reichherzer, and R. Saavedra. Coe: Tools for collaborative ontology development and reuse. In *Knowledge Capture Conference*, 2005.
5. M. Horridge. Owlviz. www.co-ode.org/downloads/owlviz/, accessed June 2009.
6. G. Stapleton, J. Flower, P. Rodgers, and J. Howse. Automatically drawing Euler diagrams with circles. *Journal of Visual Languages and Computing*, 23:163–193, 2012.
7. G. Stapleton, J. Howse, P. Chapman, A. Delaney, J. Burton, and I. Oliver. Formalizing concept diagrams. In *Visual Languages and Computing*. KSI, 2013.