8.6: Ontology Design Patterns

Ontology Design Patterns (ODPs) are a middle out way for developing ontologies. They can be viewed as an extremely lightweight version of design principles alike found in foundational ontologies, but then with less ‘clutter’. That is, they can be cleverly modularized foundational ontology fragments that serve as design snippets for good modeling practices. They also can be viewed as a way of bottom-up pattern finding that is then reused across the ontology and offered to others as a ‘best practices’ design solution for some modeling aspect. ODPs have been proposed first a while ago [BS05, Gan05], and have gained some traction in research in recent years with various ideas and proposals. There is, therefore, no clear single, neat, core to extract from it and describe at present. A clear, informal overview is described in [GP09], but terms, descriptions, and categorizations are being reworked [FGGP13], and the sub-field better characterized with respect to the issues for using ODPs and possible research directions [BHJ+15].

Let us first introduce some definitions for a pattern for a specific ontology and their uses and then proceed to types of patterns. The definitions are geared to the OWL language, but one can substitute that for another language of choice.

Definition \(\PageIndex{1}\): (Language of pattern instantiation [FK17]). OWL Ontology \(\langle O \rangle\) with language specification adhering to the W3C standard [MPSP09], which has classes \(\langle \text{Cin } V\{C\} \rangle\), object properties \(\langle \text{OPin } V\{OP\} \rangle\), data properties \(\langle \text{Dlin } V\{D\} \rangle\), data types \(\langle \text{DTin } V\{DT\} \rangle\) of the permitted XML schema types, axiom components ('language features') \(\langle Xin V\{X\} \rangle\), and such that \(\langle Axin V\{Ax\} \rangle\) are the axioms.

The ‘axiom components’ include features such as, among others, subsumption, transitivity, existential quantification, and cardinality, which can be used according to the syntax of the language. A pattern itself is a meta-level specification, in a similar fashion as stereotyping in UML. Just in case a pattern also includes ‘reserved’ entities from, say, a foundational ontology, they get their own entry in the vocabulary to clearly distinguish them.
Definition \(\PageIndex{2}\):

\(\text{Language for patterns: Vocabulary } \mathcal{V} [FK17]\). The meta-level (second order) elements (or stereotypes) for patterns are:

- class \(C \in \mathcal{V}_{C}\) as \(\mathcal{C}\) in the pattern;
- object property \(OP \in \mathcal{V}_{OP}\) as \(\mathcal{R}\) in the pattern;
- data property \(D \in \mathcal{V}_{D}\) as \(\mathcal{D}\) in the pattern;
- data type \(DT \in \mathcal{V}_{DT}\) as \(\mathcal{DT}\) in the pattern;
- reserved set of entities from a foundational ontology, as \(\mathcal{F}\) in the pattern;

where added subscripts \(i\) with \(1 \leq i \leq n\) may be different elements. Two elements in the vocabulary are called *homogeneous* iff they belong to the same type, i.e., they are both classes, or both object properties, and so on. Elements can be used in axioms \(Ax \in \mathcal{V}_{Ax}\) that consists of axiom components \(x \in \mathcal{V}_{X}\) in the pattern such that the type of axioms are those supported in the ontology language in which the instance of the pattern is represented.

With these ingredients in place, one can then define an ontology pattern \(\Pi\) as follows.

Definition \(\PageIndex{3}\):

\(\text{Ontology Pattern } \Pi [FK17]\). An *ontology pattern* \(\Pi\) consists of more than one element from vocabulary \(\mathcal{V}\) which relate through at least one axiom component from \(\mathcal{V}_{X}\). Its specification contains the:

- pattern name;
- pattern elements from \(\mathcal{V}\);
- pattern axiom component(s) from \(\mathcal{V}_{X}\);
- pattern’s full formalization.

For instance, the *basic all-some* pattern that we have seen as ‘macro’ in Section 7.2: From Spreadsheets to OWL has as specification ([FK17]):

- *pattern name: basic all-some*
- *pattern elements: \(\mathcal{C}_{1}, \mathcal{C}_{2}, \mathcal{R}\)*
- *pattern axiom component(s): \(\sqsubseteq, \exists\)*
- *pattern’s full formalization: \(\mathcal{C}_{1}\sqsubseteq\exists\mathcal{R}\mathcal{C}_{2}\)*

An instantiation of the *basic all-some* pattern in an ontology, say, the AWO, may be, e.g., \(\texttt{Giraffe}\sqsubseteq\exists\texttt{drinks.Water}\).

As can be seen from the definition, they are referred to agnostically as *patterns*—it may be a pattern realized in the ontology and some algorithm has to search for (as was the scope in [FK17]) as well as one defined separately and applied during the design phase and is therewith thus also in line with some newly proposed terminology [FGGP13]. Ontology patterns tend to be more elaborate than the *basic all-some* pattern. For instance, one could specify a pattern for how to represent attributions with DOLCE’s *Quality* rather than an OWL data property, as was discussed in Section 6.1: Foundational Ontologies “Typical Content of a Foundational Ontology”, or how to systematically...
approximate representing an \(n\)-ary into \(n\) binaries in OWL. Furthermore, there are broader options for ontology patterns. A selection of them with a few examples is as follows.

- **Architecture pattern.** This specifies how the ontology is organized. For instance, one could choose to have a modular architecture in the sense of sub-domains. An example of a fairly elaborate architecture is illustrated in Figure 7.6.1 for BioTop [BSSH08].

- **Logical pattern.** This deals with the absence of some features of a representation language and how to work with that. The issue with \(n\)-aries in OWL is such an example.

- **Content pattern.** This pattern assists with representing similar knowledge in the same way for that particular ontology. Recalling the rules-as-you-go from thesauri bottom-up development, they can be specified as content patterns. A larger example is shown in Figure 7.7.1.

- ‘Housekeeping’ patterns, including so-called lexico-syntactic patterns. They refer to ensuring clean and consistent representations in the ontology. For instance, to write names in CamelCase or with dashes, and using IDs with labels throughout versus naming the classes throughout the ontology.

There are also practical engineering tasks in the process of using ODPs, such as a workflow for using ODPs and the usual requirements of documentation and metadata; recent first proposals include [FBR\(^+\)16, KHH16].

![Figure 7.6.1: BioTop’s Architecture, which links to both DOLCE and BFO-RO and small ‘bridge’ ontologies to link the modules. (Source: http://www.imbi.uni-freiburg.de/ontology/biotop).](http://www.imbi.uni-freiburg.de/ontology/biotop)