Towards a simple but useful ontology design pattern representation language

Pascal Hitzler[1], Aldo Gangemi[2,3], Krzysztof Janowicz[4], Adila A. Krisnadhi[1,5], and Valentina Presutti[2]

[1] Data Semantics (DaSe) Laboratory, Wright State University, Dayton, OH, USA
[2] Semantic Technology Laboratory, ISTC-CNR, Italy
[3] LIPN Université Paris 13
[4] STKO Lab, University of California, Santa Barbara
[5] Universitas Indonesia

Abstract. The need for a representation language for ontology design patterns has long been recognized. However, the body of literature on the topic is still rather small and does not sufficiently reflect the diverse requirements on such a language. Herein, we propose a simple but useful and extendable approach which is fully compatible with the Web Ontology Language and should be easy to adopt by the community.

1 Introduction

Ontology Design Patterns (ODP) have become an established paradigm for ontology engineering; see [15] for a recent overview of the state of the art. At the same time, support for this approach in the form of tools, available patterns and detailed workflows is still limited and requires further development [6]. A particularly important missing piece [12] is a suitable language for the representation of ontology design patterns and their relationships.

Some proposals for such a language have already been made, however they fall short in some respects regarding what will ultimately be needed; see section 6 for a discussion. In particular, we argue that they are too complicated for easy adoption and too complex for most applications.

In this paper, we make a proposal for a simple representation language for ontology design patterns. The proposed language is practically useful, aligns with existing standards and tools, and is extensible towards a more complex representation paradigm which can be developed by the community as needed. In fact, one of the goals of our proposal is to raise questions and solicit a discussion within the community.

Central to our proposal is the systematic use of OWL annotation properties, which are available since the revision of the OWL standard in 2009 [17]. We will use such annotation properties to indicate patterns and to express relationships between patterns and ontology modules. Thus, our first version of an ontology design pattern representation language consists of a set of OWL annotation properties together with guidelines for their use. The approach has several advantages:

- OWL annotation properties are part of the OWL standard, i.e. our approach is fully compatible with OWL, and, thus, can be adopted easily by users familiar with the Web Ontology Language.
- Main ontology modeling tools such as Protégé [29] natively support the use of OWL annotation properties.
- The representation language can be further developed by the community, by extending or revising the annotation properties used or by providing processes and methodologies for their use.
- Further development of the representation language can include the definition and community adoption of *patterns* for the representation of relevant information, i.e., the community can apply its own principles to the future development.

The rest of the paper will be structured as follows. In section 2 we will discuss the desired capabilities of the representation language. In section 3 we will give a conceptual overview of the language. In section 4 we will discuss the implementation of the language by means of OWL annotation properties. In section 5 we provide some examples. Section 6 discusses related work, and section 7 provides conclusions and a path forward for the representation approach.

2 Desired Capabilities

The process of utilizing ontology design patterns for modeling has been worked out by now [5,15,20]. This modeling approach contributed to the development of modular[6] ontologies, where the modules are based on ontology design patterns [20,22,23]. Critical for this modeling approach is the reuse and adaptation [13] of already existing ontology design patterns. Unfortunately, however, the primary language for representing both ontology design patterns and the resulting modular ontologies is the Web Ontology Language (OWL) [17], which does not specify any native support for ontology design patterns or for modularization information. As a result, information about patterns and the corresponding modeling processes often gets lost after modeling has been completed, or is at best conveyed in the documentation. Consequently, a suitable language for representing such information is needed.

In this paper, we propose such a language and our guiding principles are simplicity, usefulness, adherence to existing standards, and extensibility. In more detail, we seek the following capabilities.

- Full compatibility with the OWL standard and with OWL supporting tools.
- Support for the identification of ontology design patterns as such (i.e., as distinct from ontologies), including identification of relevant parts of such patterns.

[6] We are aware that the term "module" in the context of ontologies is rather overloaded. In this paper, we mean it in the sense of [22,23], and other modularization approaches may or may not fit into what we are doing.

– Support for representing relevant relationships between patterns. For instance, one should be able to express that a pattern is a refinement or a generalization of another pattern, or that it is closely related to another pattern.
– Support for the identification of modules in ontologies generated using a modular, ontology-design-pattern-based approach.
– Support for representing relationships between ontology modules and the ontology design patterns which have been used as templates for these modules.
– Extensibility of the language by means of community-provided patterns for representing relevant information about patterns and modules.

In this first proposal, we will not address all of the mentioned aspects comprehensively. Instead, we propose selected steps towards this goal in the form of an extensible approach which can be refined by the community. For example, how exactly relationships between patterns should be recorded is still subject to research. Eventually, we envision that investigations will lead to ontology design patterns in their own right which can be used for representing such relationships. These patterns can then be used as extensions to our proposal.

3 Conceptual Overview

The approach we propose for representing ontology design patterns, ontology modules, and their relationships, is based on OWL annotation properties. Such annotations can be made for ontologies, axioms, and entities [25]. Relevant annotations for our purposes fall into three categories: (1) annotations indicating that an axiom or entity belongs to a certain module or pattern, (2) annotations indicating a pattern-relevant type of an axiom or entity, e.g. whether it is an external pattern, or a required class when the pattern is used as a template, and (3) annotations indicating relationships between patterns or between patterns and modules.

In this initial proposal, we focus on the first type, i.e. on annotations indicating that an axiom or entity belongs to a certain module or pattern. We also partially address the second type and third type, more precisely indicating pattern-relevant types and recording simple relationships between patterns or patterns and modules. In other words, we focus on representational issues which we believe should be uncontroversial and directly usable. We do not address more complex representational issues although we discuss some, as we believe that developing suitable representations requires more community discussion. However, our approach is extensible for this purpose, and we will return to this point at the end of this paper.

We define our representation language in the form of an ontology, where some of the properties (aka binary relations) are to be understood as annotation properties. This perspective naturally opens up the discussion of suitable *ontology design patterns* for extending this ontology.

Concrete competency questions which we address include the following:

1. Given a module within an ontology, which patterns was this module based on?
2. List all classes, properties, individuals, axioms, belonging to a given module or pattern.
3. What are the modules a given ontology consists of?
4. Is this pattern (or module) a specialization or generalization of another pattern (or module)?

4 The OPLa ontology

In this section, we present the OPLa ontology which will serve as ontology design pattern representation language. We first define it in terms of an ontology, and show how to use it in section 5.

Figure 1 shows a schema diagram of the OPLa. We suggest the namespace `http://ontologydesignpatterns.org/opla/` and the prefix `opla`. Individuals, Properties, Classes and Axioms are OntologicalEntities, and a collection of such entities is called an OntologicalCollection. An OntologicalCollection is either an Ontology or a Module or a Pattern. In principle one may question whether these three classes are disjoint or not. Considering that clear definitions are still to be discussed and agreed upon by the community, we feel that including disjointness axioms would be overcommitting at this stage. Our working definitions are as follows – they are not meant to be prescriptive, but capture what we believe is often general practice based on our experience: By *Pattern* we mean a conceptual model expressed in an OWL file which solves a generic modeling problem, such as how to model the roles of agents [7,11]. Patterns are inherently incomplete in the sense that they refer to external concepts which may not be modeled in detail, internally. For example, the agent role pattern would refer to a class called Agent, but would not necessarily include a formal specification for agents. Agent, in this case, would be what we call an *external class* (or *external pattern*), i.e., when using the pattern within an ontology, a refined model of Agent, e.g., based on a different pattern, will mostly be in order. By *Module* we refer to a part of an ontology which captures a complex concept or conceptual area of the formalized domain. Often, modules will be modeled by making use of one or several patterns, possibly refining them, and by placing the module within the context of a larger ontology. An example would be the module Cruise from [22], which is based on patterns for events, agent roles, and trajectories (and some others).

Now, the property isNativeTo is used to indicate what Ontology, Module, or Pattern an OntologicalEntity belongs to, i.e. it is a core element. The property is not functional, i.e. multiple assignments are allowed. Let us consider the case where we have an ontology which consists of several modules. Each OntologicalEntity may be native to one or more modules within the ontology, and this is indicated using the isNativeTo property. At the same time, this OntologicalEntity may be native also to the overall ontology. Likewise, when defining a pattern, we can use the isNativeTo property to indicate for each OntologicalEntity of the pattern that it is actually native to this pattern.

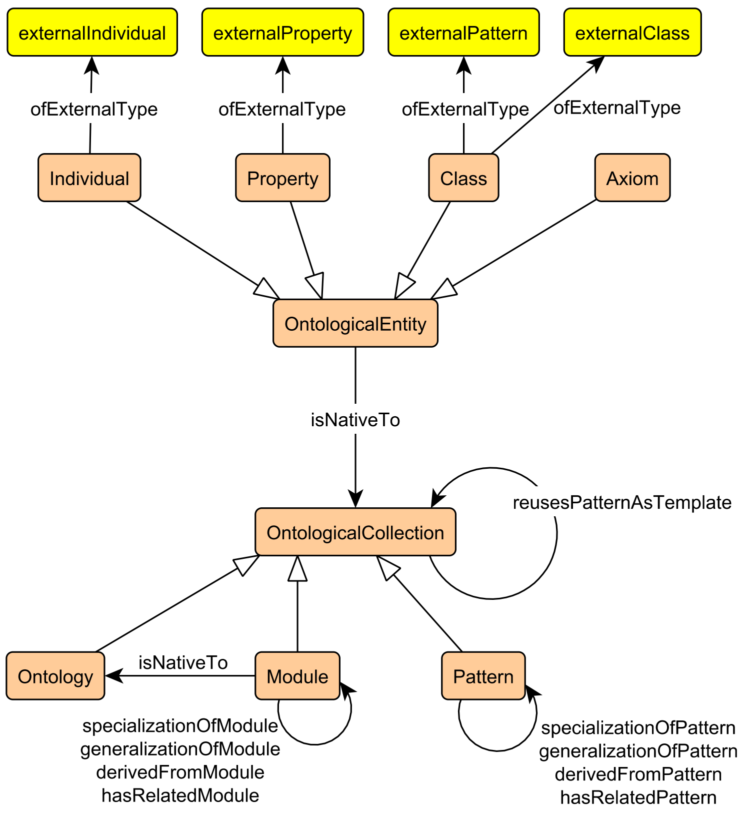

Fig. 1. Schema diagram for the OPLa ontology. White-headed arrows indicate subclass relationships. In the top row (boxes indicated in yellow) are individuals, while all remaining boxes (in orange) indicate class names.

There are several advantages of these declarations. It is now possible to programmatically extract all entities which belong to a module within an ontology. The idea – following [20,22] – is that modules represent entities which can be discussed separately, e.g., with domain experts, or are replaceable by other (more refined) modules while the remaining ontology can stay mostly untouched. Axioms can, of course, span several modules, and would thus be native to them. Also, if modules are reused elsewhere, then the corresponding part of an OWL file can be imported or even copied into the new ontology, while the annotation properties are kept and thus provide a type of provenance information for these modules. An additional advantage is the support to alignment procedures. For example, if the same module is reused by several ontologies it will be simpler to integrate them and their underlying data, because part of their knowledge is encoded by a shared formalization.

The top row in Figure 1 is meant to be used with patterns only. As indicated above, they are used to state that a class (or property or individual) used in an OWL model describing a pattern, is in fact an external class and so on. We envision that this will be helpful in particular for tools which support ontology engineers in assembling ontologies by connecting (refinements of) patterns. The identifiers externalIndividual, externalProperty, externalPattern, externalClass are individuals.

Regarding the bottom row, we use simple properties to indicate simple relationships between different OntologicalCollections. An ontological collection would usually be constructed by reusing one or several patterns as templates, indicated by the property reusesPatternAsTemplate. As defined above, we always consider a module to be part of an ontology, and this is indicated by the isNativeTo property. A pattern can be declared to be a specialization, a generalization or a derivation of another pattern, and two patterns can be declared to be related in some unspecified way, and analogous properties are available for modules.

In terms of axiomatization, we declare scoped (also called guarded) domain and range restrictions for ofExternalType and reusesPatternAsTemplate.

$$\begin{aligned}
\text{Individual} &\sqsubseteq \forall\text{ofExternalType}.\{\text{externalIndividual}\}\\
\{\text{externalIndividual}\} &\sqsubseteq \forall\text{ofExternalType}^-.\text{Individual}\\
\text{Property} &\sqsubseteq \forall\text{ofExternalType}.\{\text{externalProperty}\}\\
\{\text{externalProperty}\} &\sqsubseteq \forall\text{ofExternalType}^-.\text{Property}\\
\text{Class} &\sqsubseteq \forall\text{ofExternalType}.\{\text{externalPattern}\} \sqcup \{\text{externalClass}\}\\
\{\text{externalPattern}\} &\sqsubseteq \forall\text{ofExternalType}^-.\text{Class}\\
\{\text{externalClass}\} &\sqsubseteq \forall\text{ofExternalType}^-.\text{Class}\\
\text{OntologicalCollection} &\sqsubseteq \forall\text{reusesPatternAsTemplate}.\text{Pattern}\\
\text{Pattern} &\sqsubseteq \forall\text{reusesPatternAsTemplate}^-.\text{OntologicalCollection}
\end{aligned}$$

For isNativeTo, we declare scoped domains and ranges, and an existential for module (second line below).

$$\begin{aligned}
\text{OntologicalEntity} &\sqsubseteq \forall\text{isNativeTo}.\text{OntologicalCollection}\\
\text{Module} &\sqsubseteq \exists\text{isNativeTo}.\text{Ontology}\\
\text{Module} &\sqsubseteq \forall\text{isNativeTo}.\text{Ontology}\\
\text{Ontology} &\sqsubseteq \forall\text{isNativeTo}^-.(\text{Module} \sqcup \text{OntologicalEntity})\\
\text{Module} \sqcup \text{Pattern} &\sqsubseteq \forall\text{isNativeTo}^-.\text{OntologicalEntity}
\end{aligned}$$

For the remaining properties in the bottom row, we also only declare scoped domain and range restrictions; we do not list the axioms as they are straightforward.

Note, however, that all properties mentioned above will be annotation properties, as they do not belong to the actual specification, in terms of content, of the ontology, module, or pattern. As OWL DL does not allow us to declare complex axioms over annotation properties, inclusion of these axioms would render the ontology to be in OWL Full but not in OWL DL. Our suggested usage, however, is that the axioms are not included in the ontology (or pattern), and in fact the classes shown in Figure 1 should not be used, i.e. classes should *not* be typed as *opla:Class* etc. Rather, the schema diagram is informative only, in the sense that it tells us between which entities the annotation properties should be declared, and the axioms are meant as documentation, mainly for disambiguation for the human user [16]. They are not intended for reasoning purposes. Used in this sense, an OWL DL ontology (or pattern) endowed with these annotations will still be in OWL DL.

5 OPLa Usage Examples

In order to show how OPLa properties can be used for annotating ontology patterns, we apply them to some existing patterns and ontologies that are available at the `http://ontologydesignpatterns.org` portal. Figure 2 depicts the main classes and properties of the *Observation* pattern (prefix obs: which stands for `http://www.ontologydesignpatterns.org/cp/owl/observation.owl`). The dotted boxes with a URI on top represent an ontological collection. Everything within a box belongs to it (in OPLa terms, it is native to it). Dotted orange boxes represent restrictions, green parallelograms represent datatypes. The Observation pattern models the situations of observing objects and expressing such observations in terms of values assigned to a number of parameters. The classes Observation and Parameter belong to the pattern, which is expressed by the property *opla:isNativeTo*. The class Parameter is not defined in detail within this pattern, it could refer, e.g., to the concept formalized by a class belonging to the DOLCE+DnS UltraLite ontology (prefix dul: which stands for `http://www.ontologydesignpatterns.org/ont/dul/DUL.owl`). This is expressed by using the *opla:ofExternalType* annotation property. This simple triple allows us to explicitly state the ambiguity of a class for which we provide only a shallow or no formalization. This may have an impact when one aligns to our model through this concept, and this impact grows and affects the quality of Semantic Web ontologies as the alignments propagate in a Linked Data context. Finally, the Observation pattern specializes the Situation pattern (prefix sit: which stands for `http://www.ontologydesignpatterns.org/cp/owl/situation.owl`), which is indicated by means of the annotation property *opla:specializationOfPattern*. In summary, we use the following annotations (expressed in Turtle).

```
:Observation  opla:isNativeTo               obs: .
:Parameter    opla:isNativeTo               obs: ;
              opla:ofExternalType           opla:externalClass .
obs:          opla:specializationOfPattern  sit: .
```

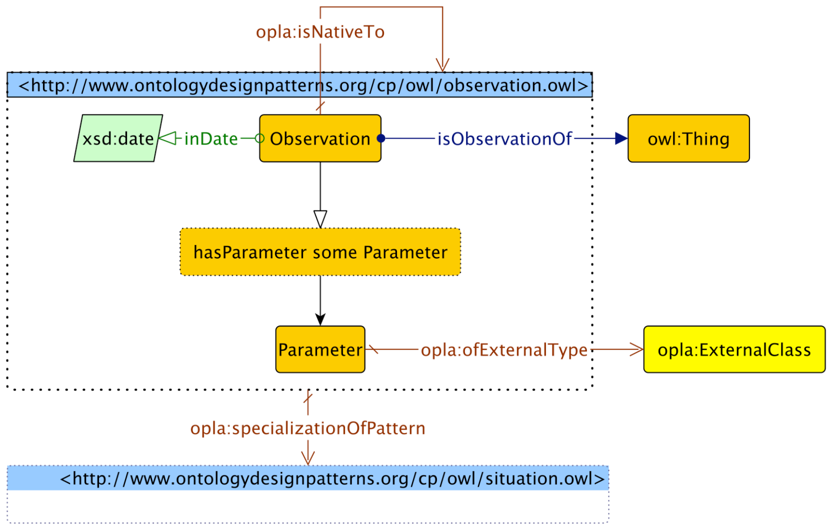
opla:isNativeTo
<http://www.ontologydesignpatterns.org/cp/owl/observation.owl>
xsd:date
inDate
Observation
isObservationOf
owl:Thing
hasParameter some Parameter
Parameter
opla:ofExternalType
opla:ExternalClass
opla:specializationOfPattern
<http://www.ontologydesignpatterns.org/cp/owl/situation.owl>

Fig. 2. The *Observation* pattern annotated with three OPLa properties: isNativeTo, ofExternalType, and specializationOfPattern.

As a second example, the Dolce+DnS UltraLite (DUL) ontology is annotated for indicating the patterns that it reuses as templates (although for the sake of space we only show five of them). To this end we use the annotation property opla:reusesPatternAsTemplate. The reused patterns are: *Situation* as mentioned before, *Description* (prefix des: which stands for `http://www.ontologydesignpatterns.org/cp/owl/description.owl`), *TimeInterval* (prefix ti: which stands for `http://www.ontologydesignpatterns.org/cp/owl/timeinterval.owl`), *Classification* (prefix cla: which stands for `http://www.ontologydesignpatterns.org/cp/owl/classification.owl`), and *AgentRole* (prefix ar: which stands for `http://www.ontologydesignpatterns.org/cp/owl/agentrole.owl`). More specifically, we include the following triples.

```
dul:  opla:reusesPatternAsTemplate  sit: .
dul:  opla:reusesPatternAsTemplate  des: .
dul:  opla:reusesPatternAsTemplate  cl: .
dul:  opla:reusesPatternAsTemplate  ti: .
dul:  opla:reusesPatternAsTemplate  ar: .
```

Finally, we show what a complete annotation for a pattern may look like. For this purpose, we use the *EventCore* pattern [21] for spatiotemporal events. The schema diagram is given in Figure 3, the dashed yellow boxes indicate external patterns.

There are several axioms to this pattern, as specified in [21]. For sake of simplicity, we use only a simple one, the range declaration for subEventOf, which

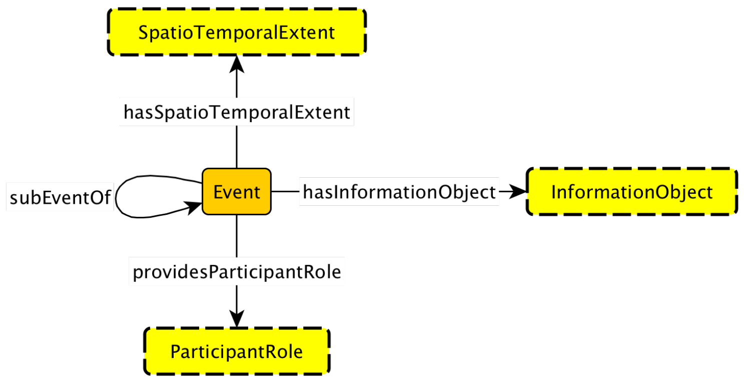

Fig. 3. Schema Diagram: Core Pattern for Events

is

$$\top \sqsubseteq \forall \text{subEventOf.Event}$$

and which can be expressed in OWL/RDF Turtle format as

```
ec:subEventOf  rdfs:range  ec:Event .
```

assuming ec: as namespace for the *EventCore* pattern.

As we see from the diagram, we have four properties and four classes. For each of these, we now record that they are native to *EventCore*, and for three classes we indicate that they are external classes.

```
ec:Event                 opla:isNativeTo      ec: .
ec:SpatioTemporalExtent  opla:isNativeTo      ec: ;
                         opla:ofExternalType  opla:externalClass .
ec:ParticipantRole       opla:isNativeTo      ec: ;
                         opla:ofExternalType  opla:externalClass .
ec:InformationObject     opla:isNativeTo      ec: ;
                         opla:ofExternalType  opla:externalClass .
ec:subEventOf               opla:isNativeTo   ec: .
ec:hasSpatioTemporalExtent  opla:isNativeTo   ec: .
ec:hasInformationObject     opla:isNativeTo   ec: .
ec:providesParticipantRole  opla:isNativeTo   ec: .
```

To express that the axiom given above is native to the pattern, we need to follow the OWL specification regarding the annotation of axioms, which in Turtle syntax requires reifying the axiom. We thus add the following.

```
[]  rdf:type                owl:Axiom ;
     owl:annotatedSource    ec:subEventOf ;
     owl:annotatedProperty  rdfs:range ;
     owl:annotatedTarget    ec:Event .
```

6 Related Work

Creating a framework to describe reusable solutions for problem solving is at least as old as Christopher Alexander's work on *design patterns* [1], which includes a template for describing them, which has been reused or adapted by the software engineering community [24,14,28,9], and eventually in ontology engineering [8,30,27,26,10,7,11], where they have been variously called *knowledge patterns* [8], *semantic patterns*[27], *ontology engineering patterns* [26,7], and finally *ontology design patterns* [10], as widely known nowadays.

The templates defined in those projects are eminently informal, following the bottom-up intuitiveness and practicality that Alexander promoted. For example (cf. [10] for a discussion), the dimensions of pattern description in software engineering have been fully inspired by the original Alexandrian ones [1]: given an `artifact type`, the pattern provides `examples` of it, its `context`, the `problem` addressed by the pattern, the involved `forces` (requirements and constraints), and a `solution` (or "approach"). In practice, what is addressed primarily by pattern templates is the ability of a pattern to deal with problems, constraints, needs, for a certain context, with examples of its advantages. The internal form, structure, nature, or content that the artifact should have is mainly addressed in the solution.

However, when using patterns in ontology design, the artifacts themselves lean to a substantial amount of combinatorial structure, typically provided by a logical representation, e.g., the adaptation of the Alexandrian template in [26] adds an OWL encoding that implements each approach suggested, while in [10] the authors admits multiple representations: an abstract encoding in a higher-order logic, a concrete implementation in a KR language, and, as further detailed in [11],[7] a simple vocabulary to annotate ontology design patterns directly with the Alexandrian dimensions represented in OWL, and basic relations between them, and is therefore a predecessor to the formal meta-model proposed herein.

This creates a difference for ontology patterns, which are not only practical problem solving schemas, but also *reusable components*. In this sense, they are closer to software libraries than software patterns. Such double nature of ontology design patterns make them sibling to ontology modules and reusable axiomatized theories, hence the reason why in this paper we tackle the relations between ontologies, modules, and patterns explicitly.

Another important terminological clarification should be made with respect to the literature: Alexander talks explicitly of a *pattern language* [2] as the result of composing, "networking" patterns, which tend to constitute constellations, collections, types, and dependencies among them. Of course, this is even more obvious in ontology design, where patterns are combined in order to obtain the best possible artifact, i.e., a qualitative ontology for a real use case.

[7] This annotation vocabulary, used to annotate most ontology design patterns from ontologydesignpatterns.org, is available at `http://www.ontologydesignpatterns.org/cp/owl/cpannotationschema.owl`

This sense inspired the *pattern languages* used in the conceptual modeling community (cf. [4]). In reference work [3] the authors also provide a complex UML framework to represent even the workflows, in which a pattern language can be used methodologically, with entry points, actions, control flows, etc.

In the ontology design community, especially for the Semantic Web, the compositional notion of *ontology pattern language* is well known, since patterns are typically collected into repositories, extracted from complex foundational or core ontologies, or organized into complex pattern frameworks. However, neither the term nor the conceptual modeling practice has been adopted, probably because Semantic Web developers are less keen to systematize their practices into large and complex methodologies. We notice that the original Alexandrian sense is less systematic and prescriptive than the one formalized, e.g., by the authors of [3].

In this paper, we call *pattern language* a specific ontology that represents the dimensions used to describe ontology patterns and their relations, while the term *pattern framework* is often used to talk about a composition of patterns. Finally, one more sense of *ontology pattern language* (OPL) is assumed by [18], and refers to a programmatic approach to deal with logical patterns in OWL. For more references, examples, discussions, and foundations, please refer to the ODP book [15].

7 Conclusions and Future Development

In this work, we have motivated and described a design pattern representation language, more specifically the OPLa ontology that makes use of OWL annotation properties. OPLa provides the means to describe the relations of classes, properties, and individuals to patterns and modules, thereby enabling the documentation of pattern and module usage during and after ontology engineering. Aside of acting as provenance records, these annotations also facilitate search for specific patterns, modules, and ontologies, and ease ontology alignment.

The work presented here is preliminary in the sense that benefits and shortcomings of OPLa can only be revealed by using it and by integrating OPLa into tool chains, e.g., that either support the annotations via a graphical user interface or by providing search capabilities. For instance, one could search for ontologies that use a common pattern or for classes that specialize a particular stub concept used (but not defined) in a pattern. As these stub concepts act as semantic hooks, they provide a natural point of contact for ontology alignment as well as a minimal interoperability fallback level for query federation more generally [19].

In the end, we suggest that the proposal presented herein be merely a stepping stone towards a more complex and sophisticated ontology or pattern language. E.g., extensions of our proposal could consist of patterns how to express complex relationships between OntologicalCollections. The next step, however, may be to work towards tool support for the simple proposal made herein.

Acknowledgements. The authors acknowledge support by the National Science Foundation under award 1440202 "Earthcube Building Blocks: Collaborative

Proposal: GeoLink – Leveraging Semantics and Linked Data for Data Sharing and Discovery in the Geosciences" and award 1540849 "EarthCube IA: Collaborative Proposal: Cross-Domain Observational Metadata Environmental Sensing Network (X-DOMES)".

References

1. Alexander, C.: The Timeless Way of Building. Oxford Press (1979)
2. Alexander, C., Ishikawa, S., Silverstein, M.: A Pattern Language. Oxford University Press (1977)
3. de Almeida Falbo, R., Barcellos, M.P., Ruy, F.B., Guizzardi, G., Guizzardi, R.S.S.: Ontology pattern languages. In: Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A., Presutti, V. (eds.) Ontology Engineering with Ontology Design Patterns – Foundations and Applications, Studies on the Semantic Web, vol. 25, pp. 133–159. IOS Press (2016)
4. de Almeida Falbo, R., Guizzardi, G., Gangemi, A., Presutti, V.: Ontology patterns: Clarifying concepts and terminology. In: WOP 2013, Workshop on Ontology and Semantic Web Patterns. Proceedings of the 4th Workshop on Ontology and Semantic Web Patterns co-located with 12th International Semantic Web Conference (ISWC 2013) Sydney, Australia, October 21, 2013. CEUR Workshop Proceedings, vol. 1188. CEUR-WS.org (2013)
5. Blomqvist, E., Hammar, K., Presutti, V.: Engineering ontologies with patterns – the eXtreme Design Methodology. In: Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A., Presutti, V. (eds.) Ontology Engineering with Ontology Design Patterns – Foundations and Applications, Studies on the Semantic Web, vol. 25, pp. 23–50. IOS Press (2016)
6. Blomqvist, E., Hitzler, P., Janowicz, K., Krisnadhi, A., Narock, T., Solanki, M.: Considerations regarding ontology design patterns. Semantic Web 7(1), 1–7 (2016)
7. Blomqvist, E., Sandkuhl, K.: Patterns in ontology engineering: Classification of ontology patterns. In: Chen, C.S., Filipe, J., Seruca, I., Cordeiro, J. (eds.) ICEIS (3). pp. 413–416 (2005)
8. Clark, P., Thompson, J., Porter, B.: Knowledge patterns. In: Cohn, A.G., Giunchiglia, F., Selman, B. (eds.) KR2000: Principles of Knowledge Representation and Reasoning. pp. 591–600. Morgan Kaufmann, San Francisco (2000)
9. Gamma, E., Helm, R., Johnson, R.E., Vlissides, J.: Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley (March 1995)
10. Gangemi, A.: Ontology design patterns for semantic web content. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) The Semantic Web – ISWC 2005, 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3729, pp. 262–276. Springer (2005)
11. Gangemi, A., Presutti, V.: Ontology Design Patterns. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies, 2nd Edition, pp. 221–243. Springer, Berlin, Germany (2009)
12. Hammar, K., Blomqvist, E., Carral, D., van Erp, M., Fokkens, A., Gangemi, A., van Hage, W.R., Hitzler, P., Janowicz, K., Karima, N., Krisnadhi, A., Narock, T., Segers, R., Solanki, M., Svátek, V.: Collected research questions concerning ontology design patterns. In: Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A., Presutti, V. (eds.) Ontology Engineering with Ontology Design Patterns –

Foundations and Applications, Studies on the Semantic Web, vol. 25, pp. 189–198. IOS Press (2016)

13. Hammar, K., Presutti, V.: Template-based content ODP instantiation. In: Proceedings of the Workshop on Ontology and Semantic Web Patterns (7th edition), Kobe, Japan, October 2016 (2017), to appear
14. Harrison, N.B., Avgeriou, P., Zdlin, U.: Using patterns to capture architectural decisions. Software 24(4), 38–45 (2007)
15. Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A., Presutti, V. (eds.): Ontology Engineering with Ontology Design Patterns – Foundations and Applications, Studies on the Semantic Web, vol. 25. IOS Press (2016)
16. Hitzler, P., Krisnadhi, A.: On the roles of logical axiomatizations for ontologies. In: Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A., Presutti, V. (eds.) Ontology Engineering with Ontology Design Patterns – Foundations and Applications, Studies on the Semantic Web, vol. 25, pp. 73–80. IOS Press (2016)
17. Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P.F., Rudolph, S. (eds.): OWL 2 Web Ontology Language Primer (Second Edition). W3C Recommendation (11 December 2012), http://www.w3.org/TR/owl2-primer/
18. Iannone, L., Rector, A.L., Stevens, R.: Embedding knowledge patterns into OWL. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E.P.B. (eds.) The Semantic Web: Research and Applications, 6th European Semantic Web Conference, ESWC 2009, Heraklion, Crete, Greece, May 31-June 4, 2009, Proceedings. Lecture Notes in Computer Science, vol. 5554, pp. 218–232. Springer (2009)
19. Janowicz, K.: Modeling ontology design patterns with domain experts-a view from the trenches. In: Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A., Presutti, V. (eds.) Ontology Engineering with Ontology Design Patterns – Foundations and Applications, Studies on the Semantic Web, vol. 25, pp. 233–243. IOS Press (2016)
20. Krisnadhi, A., Hitzler, P.: Modeling with ontology design patterns: Chess games as a worked example. In: Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A., Presutti, V. (eds.) Ontology Engineering with Ontology Design Patterns – Foundations and Applications, Studies on the Semantic Web, vol. 25, pp. 3–22. IOS Press, Amsterdam (2016)
21. Krisnadhi, A., Hitzler, P.: A core pattern for events. In: Proceedings of the Workshop on Ontology and Semantic Web Patterns (7th edition), Kobe, Japan, October 2016 (2017), to appear
22. Krisnadhi, A., Hu, Y., Janowicz, K., Hitzler, P., Arko, R.A., Carbotte, S., Chandler, C., Cheatham, M., Fils, D., Finin, T.W., Ji, P., Jones, M.B., Karima, N., Lehnert, K.A., Mickle, A., Narock, T.W., O'Brien, M., Raymond, L., Shepherd, A., Schildhauer, M., Wiebe, P.: The GeoLink Modular Oceanography Ontology. In: Arenas, M., Corcho, Ó., Simperl, E., Strohmaier, M., d'Aquin, M., Srinivas, K., Groth, P.T., Dumontier, M., Heflin, J., Thirunarayan, K., Staab, S. (eds.) The Semantic Web – ISWC 2015 – 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9367, pp. 301–309. Springer (2015)
23. Krisnadhi, A.A.: Ontology Pattern-Based Data Integration. Ph.D. thesis, Wright State University (2015)
24. Maplesden, D., Hosking, J.G., Grundy, J.C.: A visual language for design pattern modelling and instantiation. In: 2002 IEEE CS International Symposium on Human-Centric Computing Languages and Environments (HCC 2001), September 5-7, 2001 Stresa, Italy. pp. 338–339. IEEE Computer Society (2001)

25. Motik, B., Patel-Schneider, P.F., Parsia, B. (eds.): OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). W3C Recommendation 11 December 2012 (2012), available from http://www.w3.org/TR/owl2-syntax/
26. Semantic Web Best Practices and Deployment Working Group: Ontology Engineering and Patterns Task Force (OEP). http://www.w3.org/2001/sw/BestPractices/OEP/ (2004)
27. Svatek, V.: Design patterns for semantic web ontologies: Motivation and discussion. In: Abramowicz, W. (ed.) Proceedings of the 7th Conference on Business Information Systems, BIS 2004. Poznan (2004)
28. Tidwell, J.: Designing Interfaces: Patterns for Effective Interaction Design. O'Reilly (April 2007)
29. Tudorache, T., Nyulas, C., Noy, N.F., Musen, M.A.: WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the web. Semantic Web 4(1), 89–99 (2013)
30. Van Der Aalst, W., Ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow Patterns. Distributed and Parallel Databases 14, 5–51 (2003)