OPLaX: annotating ontology design patterns at conceptual and instance level*

Luigi Asprino[1], Valentina Anita Carriero[1], Christian Colonna[1], and Valentina Presutti[1,2]

[1] University of Bologna, Bologna, Italy
[2] ISTC-CNR, Rome, Italy

Abstract. In this paper, we present OPLaX, a language for annotating ontology design patterns (ODPs) in ontologies and knowledge graphs, which reuses and extends existing languages. This language allows an ontology designer to annotate ODPs implemented in ontologies, to relate these ODPs to the abstract modelling problems they are addressing (named conceptual components), and to link the ODPs with their instantiations in a knowledge graph (pattern instances). Moreover, we showcase its usefulness by means of 3 real-world use cases.

Keywords: ontology design patterns, patterns annotation, conceptual components, pattern instances, linked data visualization

1 Introduction

Ontology Design Patterns are increasingly widespread among ontologies and knowledge graphs. Pattern-based design has become a consolidated practice for guaranteeing good quality ontology engineering (see [3, 4]): it enables the modular design of ontologies, it favors the reuse of ontology design patterns and it streamlines the ontology building process. Ontologies are often used as moulds for constructing knowledge graphs (e.g. [7]), therefore ontology design patterns (ODPs) can provide meaningful views over the data of such knowledge graphs. The need of a shared language for annotating ontology design patterns used in ontologies and knowledge graphs has been recognized [6, 13], leading to some useful proposals, such as OPLa and the CP annotation schema [13, 11], and some tool support [15].

Our experience in ontology pattern discovery [1], pattern-based visualization of knowledge graphs [2] and the ArCo project [7] showed that at least two requirements are not yet addressed by the existing pattern annotation languages: (i) to indicate, at a more abstract level, a reference frame (i.e. a *conceptual component*) addressed by an annotated ODP (see the paragraph *Conceptual component* in Section 2); (ii) to annotate the data that instantiate an ODP. To address these requirements, we developed an extension of OPLa [13], called OPLaX (Ontology

Pattern Language eXtended). OPLaX provides ontology designers with a language for annotating ontology design patterns at both pattern, conceptual component, and instance level. OPLaX includes, extends and is aligned with OPLa and CP annotation schema, thus providing a novel and integrated model for annotating ontology design patterns in ontologies and knowledge graphs (KGs).

The rest of the paper is organised as follows. Section 2 motivates why such an extension is needed. Section 3 gives an overview over the related work. The proposed annotation schema is presented in Section 4. Section 5 describes three use cases demonstrating the usage of the proposed annotation model. Section 6 concludes the paper and outlines the future work.

2 Motivation

The annotation language presented in this paper can provide a basis for supporting different ontology engineering tasks: (i) an ontology designer may need to annotate ontology design patterns implemented in her ontology, or existing ontologies, in order to support ontology understanding, reuse and interoperability; (ii) an ontology designer may need to annotate an ontology (design pattern) with respect to all the *facts* that ontology (design pattern) addresses, regardless of specific OWL implementations as ODPs, enabling ontology understanding and comparison at a more abstract level; (iii) a knowledge graph engineer may need to annotate a knowledge graph, containing triples generated according to an ontology, with respect to the patterns implemented within the ontology, in order to e.g. visualize the KG with a modular and pattern-based view for easing its inspection (linked data visualization).

Pattern. A pattern-based ontology design is focused on the reuse (or creation, when needed) of ontology design patterns as small reusable components to integrate in an ontology, as e.g. in eXtreme Design methodology [5, 4]. Either an ontology is modelled following this approach since the beginning of the development, or an ontology designer may refactor her ontology by reusing existing ODPs. An ontology can also implicitly reuse patterns, e.g. by defining two relations `isPartOf` and `hasPart`, which are compatible with the *Part of*[3] ODP. Annotating patterns (re)used within an ontology eases the process of understanding and exploring an ontology e.g. for the purpose of ontology reuse: it makes explicit which groups of ontology entities are members of an ODP addressing a specific modelling issue, and it allows to represent relations (e.g. hierarchical) between different implemented ODPs. Moreover, annotations support the identification of ontology alignments, thus improving the interoperability between ontologies reusing the same patterns.

Conceptual component. An ontology design pattern provides an ontology designer with a small OWL ontology to reuse and integrate in her ontology for

[3] http://ontologydesignpatterns.org/wiki/Submissions:PartOf

addressing a specific modelling problem. Therefore, an ODP is a particular implementation of a modelling solution to a modelling problem. However, the same modelling problem could be addressed by different ODPs[4], i.e. different implementations. Let us consider the modelling problem "an object being located at a place": it can be implemented e.g. (i) as a binary relation `hasLocation` between an `Object` and a `Place`, or (ii) as an n-ary relation `Location` between the arguments `Time`, `Object` and `Place`. Therefore, (i) and (ii) are two ODPs that address, in different ways and with different levels of expressivity, the modelling problem of an object that is located somewhere. Two ontologies may reuse (i) and (ii), respectively: even if with different implementations, they address the same modelling problem. As proposed in [1], we can call these abstract relational structures, implemented by means of ODPs, *conceptual components*. An ontology can be seen as a composition of conceptual components. Annotating ontologies with conceptual components, and relating an OWL implementation to the conceptual component it implements, would ease the exploration of, and the interoperability between, ontologies at a more abstract level.

Pattern instance. Ideally, to each existing ontology corresponds one (or more than one) knowledge graph that contains triples that are modelled according to that ontology – and possibly other ontologies. The ontology may also (re)use ontology design patterns, thus the KG modelled according to the ontology will possibly contain instances of that pattern. A *pattern instance* is a collection of ABox triples, whose members are individuals and relations that comply with the ontology design pattern structure. For instance, given the pattern *Part of*[3], which defines two inverse properties `hasPart` and `isPartOf` with `Thing` as both domain and range, an instance of this pattern may be represented by the set of the 4 following triples: (i) `:whole :hasPart :part_1`, (ii) `:whole :hasPart :part_2`, (iii) `:part_1 :isPartOf :whole`, and (iv) `:part_2 :isPartOf :whole`. Annotating sets of instances with respect to the ODPs according to which they are represented, allows us to create new interesting applications, such as supporting a pattern-based exploration of KGs.

3 Related work

OPLa. In [13], the authors propose a simple and extendable language for representing information about ontology design patterns (ODPs) and the corresponding modelling process, through the use of OWL annotation properties. Some relevant competency questions addressed by OPLa are: (i) which patterns a module of an ontology is based on; (ii) which classes, properties, individuals and axioms belong to a pattern (or module); (iii) which modules an ontology consists of; (iv) is a pattern (or module) a specialization or generalization of a pattern (or module). Specifically, an entity that is an `Individual`, a `Property`,

[4] A relation between two ODPs (e.g. an ODP that is a specialization of another ODP) does not imply that the two ODPs implement the same conceptual component.

a `Class` or an `Axiom` is an `OntologicalEntity`. The property `isNativeTo` indicates that an ontological entity is member of an `OntologicalCollection` (see (ii)). This concept is further specified in `Ontology`, `Module` (intended as an ontology part capturing a conceptual area of the domain) and `Pattern`. A module can be native to an ontology (see (iii)), and an ontological collection can be annotated with `reusesPatternAsTemplate` for indicating the reused ODPs (see (i)). Finally, relations between patterns and modules are expressed via properties such as `hasRelatedPattern`, `generalizationOfPattern`, or `specializationOfModule` (see (iv)).

CP annotation schema. The *ODP Portal*[5] is a repository for collecting Ontology Design Patterns. A Content Pattern, i.e. an ODP addressing content modelling problems, needs to be annotated with the *CP annotation schema*[6] in order to be submitted to the Portal. This schema defines a set of OWL annotation properties that feed the information fields of each catalogue entry and that can be exploited by Semantic Web applications [10], and allow an ontology designer to describe an ODP by specifying: (i) its intent, i.e. the generic scope addressed by the pattern; (ii) the requirement(s) the pattern provides a solution for; (iii) scenarios, as examples of instantiation of the pattern; (iv) its consequences, as benefits or possible trade-offs when using the pattern; (v) relations between patterns (specialization, generalization, componency); (vi) unit tests to evaluate it against a requirement-based task; (vii) ontologies or concept schemas which the pattern was extracted or reengineered from.

The current versions of OPLa and CP annotation schema are not able to represent the relations between a pattern and its abstract counterpart (conceptual component), and between a pattern and its instances.

[12] proposes a reorganisation of OPLa and CP annotation schema, with some changes and extensions, into 3 namespaces[7]: `opla-core` for storing the OPLa original annotations; `opla-cp`, adapted from the CP annotation schema; `opla-sd`, with new annotation properties possibly needed by tools supporting modular graphical ontology modelling (coordinates of a node in a schema diagram).

GO-FOR. [14] introduces GO-FOR, a Goal-Oriented Framework for Ontology Reuse, which aims at supporting a pattern-based and goal-oriented reuse of ontologies. Its basic element is a goal-oriented ontology pattern (GOOP): an ontology fragment bound to a goal, intended as the scope addressed by the pattern. Existing GOOPs are stored in a dedicated repository integrated in the GOOP-HUB[8]. Based on their goals, GOOPs can be related by *part-of* relationships. The GOOP OWL metamodel[9], besides deriving from OWL the `Class`, `ObjectProperty` and `DatatypeProperty` for representing the constructs that a GOOP can consist of, defines the class `Goal`, which is further specified by `AtomicGoal`

[5] http://www.ontologydesignpatterns.org

[6] http://www.ontologydesignpatterns.org/schemas/cpannotationschema.owl

[7] https://github.com/cogan-shimizu/Extended-OPLa

[8] https://github.com/nemo-ufes/goophub

[9] https://github.com/nemo-ufes/goophub/blob/master/src/main/resources/goop-meta-model.owl

and `ComplexGoal`. A complex goal is composed of other goals by either `OR_decomposition` (at least one subgoal needs to be satisfied) or `AND_decomposition` (all subgoals are to be satisfied). Moreover, a goal is related to the `Actor` that wants to achieve it.
The concept of *goal* of a pattern in GO-FOR is similar to the concept of *intent* in the CP annotation schema. While GOOP metamodel allows the pattern designer to specify subgoals of complex goals, thus determining *part-of* relations between patterns, in OPLa and CP annotation schema the composition, specialization and generalization relations are expressed at the level of patterns. Search for patterns in GOOPR can be based on goals and specific actors (e.g. doctor, researcher), while the patterns of the ODP Portal are grouped based on their domain (e.g. multimedia, time). Like the previous ones, this metamodel focuses only on the pattern level.

OTTR. [16] presents the Reasonable Ontology Templates (OTTRs): OWL ontology macros able to represent ontology design patterns. By defining OTTRs, it is possible to formalize and instantiate ontology design patterns. An OTTR T is a parametrised ontology that can be instantiated providing arguments that fit the parameters of the template. T is formalised as a knowledge base O_T together with a list of parameters $(p_1, ..., p_n)$ of concepts, roles or individuals from O_T. Given a list $(q_1, ..., q_n)$ of constants, concepts or role expressions called arguments, $T(q_1, ..., q_n)$ represents a template instance, as a shorthand to represent an occurrence of a pattern. Defined OTTRs can also be specialised for specific domains. Moreover, specific templates can be defined for generating instances of a modelled pattern. As OTTRs are expressed in a specific syntax (stOTTR), the authors developed a tool[10] that converts OTTR templates into regular OWL ontologies. These templates aim at automatizing ODP-based ontology engineering and supporting interoperability between ontologies using the same or related templates. Moreover, it uses the concept of pattern instance that we introduce the extension of OPLa ontology. As a drawback, defining and maintaining templates may result to be expensive, and to hardly support changes in already defined ODPs.

4 OPLaX

OPLaX[11,12] (Ontology Pattern Language eXtended) aims at providing the ontology designers with a language for annotating ontology design patterns at both pattern, conceptual component, and pattern instance level (see Figure 1). OPLaX widely reuses (and is aligned with) OPLa (plain classes and properties in the figure) and CP annotation schema (properties in italics). It introduces some changes with respect to these two existing models, and integrates them with specific classes and properties (bold classes and properties) addressing the annotations related to conceptual components and pattern instances.

[10] https://gitlab.com/ottr/lutra/lutra
[11] https://w3id.org/OPLaX/
[12] https://github.com/stlab-istc-cnr/OPLaX

Pattern level. Classes and properties reused from OPLa allow the ontology designer to assign specific types to different `:OntologicalCollection`s (`:Module`, `:Ontology`, `:Pattern`), to relate them with the ontological entities that are their members (`:isNativeTo`) and to specify componency (`:componentOfPattern/Module`), derivation (`:derivedFromPattern/Module`), specialization (`:specializationOfPattern/Module`), and generalization (`:generalizationOfPattern/Module`) relations between ontological collections of the same or different types. Moreover, it can be annotated that an `:Individual`, a `:Class` or a `:Property` is used within a pattern, even if it is out of the scope of that particular pattern (`:ofExternalType`). `:reusesAsTemplate` annotates an ontological collection with other ontological collections reused as templates, e.g. an ontology reusing multiple patterns as templates. Additional properties reengineered from CP annotation schema allows to better define the scenario (`:addressesScenario`), the intent (`:hasIntent`), the requirement(s) (`:coversRequirement`), the consequence (`:hasConsequence`) and the competency question(s) (`:hasCompetencyQuestion`) of an ontological collection, and to annotate it with possible unit tests (`:hasUnitTest`). Moreover, it is possible to make it explicit that an ontological collection is `:extractedFrom` another ontological collection, e.g. a pattern that has been deeply or partially cloned by a reference ontology.

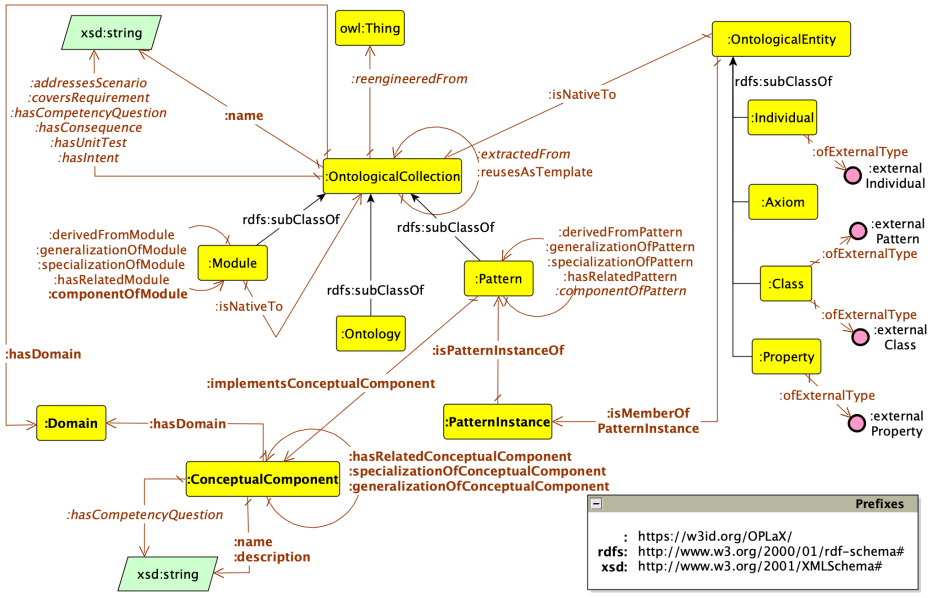


Fig. 1: OPLaX, the Ontology Pattern Language eXtended.

Conceptual component level. In OPLaX, a `:Pattern` is related to the `:ConceptualComponent` it implements by the annotation property `:implements ConceptualComponent`. Different conceptual components can be related to each other (`:hasRelatedConceptualComponent`): e.g. the conceptual component "a cultural property being located at a cultural site" would be a specialization of the more general conceptual component "an object being located at a place" (`:specializationOfConceptualComponent`), and the other way around (`:generalizationOfConceptualComponent`). A conceptual component can be annotated with its `:name` (e.g. *locating*), and a more detailed `:description`. Moreover, a conceptual component answers to a number of competency questions (`:hasCompetencyQuestion`), e.g. *where is an object?* for the component *locating*. Finally, a conceptual component has a `:Domain` (`:hasDomain`): for instance, a conceptual component *Paper award* and a conceptual component *Submitting paper* would be in the *Conference* domain.

Pattern instance level. An instance of a `:Pattern`, i.e. an entity of type `:PatternInstance`, is related to the pattern it instantiates by the annotation property `:isPatternInstanceOf`. All the individuals that are members of a pattern instance are annotated with the property `:isMemberOfPatternInstance`, so that, by retrieving all individuals related to a pattern instance by means of this property, it is possible to have the boundary of the pattern instance itself.

5 Use cases

In this section, we discuss three real-world use cases in which OPLaX has been used in practice and has proved useful.

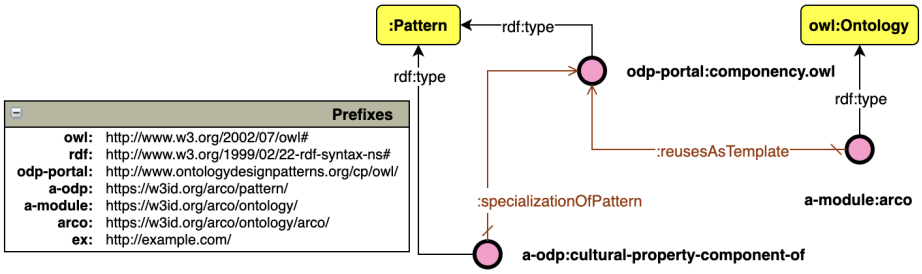
5.1 ArCo ontology network

ArCo[13] (Architecture of Knowledge) is the Italian cultural heritage (CH) knowledge graph (KG), consisting of a network of ontologies and facts on Italian cultural properties, based on the official General Catalogue of the Italian Ministry of Culture (MiC) [7, 9]. ArCo knowledge graph has been developed by using the pattern-based eXtreme Design (XD) ontology engineering methodology [5, 4]. After the requirements collection in the form of *user stories* – that are sets of sentences describing by example the facts that the resulting KG should encode – the ontology design team derives one or more *competency questions* (CQs) from a generalisation of each user story. CQs are the natural language counterpart of the queries that we want to answer against the KG, and guide the selection of the ODPs to reuse: the ontology designer tries to match one, or a set of, CQ(s) to existing ODPs.

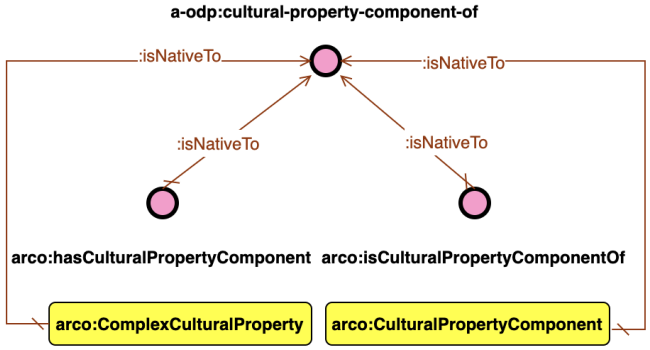
ArCo version 1.0 is composed of 7 ontology modules, which reuse and specialize 12 different ODPs. ArCo directly reuses, i.e. it directly embeds ontology enti-

[13] https://w3id.org/arco

ties in the local ontology, only two ontologies that are considered reference standards by the Italian Government and the development of which involves ArCo's team. Instead, it indirectly reuses other ontologies and ontology design patterns: these are used as templates, i.e. they are reproduced (and possibly extended) in the local ontology, and aligned with `rdfs:subClassOf/subPropertyOf` and `owl:equivalentClass/equivalentProperty` axioms [8]. However, alignment axioms allow to support interoperability by making it explicit a correspondence between single ontology entities.



(a) The property `:reusesAsTemplate` relates the *arco* module to the *Componency* ODP reused over the module. The property `:specializationOfPattern` expresses the specialization relation between the pattern *Cultural Property Component of* implemented in the module and the ODP *Componency*.



(b) The annotation property `:isNativeTo` relates the object properties and the classes of the *Cultural Property Component of* ODP to the ODP itself.

Fig. 2: An example of a specialized ODP annotated with OPLaX.

OPLaX allows an ontology designer to annotate all the ontology entities that are members of an ODP, and to generate correspondences between different ontologies at a pattern level. An example of ODP specialization and annotation through OPLaX in ArCo is shown in Figure 2. Over the *arco* ontology module, the ODP *Componency*[14] is indirectly reused from the ODP Portal[5]: the module is there-

[14] http://ontologydesignpatterns.org/wiki/Submissions:Componency

fore annotated with the property `:reusesAsTemplate` for representing the reuse of the pattern. Specifically, the ArCo's pattern *Cultural Property Component of*[15] is a specialization of the pattern *Componency*, since it represents the componency relation between a complex cultural property and its components, hence it is annotated with the property `:specializationOfPattern` (see Figure 2a). For expressing that specific properties (e.g. `arco:hasCulturalPropertyComponent`) and classes (e.g. `arco:ComplexCulturalProperty`) implemented in the module belong to this specialized ODP, the annotation property `:isNativeTo` is used (as in Figure 2b).

5.2 Conceptual Components and ODPs catalogue from a corpus of ontologies

[1] presents a method able to extract conceptual components (CCs) from multiple ontologies and the observed ontology design patterns implementing them. This method combines community detection, word sense disambiguation, frame detection, and clustering techniques. After a preprocessing step on the ontology corpus, community detection is run on each ontology, splitting it into dense communities; then, virtual documents, as bag of words disambiguated and enriched with frames, are generated from each community, and a clustering algorithm clusters all communities of the corpus in similar clusters, where each cluster represents a possible conceptual component, that is automatically given a representative name. As a result, each ontology is split into possible ontology design patterns[16], and each ontology design pattern is related to a conceptual component. Different ODPs extracted from the same or from different ontologies will eventually be linked to the same conceptual component, meaning that they are addressing the same modelling issue.

Relying on this method and starting from a corpus of ontologies, it is possible to build a resource, a *catalogue of conceptual components and observed ODPs*, that connects and organises ontologies, conceptual components and ODPs: each conceptual component in the catalogue is linked to its associated ODPs within the ontologies, thus the ontologies are classified based on the conceptual components that they implement.
Let us take as an example the catalogue generated from a corpus of 43 ontologies on Cultural Heritage[17,18]. The conceptual component named (`:name`) *Event* (Figure 3), which represents the general modelling issue of an event, is implemented by 21 observed ontology design patterns, coming from 13 distinct ontologies. Each ontology design pattern is thus related to the CC *Event* with the property `:implementsConceptualComponent`. These patterns implement the general component at different levels of specialization, thus addressing different

[15] https://w3id.org/arco/pattern/cultural-property-component-of

[16] In this context, the notion of ODP has been relaxed: adopted modelling solutions that can be observed in existing ontologies, regardless their correctness or quality level.

[17] https://stlab-istc-cnr.github.io/cc-and-odps-catalogue/

[18] See [1] for more details on the corpus.

and specific intents: for instance, a pattern extracted from the Europeana Data Model (see the ODP at the top of Figure 3) models the general concept of an `Event` that `happenedAt` some `Place`, and `occurredAt` a certain `TimeSpan`, while Cultural-ON (see the ODP at the bottom of Figure 3) focuses on a specific type of event, that is `CulturalEvents`, which involve one or more cultural entities. This conceptual component is also given a description (`:description`), that, in this specific case, is generated by concatenating all terms representing its patterns. Moreover, it is associated with a manually generated generic competency question "What happened?" by means of the property `:hasCompetencyQuestion`. Different conceptual components are organised as a hierarchical network, that is based on the inheritance relations between the frames detected from the virtual documents of the patterns. For example, in Figure 3 the CC *Event* is related with the property `:generalizationOfConceptualComponent` to the CC *Intentionally act*.

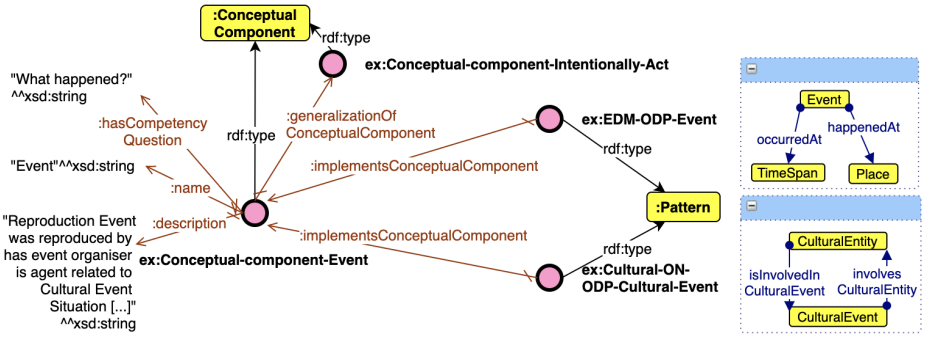


Fig. 3: An example of a conceptual component annotated with OPLaX. The property `:implementsConceptualComponent` relates two ODPs to the conceptual component *Event* they implement. Through `:generalizationOfConceptualComponent` *Event* is related to the more specific CC *Intentionally act*.

5.3 Pattern-based visualization of knowledge graphs

[2] describes a new approach to Knowledge Graph visualization based on ontology design patterns. This approach relies on ODPs as first-class citizens to explore KGs, thus providing thematic paths that guide the exploration and interaction with the KG. Moreover, the collection of the patterns instantiated in a KG is useful to concisely summarize its content. A visual frame (an intuitive standard visualization) is associated with an ODP, such that every time an ODP is used in ontology modelling, data can be visualized by means of a reusable visual frame. The described approach is divided into three levels of exploration and interaction. At the first level (*ODP level*), the user sees the ODPs the KG is shaped by and the most important concepts, namely the key concepts. At

the second level (*exploration level*), all the instances of a specific pattern are displayed in the user interface and can be filtered. The third level (*visualization level*) presents a single instance of an ODP.
The tool developed[19] by [2] relies on OPLaX annotations. In particular, at the *ODP level*, OPLaX annotations related to the pattern level are used to display the patterns in the graphical user interface. Relations between patterns are also represented: specialization (`:specializationOfPattern`) and composition (`:componentOfPattern`). The relation between a pattern and the key concept(s) belonging to that pattern is annotated through the `:isNativeTo` property. At the *exploration level*, the tool relies on the annotation property `:isPatternInstanceOf`, between a pattern instance and a pattern, to retrieve all the instances of a pattern. The user can then browse the list of instances and select them using predefined filters. The visualization of a specific pattern instance (*visualization level*) is made possible by using the property `:isMemberOfPatternInstance`, which allows to collect all the data instantiating a pattern and to present them by means of a graphical component associated with the given pattern.
ODP instances identify the meaningful neighborhood for each entity to be visualized; in other words, a visualization tool can use the instance level annotations to identify meaningful subsets of entities of a KG to be visualized together. Interestingly, the meaningful neighborhood of each entity is delimited by the instance level annotations, thus avoiding the use of query templates.

As in Figure 4, `ex:cultural-property-component-of-instance-a` is an instance of the ArCo's ODP *Cultural Property Component of*, thus it is annotated with `:isPatternInstanceOf`. The entities that are members of this pattern instance – all annotated by means of the property `:isMemberOfPatternInstance` – are the complex cultural property (`ex:cultural-property-b`) and its two components, i.e. `ex:cultural-property-component-c` and `ex:cultural-property-component-d`.

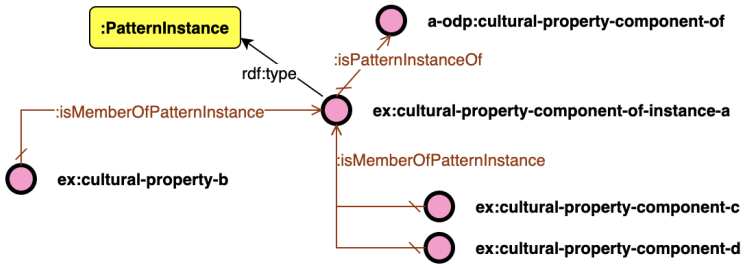


Fig. 4: An example of a Pattern Instance annotated with OPLaX. The property `:isPatternInstanceOf` relates an instance of a pattern with the pattern *Componency* itself. The property `:isMemberOfPatternInstance` relates the individuals to the pattern instance they are member of.

[19] https://github.com/ODPReactor

6 Conclusion and future work

In this paper, we proposed a language for annotating ontology design patterns (ODPs) at three levels: (i) the level of the pattern as it is implemented in an ontology, (ii) the level of the abstract conceptual component that can be implemented by different ODPs, (iii) the level of an instance of a pattern in a knowledge graph. This language reuses (is aligned with) and extends the OPLa annotation language and the CP annotation schema. We showcased the language and demonstrated its usefulness by discussing three real-world use cases.

In our future work, we aim at formally evaluating the proposed annotation language with respect to existing use cases, such as the ones presented in Section 5, in order to identify possible missing classes and properties to integrate (e.g. verifying competency questions coverage and inferences correctness).
Moreover, the annotations provided by OPLaX can be used to refine existing tools/plugins or to create new ones, aiming at supporting ontology developers in their pattern-based ontology engineering activities. Indeed, we plan to develop a tool for automatically annotating, based on OPLaX, ontology design patterns, pattern instances and conceptual components implemented in ontologies and knowledge graphs.

Acknowledgements. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101004746.

References

[1] Luigi Asprino et al. *Extraction of common conceptual components from multiple ontologies*. 2021. arXiv: 2106.12831 [cs.AI].

[2] Luigi Asprino et al. *Pattern-based Visualization of Knowledge Graphs*. 2021. arXiv: 2106.12857 [cs.HC].

[3] Eva Blomqvist, Aldo Gangemi, and Valentina Presutti. "Experiments on pattern-based ontology design". In: *Proceedings of the 5th International Conference on Knowledge Capture (K-CAP 2009), September 1-4, 2009, Redondo Beach, California, USA*. Ed. by Yolanda Gil and Natasha Fridman Noy. ACM, 2009, pp. 41–48.

[4] Eva Blomqvist, Karl Hammar, and Valentina Presutti. "Engineering Ontologies with Patterns - The eXtreme Design Methodology". In: *Ontology Engineering with Ontology Design Patterns - Foundations and Applications*. Vol. 25. Studies on the Semantic Web. Amsterdam: IOS Press, 2016.

[5] Eva Blomqvist et al. "Experimenting with eXtreme Design". In: *Knowledge Engineering and Management by the Masses. EKAW 2010* (Lisbon, Portugal). Vol. 6317. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2010, pp. 120–134.

[6] Eva Blomqvist et al. "Considerations regarding Ontology Design Patterns". In: *Semantic Web* 7.1 (2016), pp. 1–7.

[7] Valentina Anita Carriero et al. "ArCo: The Italian Cultural Heritage Knowledge Graph". In: *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part II*. Vol. 11779. Lecture Notes in Computer Science. Springer, 2019, pp. 36–52.

[8] Valentina Anita Carriero et al. "The Landscape of Ontology Reuse Approaches". In: *Applications and Practices in Ontology Design, Extraction, and Reasoning*. Vol. 49. Studies on the Semantic Web. Amsterdam: IOS Press, 2020, pp. 21–38.

[9] Valentina Anita Carriero et al. "Pattern-based design applied to cultural heritage knowledge graphs". In: *Semantic Web* 12.2 (2021), pp. 313–357.

[10] Aldo Gangemi and Valentina Presutti. "Ontology Design Patterns". In: *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, 2009, pp. 221–243.

[11] Quinn Hirt, Cogan Shimizu, and Pascal Hitzler. "Extensions to the Ontology Design Pattern Representation Language". In: *Proceedings of the 10th Workshop on Ontology Design and Patterns (WOP 2019) co-located with 18th International Semantic Web Conference (ISWC 2019), Auckland, New Zealand, October 27, 2019*. Vol. 2459. CEUR Workshop Proceedings. CEUR-WS.org, 2019, pp. 76–75.

[12] Quinn Hirt, Cogan Shimizu, and Pascal Hitzler. "Extensions to the Ontology Design Pattern Representation Language". In: *Proceedings of the 10th Workshop on Ontology Design and Patterns (WOP 2019) co-located with 18th International Semantic Web Conference (ISWC 2019), Auckland, New Zealand, October 27, 2019*. Vol. 2459. CEUR Workshop Proceedings. CEUR-WS.org, 2019, pp. 76–75.

[13] Pascal Hitzler et al. "Towards a Simple but Useful Ontology Design Pattern Representation Language". In: *Proceedings of the 8th Workshop on Ontology Design and Patterns (WOP 2017) co-located with the 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 21, 2017*. Vol. 2043. CEUR Workshop Proceedings. CEUR-WS.org, 2017.

[14] Cássio Reginato et al. "GO-FOR: A Goal-Oriented Framework for Ontology Reuse". In: *2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)*. IEEE. 2019, pp. 99–106.

[15] Cogan Shimizu, Quinn Hirt, and Pascal Hitzler. "A Protégé Plug-In for Annotating OWL Ontologies with OPLa". In: *The Semantic Web: ESWC 2018 Satellite Events - ESWC 2018 Satellite Events, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers*. Ed. by Aldo Gangemi et al. Vol. 11155. Lecture Notes in Computer Science. Springer, 2018, pp. 23–27.

[16] Martin G Skjæveland et al. "Pattern-based ontology design and instantiation with reasonable ontology templates". In: *A Higher-Level View of Ontological Modeling* (2019), p. 69.