Template-Based Content ODP Instantiation[⋆]

Karl Hammar[1,2] and Valentina Presutti[3]

[1] Department of Computer Science and Informatics, Jönköping University, Sweden
[2] Department of Computer and Information Science, Linköping University, Sweden
karl.hammar@ju.se
[3] Semantic Technology Lab, ISTC-CNR, Italy
valentina.presutti@istc.cnr.it

Abstract. Content Ontology Design Patterns (CODPs) are typically instantiated into a target ontology or ontology module through a process of specialisation of CODP entities. We find, from experiences in three projects, that this approach leads to ontologies that are unintuitive to some non-expert ontologists. An approach where CODPs are used as templates can be more suitable when constructing ontologies to be used or modified by such users, and we propose a method for such template-based ODP instantiation. We evaluate this method with positive results, and describe a tool that supports the use of the proposed method.

Keywords: Content Ontology Design Pattern, eXtreme Design, Template, Usability

1 Introduction

Content Ontology Design Patterns (CODPs) were introduced by Gangemi [7] and Blomqvist & Sandkuhl [2] in 2005, as a means of simplifying ontology development by packaging known good practices into reusable blocks of ontology functionality. Adopting CODPs and the associated eXtreme Design (hereafter XD) Ontology Engineering methodology allow developers to reuse these known good solutions to modelling problems, and thus enables developers to work in a more efficient manner.

The idea of using CODPs to support Ontology Engineering has subsequently gained traction within the academic community, as evidenced by the Workshop on Ontology Patterns series of workshops held in conjunction with the International Semantic Web Conference[4], and the recent formation of the Association for Ontology Design & Patterns (ODPA)[5]. Studies also indicate that CODP usage can help lower the number of modelling errors and inconsistencies in ontologies, and that they are by the users perceived as useful and helpful [1,3].

[⋆] This work was partly financed by the EU FP7 project Visual Analytics for Sensemaking in Criminal Intelligence Analysis (VALCRI) under grant number FP7-SEC-2013-608142.

[4] `http://ontologydesignpatterns.org/wiki/WOP:Main`

[5] `http://ontologydesignpatterns.org/wiki/ODPA`

A key step in employing the XD method lies in the instantiation and adaptation of a CODP building block (essentially a generic reusable mini-ontology) into an ontology module fit for solving the concrete modelling problem at hand. Per established practice this step typically consists of importing the building block into the target module using the *owl:imports* predicate and then, based on the existing classes and properties within that building block, specialising it by creating subclasses and subproperties that encapsulate domain semantics [11]. In the following, this way of instantiating CODPs will be referred to as *Specialisation-based.*

The Specialisation-based approach is well-known and understood by CODP researchers. However, the authors have in several recent projects noted sets of users who have different preferences regarding how CODPs should be reused – for these people the use of the Specialisation-based approach reduces the usability of the resulting CODP-based ontologies. An alternative approach to CODP instantiation is that of using the CODP building block as a template that is instantiated into the target ontology module by way of copying and renaming its constituent classes and properties [11]. We will call this *Template-based* CODP instantiation. The Template-based approach carries with it advantages and disadvantages, that may make it more or less palatable depending on the intended XD development context and ontology users. This paper reports on three projects in which CODPs have been used, the experiences from which motivate the need for better supporting the Template-based approach. It also reports an initial evaluation comparing the Specialisation-based and Template-based approaches based on resulting ontology usability and modifiability, and introduces newly developed tooling supporting Template-based CODP instantiation.

The rest of this paper is organised as follows: Section 2 introduces the foundations of CODPs and CODP usage methods. Section 3 describes the authors' experiences of applying XD and the Specialisation-based CODP instantiation approach that motivate this work, and presents the alternate Template-based approach. Section 4 describes an evaluation of the Template-based approach to CODP instantiation. Section 5 presents the authors' recent work on a tool to support this new approach, and Section 6 concludes the paper.

2 Related Work

The following section introduces the eXtreme Design method for CODP-based ontology construction, and two alternative methods to CODP formalisation and use.

2.1 Content Pattern use with XD

eXtreme Design (XD) is defined as *"a family of methods and associated tools, based on the application, exploitation, and definition of Ontology Design Patterns (ODPs) for solving ontology development issues"* [12, p. 83]. In XD, CODPs are essentially seen as mini-ontologies that fulfill certain criteria: they are small and autonomous, they are expressed in OWL, they are non-trivial in complexity (i.e., not only single classes or properties), they are cognitively relevant (i.e., intuitive to domain experts), they encode best practices (i.e., are both of high quality and reusable), etc [11].

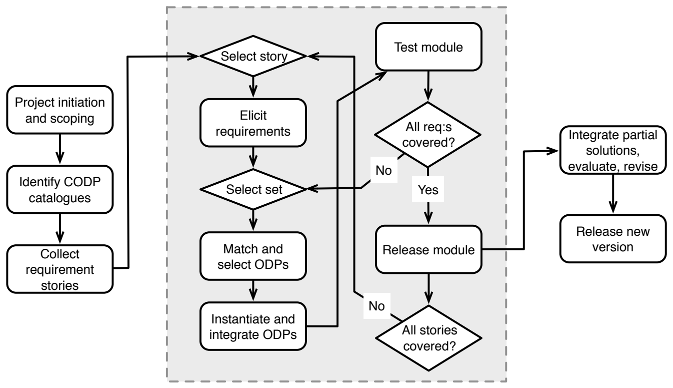


Fig. 1. XD Workflow (adapted from [12])

The XD method consists of a number of tasks, as illustrated in Figure 1. The first three tasks deal with establishing a project context, identifying a set of candidate CODP portals on the Web, and collecting initial requirements in the form of a prioritised list of user stories. The subsequent tasks are performed in developer pairs (these tasks are in Figure 1 enclosed in a grey box). A pair begins by selecting the top prioritised user story that has not yet been handled, and transform that story into more formally expressed requirements in the form of competency questions, contextual statements, and reasoning requirements. They then select a subset of these formal requirements relating to the same problem area or set of domain concepts, and try to find an existing CODP that can be used to fulfill the requirements. There may be multiple such CODPs found, in which case the development pair selects one based on their understanding of the problem domain and the modelling consequences associated with each matching pattern. The selected pattern is instantiated and integrated into the ontology module under development. The module is tested against the selected requirements, to ensure that it covers them properly. The next set of requirements from the same user story is then selected, a pattern found, adapted, and integrated, and so on. Once all requirements associated within one user story have been handled, the module is released and integrated with the prior ontology version (if one exists already) to form a new ontology release[12,11].

XD defines a set of operations on CODPs, e.g. *CODP import* (using the native owl:imports predicate), *CODP specialisation* (narrowing the semantics of the CODP by constructing new classes and properties aligned to CODP entities using rdfs:subClassOf and rdfs:subPropertyOf), and *CODP composition* (aligning two CODPs or CODP-based ontology modules using subsumption, equivalence, or disjointness axioms). The CODP instantiation and integration step in Figure 1 is in XD assumed to use these operations, but an *CODP cloning*-based approach is also mentioned, along with certain advantages (decreased import closure in the resulting ontology module) and disadvantages (lack of a shared CODP language leading to reduced support for alignment against other CODP-based ontologies) [11].

2.2 Application by Extension or Analogy

Falbo et al. [5] argue, and Ruy et al. [13] extend upon this argument, for a different perspective on CODPs and their use. Per this view, CODPs[6] can be divided into those that are extracted from ontologies covering foundational concepts (FOPs), and those that are extracted from domain-related ontologies (DROPs). Neither FOPs nor DROPs have language-specific implementations (e.g., OWL building blocks) – rather, they can be compared to Analysis Patterns [6], in that they attempt to solve a particular modeling issue in a reusable manner, regardless of technology stack used. Per this perspective, FOPs would typically be reused *by analogy*, while DROPs would typically be reused *by extension*. The former is analogous to Template-based CODP instantiation as discussed in this paper, whereas the latter is analogous to Specialisation-based CODP instantiation.

While [5] and [13] exemplify both of these types of CODP use, they do not evaluate the consequences of either approach, nor do they propose any specific method including concrete steps a person (or machine) should take to perform either. Further, we are uncertain as to whether the *FOP-analogy* and *DROP-extension* pairing necessarily holds in all cases – in fact, we think that in many cases it would for interoperability purposes be quite useful to extend foundational concepts – while cloning (i.e., reuse by analogy) may be very useful when adapting a domain-specific CODP to a related domain.

2.3 Encoding CODPs with OPPL

Yet another approach to formalising and using CODPs was developed in the CO-ODE project. This approach builds on the Ontology Pre-Processing Language (hereafter OPPL), a macro language initially designed to simplify rapid transformation of large ontologies [4]. The OPPL macro engine can add and remove ontology entities and axioms based on variables that are selected from and conditions that are evaluated against the existing ontology - in this way changes to ontology structure can easily be performed repeatedly with precision.

[6] The cited works argue that ontology patterns that include concepts or content are disjoint from patterns that concern design issues, so they prefer the use of the term *Ontology Conceptual Pattern*, rather than *Content Ontology Design Pattern* – but for the sake of not confusing the reader unduly, we will in this paper stick with the abbreviation CODP.

The same technique has been extended to formalise CODPs as OPPL macros[10]. These CODP macros contain unbound variables that the ontology developer fills with existing or new ontology entities before executing the OPPL engine that instantiates the CODP structures into the target ontology. Additionally, the CODP version of the OPPL language and tooling adds features and syntax required to better support CODP use cases, including user-friendly textual representation of CODP macros, syntax allowing macros to call one another, and annotation properties to embed metadata about CODP macro usage in target ontologies.

The OPPL-based approach to CODP use is technically impressive and from a feature perspective it would likely complement the above discussed XD method very well. All the same, OPPL-based CODP use has seen limited uptake in practice. We speculate that the reasons for this includes some drawbacks that the use of OPPL entails, namely the need to learn yet another language in addition to RDFS/OWL and the lack of tool support for developing and maintaining OPPL CODPs.

3 Template-Based CODP Instantiation

The following section recounts the authors' experiences of applying CODPs in projects which motivate thinking about CODP instantiation in a different way, and describes an approach to CODP instantiation based on these experiences.

3.1 Motivation: Experiences of Applying XD

The authors have recently used XD and CODPs in three ontology engineering projects involving developers new to ontologies or CODPs. These users can be characterised by a high degree of technical skill and in the cases of low or no experience of ontologies, they still had extensive experience of other types of conceptual modelling. A recurring theme in all three cases is the difficulty that these users have in understanding or agreeing with CODP-level entities that are imported into their models.

VALCRI Project VALCRI[7] intends to develop a system providing Visual Analytics capabilities for law enforcement analysts, supporting investigative and intelligence analysis. Ontologies are used for data integration purposes. A key requirement on those ontologies is that they need to be easily understandable by software developers. Another key requirement is that the developed system and its ontologies should be easy to modify for deployment in different contexts.

The initial versions of the ontologies were developed mainly by partners with extensive experience of Ontology Engineering work. CODPs were instantiated into the target ontologies by import and specialisation. Challenges arose when these initial ontologies needed to be communicated to and extended upon by the

[7] Visual Analytics for Sense-making in Criminal Intelligence Analysis, EU FP7 project 608142

non-expert ontology engineers in the project, e.g., professional software developers and researchers within other fields of computer science. These developers all had extensive experience of data modelling, yet they found the developed models unintuitive and poorly designed. Their complaints primarily concerned two things:

1. Many foundational entities, that are brought in via transitive imports from CODP building blocks do not immediately make sense in the target domain. For instance, criminal involvement in crimes was modelled using the ParticipantRole CODP, which in turn depends on the Situation CODP. As a consequence of this choice, the *Situation* class was brought into the target ontology as a high-level superclass. Developers expressed confusion and irritation with this design, which they argued added unnecessary complexity, as the requirements do not call for representing a general theory of situations.
2. CODP-level classes and properties, while being helpful in solving the modelling problem at hand, are not named or labelled in a cognitively relevant way for the target domain. For instance, even after being instructed in the design of the set of reused CODPs, the developers still expressed dissatisfaction that entities in these CODPs had generic names such as *Agent* rather than the domain specific term *Nominal*[8] that they were used to.

These complaints indicate that the initially developed ontologies do not fulfill the above discussed requirement of being easy to understand by software developers. Neither were the developers comfortable with modifying these ontologies for new uses or deployment contexts, seemingly due to not being confident that they properly understood the initial ontology design.

IMSK Project The goal of the IMSK project[9] was to integrate technologies for security and surveillance to provide an easily reconfigurable system capable of area security. Since the system needed to be easily reconfigurable for different deployment contexts, one of the authors explored the possible use of CODP-based modules as pluggable units of configuration, in workshops together with project participants[10].

At one particular workshop, three participants discussed the effects of *owl:imports* statements in CODPs. Participant A considered imports of foundational concepts helpful for validating the soundness of his own design. Participant C expressed an understanding of the tension between reuse and applicability presented by the import feature and large import closures. Participant B criticized the use of imports on the grounds that the expansion of CODP size that such imports imply negatively affects CODP usability, and on the grounds that the

[8] A UK Police term indicating a person who is reported as being involved in a crime in some role, including both suspected perpetrators and victims.

[9] Integrated Mobile Security Kit, EU FP7 project 218038

[10] For more details on the case, data analysis method, and other interesting findings, please see [8]

base concepts included by imported patterns may be incompatible with one's own world view:

> *"I really have to know what is there and what does it mean. And maybe it's written with some other focus, some other direction, some other goal. And I don't believe in this general modeling of the universe that fits all purposes."* – Participant B

Participant B also indicated that he would use the idea of a pattern as presented in a pattern catalogue and reimplement it, rather than reuse an existing OWL building block, if that block contained too many imports or dependencies. These differing opinions illustrate the tensions between and different preferences on how to use CODPs, possibly grounded in different backgrounds (Participant A had a background in ontologies that Participants B and C lacked).

The participants in this workshop were not required to adhere to the XD method, but were free to use CODPs in whichever way they deemed most suitable to solve the provided modelling problem. Without explicit guidance being given, the intuitive way in which the developers instantiated CODPs was consistently to first study an CODP's design documentation, then draw that CODP on a whiteboard, then modify that drawing to suit the specific modelling problem, before finally attempting to encode that illustration into a solution using Protégé. In spite of Participant A having extensive experience of working with Semantic Web ontologies, the use of *owl:imports* to instantiate CODPs was never tried.

E-care@home Project The E-care@home project[11] aims to improve home healthcare for the elderly, using ICT-rich environments to measure, record, and infer facts about people and their environment, that can be used for recommendations, reminders, etc. In reviewing a CODP-based sensor ontology developed in the project, the authors noticed that just like in the above two projects, certain classes and properties were included in the target ontology which were not strictly speaking necessary in the target context. When questioned about this design and about whether tooling or methods to support reuse without the need to import such classes would be useful, the developer responsible for the ontology in question responded:

> *"Definitely useful. I spent a considerable amount of time to find top-level classes that provide the required links to already designed ones. The lack of such tools is sensed. It can also decrease the rate of errors or inconsistencies in our design."*

3.2 Benefits of Template-Based Instantiation

Before describing our approach we would like to emphasise that the idea underlying Template-Based CODP instantiation is not new; as mentioned in Section 2,

[11] `http://ecareathome.se`

similar approaches are discussed in prior work [11,5,13,10]. Such template-based approaches have however not reached much adoption in the CODP community and the effects of their use have not been explored sufficiently.

The cases described in Section 3.1 illustrate how a certain type of ontology developers and users find the structures generated by the more common Specialisation-based approach to be unintuitive and consequently ontologies constructed using this approach to be diffiult to understand and to modify. For these users, a Template-based approach in which CODPs are instantiated by copying and adapting CODP entities to the target domain semantics (and general CODP-level concepts are left out of the resulting model) would be advantageous. Additionally, a Template-based CODP instantiation approach carries with it other advantages (and disadvantages) detailed below.

Template-based CODP instantiation does not require the target ontology to import semantics from namespaces outside of project scope, so reduces the risk of the target ontology breaking due to unforeseen changes outside of project scope. This self-containedness also simplifies tool support implementation in tooling that does not support the addition of *owl:Imports* axioms (i.e., WebProtégé) or that needs to be able to work in an offline mode.

Further, we argue that when performing Template-based instantiation, the ontology engineer in question becomes more familiar with the resulting module than when they per the Specialisation-based approach reuse a whole block of ontology functionality as-is. Consider the analogy of program code examples taken from the web – rarely do developers reuse such code straight off, but more typically, they use such examples in parts, adapting them to the target context, and in this process, begin to understand and be comfortable with the code. We argue that the same holds for CODPs; by copying and adapting an CODP to the target domain, ontology engineers make it their own code, and this reduces the complexity of and barrier to entry of subsequent debugging or refactoring tasks. This is particularly important for non-expert ontology engineers.

Another advantage of the template-based approach is that the lack of CODP-level entities in the target ontology makes it easier to validate that target ontology with domain experts who are not knowledge engineers, as the number of entities with domain-irrelevant names decreaseas.

There are of course also disadvantages to this approach, or rather, advantages to the Specialisation-based approach that the Template-based approach does not share. In Specialisation-based instantiation, the direct reuse of CODPs by *owl:imports* means that target ontologies that reuse the same CODPs (and RDF data expressed according to these ontologies) are immediately interoperable, not only in a conceptual sense, but also in that the same namespaces are used for shared concepts, which may simplify ontology integration and data sharing. This advantage could also be achieved in Template-based instantiation through alignment of local cloned CODPs to the original source CODPs, but it would then require OWL-level reasoning to be executed in order to achieve the same result, which may not be suitable in all cases. Further, in Specialisation-based instantiation, higher-level classes and properties are defined only once, while in Template-based instantiation pattern entities will be instantiated into the target ontology multiple times – which may complicate maintainability, and, particularly if done manually, may increase the risk of inconsistency.

3.3 Proposed Method of Instantiation

If CODP tooling supporting Template-based instantiation is to be developed, we need to specify what concrete steps should be taken, in which order, when instantiating CODPs in this manner. Below, we list a series of steps that have worked well in initial trials with some commonly used patterns. Due to the variance in structure of published CODPs, the below steps do not in all cases transfer all semantics from the source CODP (including its entire transitive import closure) to the target model – in some cases, additional modelling steps may be required. In the below *"copy"* implies cloning entities and associated class restrictions into a new namespace.

1. Copy CODP leaf classes into subclasses of *owl:Thing* in the target module. If two leaf classes in the source CODP have some shared parent classes below *owl:Thing* level, copy the least common subsumer also into the target module as shared parent to the copied leaves.
2. Copy those object or datatype properties that have as domain or range such classes as were copied above into the target module. For object properties: narrow a potential unmatched half of the domain/range to the least common subsumer or (in the case one does not exist) to leaf class level.
3. Copy (and similarly to above, narrow if it is an object property) any properties involved in class restrictions on classes copied in the first step above and use these copied properties to create equivalent restrictions in the target module.
4. Merge the resulting structure with existing entities in the target module, using ontology matching techniques to find candidate matches.

As mentioned, the steps proposed above have worked well in initial testing, but we have found some cases where they are insufficient and further manual work will be needed. These cases include but are not limited to:

– CODPs where leaf classes may need to be instantiated twice (e.g., the *Place* class in the Place CODP[12], or the *Object* class in Time Indexed Part Of CODP[13]).
– When higher-level (non-leaf) classes from a parent CODP are reused and specialised in a child CODP the situation can arise that the child CODP concepts are sibling leaves to parent CODP concepts, and consequently when instantiating the child CODP, some leaf nodes can exist that are not intended to be instantiated.

[12] `http://ontologydesignpatterns.org/wiki/Submissions:Place`
[13] `http://ontologydesignpatterns.org/wiki/Submissions:TimeIndexedPartOf`

4 Preliminary Evaluation

As discussed in Section 3.2, we believe that ontologies built using Template-based CODP instantiation are easier to understand and easier to modify for a certain group of non-expert users than those built using Specialisation-based instantiation. In order to evaluate this proposition, a small study was run at a workshop within an ontology engineering research project at Jönköping University. The study had five participants, all academics within Computer Science or related topics. The participants had all used ontologies, but only two participants had actually constructed ontologies themselves, and these two were beginners to the task, the project in question being the first project where they had done such work. None of the participants had worked with XD or CODPs previously.

For evaluation we constructed two sets of ontology requirements, and for each of these, we created two CODP-based ontology variants (one Template-based, and one Specialisation-based)[14]. The Specialisation-based variants were constructed using the XD method as described in Section 1, while the Template-based variants were constructed strictly adhering to the method proposed in Section 3.3. The workshop participants were given a tutorial on CODPs, XD, and the specific CODPs that the ontology variants had been constructed from. They were then given three tasks to perform individually:

1. Task 1: For requirement set A, answer which competency questions out of seven provided that the two ontology variants can answer.
2. Task 2: For requirement set B, answer which competency questions out of nine provided that the two ontology variants can answer.
3. Task 3: For the requirement set A, modify the two ontology variants by adding four object properties, specialising some of the more generic properties already in place.

After each task, the users were surveyed on which of the two ontology variants they found easiest to understand or to modify, or whether they found the two equally easy/difficult. The results of these surveys and the total percentage of correct answers to Tasks 1 and 2 given by the participants are provided in Table 1. Note that since not all users finished all tasks within the workshop time-frame, the answer frequency drops in Tasks 2 and 3.

Table 1. Evaluation Results

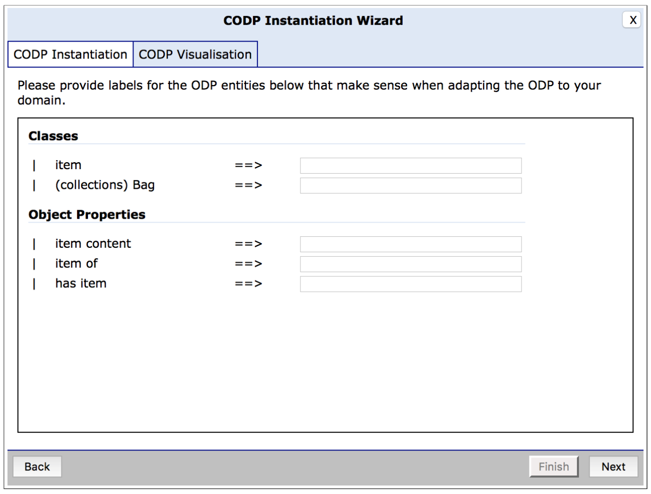
	Task 1	**Task 2**	**Task 3**
Template-based easiest	4	2	3
Equally easy/difficult	1	2	0
Specialisation-based easiest	0	0	0
Correct answer rate	83 %	81 %	

[14] Available at `http://urn.kb.se/resolve?urn=urn:nbn:se:hj:diva-30172`

Obviously a study as limited as this is insufficient to conclusively evaluate the hypotheses provided above, and the generalisability of the findings is limited. All the same, we find it interesting to note that the method proposed in Section 3.3 can be followed in practice to generate ontologies that work, and that of the experiment participants, not a single one reported that the ontologies constructed using the traditional Specialisation-based approach to CODP instantiation were the easiest or most helpful. We believe that these findings clearly motivate additional study of instantiation method effects, as well as the development of new CODP tools supporting Template-based instantiation.

5 Tool Support

In light of the findings reported in Section 4 we are presently developing tooling to support Template-based CODP instantiation. This new tooling builds on our earlier WebProtégé extension titled *XDP* [9], which includes a CODP search engine, CODP browser, and a CODP instantiation wizard. In the new version of XDP[15] we have extended the instantiation wizard, adding a step where the user selects which CODP instantiation method (Specialisation-based or Template-based) they wish to adhere to. In the case that the Template-based approach is selected, a new user interface (Figure 2) is presented where the user labels the classes and properties that will be copied into the target ontology module. Those classes and properties are selected using the method proposed in Section 3.3.



Please provide labels for the ODP entities below that make sense when adapting the ODP to your domain.

Classes

| item ==> |
| (collections) Bag ==> |

Object Properties

| item content ==> |
| item of ==> |
| has item ==> |

Fig. 2. XDP Template-based CODP Instantiation UI displaying the Bag CODP

[15] Available from `http://github.com/hammar/webprotege`

6 Conclusions

In this paper we have presented our experiences of developing ontologies with CODPs in collaboration with non-expert ontologists, noting that current practices introduce ontology structures which certain users find unintuitive to understand. We have explored an alternative Template-based approach and suggested some consequences of applying this approach (resulting ontologies are self-contained and more robust, they are easier to understand and modify for non-experts, and easier to validate by domain experts). We have studied some of those effects (those concerning ease of understanding and modification by non-experts) in a preliminary evaluation with positive initial results, and we have constructed tooling to support the Template-based approach in WebProtégé.

References

1. Blomqvist, E., Gangemi, A., Presutti, V.: Experiments on Pattern-based Ontology Design. In: Proceedings of the Fifth International Conference on Knowledge Capture. pp. 41–48. ACM (2009)
2. Blomqvist, E., Sandkuhl, K.: Patterns in Ontology Engineering: Classification of Ontology Patterns. In: Proceedings of the 7th International Conference on Enterprise Information Systems. pp. 413–416 (2005)
3. Dzbor, M., Suárez-Figueroa, M.C., Blomqvist, E., Lewen, H., Espinoza, M., Gómez-Pérez, A., Palma, R.: D5.6.2 Experimentation and Evaluation of the NeOn Methodology. Tech. rep., NeOn Project (2007)
4. Egana, M., Antezana, E., Stevens, R.: Transforming the axiomisation of ontologies: The ontology pre-processor language. Proceedigns of OWLED (2008)
5. Falbo, R.A., Guizzardi, G., Gangemi, A., Presutti, V.: Ontology patterns: clarifying concepts and terminology. In: Proceedings of the 4th International Conference on Ontology and Semantic Web Patterns. CEUR-WS. org (2013)
6. Fowler, M.: Analysis patterns: reusable object models. Addison-Wesley Professional (1997)
7. Gangemi, A.: Ontology Design Patterns for Semantic Web Content. In: The Semantic Web–ISWC 2005. pp. 262–276. Springer (2005)
8. Hammar, K.: Ontology design patterns in use: lessons learnt from an ontology engineering case. In: Workshop on Ontology Patterns in conjunction with the 11th International Semantic Web Conference 2012 (ISWC 2012) (2012)
9. Hammar, K.: Ontology design patterns in webprotégé. In: 14th International Semantic Web Conference (ISWC-2015). CEUR Workshop Proceedings (2015)
10. Iannone, L., Rector, A., Stevens, R.: Embedding knowledge patterns into owl. In: The Semantic Web: Research and Applications, pp. 218–232. Springer (2009)
11. Presutti, V., Blomqvist, E., Daga, E., Gangemi, A.: Pattern-Based Ontology Design. In: Ontology Engineering in a Networked World, pp. 35–64. Springer (2012)
12. Presutti, V., Daga, E., Gangemi, A., Blomqvist, E.: eXtreme Design with Content Ontology Design Patterns. In: Proceedings of the Workshop on Ontology Patterns (WOP), collocated with International Semantic Web Conference (ISWC) (2009)
13. Ruy, F.B., Reginato, C.C., Santos, V.A., Falbo, R.A., Guizzardi, G.: Ontology engineering by combining ontology patterns. In: Conceptual Modeling, pp. 173–186. Springer (2015)