Ontology Engineering by Combining Ontology Patterns

Fabiano B. Ruy[1,2]; Cássio C. Reginato[1]; Victor A. Santos[1];
Ricardo A. Falbo[1]; Giancarlo Guizzardi[1]

[1]Ontology and Conceptual Modeling Research Group (NEMO), Computer Science Department
Federal University of Espírito Santo, Vitória, Brazil
[2]Informatics Department, Federal Institute of Espírito Santo, Campus Serra, Serra, Brazil
{fabianoruy, cassio.reginato,victor.amsantos,
falbo, gguizzardi}@inf.ufes.br

Abstract. Building proper reference ontologies is a hard task. There are a number of methods and tools that traditionally have been used to support this task. These include foundational theories, reuse of domain and core ontologies, development methods, and software tool support. In this context, an approach that has gained increased attention in recent years is the systematic application of ontology patterns. This paper discusses how Foundational and Domain-related Ontology Patterns can be derived, and how they can be applied in combination for building more consistent ontologies in a reuse-centered process.

Keywords: Ontology Patterns, Conceptual Ontology Patterns, Ontology Reuse, Ontology Engineering.

1 Introduction

Although nowadays ontology engineers are supported by a wide range of Ontology Engineering (OE) methods and tools, building proper reference domain ontologies is still a difficult task even for experts [1]. Besides the domain knowledge, the ontology engineer needs to apply ontological foundations in order to develop well-founded ontologies.

According to their generality level, ontologies can be classified into Foundational, Core and Domain ontologies [2]. At the highest level, foundational ontologies span across many fields and model the very basic and general concepts and relations that make up the world [3]. Domain ontologies, in turn, describe the conceptualization related to a specific domain [3]. Core ontologies are located between foundational and domain ontologies, and provide a definition of structural knowledge in a specific field that spans across different application domains in this field [2]. The generality levels are not a discrete classification, but a continuum [4] from foundational ontologies, totally domain-independent, to domain ontologies, for a very particular domain. Core ontologies, though more general than domain ontologies, are domain-dependent.

Reuse is pointed out as a promising approach for OE, since it enables a speeding up of the ontology development process. Higher level ontologies can be used to support the development of lower level ontologies, e.g., foundational ontologies can be

adfa, p. 1, 2011.

used to support the development of core and domain ontologies, and core ontologies can be the basis for developing domain ontologies. However, ontology reuse, in general, is a hard research issue, and one of the most challenging and neglected areas of OE [5]. The problems of selecting the right ontologies to reuse, extending them, and composing several ontology fragments have not been properly addressed yet [6].

Ontology Patterns (OPs) are an emerging approach that favors reuse of encoded experiences and good practices. OPs are modeling solutions to solve recurrent ontology development problems [7]. There are many different types of OPs that can be used in different phases of the OE process. In this paper, we are interested in Conceptual OPs (COPs), since our focus is on developing reference ontologies. A reference ontology is constructed with the goal of making the best possible description of the domain in reality, representing a model of consensus within a community, regardless of its computational properties [8]. In other words, when developing a reference ontology, the focus is on expressivity of the representation and truthfulness to the domain being represented (domain appropriateness), even if computational properties such as tractability and decidability have to be sacrificed. In summary, in the view employed here, an ontology is a particular kind of conceptual model, namely, a reference conceptual model capturing the shared consensus of a given community. As such, although our discussion is somehow focused on domain reference ontologies, the approach advanced here should be beneficial to ontology-driven conceptual modeling in general [9].

COPs are modeling fragments extracted from either foundational ontologies (Foundational OPs - FOPs) or core / domain ontologies (Domain-Related OPs - DROPs). They are to be used during the ontology conceptual modeling phase, and focus only on conceptual aspects, without any concern with the technology or language [10]. An OP extracted from a higher level ontology can be used to support the development of lower level ontologies.

We argue that if FOPs and DROPs are systematically applied in combination, reuse is maximized, making the OE process more productive, and improving the quality of the resulting domain ontologies. In this paper, we discuss how FOPs and DROPs can be derived, and how they can be used to develop core and domain ontologies. The FOPs discussed here are derived from the Unified Foundational Ontology - UFO [8]. The DROPs, in turn, are related to the Enterprise domain [11]. Moreover, we show a tool supporting the application of COPs for developing ontologies. Our main goal is to show how the combined application of COPs could be useful for OE.

This paper is organized as follows. Section 2 briefly presents the Unified Foundational Ontology (UFO), the source for the discussed FOPs. Section 3 discusses how FOPs are derived from UFO, how DROPs are extracted from a core ontology, and how COPs can be used in combination for developing domain ontologies. Section 4 briefly presents a software tool supporting COPs definition and application. Section 5 discusses related works. Finally, Section 6 presents our final considerations.

2 The Unified Foundational Ontology - UFO

UFO [8] is a foundational ontology that has been developed based on a number of theories from Formal Ontology, Philosophical Logics, Philosophy of Language, Linguistics and Cognitive Psychology. The aspects we present here cover object-type (endurant) distinctions in UFO, namely, the distinction between sortal types, non-sortal types and relators. For an in depth presentation, formal characterization and discussion about empirical support for UFO, the interested reader should refer to [8].

By referring to a number of formal and ontological meta-properties, UFO proposes a number of distinctions among object types. Within these, sortal types are types that either provide or carry a uniform principle of identity for their instances. Within sortal types, we have the distinction between rigid and anti-rigid sortals. A rigid type is a type that classifies its instances necessarily (in the modal sense), i.e., the instances of that type cannot cease to be an instance of that type without ceasing to exist. Anti-rigidity, in contrast, characterizes a type whose instances can move in and out of the extension of that type without altering their identity. For instance, contrast the rigid type *Person* with the anti-rigid types *Student* or *Husband*. While the same individual *John* never ceases to be instance of *Person*, he can move in and out of the extension of *Student* or *Husband*, once he enrolls in/finishes college or marries/divorces, respectively.

Kinds are sortal rigid types that provide a principle of identity for their instances. Subkinds are rigid types that carry the principle of identity supplied by a Kind. Concerning anti-rigid sortals, we have the distinction between Roles and Phases. Phases are relationally independent types defined as a partition of a sortal. This partition is derived based on an intrinsic property of the type (e.g., *Child* is a phase of *Person* while she has less than 18 years). Roles are relationally dependent types, capturing that entities play roles when related to other entities (e.g., *Student*, *Husband*). Since the principle of identity is provided by a unique Kind, each sortal hierarchy has a unique Kind in the top [8].

Non-Sortals are categorizations that aggregate properties of distinct Sortals. *Furniture* is an example of Non-Sortal that aggregates properties of *Table*, *Chair* and so on. Non-Sortals do not provide principle of identity; instead, they just classify things that share common properties but that obey different principles of identity. Rigidity and anti-rigidity also applies to non-sortals. Rigid non-sortals are termed Categories (e.g., *Physical Object*); Anti-rigid sortals are RoleMixins (e.g., *Customer, Crime Weapon*). However, non-sortals can also exhibit a meta-property termed semi-rigidity. An object type is semi-rigid if it is rigid for some of its instances and anti-rigid for others. A semi-rigid non-sortal is termed a *Mixin* (e.g., *Insurable Item*).

For capturing these distinctions between object types put forth by UFO, Guizzardi [8] proposed a UML profile called OntoUML. Over the years, OntoUML has been successfully employed to build conceptual models and domain ontologies in a number of complex domains (e.g., [12], [13]). Table 1 shows the OntoUML stereotypes.

Concerning relationships, UFO's theory of relations makes a fundamental distinction between two main types of relationships, namely: *formal* and *material* relations. Whilst the former holds directly between two entities without any further intervening

individual, the latter is induced by the presence of mediating entities called Relators. Relators are individuals with the power of connecting entities. For example, an *enrollment* connects a *student* with an *educational institution*, and an *employment* connects an *employee* with an *employer organization* [8]. OntoUML has a construct for modeling relator universals. Every instance of a relator universal is existentially dependent on at least two distinct entities. The formal relations that take place between a relator universal and the object classes it mediates are termed *mediation* relations (a particular type of existential dependence relation).

Table 1. OntoUML Object Type Distinction

Stereotype	Identity	Rigidity	Allowed Supertypes
Kind	Provide	Rigid	Category, RoleMixin, Mixin
Subkind	Carry	Rigid	Kind, Subkind, Category, RoleMixin, Mixin
Role	Carry	Anti-Rigid	Kind, Subkind, Role, Phase, Category, RoleMixin, Mixin
Phase	Carry	Anti-Rigid	Kind, Subkind, Role, Phase, Category, RoleMixin, Mixin
Category	-	Rigid	Category
Mixin	-	Semi-Rigid	Mixin, RoleMixin
Role Mixin	-	Anti-Rigid	Mixin, RoleMixin

3 Conceptual Ontology Patterns

In the Ontology Engineering process, pattern reuse can occur in many possible ways [10]. Our focus is on FOPs and DROPs, how they are extracted, and how they are applied for building core and domain ontologies. Figure 1 shows, on the left side, the ontology generality levels, and on the right, the corresponding types of COPs.

FOPs are extracted from the foundations and rules of a foundational ontology. A FOP, capturing foundational structural content, can be reused **by analogy** [14], reproducing their structure in the ontology being developed, by matching the referenced concepts of the pattern to the corresponding ones in the domain. The result is an ontology fragment with the FOP structure shaping the structures at the level of domain concepts. FOPs can be applied for building both core and domain ontologies.

On the other hand, DROPs capture the core knowledge of the target domain, and can be extracted directly from core/domain ontologies fragments. It is worth to note that when a DROP is extracted from a core/domain ontology modeled already reusing FOPs, we have as result a richer DROP, carrying both structural and domain knowledge, characterizing a chained COP application at the domain level. DROP reuse occurs **by extension** [15], i.e., their concepts and relations become part of the domain ontology and they are typically extended, giving rise to new more specific concepts and relations.

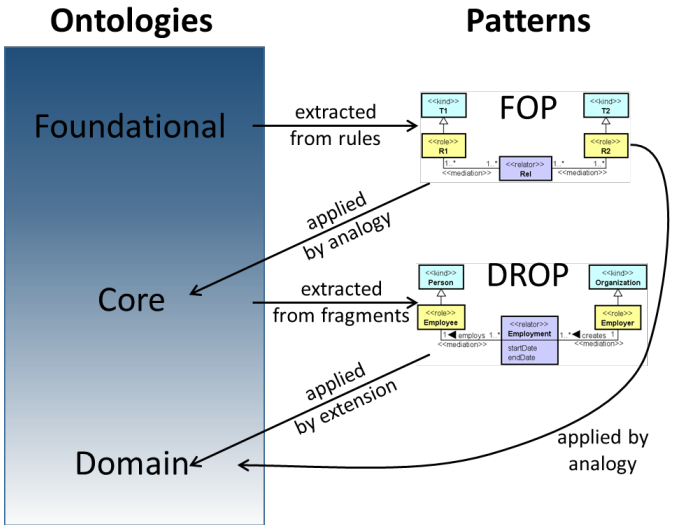


Fig. 1. Generality Levels and Ontology Patterns.

Both DROPs and FOPs can be applied for engineering domain ontologies. When developing domain ontologies, for each aspect to be modeled, the ontology engineer should first try to find an applicable DROP, which captures the core knowledge, and possibly a foundational structure, suitable for the modeling problem at hand. If there is no DROP satisfying the domain requirements, she needs to build a new ontology fragment. In this case, she can apply a FOP and reuse its structure.

Of course, modeling domain ontologies is not limited to the direct application of patterns. Domain ontology fragments created from COPs are interrelated and need to be put together. To do that, the ontology engineer can look for related DROPs, and also use FOPs for combining the structure inherent to the different fragments.

3.1 Deriving and Applying Foundational Ontology Patterns (FOPs)

FOPs pack the structure and rules of some foundational aspects recurrently applied when building core and domain ontologies. A FOP is not a foundational ontology fragment; instead, it is a self-contained set of related foundational rules and constraints that is applied to solve a common modeling problem independently of domain. By self-contained we mean that the pattern satisfies some foundational ontology constraints imposed to the elements involved, not contradicting any foundational ontology rule.

We distinguish between two main types of FOPs, namely Structural FOPs and Derivation FOPs. A Structural FOP captures a template structure that comes from the application of certain foundational ontology aspects that enforce this structure to take place. In order to exemplify, let us consider the following foundational rules from UFO: (r1) A *relator* mediates at least two distinct entities; (r2) A *mediation* is a bidirectional mandatory relation between a *relator* and an *endurant*; (r3) A *role* is an anti-

rigid type and must inherit the identity principle of exactly one *kind*; and (r4) A *role* is externally dependent on a *relator*. A combination of (r1) and (r2) gives rise to the Relator FOP, whilst a combination of (r3) and (r4) gives rise to the Role FOP. Since the Relator FOP is generic, allowing mediations with different types of *endurants* (*kinds*, *roles*, *phases*, *mixins*, etc.), it enables some variations combining other ontological rules or even other FOPs. It is the case of the Role-Relator FOP, which combines the Relator and the Role FOPs (or the four foundational rules mentioned above) to represent the case where a *relator* mediates two *roles*. These patterns are shown in Figure 2, as templates. They can be applied during the development of core or domain ontologies by analogy, i.e., by reproducing their structure in the ontology being developed [10], and completing the structure with the specific knowledge. For instance, to represent the relation of a Person booking a Room, the *relator* Reservation mediates two *roles*: Customer as a *role* of a Person *kind*, and Reserved Room as a *role* of a Room *kind*. The Role-Relator FOP can be applied to many situations relating things, such as in a marriage, in an employment, etc.

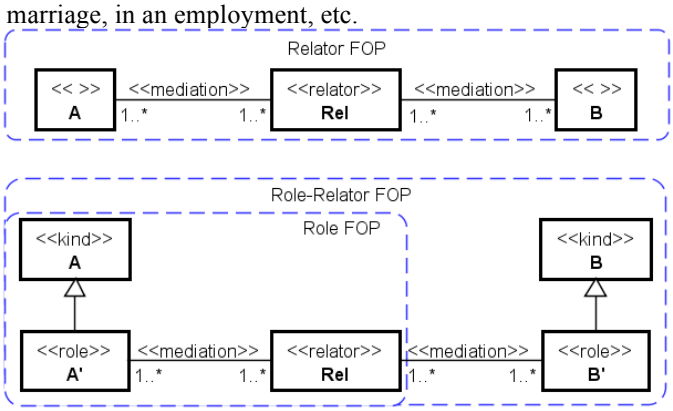


Fig. 2. The Relator FOP.

Another recurrent applied variation of the generic Relator FOP is the Kind-Relator FOP. It takes place when the *relator* mediates a *role* and a *rigid sortal* (*kind* or *subkind*). This FOP is useful when the ontology engineer needs to represent a *relator* between two entities but the *role* that could be assumed by the instances of the *rigid type* is not representative in the domain (e.g., a *role* "University with Students", since it is supposed that all University has Students), and can be discarded. Box D in Figure 6 shows an example of the application of the Kind-Relator FOP.

The Role Mixin FOP [8], which addresses the problem of modeling roles with multiple allowed types is another example of Structural FOP. As Figure 3 shows, the abstract class C is the *role mixin* that covers different *role* types. Classes A' and B' are the disjoint subclasses of C that can have direct instances, representing the *roles* that carry the principles of identity that govern the individuals that fall in their extension. Classes A and B are the ultimate sortals (*kinds*) that supply the principle of identity carried by A' and B', respectively. The material relation R represents the common specialization condition for A' and B', which is represented in C. Finally, class D represents a *endurant* (such as *kind*, *role*, *phase*) that C is *relationally dependent on*.

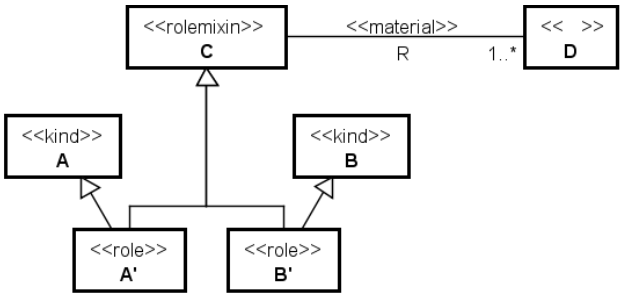


Fig. 3. Role Mixin FOP [8]

Another important class of FOPs regards derived concepts [16]. Derivation regards how someone can derive concepts from operations applied to other concepts. For instance, in [17], Guizzardi demonstrates that ontological meta-properties can be derived from derived Object-Types. For instance, any object-type derived by union from two rigid Object-Types is also a rigid type. In this paper, we approach two cases of derivation: derivation by union and derivation by intersection. Table 2 shows four possible derived concepts obtained by union and intersection, and thus four Derivation FOPs.

Table 2. Derivation FOPs: Union and Intersection

Type 1	Operation	Set of Types	Result
Kind	∪	Kind,…, Kind	Category
Category	∪	Category,…,Category	Category
Subkind	∪	Subkind,…,Subkind	Subkind, Kind
Role	∩	Role,…, Role	Role

In order to exemplify the use of FOPs, we choose the Enterprise domain. Figure 4 depicts a portion of an Enterprise Core Ontology (adapted from [11]), answering the following competency questions: (CQ1) Who are the employees of an Organization? (CQ2) What is the time period of an Employment? (CQ3) How is an Organization structured in terms of Organizational Units? (QC4) Which are the Projects of an Organization? (CQ5) Which are the parties involved in a Project?

The first competence question (CQ1) points the need to relate people to an organization, representing the employment relation. The Role-Relator FOP is suitable for this case. By analogy, we can say that Person (a *kind*) plays the *role* Employee while participating in the Employment *relator*, which is created by an Employeer, a *role* of the Organization *kind*. In the same Enterprise domain, the Relator FOP applies several times, being useful to represent, for instance, Team Membership. The competency question CQ2 is solved by adding the *startDate* and *endDate* properties to the Employment concept. CQ3 can be solved by applying the Composition FOP, considering two subtypes of Organization: Simple Organization, representing those organizations not broken down into Organizational Units; and Complex Organization, representing organizations composed of two or more Organizational Units (satisfying the so-called

weak supplementation [18] axiom for compositional structures). CQ4 can be answered by the same fragment of CQ5, since the Organization itself is a party involved in its Projects. The parties involved in Projects have diverse nature, such as Organizations, Organizational Units and People. Since each party type assumes a role applying different principles of identity, the Role Mixin FOP applies. Thus, Project Party is a *role mixin* with a material relation with the Project *kind*.

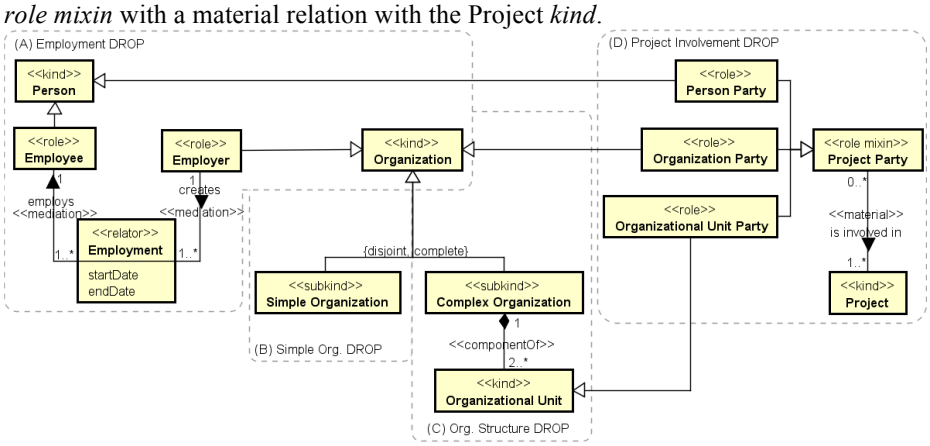


Fig. 4. Enterprise Core Ontology fragmented in DROPs (partial version, adapted from [11]).

To exemplify the application of a derivation pattern, let us take a fragment of the Enterprise Core Ontology, where Team, Organizational Unit and Organization give rise by union to the concept of Institutional Agent. Since these three concepts are classified as *kind* in the ontology, the resulting derived by union type (Institutional Agent) should be a *category* (see the first line of Table 2), as Figure 5 shows.

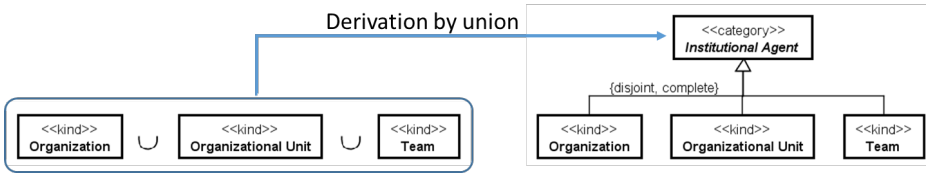


Fig. 5. The Kind Union FOP.

3.2 Extracting and Applying Domain-related Ontology Patterns (DROPs)

DROPs are reusable fragments extracted from reference core/domain ontologies. Ideally, DROPs capture the core knowledge related to a domain, and thus they can be seen as fragments of a core ontology of that domain [10]. Since core ontologies should be created grounded on foundational ontologies, it is natural to build them applying FOPs.

Once we have a core ontology, DROPs can be extracted from it through a fragmentation process. Each fragment meaningful for the domain can be packaged as a DROP. DROP complexity can vary greatly depending on the domain fragment being represented. Sometimes a DROP contains only two related concepts; in other situa-

tions they can contain a complex combination of concepts and relations. Sometimes the same fragment gives rise to two (or more) variant and alternative patterns; sometimes a DROP is structurally open in order to be completed by another DROP.

Regarding the DROP derivation process, while FOPs are focused on foundational and structural aspects, when deriving a DROP, domain aspects come first. The main rule for a DROP is to represent a recurrent fragment in the field, regardless of its foundational structure. Thus, while FOPs tend to be generally applied, DROPs for a specific field are very interrelated [4]. For this reason, it is usual to apply many DROPs in combination or in a sequence for engineering domain ontologies. Ontology Pattern Languages (OPLs) [4], for instance, are used to organize DROPs in a guided application process.

With a core ontology in hands, fragments can be extracted to form DROPs. In order to illustrate the extraction of DROPs from core ontologies, in Figure 4, the dotted boxes show four DROPs: (A) Employment, (B) Simple Organization, (C) Organizational Structure, and (D) Project Involvement. Although each DROP represents a distinct aspect of the core domain knowledge, they can be related in different ways. For instance, the DROPs Simple Organization (B) and Organizational Structure (C) are very related from the foundational point of view, since they model two complementary *subkinds* of the same *kind*. However, they can also be applied in isolation, depending on the application domain. Another interesting aspect is that different DROPs partially overlap between them. This is the case of the Organization concept, which is shared by three distinct DROPs. This characteristic shows that DROPs have a more strict relation and are often applied in combination or even in sequence. Additional examples of DROPs of the Enterprise domain and possible combinations between them can be found in [11].

In order to illustrate the combined application of COPs for building a domain ontology, Figure 6 presents a portion of a domain ontology for Universities. This example explores some different situations of COP reuse at the domain level. Since University is a specific type of Enterprise, the Enterprise DROPs are applicable. Three competency questions were defined for this portion of the domain ontology: (CQ1) Who are the employees of a University? (CQ2) Who are the students of a University? (CQ3) Which are the parties involved in a Research Project? As we can notice, CQ1 and CQ3 are, indeed, specializations of the competency questions of the Enterprise core ontology presented before. Two DROPs of Figure 4 applies to answer CQ1 (Employment DROP) and CQ3 (Project Involvement DROP).

CQ1 can be solved directly by applying the Employment DROP. This DROP is added to the domain model (box A in Figure 6) and its concepts and relations are extended (in box C): University is an Organization, University Employment is an Employment, and the *roles* Employer and Employee are specialized into the specific *roles* University Employer and University Employee, respectively. The relations "creates" and "employs" are also specialized. This reuse by extension mechanism allows some modifications, such as adding new properties to the concepts, restricting cardinalities and adding new axioms. It is worthwhile to point out that the Employment DROP inherits the structure of the Role-Relator FOP. Thus, the university employment fragment being modeled (box C) reuses these two COPs, from different levels.

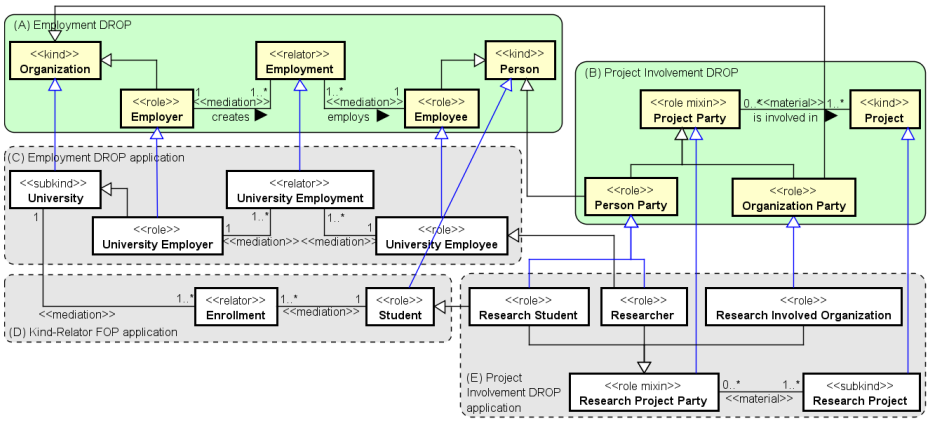


Fig. 6. Building a Domain Ontology with different types of COPs.

For CQ2, there is not a DROP available, since Student is a notion of the University domain, and is not modeled for Enterprises in general. However, the question treats a relation between a specific role assumed by people and a University. Thus, the Kind-Relator FOP applies. Reusing this FOP structure, the *role* Student extends Person, and the *relator* Enrollment establishes the participation of Students in a University (box D). It is important to observe that only two new concepts, Student and Enrollment, were added to the model, taking the advantage of reusing the concepts of Person and University already modeled.

The third question, CQ3, addresses the different parties involved in a (Research) Project, and can be solved applying the Project Involvement DROP. As in the case of CQ1, the DROP is added to the model (box B) and its concepts and relations are extended (box E). Research Project is a Project, and Research Project Party is a specialization of the Project Party *role mixin*. The University domain considers three types of involved parties: Researchers, a more specific *role* of University Employee; Research Student, a specialization of the Student *role*; and Research Involved Organization, a *role* for Organizations involved in research projects. The first two types, have their identity principle provided by the Person *kind*, thus extend Person Party. The last one is an Organization, and extends Organization Party. This third fragment is another case of combined reuse of COPs, since the fragment of the Project Involvement DROP reuses the structure of the *Role Mixin* FOP. In this case, the joint application of the FOP is more explicit, since the modeler needs to know the *Role Mixin* FOP for assuring that the "is involved in" material relation is enough to characterize the three new *roles* created in the domain.

The combined reuse of COPs can also directly support the axiomatization of the domain ontology at hand. This is because, by reusing and adapting the COPs, one can also directly reuse and adapt the axioms defined for that COP. For instance, in the university employment fragment (box C in Figure 6), there are a number of axioms that come directly from the application of the corresponding COP. In this example, by instantiating with domain concepts parts of the generic axiomatization defined for the Role-Relator FOP, we can have the automatic generation of formal constraints stating,

for instance, that an University Employment is multiply *existentially dependent* on exactly one University Employee and exactly one University Employer; that a University Employer is a Person that contingently (in the modal sense) instantiates that role when mediated by at least a University Employment, among other constraints [8].

4 Tool Support for Ontology Pattern Application in OLED

For the definition and application of FOPs and DROPs, we extended the OLED tool. OLED[1] is an ontology framework that supports the OntoUML language [8], providing a number of features such as construction, evaluation, simulation and transformations of OntoUML models. In order to provide support for pattern-based engineering of OntoUML models, OLED allows the definition of FOPs and DROPs libraries. These libraries can be created in the tool by defining each pattern as a model. Then, the suitable libraries can be imported and applied to build core and domain ontologies. This feature also turns possible the combination of the ontology patterns aforementioned, since the OLED extension enables to build a core ontology applying FOPs libraries and then saving fragments of this core ontology in a DROP library. These DROPs (and FOPs), in turn, can be reused to build domain ontologies.

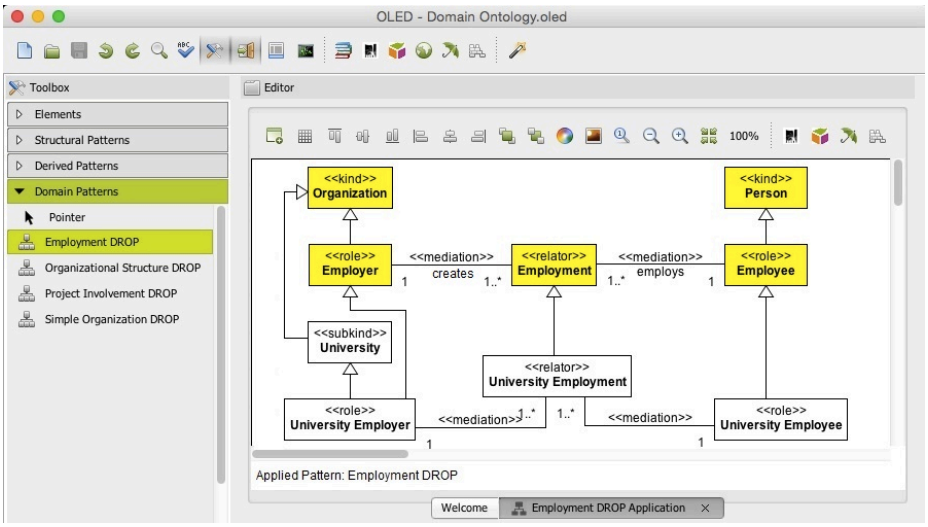
Figure 7 illustrates these pattern supporting features for creating a domain ontologies with a DROP library. First, the Employment DROP is selected for reuse in the pallet (left side of Figure 7a). A new window for COP configuration opens (Figure 7b), where some adaptations can be made (changing names, picking up from the model, adding new classes, etc.). Next, the DROP classes are included in the current model (highlighted classes), and can be specialized with specific concepts and relations from the application domain.

5 Related Work

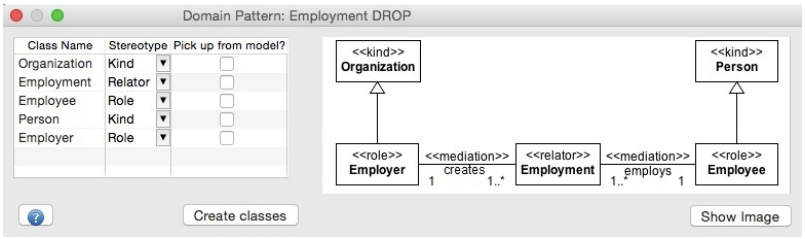
There are few works addressing the combined application of OPs. Most of them focuses on reusing patterns of the same generality level, or of the same type.

Uschold and colleagues [19] consider the use of an implementation guide for building domain ontologies. This process has been applied, for instance, in the Gist ontology [20]. Gist has the main purpose of being used as a catalog of generic concepts for the enterprise domain. The Gist's intents are similar to the idea of DROPs, but restricted to the enterprise domain. Gist can be considered an enterprise core ontology that establishes a method to apply its concepts. Hence, it would be plausible to consider the Gist's fragments as DROPs. As Gist, others trustful generic ontologies are capable to be a source of DROPs for specific domains. One of our main concerns here is to show that core ontologies can take advantage of using foundational ontologies and consequently FOPs.

[1] OLED: https://code.google.com/p/ontouml-lightweight-editor



(a)



(b)

Fig. 7. OLED's support for reusing COPs.

In [5], Gangemi and Presutti present six categories of patterns (namely: structural, content, lexico-syntactic, reasoning, presentation and correspondence) for ontologies at the design and implementation levels. Their Content Patterns are quite similar to our DROPs. However, they do not consider foundational patterns nor discuss how patterns can be combined.

Falbo and colleagues [10] extended Gangemi and Presutti's pattern classification, by considering DROPs and FOPs as COPs. They also briefly show how these types of COPs can be used for building ontologies (reuse by analogy and by extension). In fact, we took this work as a baseline. With respect to that work, we define here a new type of Foundational OP, the Derivation FOPs, and discussed how the mechanism for deriving FOPs and DROPs works. We also discuss, in details, how FOPs are applied by analogy and DROPs by extension. Finally, we present a software tool supporting the definition and application of these COPs.

In another work of the same group [4], Falbo and colleagues discussed how core ontologies can be organized as Ontology Pattern Languages (OPLs). An OPL provides guidance for the application of DROPs to build domain ontologies. OPLs are a relevant example of COPs joint application, when DROPs are reused in combination,

or even in sequence. However, OPLs explore only DROP application. As contributions, we presented how FOPs, combined with DROPs, are useful for supporting the development of domain ontologies. This combination of different level COPs is useful especially for (i) applying a FOP when there is no DROP suitable for meeting a domain requirement; (ii) applying a DROP enriched by a FOP background (combined application of a sequence of DROPs and FOPs); (iii) combining different DROP applications in structurally valid ontology configurations; and (iv) combining DROPs from different related OPLs. Considering all these situations, we claim that OPLs should also consider including FOPs for building domain ontologies.

6 Final Considerations

Ontology Patterns have been recognized as a beneficial approach for Ontology Engineering [5,6]. This paper discussed how FOPs and DROPs can be derived, respectively from foundational and core ontologies, and how they can be applied in combination for creating ontology-driven conceptual models, in general, and domain ontologies, in particular. The combined application of these two different levels of COPs enriches the model development process in many ways: (i) FOPs can complement a domain ontology fragment by adding the necessary concepts to make the model consistent; (ii) FOPs can be used when there is no suitable DROP for the problem at hands, and (iii) the combined reuse of COPs can enrich a domain ontology fragment with structural and domain knowledge. Besides the advantages for modeling, COPs can also contribute for reusing competency questions and axioms from foundational and core ontologies. The key point is that when developing domain ontologies, foundational aspects count as much as the domain aspects. Thus, for building well-founded and domain compliant ontologies, it is essential to reuse both aspects. This reuse can be achieved by applying foundational and domain-related pattern in combination.

As future work, we are studying a proper way to include FOPs into OPLs, improving the language guidance for building domain ontologies. Some efforts are also invested on evolving some aspects of the OLED tool, mainly for integrating different patterns libraries and improving the model construction usability. We are also planning an empirical study to demonstrate the advantages of this pattern-based approach in which FOPs and DROPs are combined, both in terms of the productivity gained and in terms of the cognitive tractability of the resulting models.

Acknowledgements

This research is funded by the Brazilian Research Funding Agency CNPq (Processes 485368/2013-7 and 461777/2014-2).

References

1. O. Noppens and T. Liebig, "*Ontology Patterns and Beyond Towards a Universal Pattern Language*", in WOP, 2009.
2. A. Scherp, C. Saathoff, T. Franz, and S. Staab, "*Designing core ontologies*", Applied Ontology, vol. 6, pp. 177–221, 2011.
3. N. Guarino, "*Formal Ontology and Information Systems*", in FOIS'98, 1998, vol. 46.
4. R. A. Falbo, M. P. Barcellos, J. C. Nardi, and G. Guizzardi, "*Organizing ontology design patterns as ontology pattern languages*". In Proceedings of the 10th Extended Semantic Web Conference - ESWC 2013 (Montpellier, France) 2013.
5. A. Gangemi and V. Presutti, "*Ontology Design Patterns*", in Handbook on Ontologies, Second, S. Staab and R. Studer, Eds. Springer, 2009, pp. 221–243.
6. E. Blomqvist, A. Gangemi, and V. Presutti, "*Experiments on pattern-based ontology design*", in Proc. of 5th International Conference on Knowledge Capture, K-CAP'09, 2009.
7. V. Presutti, E. Daga, A. Gangemi, and E. Blomqvist, "*eXtreme Design with Content Ontology Design Patterns*", Proc. Work. Ontol. Patterns (WOP 2009), 2009.
8. G. Guizzardi, "*Ontological foundations for structural conceptual models*", Enschede: Telematica Instituut Fundamental Research Series, 2005.
9. G. Guizzardi, "*Ontology Patterns, Anti-Patterns and Pattern Languages for Next-Generation Conceptual Modeling*", Proceedings of the 34th International Conference on Conceptual Modeling (ER 2014), Atlanta, USA, 2014.
10. R. A. Falbo, G. Guizzardi, A. Gangemi, and V. Presutti, "*Ontology patterns: clarifying concepts and terminology*". In Proceedings of the 4th Workshop on Ontology and Semantic Web Patterns, Sidney, Australia, 2013.
11. R. A. Falbo, F. B. Ruy, G. Guizzardi, M. P. Barcellos, and J. P. A. Almeida, "*Towards an enterprise ontology pattern language*", in Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14, 2014, pp. 323–330.
12. U.S. Department of Defense (DoD), Data Modeling Guide (DMG) For An Enterprise Logical Data Model, V2.3; 15 March 2011.
13. G. Guizzardi, M. Lopes, F. Baião, and R. A. Falbo. "*On the importance of Truly Ontological Distinctions for Ontology Representation Languages: An Industrial Case Study in the Domain of Oil and Gas*", 14th International Conference on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD´09), Amsterdam, The Netherlands, 2009.
14. N.A. Maiden and A.G. Sutcliffe, "*Exploiting Reusable Specifications Through Analogy*", Communications of the ACM, Vol. 35, No. 4, April 1992.
15. M. Mattson, J. Bosch, and M. Fayad, "*Framework Integration Problems, Causes, Solutions*". Communications of ACM, 42(10), pp. 80–87, 1999.
16. A. Olivé, "*Conceptual modeling of information systems*". Springer Science & Business Media, 2007.
17. G. Guizzardi, "*Ontological meta-properties of derived object types*", Proceedings of the 24th International Conference on Advanced Information Systems Engineering (CAISE 2012), Gdansk, 2012.
18. A. Varzi, "*Mereology*", in The Stanford Encyclopedia of Philosophy, Spring 2015. E. N. Zalta, Ed. 2015.
19. M. Uschold, M. King, S. Moralee, and Y. Zorgios, "*The enterprise ontology*", Knowl. Eng. Rev., vol. 13, no. 01, pp. 31–89, 1998.
20. M. Uschold and D. McComb, "*Introduction to Gist*", IAOA, 2013. [Online]. Available: http://iaoa.org/isc2014/uploads/Whitepaper-Uschold-IntroductionToGist.pdf.