From Reference Ontologies to Ontology Patterns and Back

Fabiano B. Ruy[1,2], Giancarlo Guizzardi[1,3], Ricardo A. Falbo[1], Cássio C. Reginato[1], Victor A. Santos[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
[3]Faculty of Computer Science, Free University of Bozen-Bolzano, Italy
{fabianoruy, gguizzardi, falbo}@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 the use of foundational theories, the reuse of domain and core ontologies, the adoption of development methods, as well as the support of proper software tools. In this context, an approach that has gained increasing attention in recent years is the systematic application of *ontology patterns*. However, a pattern-based approach to ontology engineering requires: the existence of a set of suitable patterns that can be reused in the construction of new ontologies; a proper methodological support for eliciting these patterns, as well as for applying them in the construction of these new models. The goal of this paper is twofold: (i) firstly, we present an approach for deriving conceptual ontology patterns from ontologies. These patterns are derived from ontologies of different generality levels, ranging from foundational to domain ontologies; (ii) secondly, we present guidelines that describe how these patterns can be applied in combination for building reference domain ontologies in a reuse-oriented process. In summary, this paper is about the construction of ontology patterns from ontologies, as well as the construction of ontologies from ontology patterns.

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

1 Introduction

Nowadays, we have a wide range of Ontology Engineering (OE) methods and tools to support ontology engineers in the development of domain ontologies. Despite of that, building proper reference domain ontologies is still a difficult task even for ontology experts [1]. In addition, to properly representing domain specific knowledge, the ontology engineer needs to comply with domain-independent ontological principles in order to develop well-founded ontologies.

Reuse is pointed out as a promising approach for OE, since it enables speeding up the ontology development process. However, ontology reuse, in general, is a hard research issue, and one of the most challenging and neglected areas of OE [2]. The problems of selecting the right ontologies to reuse, extending them, and composing several ontology fragments have not yet been properly addressed [3].

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 [4]. There are many different types of OPs that can be used in different phases of the OE process. In this paper, since our focus is on developing the so-called *reference ontologies* (as opposed to operational ontologies) [5], we are interested in *Conceptual OPs* (henceforth, *COPs*). COPs are reusable modeling fragments extracted from either *foundational ontologies* (*Foundational OPs - FOPs*) or *core* and *domain ontologies* (*Domain-Related OPs - DROPs*). They are intended for use during the conceptual modeling phase of OE and, hence, focus only on conceptual aspects, i.e., without any concern with technological or computational issues [6].

As demonstrated in the literature, the use of higher-level ontologies can bring several benefits to the development of lower-level ontologies [7]. Given this, our basic premise is that, in a reused-oriented approach, COPs extracted from higher-level ontologies can also be used to support the development of lower-level ontologies.

The extraction and application of COPs cannot be done in an arbitrary manner. Firstly, COPs do not simply exist as freestanding fragments of a higher-level ontology. Typically these patterns form a system of interrelated components [8]. So, a proper Pattern-based OE approach should provide methodological support for systematically eliciting patterns from these higher-level ontologies. Secondly, it is not enough for us to provide the ontology engineer with a set of patterns and expect her to always be able to conduct the combined application of the patterns that is more suitable for a given situation. Hence, a proper Pattern-based OE approach should also be accompanied with a methodological support for guiding the modeler in the correct application of these patterns. In summary, we need a Pattern-based OE approach that supports systematic pattern development *for reuse* and systematic ontology development *with reuse*. Contributing to the development of such an approach is the goal of this paper. In particular, we present an approach for deriving DROPs from core/domain ontologies, and guidelines for applying FOPs and DROPs in the development of reference domain ontologies. In the approach proposed here, we argue that, if FOPs and DROPs are systematically applied in combination, reuse is drastically improved, making the OE process more productive, and improving the quality of the resulting domain ontologies.

The FOPs presented in this paper have been derived from the Unified Foundational Ontology (UFO) [9]. In contrast, the DROPs presented here have been derived from a core ontology in the Service Domain, which is itself based on UFO (termed UFO-S) [10].

UFO [9] 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. It has been successfully employed as a basis for analyzing, reengineering and integrating many modeling languages and standards in different domains (e.g., UML, TOGAF, ArchiMate, RM-ODP, TROPOS/i*, AORML, ARIS, BPMN), as well as for the development of Core and Domain Ontologies in different areas. Examples of targeted domains include Services, Capabilities, Organizational Structures, Communities, Goals and Motivations, Constitutional Law, Business Processes, Discrete Event Simulation, Simulation for Land Covering and Use, Measurement, and Software Engineering, among many others [11]. Of all applications of UFO, one deserves special attention, namely, the use of UFO categories and axiomatization in the design of an ontology-driven conceptual modeling language, which later came to be known as OntoUML [9].

OntoUML was conceived as an ontologically well-founded version of the UML 2.0 fragment of class diagrams. Over the years, it has been adopted by many research, industrial and government institutions worldwide, in areas ranging from Geology to Organ Donation, from Biodiversity Management to Logistics, from Software Engineering to Telecommunications [12]. In particular, it has been considered as a candidate for addressing the OMG SIMF (Semantic Information Model Federation) Request for Proposal, after a report of its continuous successful use by a branch of the U.S. Department of Defense [12]. The UML-like models shown in the remainder of this paper are OntoUML models.

This paper is organized as follows. Section 2 introduces the main notions regarding ontology and ontology patterns. In particular, it briefly introduces the Unified Foundational Ontology (UFO) and the OntoUML language, which are used throughout this work. Section 3 presents the rationale of our approach for deriving FOPs from UFO, by using OntoUML as a pattern language. Section 4, in turn, presents guidelines for deriving DROPs from core and domain ontologies. Section 5 demonstrates, in a comprehensive example in the service domain, how FOPs and DROPs can be used in combination for developing a reference domain ontology. General guidelines for applying these patterns in combination are also provided. Section 6 discusses how the presented approach for extracting DROPs from core ontologies has been employed for eliciting patterns for Ontology Pattern Languages (OPLs), as well as some results of using them in combination with FOPs to build reference domain ontologies. Section 7 presents a computational tool supporting the definition and application of COPs following the approach presented here. Section 8 discusses related works. Section 9 presents our final considerations.

2 Background: Ontologies and Ontology Patterns

Ontologies have been classified in diverse perspectives in the literature, for example, according to their levels of generality, formality, applicability, etc. In this paper, we are mainly interested in the

classification criterion regarding generality levels, in which ontologies can be classified into *Foundational*, *Core* and *Domain ontologies* [13]. At the highest generality level, Foundational Ontologies span across many fields and model the most basic and general concepts and relations that make up the world (including domain-independent notions, such as object, event, dependence, classification, parthood relation etc.) [14]. Domain Ontologies, in turn, describe the conceptualization related to a specific domain (such as *Requirements* and *Testing* in *Software Engineering*) [14]. Core Ontologies, located between the foundational and domain levels, provide a definition of structural knowledge in a specific field but one that still spans across different application areas in this field (such as *Service*, *Enterprise* and *Measurement*). These ontologies are typically built based on foundational ontologies and provide a refinement to them by adding detailed concepts and relations in their specific fields [13]. The different generality levels do not amount to a discrete classification, but to a continuum [8], ranging from foundational ontologies that are totally domain-independent (such as DOLCE [14] and UFO [9]), to domain ontologies, for a very particular domain. Finally, core ontologies, despite being more general than domain ontologies, are also domain-dependent.

Higher-level ontologies can be used to support the development of lower-level ontologies, e.g., foundational ontologies can be used as basis for building core and domain ontologies, and core ontologies can support the development of domain ontologies. In fact, considering the continuous nature of the aforementioned classification, some ontologies can be used for supporting the development of more specific ontologies even within the same level of generality. For example, UFO-A (an ontology of endurants) [9] and UFO-B (an ontology of events) [15], both of which are foundational ontologies, have been used as basis for building UFO-C (an ontology of social entities) [7]. The latter, albeit being more specific, is still considered to be a foundational ontology. UFO-S (a core ontology on services) [10] is grounded in UFO-C, while the *Car Rental* domain ontology presented in Section 5 of this paper is developed by extending UFO-S. The *Software Requirements Ontology* (SRO), in turn, is a domain ontology that is grounded directly in UFO-C. Finally, the *Runtime Requirements Ontology* [16] (RRO), a domain ontology on requirements at runtime, was developed based on SRO. Figure 1 illustrates the view of ontology generality levels as a continuum using the aforementioned ontologies. The dashed arrows show the grounding dependencies between the ontologies in different levels.
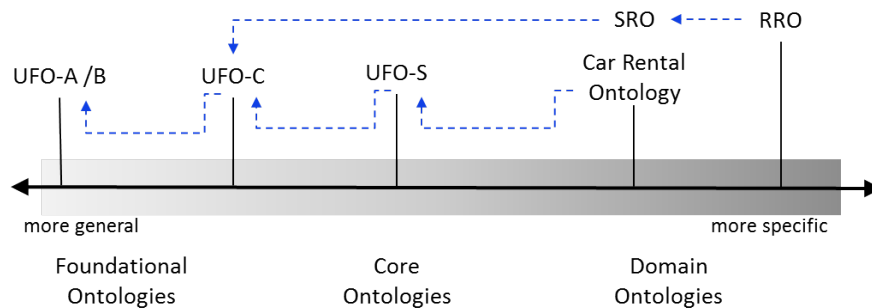

Figure 1. Ontology Generality Levels as a Continuum.

Another relevant classification criterion concerns the intended application of ontologies. Guizzardi [5] makes an important distinction between ontologies as conceptual models (termed *reference ontologies*) and ontologies as coding artifacts (*operational ontologies*). A reference domain ontology is constructed with the goal of making the best possible description of the domain in reality. It is a special kind of conceptual model, an engineering artifact with the additional requirement of representing a model of consensus within a community [5]. On the other hand, once users have already agreed on a common conceptualization, different operational versions of a reference ontology can be created. Contrary to reference ontologies, operational ontologies are designed with the focus on maximizing particular non-functional requirements (e.g., the maintainability of certain desirable computational properties). 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 at the expenses of computational characteristics such as tractability and decidability [5]. In summary, in the view employed here, a reference 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 (which, again, is a particular kind of

conceptual model), the approach advanced here should be beneficial to ontology-driven conceptual modeling in general [17].

2.1 Ontology Patterns

An Ontology Pattern (OP) describes a particular recurring modeling problem that arises in specific ontology development contexts and presents a well-proven solution for the problem [4, 6]. OPs also can be classified using different criteria. Falbo and colleagues [6] present a classification (shown in Figure 2) considering the ontology development phases where these patterns are applied.
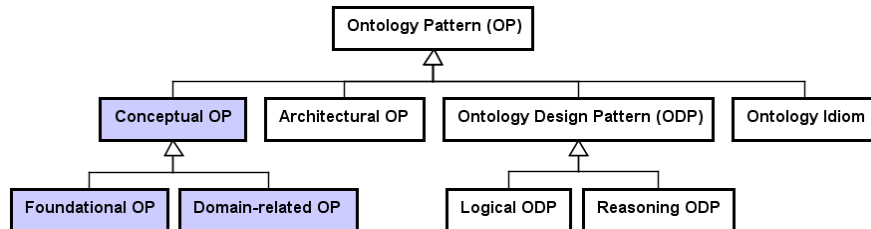

Figure 2. Ontology Pattern Types [6].

Given the objectives of this paper, we focus here only on Conceptual Ontology Patterns (COPs), on their nature and, more specifically, on how they are extracted and applied. For a detailed discussion on the role of Architectural OPs, Ontology Design Patterns and Ontology Idioms, the interested reader is referred to [6].

COPs are modeling fragments extracted of either foundational ontologies (Foundational OPs, henceforth, FOPs) or core/domain reference ontologies (Domain-related OPs, henceforth, DROPs). They are intended for use during the ontology conceptual modeling phase, and focus only on conceptual aspects, without any concern with technological or computational issues [6]. A COP extracted from a higher-level ontology can be used to support the development of lower-level ontologies. FOPs are reusable fragments derived from foundational ontologies [6], packaging a small portion of foundational structural knowledge. Since foundational ontologies are in the top-most generality level, FOPs can potentially be applied to any domain. An example of a FOP is the pattern for the problem of *specifying roles with multiple disjoint allowed types*, which has been extracted from the ontology of substantial universals of the Unified Foundational Ontology (UFO), as detailed in [9]. DROPs are reusable fragments extracted from reference core/domain ontologies, packaging the core knowledge related to a domain [6]. Thus, DROPs can be seen as fragments of a core/domain ontology that are applicable for designing ontologies of a lower generality level.

There are two main ways of reusing ontology patterns [6]: by analogy and by extension. In reuse by analogy, with an ontology modeling problem at hands, we look for OPs that describe knowledge related to the type of situation we are facing. Once selected the pattern, we have to identify which concepts in our domain correspond to the concepts in the pattern, and we reproduce the structure of the pattern in the domain ontology being developed. FOPs are reused by analogy. DROPs, in turn, can be reused both by analogy and by extension. In reuse by extension, the DROP is incorporated in the domain ontology being developed, and it can be extended by means of specialization of its concepts and relations, and also by including new properties and relationships with the extended concepts.

2.2 The Unified Foundational Ontology - UFO

UFO consists of three parts: UFO-A, an ontology of endurants (objects) [9], UFO-B, an ontology of events (perdurants) [15] and UFO-C, an ontology of social entities [7] built on top of UFO-A and UFO-B. In this paper, our discussion is centered in the ontological distinctions comprising UFO-A. Thus, we limit our presentation to the aspects that are germane for the purposes of this article. For a complete description, formal characterization and empirical support for the distinctions and axiomatization comprising UFO-A, the reader should refer to [9].

UFO makes a fundamental distinction between Individuals and Universals. **Individuals** are entities that exist in reality and obey a unique and determinate principle of identity, while **Universals** are ab-

stract patterns of features that can be realized in a number of different individuals. Figure 3 presents a fragment of UFO that focuses on different categories of Individuals.
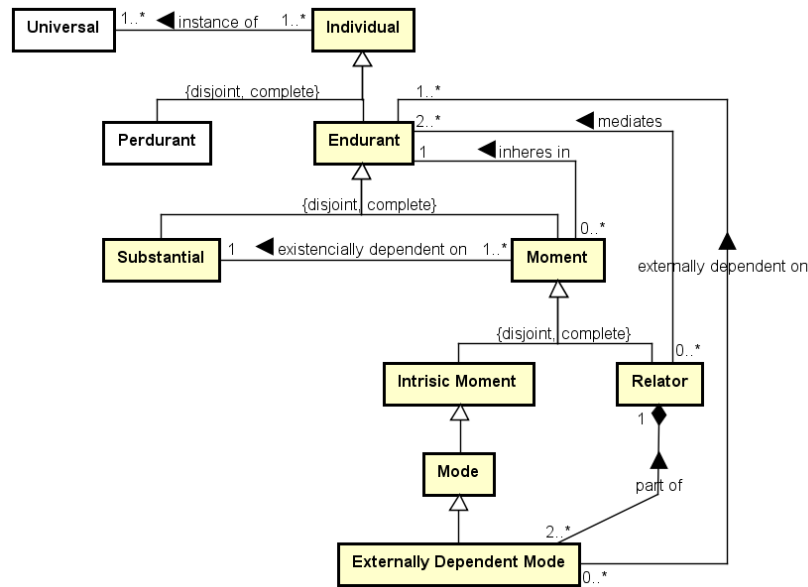

Figure 3. A UFO-A Fragment (Individuals).

Endurants are individuals that are wholly present whenever they are present. They can be understood in contrast with the category of Perdurants (Processes, Events). The category of endurants can be further specialized into Substantials (Objects) and Moments. **Substantials** are existentially-independent Endurants (e.g., *a person*, *a car*). **Moments**, in contrast, are individuals that can only exist by inhering in other individuals. In other words, moments are individuals that are existentially-dependent on their bearers (e.g., *a person's headache*, *a covalent bond between atoms*). **Intrinsic Moments** are moments that inhere in one single individual (e.g., *an apple's color*). An example of an intrinsic moment is a **Mode** (e.g., *John's desires, intentions, perceptions, symptoms, skills*) [9].

   Relators, introduced in [9], are moments that existentially depend on two or more endurants (e.g, *marriages*, *service agreements*, *enrollments*, *employments*, *presidential mandates*). Relators are considered to be the truthmakers of domain (material) relationships [18]. For example, assume that *John* and *Mary* get married. In this case, several *externally-dependent (i.e., relational) modes* come into existence, such as all emotions, commitments and claims towards each other. These co-dependent modes are originated from the same foundational event (in this case the wedding event). The relator is an endurant that, at each time that marriage relationship exists, it aggregates all the externally-dependent modes that the two persons acquire in virtue of participating in the corresponding relation. In the UFO literature, relator names are commonly nominalizations of the verb that expresses the underlying relation (e.g., *married-to*/*marriage*). It is important, nonetheless, to stress that, despite such nominalizations are often understood as referring to perdurants (e.g., *marriage* denotes the life of the couple after the wedding), a relator is not a perdurant. Rather, the perdurant referred by this nominalization can be seen as the constitutive subject of the "relator's life", whose changes in time account for the way the relator evolves [18].

   Figure 4 depicts the Endurant Universals hierarchy in UFO. **Endurant Universals** are distinguished into **Substantial Universals** and **Moment Universals**. Naturally, these are kinds of universals whose instances are **Substantial** Individuals and **Moments** [9], respectively. Concerning the Substantial Universal hierarchy, **Sortal Universals** are the ones that either provide or carry a uniform *principle of identity* for their instances. A principle of identity supports the judgment whether two individuals are the same, i.e., in which circumstances the identity relation holds. In particular, it also informs which changes an individual can undergo without changing its identity. The **Mixin Universals**, or Non-Sortals, are universals that aggregate properties of distinct Sortals, i.e., it can have as instances individuals obeying different principles of identity.
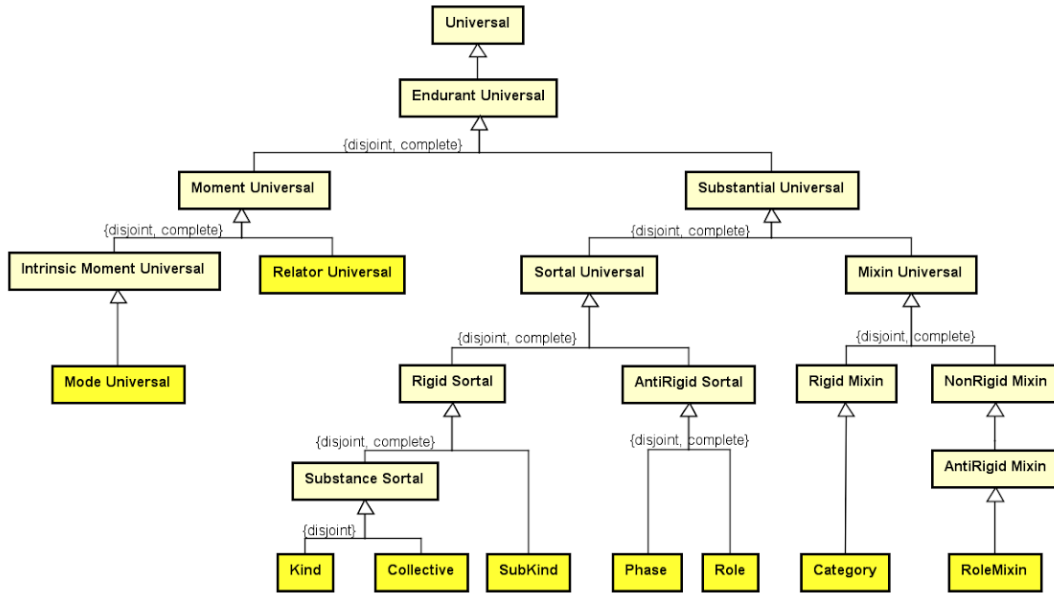
Figure 4. A UFO-A Fragment (Endurant Universals).

Within the category of **Sortal Universals**, we have the distinction between rigid and anti-rigid universals. A rigid universal is one that classifies its instances necessarily (in the modal sense), i.e., the instances of that universal cannot cease to be so without ceasing to exist. Anti-rigidity, in contrast, characterizes a universal whose instances can move in and out of its extension without altering their identity. For instance, contrast the rigid universal *Person* with the anti-rigid universals *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*, depending on whether he *enrolls in/finishes college* or *marries/divorces*, respectively.

Figure 4 depicts the Sortal distinctions in UFO. **Kinds** are sortal rigid universals that provide a uniform principle of identity for their instances (e.g., *Person*). **Subkinds** are sortal rigid universals that carry the principle of identity supplied by a unique **Kind** (e.g., a Kind *Person* can have the Subkinds *Man* and *Woman* that carry the principle of identity provided by *Person*).

Concerning anti-rigid sortals, we have the distinction between Roles and Phases. **Phases** are relationally independent universals defined as a partition of a sortal. This partition is derived based on an intrinsic property of that universal (e.g., *Child* is a phase of *Person*, instantiated by instances of person who are less than 12 years). **Roles** are relationally dependent universals, capturing relational properties shared by instances of a given kind, i.e., putting it baldly: 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** at the top [9].

The relational dependence of **Roles** is manifested by the presence of a **Relator** in the model. **Relators** are individuals with the power of connecting entities. For example, an *Enrollment* relator connects a *Student* role with an *Educational Institution*. 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 relation that take place between a relator universal and the object classes it connects is termed *mediation* (a particular type of *existential dependence* relation) [9].

Non-Sortals or **Mixins** are universals that aggregate properties that are common to different Sortals, i.e., that ultimately classify entities that are of different **Kinds**. Non-Sortals do not provide a uniform principle of identity for their instances; instead, they just classify things that share common properties but which obey different principles of identity. *Furniture* is an example of Non-Sortal (a Category) that aggregates properties of *Table*, *Chair* and so on. Other examples include Works of Art (including paintings, music compositions, statues), insurable items (including works of arts, buildings, cars, body parts, etc.) and social and legal objects (including people, organizations, contracts, legislations, etc.).

The meta-properties of rigidity and anti-rigidity can also be applied to distinguish different types of **Non-Sortals (Mixins)**. A **Category** represents a rigid and relationally independent mixin, i.e., a dis-

persive universal that aggregates essential properties that are common to different rigid sortals [9] (e.g., *Physical Object* aggregates essential properties of *Table*, *Car*, *Glass*, etc). A **RoleMixin** represents an anti-rigid and externally dependent Non-Sortal, i.e., a dispersive universal that aggregates properties that are common to different **Roles** *(*e.g., *Customer* that aggregates properties of *Individual Customer* and *Corporate Customer)* [9].

As discussed in depth in [9], by using the language engineering method proposed in [19], UFO-A has been employed in the design of an ontologically well-founded version of UML 2.0 termed OntoUML. OntoUML has modeling constructs that reflect all the leave categories in the hierarchy of Figure 4. Moreover, its metamodel contains a number of formal constraints derived from the axiomatization of UFO that prescribe that rules that govern the allowed combination of these constructs. These rules constrain possible combination of constructs in subsumption hierarchies (e.g., an anti-rigid universal cannot be a supertype of a rigid universal; a sortal universal cannot be a subtype of a mixin universal; every sortal is either a kind or a direct or indirect subtype of a unique kind); reinforce the necessary disjointness between instances of certain modeling constructs (e.g., all kinds are mutually disjoints; all phases of a given kind must appear in a disjoint, complete generalization set specializing that kind); and reinforce the existence of relators representing the relational dependence of types such as Roles and RoleMixins [9].

The representation of the axiomatization of UFO-A that gave rise to the OntoUML metamodel was firstly presented in [9]. In [20], however, we initiated a trend of looking at OntoUML as a pattern language, i.e., we realized that the modeling primitives of the language are actually higher-granularity building blocks (ontology patterns) that reflect the different ontological micro-theories in UFO. In that initial paper, however, we merely illustrated this approach by recognizing the correspondence between content patterns in OntoUML and micro-theories in UFO. Moreover, that paper is limited to discussing only the sortal fragment of UFO-A. In the next section, as the first contribution of this paper, we present a version of OntoUML in terms of intuitive form of graph grammar that describes OntoUML as a pattern language.

3   Deriving FOPs from the Unified Foundational Ontology

A Foundational Ontology Pattern (FOP) in OntoUML is a structure that reflects the micro-theories [17] put forth by its underlying foundational ontology (UFO). As discussed in [11], UFO is a system of micro-theories addressing basically all the classic conceptual modeling concepts including: Types and Taxonomic Structures [21], Attributes and Weak Entities [22], Datatypes [23], Relations [20, 24], Parthood [25, 26, 27], Events [28], Higher-Order types (Powertypes) [29], among others. For each of the ontological distinctions present in UFO and which are reflected as modeling constructs in OntoUML, we have a corresponding axiomatization. This axiomatization makes that OntoUML constructs can only appear in a model forming clusters of constructs with their ties and associated constraints. In other words, in general purpose languages such as ER, UML or OWL, the actual modeling building blocks of the language are low-granularity modeling primitives such as class, association, attribute, etc. In OntoUML, in contrast, the actual modeling primitives are these structures (and their corresponding axiomatization) reflecting the underlying ontological micro-theories. As a consequence, OntoUML is a *pattern language* whose models are constructed via the combined instantiation of the foundational patterns.

We here present a nearly complete fragment of OntoUML, leaving out a few constructs that are not germane for the purposes of this paper. Table 1 and 2 present the FOPs comprising this fragment of OntoUML. Table 1 presents the language in the form of sort of a "Graphical EBNF[1]", while Table 2 shows the patterns templates with example fragments. The interpretation of this grammar description is rather intuitive and shall become clear in its description that follows. We decided to use this intuitive representation (as opposed to some existing formal system of graph grammar description, e.g., a formal graph-rewriting system) to keep the paper self-contained.
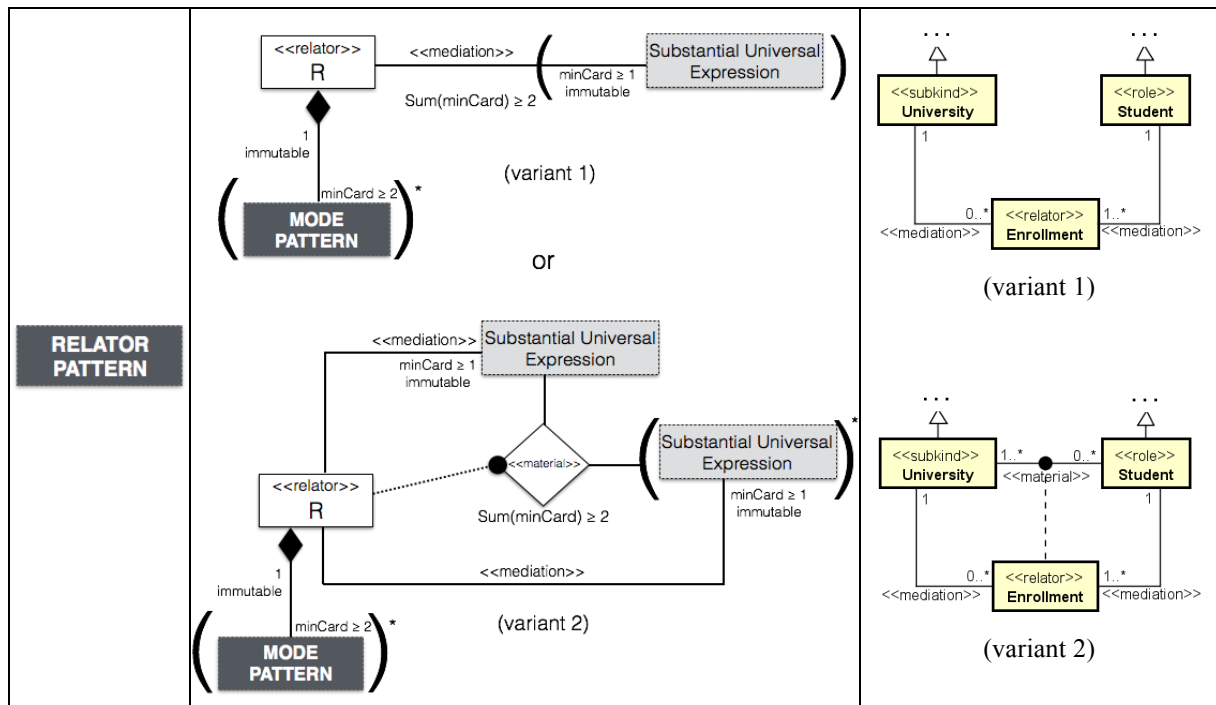
---

[1] EBNF: Extended Backus–Naur Form

Table 1. A Graph Grammar Description of OntoUML as a Pattern Language.

Expression	Expression Structure
OntoUML Structural Model	(Endurant Universal Expression)$^{1..*}$
Endurant Universal Expression	Substantial Universal Expression **or** Moment Universal Expression
Substantial Universal Expression	Sortal Expression **or** Mixin Expression
Sortal Expression	Rigid Sortal Expression **or** Anti-Rigid Sortal Expression
Rigid Sortal Expression	Substance Sortal Expression **or** SUBKIND PATTERN
Substance Sortal Expression	<<kind>> T **or** COLLECTIVE PATTERN
Anti-Rigid Sortal Expression	PHASE PATTERN **or** ROLE PATTERN
Mixin Expression	CATEGORY PATTERN **or** ROLEMIXIN PATTERN
Moment Universal Expression	MODE PATTERN **or** RELATOR PATTERN
Relationally Dependent Universal Expression	ROLE PATTERN **or** ROLEMIXIN PATTERN

Table 2. OntoUML Patterns' Templates and Examples.

Pattern	Pattern Template	Application Example Fragment
SUBKIND PATTERN	(variant 1) <<subkind>> T ▷ Rigid Sortal Expression **or** (variant 2) (<<subkind>> T_i)$^{2..*}$ {disjoint, [complete]} ▷ Rigid Sortal Expression	<<kind>> Person {disjoint, complete} <<subkind>> Man / <<subkind>> Woman (variant 2)
COLLECTIVE PATTERN	<<collective>> T ◇ <<memberOf>> (Endurant Universal Expression)$^{1..*}$ Sum(minCard) ≥ 2	<<collective>> Group 0..* <<memberOf>> 2..* <<kind>> Person
PHASE PATTERN	(<<phase>> T_i)$^{2..*}$ disjoint complete ▷ Sortal Expression	<<kind>> Person {disjoint, complete} <<phase>> Child / <<phase>> Adolescent / <<phase>> Adult

Pattern	Pattern Diagram	Example
ROLE PATTERN	Sortal Expression ▷ <<role>> T — RELATIONAL DEPENDENCE PATTERN	<<kind>> Person ▷ <<role>> Student 1 <<mediation>> 1..* <<relator>> Enrollment <<mediation>> ...
CATEGORY PATTERN	<<category>> T ▷ Mixin Expression (variant 1) **or** <<category>> T ▷ {disjoint} (Rigid Sortal Expression)2..* (variant 2)	<<category>> Artifact ▷ {disjoint} <<kind>> Document <<kind>> Model <<kind>> Software Item (variant 2)
ROLEMIXIN PATTERN	<<rolemixin>> T — RELATIONAL DEPENDENCE PATTERN ▷ ROLEMIXIN PATTERN (variant 1) **or** Sortal Expression ▷ <<rolemixin>> T — RELATIONAL DEPENDENCE PATTERN {disjoint, complete} (<<role>> T')2..* (variant 2)	<<rolemixin>> Customer <<material>> ... ▷ <<kind>> Person <<kind>> Organization {disjoint, complete} <<role>> Personal Customer <<role>> Corporate Customer (variant 2)
MODE PATTERN	Endurant Universal Expression <<inheresIn>> 1 immutable <<mode>> T (Endurant Universal Expression)* minCard ≥ 1 immutable <<externallyDependsOn>>	<<kind>> Person 1 inheres in 0..* <<mode>> Language Fluency 0..* <<kind>> Language 1 externally depends on
RELATIONAL DEPENDENCE PATTERN	Relationally Dependent Universal Expression <<mediation>> minCard ≥ 1 immutable RELATOR PATTERN (variant 1) **or** Relationally Dependent Universal Expression ◇ Substantial Universal Expression (variant 2) **or** Relationally Dependent Universal Expression ◇ Substantial Universal Expression (variant 3)	... ▷ <<role>> Student 1 <<mediation>> 1..* <<relator>> Enrollment ... <<mediation>> (variant 1)

As Table 1 shows, an OntoUML structural model is a non-empty set of **Endurant Universal Expressions**, each of which is either a **Substantial Universal Expression** or a **Moment Universal Expression**. A **Moment Universal Expression** is either an occurrence of the **Mode Pattern** or an occurrence of the **Relator Pattern**.

The **Mode Pattern** represents a Mode Universal connected to an **Endurant Universal Expression** via an existential dependence (*inherence*) relation. This **Endurant Universal Expression** is then used to describe the universals whose instances are the bearers of the instance of this mode universal. Since a Mode can be an *externally-dependent mode*, the Mode Pattern also contains a (possibly empty) set of relationships of *external dependence* connecting the instances of the mode universal at hand with their sources of external dependence (e.g., it connects a commitment with the entities which are referred by this commitment).

The **Relator Pattern** has two variants. In the first variant, we have a representation of relators connected via mediation relations (i.e., existential dependence relations) to possibly a number of substantial universals whose instances are entities mediated by this relator. In a second (and more complete) version of this pattern, we also have the description of a material relation derived from this relator universal and which is established between the relata mediated by the instances of that relator universal. For instance, we could have the relation of *married-with* derived from the relator universal *Marriage* and connecting the endurant universals *Husband* and *Wife*, whose instances, in turn, are mediated by instances of *Marriage*. As represented in the **Relator Pattern**, the material relation at hand can be of any arity (obviously, higher than two). In both variants of this pattern, we can optionally specify the connection between a relator and the modes that constitute this relator by recursively defining them via the **Mode Pattern**. In Figure 5 below, we show another depiction of the **Relator Pattern** in its second variant. In this case, we have a binary material relation with the two relata universals involved (F and G). As this figure illustrates, the cardinality constraints of the material relation and of the derivation relation are drastically constrained by the cardinality constraints of the mediation relations in the pattern (see detailed discussion in [9, 17]).
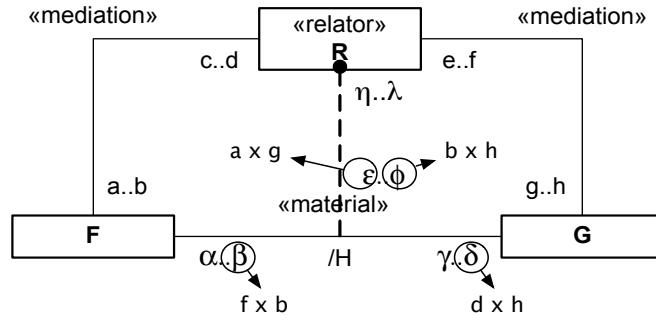
Figure 5. A representation of the Relator Pattern with dependencies between cardinality constraints.

A **Substantial Universal Expression** can be either a **Sortal Expression** or a **Mixin Expression**. The former can be either a **Rigid Sortal Expression** or an **Anti-Rigid Sortal Expression**.

A **Rigid Sortal Expression** is either a **Substance Sortal Expression** or an occurrence of the **Subkind Pattern**. A **Substance Sortal Expression** describes the *identity provider* universals of an OntoUML model, which can be either a **Kind** or a **Collective** (indirectly specified via the **Collective Pattern**). A **Kind** is the only terminal symbol of this grammar. In other words, a recursive thread of definitions of these expressions only stops in an OntoUML model when a **Kind** construct is reached.

The **Collective Pattern** describes a Collective Universal and the universals whose instances are members of these collectives. Due to the so-called *Weak Supplementation Principle*, required for all parthood relations [25], the sum of the minimum cardinality constraint on the side of the members must be equal or higher than 2.

The **Subkind Pattern** appears in two variants. In the first of these, we have simply a subkind specializing a **Rigid Sortal Expression**; in the second, we have a subkind generalization set collecting a disjoint (and optionally complete) set of subkinds that specialize the same universal, once more, described by a **Rigid Sortal Expression**. A subkind can only specialize a rigid sortal. The recursive definition of these patterns guarantees that a subkind either directly or indirectly specialize a substance sortal that provides a uniform principle of identity for its instances.

An **Anti-Rigid Sortal Expression** is either an occurrence of the **Phase Pattern** or an occurrence of the **Role Pattern**. The **Phase Pattern** consists of a phase partition, i.e., a disjoint and complete set of two or more complementary phases that specialize the same sortal, which is specified by a **Sortal Expression**. Notice that, once more, the recursive definition of this pattern guarantees that a substance sortal providing a common principle of identity for the instances of these phases is always specified in the model. Analogously, in the **Role Pattern**, we have a role that specializes a sortal universal (again, specified by a **Sortal Expression**). However, since roles are relationally dependent universals, we have here also that a role must be part of an occurrence of the **Relational Dependence Pattern**.

The **Relational Dependence Pattern** is a complex pattern that, on one hand, describes the relational dependence condition of a relationally dependent universal (i.e., either a Role or a RoleMixin), which, in turn, is specified by a **Relationally Dependent Universal Expression**. This relational dependence conditions is captured either: (i) via a connection to (one of the variants of) the **Relator Pattern**, in which the relationally dependent universal at hand appears as one of the mediated types; (ii) via a parthood relation, in which the relationally dependent universal at hand appears either as a part of or a whole universal. A **Relationally Dependent Universal Expression** is either an occurrence of the **Role Pattern** or an occurrence of the **RoleMixin Pattern**.

A **Mixin Expression** can be either an occurrence of the **RoleMixin Pattern** or an occurrence of the **Category Pattern**. A RoleMixin ultimately captures common contingent and relationally dependent properties of entities of multiple Kinds. In other words, a RoleMixin can be seen as an abstraction capturing common characteristics of roles played by entities of different Kinds. The **RoleMixin Pattern** appears in two variants. One of them (variant 2) defines a RoleMixin by a partition of two or more Roles, each of which is connected to a kind (directly or indirectly) via a **Sortal Expression**. The common relational dependence of these roles is captured by connecting the RoleMixin to an occurrence of the **Relational Dependence Pattern**. Finally, a RoleMixin can (optionally) appear in a model recursively applying the **RoleMixin Pattern**, i.e., specializing another RoleMixin with its associated relational dependence (variant 1).

Finally, a Category ultimately captures common essential properties of entities of multiple Kinds. The **Category Pattern** represents this by either: (i) directly having a Category as a common abstraction of two or more disjoint **Rigid Sortal Expressions** (variant 2); (ii) indirectly having a Category as a common abstraction of another **Mixin Expression** (either another recursive occurrence of the **Category Pattern** or an occurrence of the **RoleMixin Pattern**), which will, in turn, eventually be connected to set of **Sortal Expressions**.

In order to briefly illustrate how a valid OntoUML structural model can be built using the grammar just described, we use the example of Figure 6. Suppose that a modeler initiates this model by instantiating the **RoleMixin Pattern** (area 1 in the figure). She then creates the RoleMixin universal *Customer* and a partition of Roles defined by the universals *Personal Customer* and *Corporate Customer*. Each of these roles is a specialization of a corresponding Sortal Expression (areas 2 and 3 in the figure). Following the **RoleMixin Pattern**, *Customer* must also be connected to an occurrence of the **Relator Pattern** (area 4). In area 2, we have a **Sortal Expression** that is an occurrence of the **Phase Pattern**. In this occurrence of this pattern, we have a phase partition containing phases *Living Person* and *Deceased Person*, which in turn specialize another **Sortal Expression**. This latter **Sortal Expression** is the definition of the Kind *Person*. Analogously, in area 3, we have a **Sortal Expression** that subsumes *Corporate Customer*, which is again another occurrence of the **Phase Pattern**. In this case, we have a phase partition containing phases *Active Organization* and *Extinct Organization*, which in turn specialize another **Sortal Expression**, namely, a **Substance Sortal Expression** of the Kind *Organization*. In area 4, we have the occurrence of variant 2 of the **Relator Pattern**, in which we have the definition of the relator universal *Purchase Contract* that mediates the RoleMixin *Customer* as another **Substantial Sortal Expression**, namely, another occurrence of the **Role Pattern**. In this new occurrence of the **Role Pattern**, we have the definition of the role universal *Supplier*, whose relational conditional is already satisfied by the aforementioned occurrence of the **Relator Pattern** with *Purchase Contract*, and whose subsuming type condition (a **Sortal Expression**) is already satisfied by an occurrence of the **Phase Pattern** containing *Active Organization*. By applying these patterns in areas 1-4, we have a grammatically correct OntoUML model and one in which all instances of the model have a unique principle of identity that they obey (provided by the object kinds *Person* and *Organization* and the relator universal *Purchase Contract*).
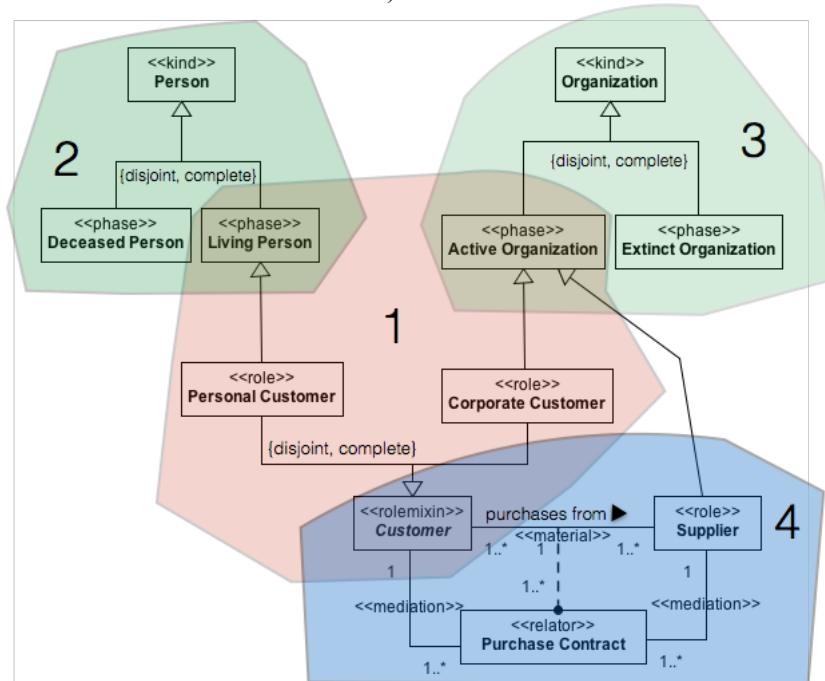

Figure 6. An example of an OntoUML model as a combination of FOPs.

4   Extracting DROPs from Core and Domain Ontologies

While FOPs are focused on foundational structural aspects, domain aspects are the drivers for deriving DROPs. The main rule for a DROP is to represent a recurrent modeling fragment in a given domain. Thus, DROPs are extracted from core and domain ontologies by packaging the core knowledge of a target domain in meaningful reusable fragments.

4.1   Deriving DROPs from Core Ontologies

Core ontologies are important sources of DROPs, since they describe the core knowledge of a wide domain that spans across different subdomains. Their models contain fragments of knowledge that can be reused when modeling more specific domain ontologies. The main issues related to extracting DROPs from core ontologies are how to achieve these fragments in a proper way and how to relate them. With a core ontology in hands, some of its information can be used for supporting a fragmentation process for creating DROPs. In particular, competency questions (CQs) [30] can reveal modeling needs in small (and still connected) pieces. Therefore, a general process comprising four steps and providing some guidelines is presented in the sequel.

A. Modularize the core ontology.
If the core ontology to be fragmented in DROPs is a complex one, we should start by splitting it into modules (sub-ontologies), for example, using an approach of Ontology Partitioning [31]. As pointed out by d'Aquin [31], there is no universal way to modularize an ontology and the choice of a particular approach should be guided by the ontology requirements. The general approach proposed by d'Aquin in [31] can be applied for this purpose. The same can be said for techniques and criteria for ontology partitioning, such as the ones proposed in [32]. Regarding the criteria, we suggest considering at least the following: independence, cohesion, and size. If the core ontology is already modularized, this step can be skipped.

B. Fragment each sub-ontology model into small pieces still meaningful for the domain.
Consider splitting a sub-ontology into even smaller fragments suitable to be considered DROPs. By looking again at ontology requirements, we can look at each sub-ontology and search from lower-granularity reusable fragments. A pattern, in general, describes a particular recurring problem that arises in specific contexts and presents a well-proven solution for the problem [33]. Thus, the process of extracting DROPs from a sub-ontology should start by looking for the problems being addressed by fragments of the sub-ontology. If the core ontology is guided by the competency questions (CQs) that it aims answering, then these CQs are the natural guide for driving the process of splitting the sub-ontology into DROPs. As a starting point, we suggest defining one DROP for each CQ. However, it is important to keep in mind that CQs and DROPs have different concerns. While CQs aim at defining the ontology requirements, DROPs are focused on finding the best configuration for being recurrently applied in the domain. Thus, a one-to-one relation (with each CQ attached to a unique DROP and vice-versa) could not be the best organization for all the fragments, and should be reviewed as described in the next step. Moreover, when the core ontology CQs are not available, some reengineering may be needed, for making explicit the ontology requirements before trying to fragment the ontology model into DROPs.

   The complexity of the fragments can vary depending on the problem/solution that they are addressing. Sometimes, a fragment that is a candidate for becoming a DROP contains only two related concepts; in other situations, it can contain a complex combination of concepts and relations. Sometimes the same fragment gives rise to two (or more) variant and alternative patterns; and sometimes a pattern is structurally open (Partial DROP) in order to be completed by another DROP or FOP. Complete self-contained fragments are candidates for Complete DROPs; fragments that need to be completed by other DROPs or FOPs are said Partial DROPs.

C. Review the model fragments and select the DROPs
Fragments too big, addressing more than one modeling problems (for example, connected to unrelated CQs), should be analyzed to check if they would be better handled in distinct DROPs. Domain aspects, as well as foundational aspects should be taken into account in this analysis. For example, sup-

pose that the problems being addressed by the fragment are too interrelated, so that the resulting DROPs should always to be used in conjunction. In this case, it is better to maintain the whole fragment in a single DROP. Otherwise, i.e., if each of the two DROPs can be used independently of each other, it is better to break the fragment into two DROPs.

Fragments too small should also be analyzed to check if they really address relevant problems. If not, then the fragment does not deserve to become a DROP. Moreover, if the problem being addressed is too interrelated to another problem, then consider merging it with the other DROP on which it depends heavily.

A complementary approach is to use FOPs for helping to define the boundaries of DROPs. Since a FOP is applied to solve general modeling problems, several of the domain problems solved by DROPs can be mapped to an underlying FOP structure. Thus, often, a DROP is delimited as the application of a FOP for solving a domain-related problem. This is, however, not a strict rule, since a DROP can be structurally open in a way that it should be completed by another DROP or FOP (thus not directly fitting in any FOP). In other cases, a DROP can apply more than one FOP.

Alternative and useful variants of the same fragment can also be considered. Frequently, a modeling fragment can be represented in different ways, depending on possible variations on what an ontology engineer may want to represent in the target domains. The existence of variants for the same ontology pattern was already illustrated for FOPs in Section 3. To cite one example, relational properties between two entities can be modeled: by simply representing a *relator* with the associated *mediation* relations connecting the relata; or by a *relator*, with associated *mediation* relations and with a derived *material relation* connecting the relata. In an analogous manner, different domain-level modeling problems can give rise to different alternative DROPs.

D. Pack the DROP with its associated useful information.
Information for locating, understanding and using the DROP needs to be attached to it. This includes: name, intent, rationale, CQs addressed by the DROP, the conceptual model fragment, axioms, related COPs (mandatory or optional), and definitions of the types of entities considered in the DROP. It is useful to create a Pattern Specification as shown in Table 3.

4.1.1 Engineering *for Reuse*: Extracting DROPs from a Service Core Ontology

In order to illustrate this fragmentation process resulting in a set of DROPs, we present an application of the proposed approach to a complex reference ontology, namely, the UFO-based Core Ontology for Services (UFO-S) [10]. UFO-S was designed for addressing the notion of service broadly, harmonizing different service perspectives. As a core ontology, it can be reused for representing a variety of specific domains concerning services (such as Car Rental, Hostage, Training and Cable TV Services). UFO-S is divided into 3 sub-ontologies concerning the domain aspects it represents: *Service Offering*, *Service Negotiation and Agreement*, and *Service Delivery*. In the sequel, we present the CQs for the *Service Offering* sub-ontology (CQ01-CQ04) and *Service Negotiation and Agreement* sub-ontology (CQ05-CQ13). Figures 7 and 8 present the corresponding conceptual models of these UFO-S sub-ontologies.

CQ01. What is a service offering?
CQ02. Who is involved in a service offering?
CQ03. Which are the descriptions of a service offering?
CQ04. Which are the terms and conditions of a service offering?
CQ05. What is a service negotiation?
CQ06. Who is involved in a service negotiation?
CQ07. To which service offering does a service negotiation regard?
CQ08. What is a service agreement?
CQ09. Who is involved in a service agreement?
CQ10. To which service offering does a service agreement conform?
CQ11. From which service negotiation did a service agreement result?
CQ12. Which are the descriptions of a service agreement?
CQ13. Which are the commitments and claims of the parties involved in a service agreement?
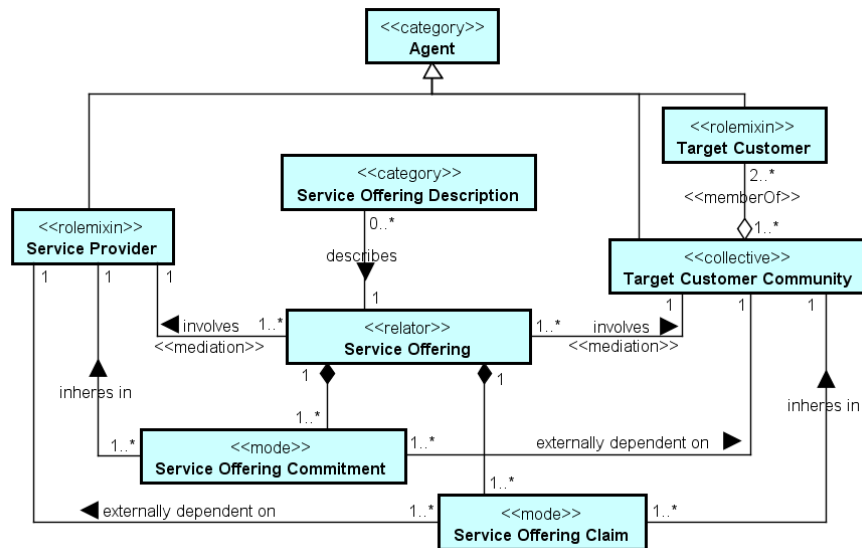
Figure 7. UFO-S Service Offering sub-ontology (adapted from [10]).

The Service Offering sub-ontology represents how a **Service Offering** is offered from a **Service Provider** to a **Target Customer Community**. A **Target Customer Community** has **Target Customers** as members. A **Service Offering** can be described by **Service Offering Descriptions**. An offering is composed of **Service Offering Commitments** from the **Service Provider** towards the **Target Customer Community** and the corresponding **Service Offering Claims** from the **Target Customer Community** towards the **Service Provider**. For example, a *Taxi Driver* offers *Taxi Services* to a *Community*, and has the *Commitment* to take a *Customer* from a place to another receiving a payment for that, while the *Customers* have the complementary *Claim* of being taken from one place to another paying for that.
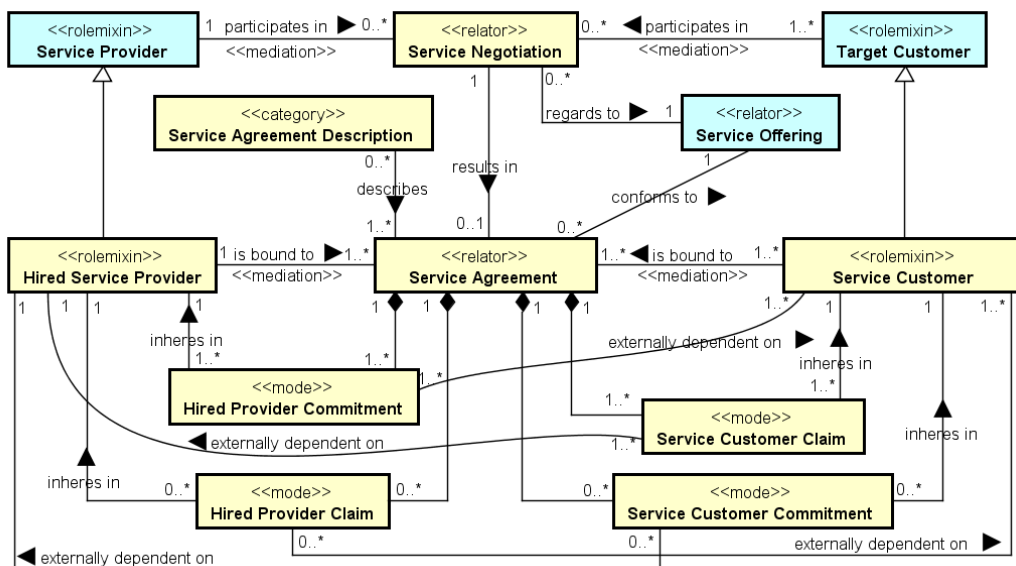
Figure 8. UFO-S Service Negotiation and Agreement sub-ontology (adapted from [10]).

The Service Negotiation and Agreement sub-ontology depicts the negotiation of an offering towards a possible agreement between the involved parts. A **Service Negotiation** is an interaction involving the participations of a **Service Provider** and **Target Customers**. When the negotiation succeeds, it results in a **Service Agreement** between the parts, acting as **Hired Service Provider** and **Service Customer**, respectively. Analogously to the **Service Offering**, a **Service Agreement** can have **Service Agreement Descriptions** and is composed of the **Hired Provider Commitments** and **Claims**, as well as the **Service Customer Commitments** and **Claims**.

Since UFO-S is already modularized into sub-ontologies, and the related CQs are provided, the first step for the fragmentation process is dividing its conceptual modules into meaningful fragments. For example, by analyzing the Service Offering sub-ontology and how each CQ is solved, we have:

- *CQ01. What is a service offering?* This is a general question and its answer is given mainly by the **Service Offering** concept. A Service Offering, as a social relator, cannot be detached from the involved parties, and thus we cannot answer this CQ without answering also CQ2.

- *CQ02. Who is involved in a service offering?* This question is addressed by a fragment relating **Service Offering** to **Service Provider** and **Target Customer Community** and the **Target Customer** members. **Agent** is included as generalization of provider, community and customer. Thus, the model fragment including these five concepts and the relations between them comprises a meaningful DROP named SOffering (Figure 9 (a)). SOffering should always be used when an ontology engineer is interested in describing a Service Offering.

- *CQ03. Which are the descriptions of a service offering?* This question is addressed by a fragment relating **Service Offering** to **Service Offering Description**. It is worth noting that there are cases in which the ontology engineer may not be interested in representing service offering descriptions in her ontology. Thus, it is not recommended to include the **Service Offering Description** concept in the SOffering pattern. Thus, the model fragment relating a service offering to its descriptions is better handled as another DROP, called SODescription (Figure 9 (b)).

- *CQ04. Which are the terms and conditions of a service offering?* This question is addressed by a fragment defining **Service Offering Commitment** and **Service Offering Claim** as parts of **Service Offering**, plus representing **Service Provider** and **Target Customer Community** as the bears and external dependencies of the commitments and claims. As pointed by Guizzardi et al. [7], commitments and claims always form a pair that refers to a unique propositional content. From a practical point of view, however, the ontology engineer, while modeling a specific domain, may want to represent only the commitments, only the claims, both, or none of them. Thus, it is better to split the fragment involving these concepts, creating two DROPs, namely SOCommitments (Figure 9 (c)) and SOClaims (Figure 9 (d)).
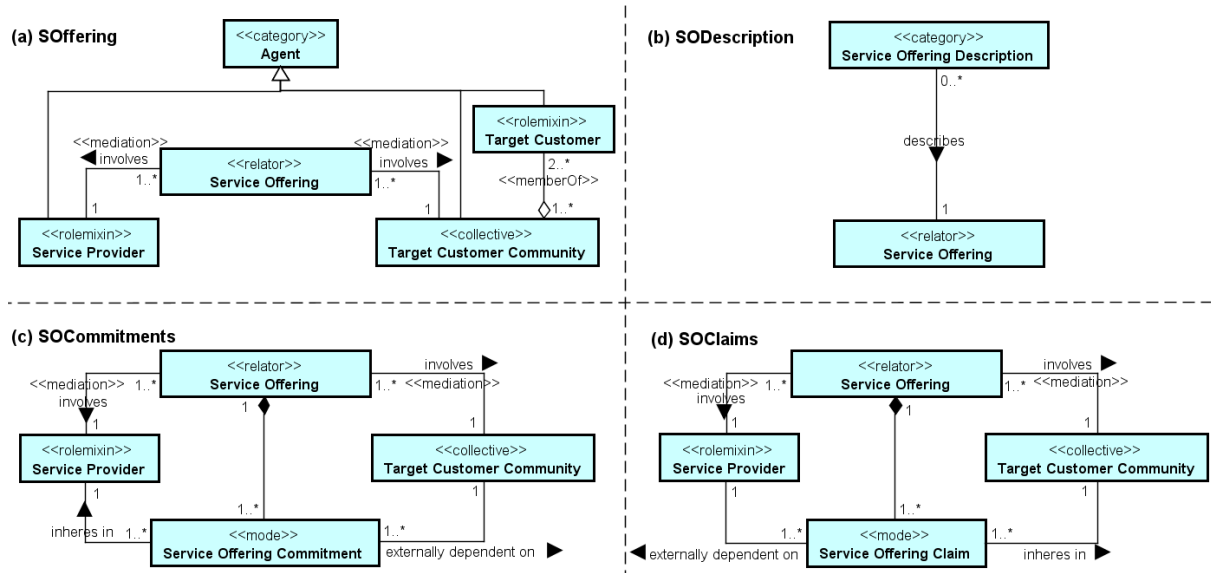

Figure 9. DROPs extracted from the Service Offering sub-ontology (adapted from [34]).

Concerning the foundations for the SOffering DROP (Figure 9 (a)), we should highlight that this DROP contains a complete application of the following FOPs: **Relator Pattern, Collective Pattern** and **Category Pattern** (see Table 2). It also includes two *rolemixins* without any specialization, generalized into a *category*. One should notice that the specification of these rolemixins does not fully comply with the **RoleMixin Pattern** in Table 2. This, however, does not constitute a problem at this stage, since the idea is to let these fragments be completed only in the specific domain ontology. For this reason, this DROP is said a Partial DROP, i.e., the Rolemixin FOP should be applied in its integ-

rity to achieve a complete and well-founded domain ontology. On the other hand, by not committing to a full specification of this DROP at this stage, this approach gains in flexibility by allowing diverse types of service providers and target customers to be specified when the DROP is reused in specific domain ontologies.

Regarding the SODescription DROP (Figure 9 (b)), we should point out that, although it involves only two concepts linked by a formal relation, it depicts a useful model fragment of the domain, being thus a good choice for a DROP.

Finally, the last two DROPs (SOCommitments (Figure 9 (c)) and SOClaims (Figure 9 (d))) are, in fact, derived from the Social Relator FOP, extracted from UFO-C. This pattern is a combined application of the **Relator Pattern** and the **Mode Pattern** (see Table 2) for describing commitments and claims that comprise a social relator. A Social Relator involves at least two agents and it is composed of pairs of commitments and claims. These, in turn, are social moments that inhere in an agent and are externally dependent on the other agent involved in the social relator. We should highlight that, as previously mentioned, we decided to split the manifestation of this FOP in UFO-S in two DROPs, since the ontology engineer, while modeling a specific service domain, may want to represent only the commitments, only the claims, both, or none of them.

Although each DROP represents a distinct aspect of the core domain knowledge, there are a number of possible relations that can occur between them. For instance, some DROPs partially overlap. This is the case of the **Service Offering** concept, which is shared by all the presented DROPs, as well as the concepts of **Service Provider** and **Target Customer Community**, which are part of three of the four DROPs in Figure 9. Another situation is illustrated by the SODescription DROP, which aims at describing a service offering, typically including information about the service provider and customers. Thus, this DROP should be dependent on the SOffering DROP (SOffering should be applied before SODescription). Similar rationale applies to the SOCommitments and SOClaims DROPs, which are typically used in consequence of establishing a service offering (SOffering DROP). These characteristics show that DROPs typically bear strict relations to each other and, thus, are very often applied in combination or even in sequence. Considering the *Service Negotiation and Agreement* sub-ontology, the model presented in Figure 8 has nine related CQs (CQ05 – CQ13). Performing the fragmentation process, we have followed a similar rationale of the work done in the case of the *Service Offering* sub-ontology in order to derive DROPs. The results and complementary rationale are discussed in a brief manner in the sequel. Some of the DROPs extracted from this sub-ontology are presented in Figure 10.

The SNegotiation pattern (Figure 10 (a)) comes from three CQs: CQ5 (*What is a service negotiation?*), CQ6 (*Who are involved in a service negotiation?*) and CQ7 (*To which service offering does a service negotiation regard?*). CQ5 is general, and it is answered in tandem with CQ6 and CQ7. It is important to keep the Service Offering concept in the same fragment of Service Negotiation concept. This is because there is no negotiation disconnected from an offering, i.e., the subject of a service negotiation in this domain is always a service offering.

The SAgreement DROP (Figure 10 (b)) comes from CQ8 (*What is a service agreement?*) and CQ9 (*Who are involved in a service agreement?*). It models the provider and customers involved in a Service Agreement, despite of any previous offering or negotiation. From this fragment, some variations are created, taking other CQs into account. For considering CQ10 (*To which service offering does a service agreement conform?*), the SOfferAgree pattern (Figure 10 (c)) is proposed as a variation of SAgreement. SOfferAgree also considers the Service Offering to which the agreement conforms (not taking any negotiation into account). Finally, for considering also CQ11 (*From which service negotiation did a service agreement result?*), the SNegAgree pattern is created, representing the whole situation where both the negotiation and agreement are important for the ontology being developed.

Besides the patterns discussed above, other five patterns were extracted from the Service Negotiation and Agreement sub-ontology, namely: SADescription, which is analogous to SODescription, but describing an agreement (CQ12); and HPCommitments, SCCommitments, HPClaims and SCClaims, which are analogous to SOCommitments and SOClaims, but addressing commitments and claims from the hired provider and the service customer (CQ13). Since these patterns are analogous to others already discussed in this paper, we do not elaborate further on them here. Their full description can be found in [34].
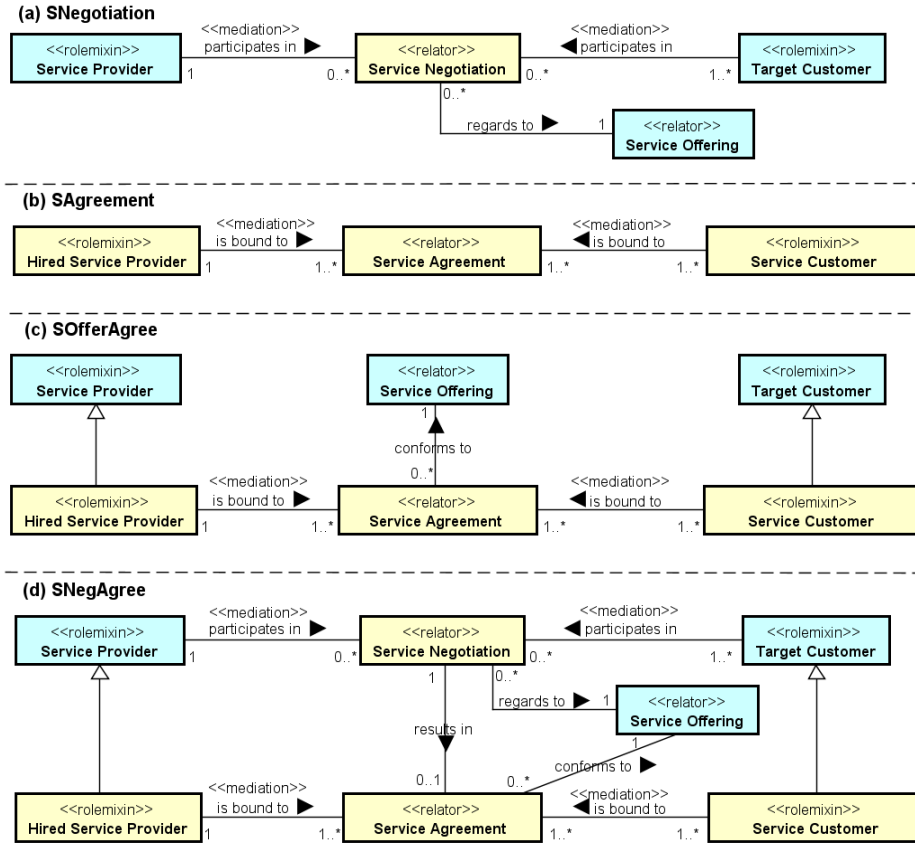
Figure 10. DROPs extracted from the *Service Negotiation and Agreement* sub-ontology (adapted from [34]).

Besides extracting these patterns from the core ontology, we have to develop a specification for each DROP. The DROP specification should include the following information:

- **Name**: the name of the DROP (complete and acronym).
- **Intent**: describes the pattern purpose.
- **Rationale**: describes the rationale underlying the pattern.
- **Competency Questions**: describes the CQs that the pattern aims to address. These CQs point out the modeling problem that is solved by the pattern.
- **Conceptual Model**: depicts the conceptual model representing the pattern elements. The conceptual model plus the concept definitions and axioms described in the next items capture the solution for the problem posed as CQs.
- **Concept Definitions**: definitions of the ontology concepts in natural language.
- **Axiomatization**: presents complementary formal axioms related to the DROP conceptual model. These axioms typically capture constraints and other aspects of the pattern that cannot be directly represented by the diagram depicting the conceptual model. We should mention that the axioms constraining the interpretation of the elements of the DROP come directly from the axiomatization of the corresponding elements in the core ontology from which the pattern has been extracted. We advocate that, in any case, they should be explicitly represented as part of the DROP's documentation.
- **Force:** indications regarding when to use or not use the pattern.
- **Related Patterns**: list other COPs (DROPs or FOPs) related to the pattern being presented, specifying the corresponding relation between these patterns (e.g., if the occurrence of the associated pattern is optional or mandatory).

As an example, Table 3 shows the specification of the SOfferAgree pattern. In this table, axioms are written in First Order Logics. Additional examples of DROPs extracted from UFO-S and possible combinations between them can be found in [34].

Table 3. The Specification of the SOfferAgree DROP.

Service Offering and Agreement

Name: Service Offering and Agreement (SOfferAgree)

Intent: Representing a *Service Agreement* in conformance with a *Service Offering*, without addressing service negotiation aspects.

Rationale: A *Service Agreement* is established between a *Hired Service Provider* and a *Service Customer*. The terms and conditions that may take part of a *Service Agreement* must be in conformance to those of the correspondent *Service Offering*.

Competency Questions:

- Which are the parties involved in a service agreement?
- To which service offering does a service agreement conform to?

Conceptual Model

Concept Definitions:

- **Service Offering**: A promise of the *Service Provider* to provide a service under certain conditions to a *Target Customer Community*.
- **Service Provider**: The role played by agents when these agents commit themselves to a *Target Customer Community* by means of a *Service Offering*.
- **Target Customer**: The role played by agents that are members of a *Target Customer Community*.
- **Service Agreement**: An agreement established between a *Hired Service Provider* and *Service Customers*, regarding a *Service Offering*.
- **Hired Service Provider**: The role played by a *Service Provider*, when the *Service Provider* commits itself to a *Service Customer* to perform actions or to achieve the results determined in the *Service Agreement*.
- **Service Customer**: The role played by a *Target Customer* that hires a service in the context of a *Service Agreement*.

(Partial) Axiomatization

A1: ∀ sa:ServiceAgreement, hsp, sc (isBoundTo(sa, hsp, HiredServiceProvider) ∧ isBoundTo(sa, sc, ServiceCustomer) → (hsp≠sc))

An agent cannot simultaneously play the roles of *Hired Service Provider* and *Service Customer* in the same *Service Agreement*.

A2: ∀ sa:ServiceAgreement, hsp (isBoundTo(sa, hsp, HiredServiceProvider) → (∃so: ServiceOffering (offers(hsp,so) ∧ conformsTo(sa,so)))

The *Service Provider* that is bound to a *Service Agreement* offers the *Service Offering* to which the agreement conforms.

Forces

- This pattern should be used if you need to model an agreement established between a **Service Provider** (who starts to play the role of **Hired Service Provider**) and a **Target Customer** (who starts to play the role of **Service Customer**) in conformance with a previously modeled **Service Offering**.
- If besides modeling the agreement and its conformance to a previously established offering, you need to model the negotiation between **Service Provider** and **Target Customer**, then you should not use this pattern, but the **SNegAgree** pattern.
- If you are interested only in modeling an agreement, disregarding the offering, then you should not use this pattern, but the **SAgreement** pattern.

Related Patterns

- The **SOffering** DROP should be applied before this for properly defining the involved offering.
- In a refinement specification for this DROP, the **RoleMixin** FOP should be fully applied for defining the allowed types of *Hired Service Providers* and *Service Customers* (e.g. Person, Organization etc.).

4.2   Deriving DROPs from Domain Ontologies

DROPs can also be extracted from Domain Ontologies. Although these ontologies are in the lower generality level, it is possible to observe very similar modeling problems in related domains. The modeling solutions for these problems, even containing domain peculiarities, can be analyzed to check if a pattern is manifested in them. If this is the case, the modeling solutions can be generalized as a DROP. Of course the identification of a potentially reusable fragment in a single domain ontology is not enough to make it a pattern. However, once assured that this pattern is indeed recurrent in ontologies of related domains, this fragment can be generalized and packaged as a new DROP.

Hence, in a nutshell, is the following: we first identify a model fragment that is significantly similar to other fragments already used to solve the same problem in other domains (in particular, the structure of the model fragments in the corresponding domain ontologies should be the same); then this fragment must be generalized in order to become applicable in similar situations in which it is useful. Finally, this general fragment can be packaged as a DROP, gathering all associated information, as explained in Subsection 4.1.

In order to show the extraction of DROPs from domain ontologies, let's take two domains related by their organizational characteristics: Software Organizations and Universities. Figure 11 presents portions of the ontologies for these domains, focusing on how people are involved in these two types of organizations.
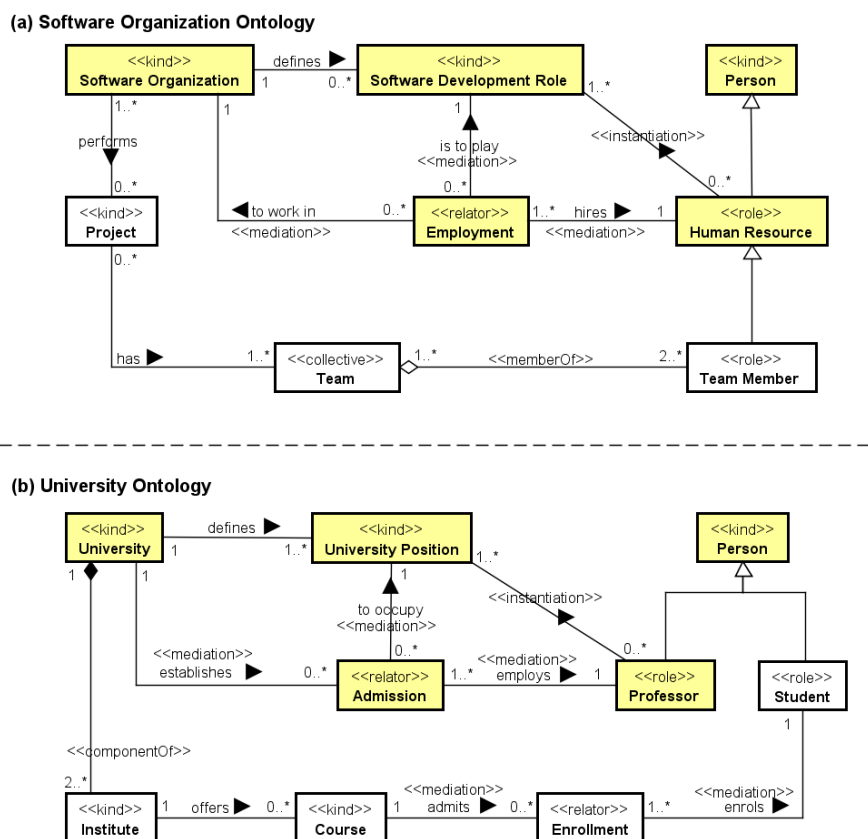

Figure 11. Model fragments from the Software Organization and University domain ontologies.

Considering these models fragments, we can see that, besides dealing with particularities of the related domains (such as projects and teams in software organizations, and students, courses and enrollments in universities), they address a problem shared by both domains: managing people in employments. Employment is a general modeling problem for the broader domain of organizations. A Software Organization employs Developers assigning to them Software Development Roles; A University admits Professors, for occupying University Positions. Generally observing the given models, one can see an Organization employing People for playing an Organizational Role. Thus, the model fragments shown in Figure 11 can be generalized giving rise to the Employment DROP shown in Figure 12. In

the Employment DROP, the **Employment** *relator* mediates **Organization**, **Person** (which plays the role of **Employee** in the context of an **Employment**), and **Organizational Role**. **Organizational Role** is a social role defined by and recognized by an **Organization**.
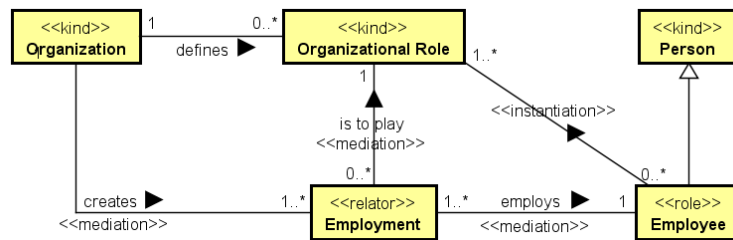

Figure 12. Employment DROP.

It is worth to mention that if a core ontology on organizations is available, an equivalent pattern could be extracted by the process explained in Subsection 4.1. Thus, the same resulting DROP can be achieved from two different ways: by fragmenting a core ontology, or by generalizing recurrent fragments of domain ontologies.

5 Using Conceptual Ontology Patterns in Building Reference Ontologies

Throughout the Ontology Engineering process, different types of pattern can be reused in a variety of ways. Our focus here is on reusing FOPs and DROPs for building reference core and domain ontologies. Figure 13 shows, on the left-hand side, the ontology generality levels, and on the right-hand side, the corresponding types of COPs that can be extracted and where they apply. We should emphasize that here we do not aim at defining a pattern-based Ontology Engineering method. However, any method that considers the application of patterns can be used (e.g., [4] and [35]).
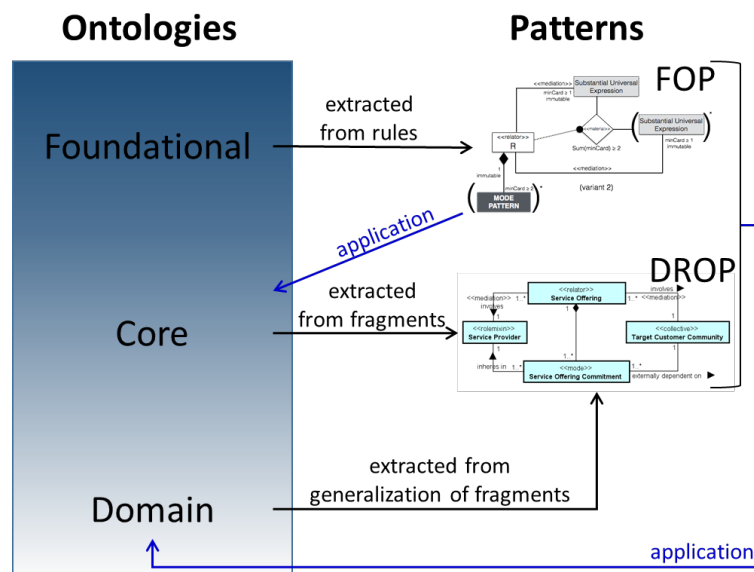

Figure 13. Conceptual Ontology Patterns Extraction and Application.

5.1 Engineering *with Reuse*: Building a Service Domain Reference Ontology by reusing DROPs and FOPs

In order to illustrate the combined application of DROPs and FOPs for building a reference domain ontology, let's consider the case of a reference ontology for the Car Rental domain, aiming at answering the following competency questions (CQs):

CQ01. How is a rent-a-car agency structured?
CQ02. What is a rent-a-car service offering?
CQ03. Who is involved in a rent-a-car service offering?
CQ04. How is a rent-a-car service offering described?
CQ05. Which are the terms and conditions of a rent-a-car service offering?
CQ06. What is a rent-a-car service agreement?
CQ07. Who is involved in a rent-a-car service agreement?
CQ08. To which rent-a-car service offering does a rent-a-car service agreement conform?
CQ09. Which car is located in a car rental?
CQ10. Who are the enabled drivers in a car rental?

These competency questions have been analyzed, and many of them are, in fact, adapted from competency questions associated to previously presented patterns, namely: CQ02 to CQ08. The reuse of CQs indeed has oriented the DROPs application during our modeling efforts here.

The first question, CQ01, does not deal with services, addressing organizations composed of organizational units. For modeling this, we have applied twice the first variant **Subkind Pattern** (see Table 2), creating only one specialization for each *kind*. Thus, considering the Car Rental domain, a *Car Rental Agency* is a *subkind* of the *Organization kind*, and is composed of *Car Rental Branch*, which is a *subkind* of the *Organizational Unit kind*. The two applications of this FOP application are shown in Figure 14 (E).

CQ02 and CQ03 deal with a car rental service offering and are addressed by the application of the SOffering DROP discussed in Subsection 4.1. Thus, the pattern is added to the domain model and its concepts and relations are extended, as shown in Figure 14 (A) and (B): *Car Rental Offering* is subtype of **Service Offering**, *Car Rental Provider* is a **Service Provider**, *Potential Car Rental Community* is a **Target Customer Community**, and *Potential Car Renter* is subtype of **Target Customer**. The relations are also specialized, resulting in the fragment: *Car Rental Provider* offers *Car Rental Offerings* to a *Potential Car Rental Community* composed of *Potential Car Renters*. Moreover, the application of SOffering requires the application of other patterns, because two modeling problems remain opened. Firstly, the *Car Rental Provider* and the *Potential Car Renter* are *rolemixins* that must be completed in the domain ontology to keep the model consistency. Thus, the **RoleMixin Pattern** (Table 2) should be applied in both cases in order to achieve a well-founded solution. The provider is a role that can be played by a *Car Rental Agency* or a *Car Rental Branch* (see Figure 14 (C)), reusing the *subkinds* previously defined. The renter is a role played by a **Person** or an **Organization** (see Figure 14 (D)).
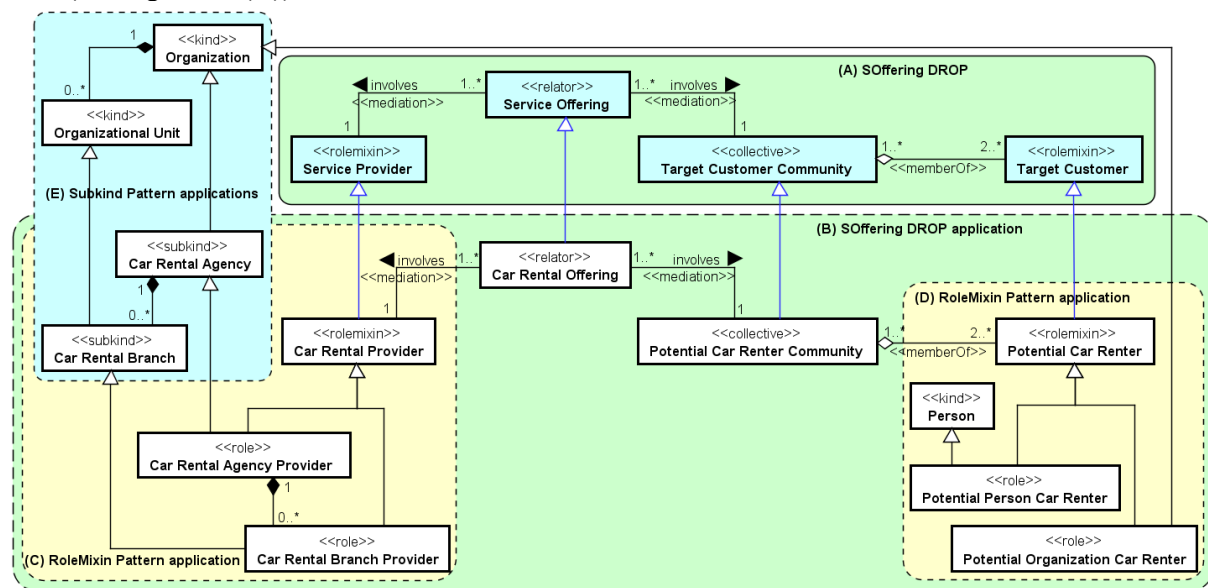

Figure 14. Combined application of DROPs and FOPs in the Car Rental domain.

The model fragment shown in Figure 14, addressing CQ01-CQ3, is useful to show the combined application of both DROPs and FOPs. There are two main perspectives. First, the application of the

SOffering DROP had to be combined with applications of the **RoleMixin Pattern**, in order to produce a complete consistent model fragment. It allows the definition of more flexible patterns, covering a larger range of situations, but still offering a reuse-oriented support for the ontology engineer. Second, as aforementioned, the SOffering DROP is, by itself, an application of following FOPs: **Relator Pattern, Collective Pattern** and **Category Pattern** (see Table 2). Thus, the foundational knowledge of a set of FOPs is inherited by a DROP, where core knowledge is added, and then the accumulated knowledge is transferred to the domain representation in a chain.

CQ04 (*How is a rent-a-car service offering described?*) is addressed by applying the SODescription DROP, where a ***Car Rental Offering Description*** is created by specializing **Service Offering Description** and describing the already defined ***Car Rental Offering*** (a subtype of **Service Offering**), as shown in Figure 15 (A) and (B). This DROP application also requires to be complemented by a FOP, since **Service Offering Description** is a *category*, which shall be specialized into different types of descriptions for a service. Thus, by applying the **Category Pattern**, the complete answer for CQ04 comes specializing the ***Car Rental Offering Description*** into two *kinds* of offering descriptions: ***Car Rental Offering Folder*** and ***Car Rental Offering Web Page***, as shown in Figure 15 (C).

To address CQ05 (*Which are the terms and conditions of a rent-a-car service offering?*), we applied the SOCommitment DROP. Since we have previously applied the SOffering DROP, the SOCommitment introduces only one new concept and three relations for the domain model, reusing the other already defined elements. Thus, the ***Car Rental Offering Commitment*** concept is created extending **Service Offering Commitment**, and the corresponding relations (*partOf*, *inheres in*, and *externally depends on*) are also specialized (as shown in Figure 15 (D) and (E)).

Figure 15 shows the complete Car Renter Offering sub-ontology complementing Figure 14 with the new DROPs and their applications.
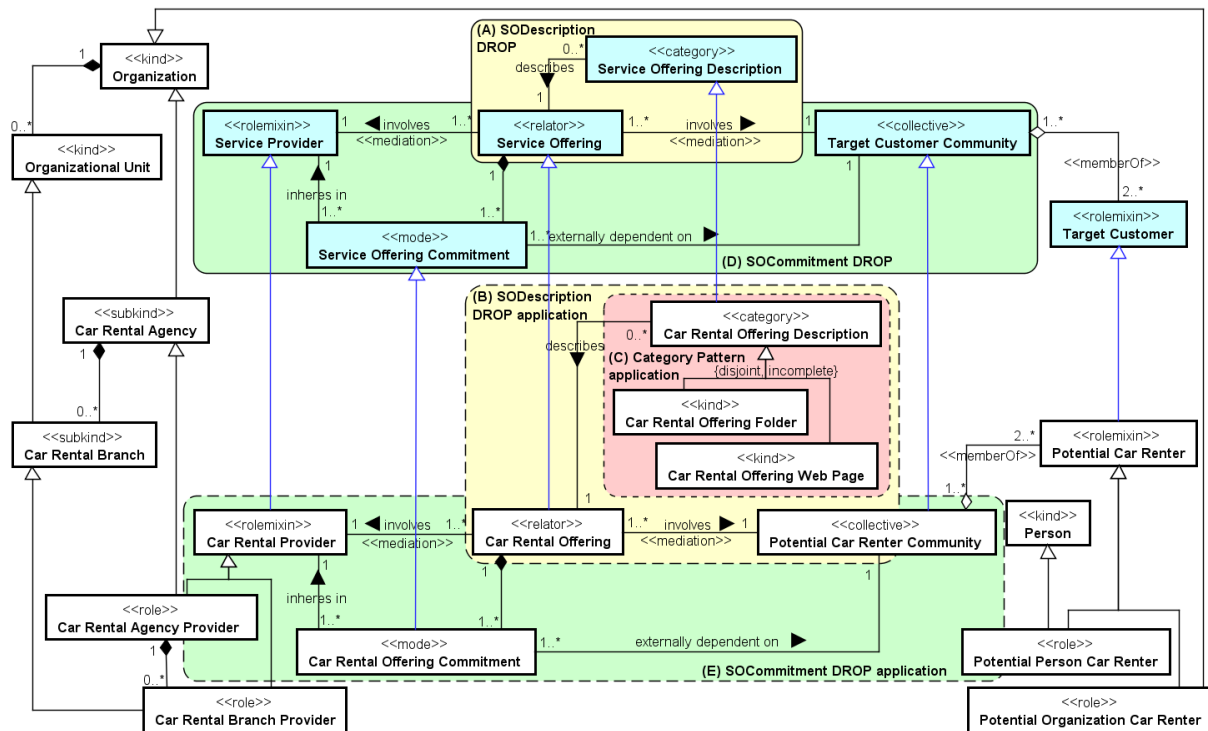

Figure 15. Car Rental Offering sub-ontology, with COPs applied.

CQ06-CQ10 are addressed by the Car Rental Agreement sub-ontology, as Figure 16 shows. Starting from the three first questions (*CQ06. What is a rent-a-car service agreement?*; *CQ07. Who are involved in a rent-a-car service agreement?*; and *CQ08. To which rent-a-car service offering does a rent-a-car service agreement conform?*), they are covered by the SOfferAgree DROP (Figure 16 (A)). Briefly, a car rental agreement is an agreement between a car lessor and a car renter, conforming to a previously established car rental offering. This way, by applying the SOfferAgree DROP through extension (see Figure 16 (B)), we modeled ***Car Rental Agreement*** as a subtype of **Service Agree-**

ment; *Car Lessor* as a subtype of **Hired Service Provider**, and *Car Renter* as a subtype of **Service Customer**. The *Car Rental Offering* (**Service Offering**), the *Car Rental Provider* (**Service Provider**), and the *Potential Car Renter* (**Target Customer**) are imported from the Car Rental Offering sub-ontology.

However, the SOfferAgree DROP requires the application of the **RoleMixin FOP** in order to complete the model fragment, defining the possible types of **Hired Service Providers** and **Service Customers**. Figure 16 (C) shows the **RoleMixin Pattern** application, defining the two types of *Car Lessor* as an agency or a branch; and Figure 16 (D) shows the **RoleMixin Pattern** application, defining person and organization as the two types of *Car Renter*.

Finally, let's consider CQ09 (*Which car is located in a car rental?*) and CQ10 (*Who are the enabled drivers in a car rental?*). These questions concern domain specific characteristics, not covered by services in general, and thus, not supported by the service DROPs extracted from UFO-S. Hence, the option is to apply a FOP. CQ09 is addressed by modeling cars that are assigned to a car rental. Thus, the **Role Pattern** applies. From a *kind Car*, we define the *role Rented Car* in a *Car Rental Agreement* (Figure 16 (E)). CQ10 is addressed similarly, applying the **Role Pattern** to specialize *Person* as the *role Enabled Driver*, indicated in a *Car Rental Agreement* (Figure 16 (F)).
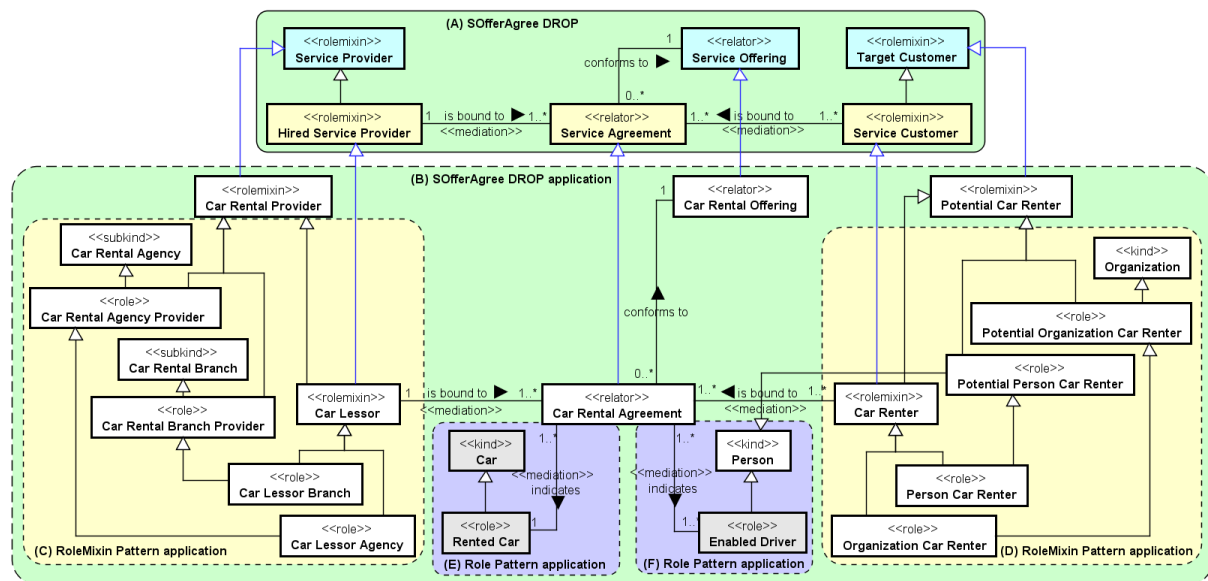

Figure 16. Car Rental Agreement sub-ontology, with COPs applied.

Figures 14 to 16 show some of the concepts of the domain ontology as specializations of concepts from the DROPs (and, thus, from the core ontology). This occurs because, in this case, we have adopted the *reuse by extension* approach for reusing DROPs. If we had adopted the *reuse by analogy* approach for reusing DROPs, this would not have happened. Figure 17 shows the resulting Car Rental Agreement sub-ontology, if the reuse by analogy approach had been used. In this case, the concepts related to the car rental agreement (see Figure 17 (A)) replace the corresponding concepts of the SOfferAgree DROP, instead of specializing them (as in Figure 16). We should reinforce that, in both cases, FOPs from Table 2 are always reused by analogy, independently from the approach chosen for reusing DROPs, as Figures 16 and 17 show.
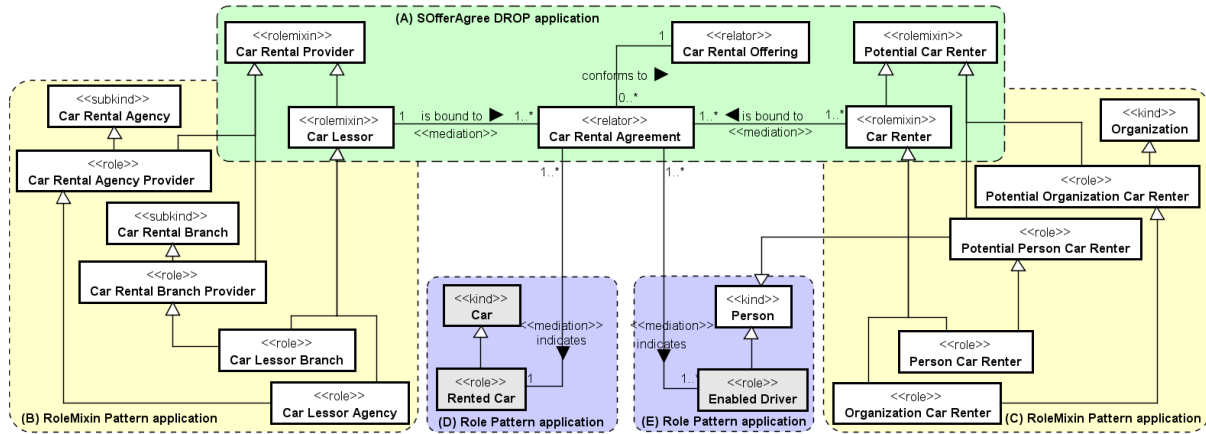
Figure 17. Car Rental Agreement sub-ontology following the Reuse by Analogy Approach.

Besides reusing conceptual model fragments, axioms defined in a COP can also be reused. This is a striking feature when applying DROPs, due to similarities between the domains being modeled, where, in general, concepts and relations are extended. To illustrate this process, Table 4 shows, in the first line, an axiom of the <u>SOCommitments DROP</u> and its description and, in the second line, a specialization of this axiom applied to the Car Rental domain.

Table 4. Reusing Axioms from DROPs.

Source	Axiom	Description
<u>SOCommitments DROP</u>	$\forall$so: ServiceOffering, sp: ServiceProvider, tcc: TargetCustomerCommunity, soco: ServiceOfferingCommitment (involves(so, sp, ServiceProvider) $\wedge$ involves(so, tcc, TargetCustomerCommunity) $\wedge$ partOf(soco, so)) $\rightarrow$ (inheresIn(soco, sp) $\wedge$ externallyDependentOn(soco, tcc))	Each *Service Offering Commitment* that is part of a *Service Offering* inheres in the *Service Provider* that offers the *Service Offering*, and is externally-dependent on the *Target Customer Community* to which this offering is offered.
Domain Ontology	$\forall$ cro: CarRentalOffering, crp: CarRentalProvider, pcrc: PotentialCarRenterCommunity, croc: CarRentalOfferingCommitment (involves(cro, crp, CarRentalProvider) $\wedge$ involves(so, pcrc, PotentialCarRenterCommunity) $\wedge$ partOf(croc, cro)) $\rightarrow$ (inheresIn(croc, crp) $\wedge$ externallyDependentOn(croc, pcrc))	Each *Car Rental Offering Commitment* that is part of a *Car Rental Offering* inheres in the *Car Rental Provider* that offers the *Car Rental Offering*, and is externally-dependent on the *Potential Car Renter Community* to which this offering is offered.

As one can see, both axioms are equivalent, but the second one is more specific, once it is applied to a more specific domain.

The reuse of axioms also has its origin at the foundational level. As discussed in depth and empirically demonstrated in [36], when the **Relator Pattern** is applied in a configuration in which it mediates a set of *Roles*, a formal constraint that modelers typically want to include in the model is one that dictates that these roles are mutually disjoint in the scope of that relator. For instance, a criminal investigation can connect a suspect, a victim and an investigator; all these roles are played by individuals of the same Kind (Person); however, it is not the case that, in the scope of a given investigation, these roles can be played by the very same individual! In Table 5, we use the example of this FOP to illustrate an axiom defined for a FOP, reused by a DROP, and then by a domain ontology.

Table 5. Reusing Axioms from FOPs and DROPs.

Source	Axiom	Description
Relator Pattern extended with a *formula schema* defining the mutual disjointness of Roles in the scope of a Relator	$\forall r$: RelatorUniversal$_1$ → ($\exists e_1, e_2$ mediates(r, e_1, Role$_1$) ∧ mediates (r, e_2, Role$_2$)) ∧ ($\forall r, e_1, e_2, r$: RelatorUniversal$_1$ ∧ mediates(r, e_1, Role$_1$) ∧ mediates (r, e_2, Role$_2$) → $e_1 \neq e_2$)	For every relator r of type RelatorUniversal$_1$, we have that r mediates endurants playing the roles Role$_1$ and Role$_2$. Moreover, for every relator r of type RelatorUniversal$_1$, we have that the individuals playing roles R$_1$ and R$_2$ in the scope of r must be distinct.
SOfferAgree DROP	$\forall a$: ServiceAgreement → ($\exists p, c$ isBoundTo(a, p, HiredServiceProvider) ∧ c isBoundTo(a, c, ServiceCustomer)) ∧ ($\forall a, p, c$ a: ServiceAgreement ∧ isBoundTo(a, p, HiredServiceProvider) ∧ isBoundTo(a, c,ServiceCustomer) → $p \neq c$)	For every *Service Agreement a*, there are individuals bound to *a* playing the roles of *Hired Service Provider* and *Service Customer*. Moreover, in the scope of a given *Service Agreement a*, these roles cannot be played by the same individual.
Domain Ontology	$\forall a$: CarRentalAgreement → ($\exists l, r, c, d$ isBoundTo(a, l,CarLessor) ∧ isBoundTo(a, r,CarRenter) ∧ **indicates(a, c,RentedCar) ∧ indicates(a, d,EnabledDriver)**) ∧ ($\forall a, l, r$ a: CarRentalAgreement ∧ isBoundTo(a, l,CarLessor) ∧ isBoundTo(a, r,CarRenter) → $l \neq r$))	For all *Car Rental Agreement*, there are a *Car Lessor*, a *Car Renter*, a *Rented Car*, and an *Enabled Driver*. The *Car Lessor* and the *Car Renter* must be distinct individuals in the scope of a given *Car Rental Agreement* to which they are bound.. Moreover, this *Car Rental Agreement* indicates the *Rented Car* and the *Enabled Driver* for that rental.

The first line presents an axiom defined to exclude the aforementioned situation at the level of the **Relator Pattern**. We define a *formula schema* (line 1 of this table) that makes use of a ternary predicate *mediates(r,e,R)* (meaning that endurant *e* is mediated by relator *r* instantiating the type *R*)[2] in order to define a case where two roles[3] are disjoint in the scope of relator of a given type. In the second line of that table, we instantiate that formula schema by replacing RelatorUniversal$_1$, Role$_1$ and Role$_2$ with the core-level universals *Service Agreement*, *Hired Service Provider* and *Service Customer*, respectively. Moreover, we replace the *mediates(r,e,R)* relation with the *isBoundTo(r,e,R)* relation in a way that the latter implies the former, plus the constraint that reinforces that *R* is of the proper type (i.e., either *Hired Service Provider* or *Service Customer*). Finally, in the third line of this table, when applying the SOfferAgree DROP to the Car Rental domain, we, once more, specialize this formula schema by having the universals *Car Rental Agreement*, *Car Renter* and *Car Lessor* as domain-level specializations of *ServiceAgreement*, *Hired Service Provider* and *Service Customer*, respectively. Moreover, we extend this domain-level instantiation of that schema by including two new domain-specific concepts (*Enabled Driver* and *Rented Car*) and a two new specializations of the *mediates* relation: *indicates*. The bold fragment of the formula in line 3 of this table connects an instance of *Car Rental Agreement* with instances of *Rented Car* and *Enabled Driver* (see Figure 15).

5.2 Engineering *with Reuse*: Guidelines for Using DROPs and FOPs

The approach followed in the previous example is now defined as a guideline, which can be used in tandem with any ontology engineering method that professes the use of ontology patterns. The modeler should start by trying to reuse DROPs. For reusing DROPs, the following steps should be done:

---

[2] Of course, if *mediates(r,e,R)* then *mediates(r,e)*, where the latter is the previously defined existential dependence relation of mediation between relators and their relata.

[3] We use the case of two roles here just for the sake of simplicity. The formula schema can obviously be extended to define mutual disjointness across an arbitrary number of roles in the scope of a relator.

A. Know the target domain and the related DROPs. Once the ontology engineer knows the domain and the scope of the ontology to be built, she has to look for DROPs related to that domain. By studying the candidate patterns, many modeling decisions can be expedited and more elements can be reused (e.g., model fragments, CQs, axioms and concept definitions).

B. Find the applicable DROPs. For each domain-specific modeling problem, look for a DROP that can be reused to solve it. This can be done by looking the pattern's intent, model fragment and CQs.

C. Select the Reuse Approach to Adopt and Apply the DROPs. As discussed in Section 2, there are two reuse approaches that can be adopted: reuse by extension or reuse by analogy. Before applying any DROP, the ontology engineer should choose which reuse approach to adopt. The approach should be used for all DROPs, i.e., it is not recommended to reuse some DROPs by extension and others by analogy (notice that we are only talking about DROPs, since FOPs are typically reused by analogy). When reuse by extension is chosen, if a DROP matches the problem at hand, select it and add the corresponding model fragment directly to the domain model. Typically, the DROP fragment is extended in the domain model, creating new specialized concepts and relations. This reuse approach allows for some modifications to be applied to the original pattern, such as adding new properties to the concepts and restricting cardinality constraints. This approach was used in Subsection 5.1 (Figures 14 to 16). Another possibility is to take the pattern fragment, replace their concepts by the corresponding domain concepts, and to do the necessary changes in the resulting model fragment (*reuse by analogy*). The result of applying such approach is partially shown in Figure 17 for the case of the Car Rental Agreement sub-ontology. The first approach (reuse by extension) is preferred, mainly if some DROPs are applied many times in the same model, since this approach preserves the original domain types of the modeled elements. Reuse by extension is also useful for keeping track to the DROPs reused (and to the corresponding core ontology, if the DROPs are extracted from a core ontology). This could be useful for latter integrating the domain ontology with others developed using the same DROPs (and, thus the same core ontology) as basis. On the other hand, reuse by analogy makes the resulting domain ontology simpler, what certainly is valuable. At this point, axioms can also be reused (incorporated or specialized), and CQs can be rewritten for standardization. Observe the **related patterns** item in the DROP specification, and apply the mandatory patterns, and, if needed, the optional patterns. This can be done as many times as needed, reusing a number of DROPs and FOPs.

D. In a case that no DROP is applicable: check if there is a FOP to support the modeling problem, and apply it if necessary (see FOP application guidelines). Remember that the FOPs shown in Table 2 are always reused by analogy.

E. Check the domain model consistency and coverage. Check if the whole model is consistent and if it covers all the planned scope (all CQs are answered). At this point, besides the reused axioms, new domain-specific axioms can be added. If DROPs are not available to address some of (or even all) competency questions for the ontology being developed, the approach illustrated in Figure 6 should be followed for achieving a valid OntoUML structural model. For using the FOPs presented in Table 2, the following steps should be done:

F. Categorize the main concepts with OntoUML categorizations. Once a main domain concept is identified, we must detect which of the OntoUML meta-types should be applied to model this concept. Determining the correct meta-type, involves a precise analysis of ontological aspects covered by OntoUML meta-properties presented in Section 2.2. The systematic choice of the proper stereotype for a given concept in a given situation can be precisely identified by using the techniques presented in [20].

G. Identify a proper FOP. The proper choice of which meta-type should be used for modeling a domain concept automatically triggers the application of the corresponding OntoUML FOP (see Table 2). The only exception when this domain concept is should be modeled as a kind (the only terminal symbol in the language). For instance, if a concept is categorized as a Relator, we need to apply one of the possible variations of the **Relator Pattern**. Once we select the possible FOPs to be applied for this categorization, we can use the competency question to find out which specific FOP variant must be applied.

H. Apply the FOP. The application of a FOP implies the instantiation of the general meta-types comprising it with domain concepts (*reuse by analogy* [6]). Frequently, these instantiations can be done by reusing domain classes and relations that are already represented in the model. As specified in Table 2, the application of these FOPs frequently triggers the application of other associated FOPs in a recursive manner. The recursive application of series of FOPs terminates when a terminal symbol (a Kind) is reached. In this manner, OntoUML guarantees that all elements of domain-level model fragment represented in the language obey a unique and determinate principle of identity. An illustration of the process of building a domain conceptual model just by recursively applying OntoUML FOPs was shown in Figure 6. In that model, the choice of using the second variant of the **Relator Pattern** could be motivated by a CQ such as: "*What kind of relation is established between a Customer and a Supplier?*".

6 Experience Report: Evaluating the Proposed Approach through the Design of Ontology Patterns Languages

In order to evaluate and refine the research reported in this paper, we have applied it in a series of initiatives in different domains. One important part of our strategy for exploring and promoting the creation and application of COPs in a reuse-oriented process is the development of *Ontology Pattern Languages (OPLs)* [2, 37]. While FOPs tend to be generally applied, DROPs for a specific field are very interrelated [8]. For this reason, DROPs are usually applied in combination (with other DROPs and FOPs), bearing to each other relations such as co-occurrence dependence (patterns must be applied together), temporal precedence (a pattern must be used before another one can be used) or mutual exclusion (patterns are alternatives to each other as a solution to a given instance of a modeling problem). An OPL organizes DROPs in a guided application process by explicitly representing these relations.

An Ontology Pattern Language (OPL) is a network of interconnected DROPs that provides holistic support for solving ontology development problems for a specific domain. An OPL contains a set of interrelated DROPs, plus a modeling workflow guiding on how to use and combine them in a specific order, and suggesting patterns for solving some modeling problems in that domain [37]. We have applied the guidelines discussed in Subsection 4.1 for deriving DROPs for OPLs in five different complex domains. One of them is Service OPL (S-OPL), whose patterns were extracted from UFO-S, a commitment-based core ontology for services [10], as discussed in Subsection 4.1. The other four OPLs are briefly described in the sequel. More details about them can be found in [37]:

- **Software Process OPL (SP-OPL)** [8]: consists in a network of ontology modeling patterns covering the software process domain. Its patterns were extracted from the Software Process Ontology [38], and has more than 40 patterns addressing software process definition in three levels: standard, intended and performed processes
- **ISO Software Process OPL (ISP-OPL)** [39]: is a specialization of SP-OPL focusing on the terminology and process structure provided by ISO standards devoted to software processes. It is based on an ontological analysis of the ISO/IEC 24744 metamodel (Software Engineering Metamodel for Development Methodologies – SEMDM), and involves a set of 27 patterns regarding work units, work products and human resources.
- **Measurement OPL (M-OPL)** [40]: provides ontology modeling patterns addressing the core conceptualization about measurement. It is composed of 19 patterns regarding measurable entities, measures, measurement units & scales, measurement procedures, measurement planning, and measurement & analysis.
- **Enterprise OPL (E-OPL)** [41]: provides ontology modeling patterns for enterprise ontology modeling. It is composed of 22 patterns addressing common aspects to several enterprises: organization arrangement, team definition, institutional roles, institutional goals, and human resource management.

As discussed in Section 4, several DROPs are supported by FOPs, thus, inheriting the FOP structure. However, in our first OPLs, we tried to make it transparent for the user, which was not supposed to know the FOPs for using an OPL. After some experimentation, however, we observed that knowing

the FOPs that can be applied in combination with each DROP reduces modeling errors and results in more consistent domain ontologies [39]. Thus, we decided also to include FOPs as related patterns for supporting the combined application of COPs when using OPLs.

Regarding pattern application, these OPLs have being used for building domain ontologies in several initiatives. For instance, in an experiment, S-OPL was applied by a group of 10 people for building 5 domain ontologies in service related domains (condominium administration, cable TV, cleaning services, hotel hosting, and passengers transport). Most of the participants classified the ontology development aided by S-OPL as simple and intuitive, resulting in better quality ontologies. S-OPL was used also in a in a real case study to model an email service in a big Italian company [34]. In another experiment, ISP-OPL was applied by a group of 19 people, resulting in 8 software process domain ontologies comprising human resource management, measurement, risk management, software architectural design, software configuration management, software documentation management, software maintenance, and stakeholder requirements definition. In the ISP-OPL experiment, when comparing the development of ontologies with and without the use of the ISP-OPL patterns, the majority of the participants that had a previous experience developing ontologies reported an expressive gain in productivity and quality of the results [39]. In addition, the reuse of patterns has promoted some standardization between the resulting ontologies. In general, these experiment results have shown that pattern application can improve the ontology engineering process by reducing complexity, improving productivity and resulting in better quality ontologies. Moreover, we have been verifying the real recurrence of the DROPs and collecting results and insights that are helping to improve the approaches for creation and application of DROPs. As we advance in the pattern application guidelines, the use of OPLs also advances, given the close relation between the two approaches.

It is important to highlight that it is not intent of this paper to discuss how COPs can be collectively organized (e.g. in a catalog or OPL). For a detailed discussion on OPLs, see [2, 37].

7 Tool Support for Ontology Pattern Application in OLED

In order to support the approach proposed here, we have extended the OLED[4] (OntoUML Editor) tool [42]. OLED is a model-based open-source computational tool, which provides a complete framework to support the modeling, design, verification, validation and implementation activities of the ontology development process. It provides a number of features, such as construction, evaluation, simulation and transformations of OntoUML models [9]. The extension proposed in the context of the present work automates the process of definition and application of FOPs and DROPs, following the approaches proposed in this paper.

In order to provide support for a pattern-based approach for ontology modeling, OLED was equipped with a built-in FOP library. Moreover, it supports the modeler in the definition of her own libraries of DROPs.

For the case of FOPs, we collected the main foundational patterns of UFO-A (see Section 3) and defined a pattern palette in the tool. Picking up one pattern from the palette, the users can configure and apply the patterns guided by the Pattern Application feature, as depicted in Figure 18. As one can see in this figure, the right-hand side of the window illustrates the model fragment that represents the applied FOP (in this case, the **Subkind Pattern** – variant 2). The left-hand side allows customizing the specific application of a FOP. It is possible to set the name of the domain concepts, to choose one of the possible stereotypes for a type (since a generic stereotype is used to define a pattern, e.g. *Sortal, Moment, Rigid Sortal*). The modeler can also choose to instantiate the classes forming the pattern by selecting a class that is already present in the model being built. This last option is one of the most important features in order to combine patterns, since concepts already introduced in the current conceptual model can be reused. In order to improve the usability of OLED, we also implemented the buttons "Add New Concept", "Remove Concept" and "Create Concepts". The addition of new concepts depends on which pattern is being applied. For the case of partitions or generalizations, this button adds new subtypes; for other cases, such as the **Relator Pattern**, it can add a new related concept.

---

[4]OLED: https://github.com/nemo-ufes/ontouml-lightweight-editor

Now, suppose in this example that the class named "General" in Figure 18 is not stereotyped as a Kind but as a Subkind. In this case, the tool would again call for another instantiation of this window in which a new recursive instantiation of **Subkind Pattern** could be configured. This process would be repeated in a coordinated way by the tool until a terminal symbol (i.e., a Kind) is reached.
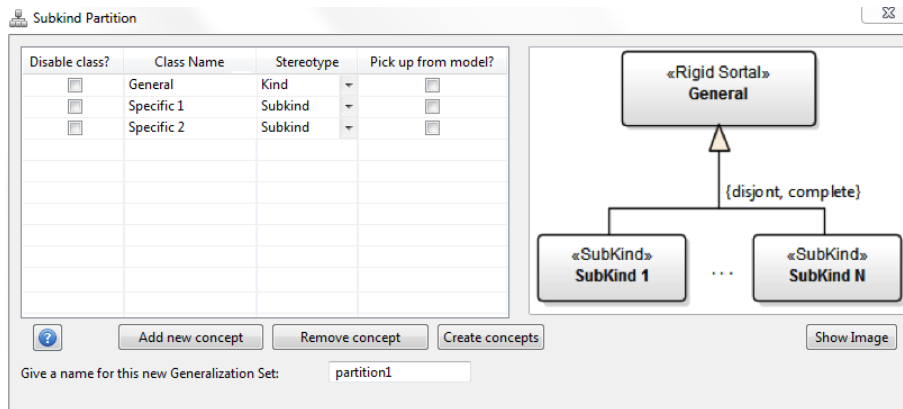

Figure 18. FOP Application window – Subkind Partition FOP.

Considering DROP libraries, instead of defining a default palette for the patterns, it is possible to create customized libraries. These libraries can be created in the tool by defining each pattern in a distinct diagram and adding the related information. The suitable libraries can be imported and applied to build core and domain ontologies. In order to improve the sharing of libraries of DROPs, a XML file saves the information related to each pattern, namely: description, competency questions, and axioms (as OCL rules). This feature also makes possible for the combination of different types of COPs, since the OLED extension enables to build a core ontology by applying FOPs, 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.

As well as for FOPs, OLED has also a DROP Application feature. Once a DROP library is chosen, the palette shows its patterns. When a DROP is selected for application, the DROP Application window shows the DROP model and related information, as illustrated in Figure 19.
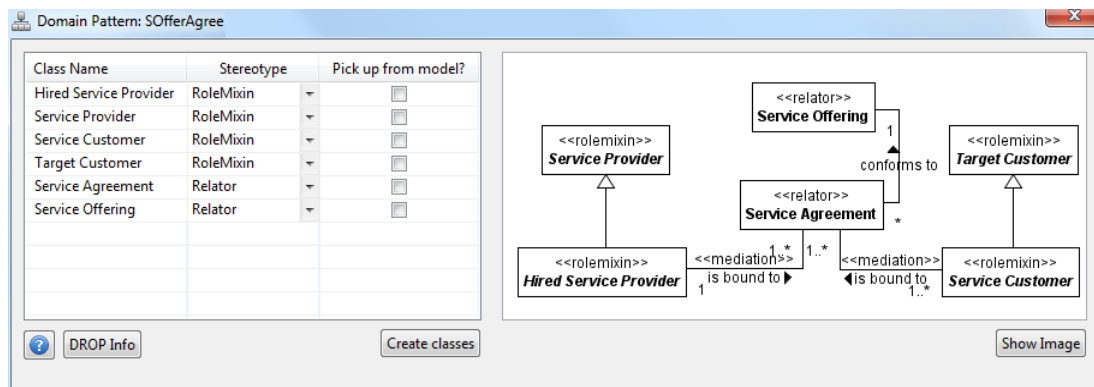

Figure 19. DROP Application window – SOfferAgree DROP

As one can see, this window is very similar to the FOP Application window. In the right-hand side, the window presents the DROP conceptual model, and, in the left-hand side, the configuration options for that pattern. In addition, as depicted in Figure 20, the tool also provides a way to verify the DROP description (through the button "DROP info").
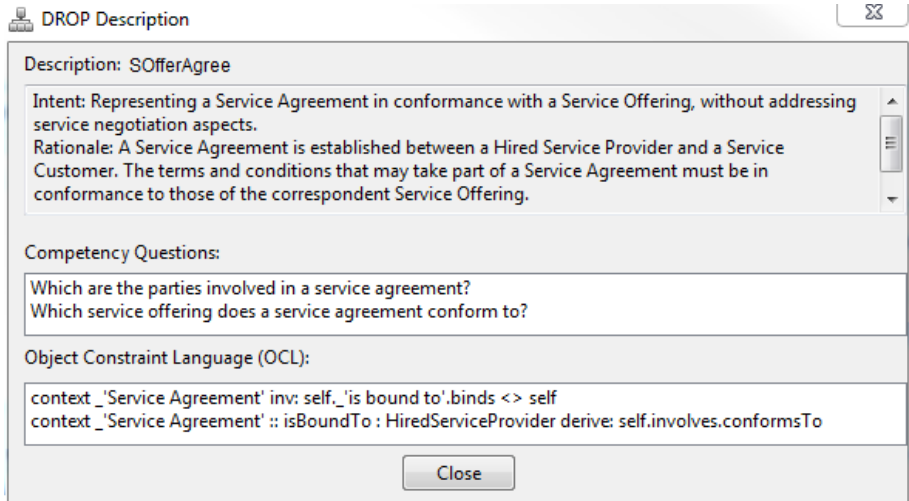
Figure 20. Description Window for SOfferAgree DROP.

This window presents the specific information for the selected DROP, in order to provide a simple way to consult the applicability of the pattern. Here, we have information about the pattern description, its associated competency questions and axioms. Regarding the latter, the axioms are written in OCL and added to the model when the pattern is applied. With this feature, the tool incorporates the axioms defined in each DROP to the domain ontologies being built.

Finally, Figure 21 shows an example of how these features contribute to building domain ontologies. First, the SOfferAgree DROP is selected in the palette (left-hand side of Figure 21) to be reused. A new window for DROP configuration opens (Figure 19). Next, the DROP concepts are included in the current model (highlighted concepts), and can be specialized with domain specific concepts and relations, which can be complemented with the application of a FOP or another DROP.
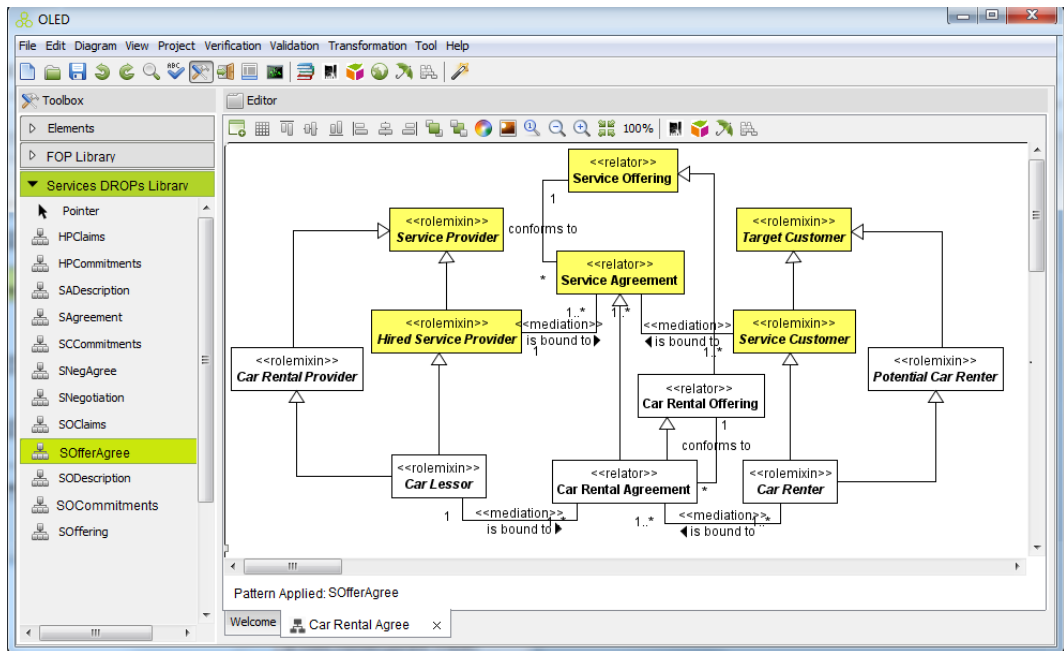

Figure 21. OLED's support for reusing COPs.

8   Related Work

There are few works addressing the combined application of OPs, most of which focus on reusing patterns of the same generality level, or of the same type.

Uschold and colleagues [43] consider the use of an implementation guide for building domain ontologies. This guide intends to provide a computer-based toolset that helps to capture aspects of a business and to analyze them to identify and compare options for meeting the business requirements. This process has been applied, for instance, in the Gist ontology [44]. Gist has the main purpose of laying a foundation for improving interoperability and integration among IT systems for business domains. Once the generic concepts for business are widespread and well known, the Gist ontology provides 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 for selecting and applying its concepts. Hence, it would be plausible to consider the Gist's fragments as DROPs. As Gist, others reliable generic ontologies are capable of serving as a potential 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. Moreover, we also focus on providing guidelines for extracting DROPs from core ontologies

In [2], 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. An evolution of this work is presented by Falbo and colleagues [6], which extend the Gangemi and Presutti's work [2]. In [6], they change the pattern classification, which now consider DROPs and FOPs as Content OPs (which were renamed to Conceptual OPs - COPs). They also briefly show how these types of COPs can be used for building ontologies (*reuse by analogy and reuse by extension*). In fact, we took that work as our baseline. Here, we presented approaches including the mechanism for deriving FOPs and DROPs. We also discussed how FOPs and DROPs are applied in a systematic and combined way. Finally, we present a software tool supporting the definition and application of these COPs.

In another work of the same group, Falbo and colleagues [8] 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 the combined application of COPs, when DROPs are reused in forming a complex pattern, or even in sequence. However, typically, in this line of work, OPLs explore only the application of DROPs (i.e., domain-related OPs). As a contribution of this paper, we present how FOPs, combined with DROPs, are useful for supporting the development of domain ontologies. This combination of COPs at different level 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; (iii) combining different DROP applications in structurally valid ontology configurations; and (iv) combining DROPs from different related OPLs. Considering all these situations, we demonstrate here how the inclusion of FOPs can be beneficial for OPLs. Moreover, the approach for extracting DROPs from core ontologies, presented in Subsection 4.1, systematizes the way that DROPs that constitute an OPL can be derived. In fact, making explicit the rationale underlying this process is an important contribution of this work.

In another work [45], Fernández-López et al. present a systematic method for reusing operational ontologies. Different from the work presented here, they tried to formalize the method of building operational ontologies, reusing ontologies from different domains, in order to create a network of ontologies. This method analyzes the modeled concepts, as well as related axioms, in order to find the more suitable correlated ontologies to be applied. Afterwards, two (or more) ontologies are mapped by the concepts that they share. In that approach, the authors do not consider a unique core ontology as source to build the ontology at hand. Instead of this, they provide a method to identify the suitable ontologies and join them. Although in our approach we also apply guidelines to the extraction and application of COPs, we consider that core ontologies should derive a wide set of generic patterns to model the intended domain and then, whenever necessary, these patterns might be extended to reflect each specific domain.

9   Final Considerations

Ontology Patterns have been recognized as a beneficial approach for Ontology Engineering [2, 3]. This paper discussed how Content Ontology Patterns (FOPs and DROPs) can be systematically derived and reused for developing reference domain ontologies. The presented approaches for deriving FOPs and DROPs have helped achieving a consistent and useful set of patterns that are being applied in different initiatives, including the development of OPLs as well as the evolution of an OntoUML computational supporting tool.

The application of these two types of COPs from different generality levels enriches the ontology development process in many ways: whilst FOPs add the foundational grounding for the domain models, DROPs support the reuse of core domain knowledge in a systematic manner. By combining these types of patterns, we have that: (i) FOPs can be applied when there is no suitable DROP for the problem at hand; (ii) FOPs can complement a DROP application by adding the necessary concepts to make the model consistent; and (iii) when a DROP is built carrying a FOP structure, its use causes a chained application of both foundational and domain aspects in the same fragment. In general, the combined use of these two types of COPs can enrich a domain ontology model with structural foundational and core domain knowledge.

Besides these advantages from a modeling point of view, COPs can also contribute for reusing competency questions and axioms from higher-level 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 patterns in combination.

Concerning evaluation, the proposed approach for extracting DROPs from core ontologies (see Subsection 4.1) has been applied for deriving the constituent DROPs of five Ontology Pattern Languages. In this paper, we showed its application for deriving the patterns of an OPL in the service domain (S-OPL) [34]. This series of evaluation studies provide us with invaluable feedback for evolving this approach, culminating with a proposal that makes explicit the rational behind the process through which DROPs are elicited for OPLs. The approach presented here is now being used by independent researchers to derive the DROPs for an OPL in the domain of collaboration, taking as basis a Collaboration Core Ontology [46]. This initiate shall serve as yet another evaluation study for our approach. Regarding the guidelines for using DROPs and FOPs in combination for building domain ontologies, they have been applied in several initiatives using OPLs (see, for instance, [39] and [34]).

We claim that when DROPs are extracted from a core ontology and organized as an OPL, we can achieve higher levels of reuse. In particular when, besides the models and concept definitions, the DROPs include competency questions (CQs) and axioms. OPLs guide the ontology engineer through the set of DROPs. By providing the CQs, it is easier for she to identify the useful patterns and adapt them for the case in hands. Thus, the reuse process becomes straightforward, as in the case discussed in Section 5.1. On the other hand, if DROPs are extracted from domain ontologies and put into catalogues, containing less information (for instance, without clearly presenting the CQs, definitions and axioms), reuse becomes harder.

As future work, we are using the COPs derivation and application approaches for improving and increasing the available set of patterns (especially in OPLs). In this context, a comprehensive and navigable specification is to be produced and published in order to improve the COPs availability. We are also investing efforts in evolving some aspects of the OLED tool, mainly for integrating different patterns libraries and improving the model construction usability. Finally, we are planning additional empirical studies to better 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.

10   Acknowledgements

11   References

1. Noppens, O. and Liebig, T., "Ontology Patterns and Beyond Towards a Universal Pattern Language", in WOP, 2009.
2. Gangemi, A. and Presutti, V., "Ontology Design Patterns", in Handbook on Ontologies, Second, S. Staab and R. Studer, Eds. Springer, pp. 221–243, 2009.
3. Blomqvist, E., Gangemi, A. and Presutti, V., "Experiments on pattern-based ontology design", in Proceedings of 5th International Conference on Knowledge Capture (K-CAP'09), 2009.
4. Presutti, V., Daga, E., Gangemi, A. and Blomqvist, E., "eXtreme Design with Content Ontology Design Patterns", in Proceedings of the Workshop of Ontology Patterns (WOP'09), 2009.
5. Guizzardi, G., "On Ontology, ontologies, Conceptualizations, Modeling Languages and (Meta)Models", In: Vasilecas, O., Edler, J., Caplinskas, A. (eds.) Databases and Information Systems IV, pp. 18-39, IOS Press, Amsterdam, 2007.
6. Falbo, R.A., Guizzardi, G., Gangemi, A. and Presutti, V., "Ontology patterns: clarifying concepts and terminology", in Proc. of the 4th Workshop on Ontology and Semantic Web Patterns, Sidney, Australia, 2013.
7. Guizzardi, G., Falbo, R.A., Guizzardi, R.S.S., "Grounding Software Domain Ontologies in the Unified Foundational Ontology (UFO): The Case of the ODE Software Process Ontology". In: Proc. XI Iberoamerican Workshop on Requirements Engineering and Software Environments, pp.244-251. Recife, Brazil, 2008.
8. Falbo, R.A., Barcellos, M.P., Nardi, J.C., Guizzardi, G., "Organizing ontology design patterns as ontology pattern languages", in Proceedings of the 10th Extended Semantic Web Conference (ESWC'13), Montpellier, France, 2013.
9. Guizzardi, G., "Ontological foundations for structural conceptual models", Enschede: Telematica Instituut Fundamental Research Series, 2005.
10. Nardi, J.C., Falbo, R.A., Almeida, J.P.A., Guizzardi, G., Pires, L.F., van Sinderen, M.J., Guarino, N. and Fonseca, C.M., "A commitment-based reference ontology for services", Information Systems, 54, pp.263-288, 2015.
11. Guizzardi, G., Wagner, G., Almeida, J.P.A., Guizzardi, R.S.S., "Towards Ontological Foundation for Conceptual Modeling: The Unified Foundational Ontology (UFO) Story", Applied Ontology, Vol. 10, issues 3-4, IOS Press, 2015.
12. U.S. Department of Defense (2011), "Data Modeling Guide (DMG) for an Enterprise Logical Data Model (ELDM)", available online: http://www.omgwiki.org/architecture-ecosystem/lib/exe/fetch.php?media= dmg_for_enterprise_ldm_v2_3.pdf.
13. Scherp, A., Saathoff, C., Franz, T., Staab, S., "Designing core ontologies", Applied Ontology, vol. 6, pp. 177–221, 2011.
14. Guarino, N., "Formal Ontology and Information Systems", in FOIS'98, vol. 46, 1998.
15. Guizzardi, G., Wagner, G., Falbo, R.A., Guizzardi, R.S.S, Almeida, J.P.A., "Towards Ontological Foundations for the Conceptual Modeling of Events", In: 32th Int. Conference on Conceptual Modeling (ER'13), pp.327-341. Hong-Kong, China, 2013.
16. Duarte, B.B., Souza, V.E.S., Leal, A.L.C., Falbo, R.A., Guizzardi, G., Guizzardi, R.S.S., "Towards an Ontology of Requirements at Runtime", in Proc. of the 9th International Conference on Formal Ontology in Information Systems, Annecy, France, 2016.
17. Guizzardi, G., "Ontology Patterns, Anti-Patterns and Pattern Languages for Next-Generation Conceptual Modeling", in Proc. of the 34th Int. Conference on Conceptual Modeling (ER'14), Atlanta, USA, 2014.
18. Guarino, N., Guizzardi, G., "We need to Discuss the Relationship: Revisiting Relationships as Modeling Constructs", 27th International Conference on Advance Information Systems Engineering (CAISE 2015), Stockholm, Sweden, 2015.
19. Guizzardi, G., Ferreira Pires, L., van Sinderen, M., "An Ontology-Based Approach for Evaluating the Domain Appropriateness and Comprehensibility Appropriateness of Modeling Languages", ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems, Montego Bay, Jamaica, 2005, Lecture Notes in Computer Science LNCS 3713, Springer-Verlag.
20. Guizzardi, G., das Graças, A.P. and Guizzardi, R.S., "Design patterns and inductive modeling rules to support the construction of ontologically well-founded conceptual models in OntoUML", In International Conference on Advanced Information Systems Engineering, pp. 402-413. Springer Berlin Heidelberg, 2011.
21. Guizzardi, G., Wagner, G., Guarino, N. and van Sinderen, M., "An Ontologically Well-Founded Profile for UML Conceptual Models", 16th International Conference on Advances in Information Systems Engineering (CAiSE), Latvia, 2004. Springer-Verlag, Berlin, Lecture Notes in Computer Science 3084, ISBN 3-540-22151-4.
22. Guizzardi, G., Masolo, C., Borgo, S., "In the Defense of a Trope-Based Ontology for Conceptual Modeling: An Example with the Foundations of Attributes, Weak Entities and Datatypes", 25th International Conference on Conceptual Modeling (ER'2006), Arizona, USA, 2006.

23. Albuquerque, A., Guizzardi, G., "An Ontological Foundation for Conceptual Modeling Datatypes based on Semantic Reference Spaces", In: 7th IEEE International Conference on Research Challenges in Information Science (RCIS 2013), Paris, 2013.

24. Guizzardi, G., Wagner, G., "What's in a Relationship?: An Ontological Analysis", 27th International Conference on Conceptual Modeling (ER'08), Barcelona, Spain, LN Computer Science, v.5231, p.83-97, 2008.

25. Guizzardi, G., "Ontological Foundations for Conceptual Part-Whole Relations: The Case of Collectives and their Parts", 23rd Int. Conf. on Advanced Information System Engineering (CAiSE'11), London, UK, 2011.

26. Guizzardi, G., "The Problem of Transitivity of Part-Whole Relations in Conceptual Modeling Revisited", 21st Int. Conf. on Advanced Information Systems Engineering (CAISE'09), Amsterdam, 2009.

27. Guizzardi, G., "On the Representation of Quantities and their Parts in Conceptual Modeling", 6th International Conference on Formal Ontologies in Information Systems (FOIS'10), Toronto, 2010.

28. Guizzardi, G., Wagner, G., Falbo, R.A., Guizzardi, R.S.S., Almeida, J.P.A., "Towards Ontological Foundations for the Conceptual Modeling of Events", 32nd International Conference on Conceptual Modeling (ER'13), Hong Kong, 2013.

29. Guizzardi, G., Almeida, J.P., Guarino, N., Carvalho, V.A., "Towards an Ontological Analysis of Powertypes", International Workshop on Formal Ontologies for Artificial Intelligence (FOFAI 2015), 24th International Joint Conference on Artificial Intelligence (IJCAI 2015), Buenos Aires, 2015.

30. Grüninger, M., Fox, M.S., "Methodology for the Design and Evaluation of Ontologies. Workshop on Basic Ontological Issues in Knowledge Sharing", 1995.

31. d'Aquin, M., "Modularizing Ontologies", In: Suarez-Figueroa, M. C., et al. (eds), Ontology Engineering in a Networked World. Springer, Berlin, 2012.

32. d'Aquin, M., Schlicht, A., Stuckenschmidt, H., Sabou, M., "Criteria and Evaluation for Ontology Modularization Techniques", In: Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization, Springer-Verlag, pp. 67–89, 2009.

33. Buschmann, F., Henney, K., Schmidt, D.C., "Pattern-Oriented Software Architecture: On Patterns and Pattern Languages", John Wiley & Sons Ltd, 2007.

34. Falbo, R.A., Quirino, G.K., Nardi, J.C., Barcellos, M.P., Guizzardi, G., Guarino, N., Longo, A. and Livieri, B., "An ontology pattern language for service modeling". In Proceedings of the 31st Annual ACM Symposium on Applied Computing, pp. 321-326, 2016.

35. Falbo, R.A., "SABiO: Systematic Approach for Building Ontologies". In Guizzardi, G., Pastor, O., Wand, Y., de Cesare, S., Gailly, F., Lycett, M., and Partridge, C., editors, Proc. of the Proceedings of the 1st Joint Workshop ONTO.COM / ODISE on Ontologies in Conceptual Modeling and Information Systems Engineering, Rio de Janeiro, Brazil, 2014.

36. Salles, T.P., Guizzardi, G., "Ontological Anti-Patterns: Empirically Uncovered Error-Prone Structures in Ontology-Driven Conceptual Models", Data & Knowledge Engineering (DKE) Journal, 2015.

37. Falbo, R.A., Barcellos, M.P., Ruy, F.B., Guizzardi, G., Guizzardi, R.S.S., "Ontology Pattern Languages". In Gangemi, A., Hitzler, P., Janowicz, K., Krisnadhi, A., and Presutti, V., editors, Ontology Engineering with Ontology Design Patterns: Foundations and Applications. IOS Press, 2016.

38. Bringuente, A.C.O., Falbo, R.A., Guizzardi, G., "Using a Foundational Ontology for Reengineering a Software Process Ontology", Journal of Information and Data Management, vol. 2, n. 3, pp. 511-526, 2011.

39. Ruy, F.B., Falbo, R.A., Barcellos, M.P., Guizzardi, G., and Quirino, G.K.S., "An ISO-based Software Process Ontology Pattern Language and its Application for Harmonizing Standards". ACM SIGAPP Applied Computing Review, 15(2):27--40, 2015.

40. Barcellos, M.P., Falbo, R.A. and V. Frauches. "Towards a measurement ontology pattern language". In Guizzardi, G., Pastor, O., Wand, Y., de Cesare, S., Gailly, F., Lycett, M., and Partridge, C., editors, Proc. of the Proceedings of the 1st Joint Workshop ONTO.COM / ODISE on Ontologies in Conceptual Modeling and Information Systems Engineering, Rio de Janeiro, RJ, Brasil, 2014.

41. Falbo, R.A., Ruy, F.B., Guizzardi, G., Barcellos, M.P., Almeida, J.P.A., "Towards an enterprise ontology pattern language", in Proc. 29th ACM Symposium on Applied Computing (SAC'14), pp. 323–330, 2014.

42. Guerson, J., Sales, T.P., Guizzardi, G. and Almeida, J.P.A., "OntoUML Lightweight Editor: A Model-Based Environment to Build, Evaluate and Implement Reference Ontologies." IEEE 19th International Enterprise Distributed Object Computing Workshop. IEEE, 2015.

43. Uschold, M., King, M., Moralee, S. and Zorgios, Y., "The enterprise ontology", Knowledge Engineering Rev., vol. 13, no. 01, pp. 31–89, 1998.

44. Uschold, M. and McComb, D., "Introduction to Gist", IAOA, 2013. [Online]. Available: http://iaoa.org/isc2014/uploads/Whitepaper-Uschold-IntroductionToGist.pdf.

45. Fernández-López, M., Gómez-Pérez, A. and Suárez-Figueroa, M.C., "Methodological guidelines for reusing general ontologies", Data & Knowledge Engineering, 86, pp.242-275, 2013.

46. Frechiani, F., Antunes, J., Guizzardi, R.S.S., "Towards a Collaboration Ontology", In: 2nd Workshop on Ontologies and Metamodels in Software and Data Engineering, João Pessoa, Brazil, 2007.