OPL-ML: A Modeling Language for Representing Ontology Pattern Languages

Glaice K.S. Quirino[1,2], Monalessa P. Barcellos[1] and Ricardo A. Falbo [1]

[1] Ontology and Conceptual Modeling Research Group (NEMO), Department of Computer Science, Federal University of Espírito Santo– Vitória – ES – Brazil

[2]Federal Institute of Espírito Santo – Cachoeiro de Itapemirim – ES - Brazil

glaice.monfardini@ifes.edu.br; {monalessa, falbo}@inf.ufes.br

Abstract. Reuse has been pointed out as a promising approach for Ontology Engineering. Reuse in ontologies allows speeding up the development process and improves the quality of the resulting ontologies. The use of patterns as an approach to encourage reuse has been explored in Ontology Engineering. An Ontology Pattern (OP) captures a solution for a recurring modeling problem. Very closely related OPs can be arranged in an Ontology Pattern Language (OPL). An OPL establishes relationships between the patterns and provides a process guiding the selection and use of them for systematic problem solving. To make it easier using an OPL, the relationships between the patterns and the process for navigating them should be represented in a clear and unambiguous way. A visual notation can be used to provide a visual representation of an OPL, aiming at improving communication. To facilitate understanding an OPL and strengthen its use, this visual notation must be cognitively rich. This paper presents OPL-ML, a visual modeling language for representing OPLs.

Keywords: Ontology Pattern Language, Ontology, Visual Notation, Visual Modeling Language.

1 Introduction

Nowadays, ontology engineers are supported by a wide range of methods and tools. However, building ontologies is still a difficult task. In this context, an approach that has gained increasing attention in recent years is the systematic application of *ontology patterns* (OPs), which favors the reuse of encoded experiences and promotes the application of quality solutions already applied to solve similar modeling problems [1, 2, 3, 4]. An OP describes a recurring modeling problem that arises in specific ontology development contexts, and presents a well-proven solution for this problem [2]. Experiments, such as the ones presented in [5], show that ontology engineers perceive OPs as useful, and that the quality and usability of the resulting ontologies are improved.

As pointed by Blomqvist et al. [3], although the ideas behind OPs are not completely realized in practice yet, the process is ongoing. As OPs become more mature and the community collects more and more experience using them, a situation that already occurs in Software Engineering, where patterns have been studied and applied for a long

time, will emerge in Ontology Engineering. In a pattern-based approach to Ontology Engineering, several patterns can be combined to derive a new ontology. Such approach requires the existence of a set of suitable patterns that can be reused in the development of new ontologies, and a proper methodological support for selecting and applying these patterns [4]. In this context, we need to record how different patterns relate to each other in a more abstract level, and thus we need an ontology pattern representation language [3]. The first steps towards developing such a representation language have already been undertaken, giving rise to Ontology Pattern Languages (OPLs) [6, 7]. An OPL is a network of interconnected OPs that provides holistic support for solving ontology development problems. An OPL contains a set of interrelated OPs, plus a modeling process guiding on how to use and combine them in a specific order [6].

To facilitate ontology engineers understanding an OPL, the relationships between the patterns, as well as the process for navigating them, should be represented in a clear way. Ideally, such representation should be visual, since visual representations are effective, by tapping into the capabilities of the human visual system [8].

The use of OPLs is a recent initiative. There are still only few works defining OPLs, all of them from the same research group: the Ontology and Conceptual Modeling Research Group (NEMO). This research group has developed the following OPLs [7]: Software Process OPL (SP-OPL), ISO-based Software Process OPL (ISP-OPL), Enterprise OPL (E-OPL), Measurement OPL (M-OPL), and Service OPL (S-OPL). Since these works were done by the same research group, they share commonalities. All of them use extensions of UML activity diagrams for representing the OPL modeling processes [7]. However, there are still inconsistencies and problems in such representations, as we could perceive by applying two experimental studies. Trying to overcome these problems, we developed OPL-ML, an OPL Modeling Language. For developing OPL-ML, we rely on the results of a systematic mapping of the literature that investigated visual notations for Software Pattern Languages. Moreover, OPL-ML was designed according to the principles of the Physics of Notation (PoN) [8], and following the design process defined by PoN-S (PoN Systematized) [9].

This paper aims at presenting OPL-ML, and is organized as follows: Section 2 discusses patterns, OPs, pattern languages and OPLs; Section 3 discusses the Design Science [10] methodological approach we followed to develop OPL-ML; Section 4 presents OPL-ML; Section 5 discusses a preliminary evaluation of OPL-ML; Section 6 concerns related works, and Section 7 presents our final considerations.

2 From (Ontology) Patterns to (Ontology) Pattern Languages

Patterns are vehicles for encapsulating design knowledge, and have proven to be beneficial in several areas. “Design knowledge” here is employed in a general sense, meaning design in different areas, such as Architecture and Software Engineering (SE). In SE, for instance, patterns have been studied and applied for a long time, and there are several types of patterns, such as analysis patterns, design patterns and idioms. The main principle behind patterns is not having to reinvent the wheel.

Ontology Patterns (OPs) follow the same idea: capturing a well-proven solution for a recurring modeling problem that arises in ontology development contexts. There are also several types of OPs, such as [2]: content patterns (foundational and domain-related patterns), design patterns (logical and reasoning patterns), and idioms.

Patterns, in general, can exist only to the extent that they are supported by other patterns. There is a need to describe the context of larger problems that can be solved by combining patterns, and to address issues that arise when patterns are used in combination. This context can be provided by what in Software Engineering has been termed a *Pattern Language* (PL) [11]. According to Schmidt et al. [11], the trend in the SE patterns community is towards defining pattern languages, rather than standalone patterns. We have advocated that this approach should also be followed in Ontology Engineering, by defining Ontology Pattern Languages (OPLs). An OPL aims to put together a set of very closely related ontology patterns (OPs), in a system of patterns that provides, besides the OPs themselves, a process describing how to navigate, select and apply them in a consistent way. The term "pattern language" was borrowed from Software Engineering (SE). However, it is important to say that we are not talking about a language properly speaking. In "pattern language", the term "language" is, in fact, a misnomer, given that a pattern language does not typically define per se a grammar with an explicit associated mapping to a semantic domain [6].

The use of OPLs is a recent initiative. At the best of our knowledge, all existing OPLs were developed by the same research group and use extensions of UML activity diagram for representing the OPL processes [7]. However, there are still inconsistencies and problems in such representations. Thus, to help developing new OPLs, it is essential to solve these problems and to provide a well-defined modeling language for representing OPLs. The use of PLs in SE is not new. Thus, aiming to get knowledge about visual notations for PLs, we carried out a systematic mapping to investigate visual notations used to represent software PLs. A systematic mapping provides an overview of a research area, and helps identify gaps that can be addressed in future research [12].

In our systematic mapping, we searched seven digital libraries, namely: Scopus (*www.scopus.com*), Engineering Village *(www.engineeringvillage.com)*, ACM (*dl.acm.org*), IEEE Xplore (*ieeexplore.ieee.org*), Springer (*link.springer.com*), ScienceDirect (*www.sciencedirect.com*) and Web of Science (*www.webofknowledge.com*). We identified 54 software-related PLs represented by visual notations and we investigated them. Next, we summarize some results and perceptions gotten from the study.

Different elements are addressed in PLs. We identified 13 different elements, namely: pattern, pattern group, pattern subgroup, flow, mandatory flow, alternative flow, parallel outputs, parallel inputs, structural relation, optional relation, and variant patterns. Many different representations for the same element were found. For instance, we found seven different symbols for representing patterns and nine for pattern groups. Several PLs include only few elements (e.g., pattern and flow). We noticed that in most cases, PLs use less elements than necessary. Consequently, elements meaning is overloaded and their symbols tend to be cognitively poor. As a result, due to lack of clarity, it is often difficult to understand the PL.

Patterns can relate to others in different ways. The most common relations in the investigated PLs are dependency, usage and specialization. Most of the investigated

PLs include dependency relations. Dependency can be understood as a broad relation and, if more specific relations are not defined, it can be not clear what dependency really means. For instance, if a pattern A depends on a pattern B, it is not clear if A depends on B because the solution given by A requires the solution given by B (thus, B must be applied before A), or because B is part of the solution given by A. In both cases, A depends on B, however, in the second case the dependency is, more specifically, a composition relation. The investigated PLs often do not explicitly define the different kinds of relations, making it difficult to understand relations among patterns and, consequently, to properly select and apply them.

Defining groups is particularly important when defining large and complex PLs. Groups can be represented in a transparent way (i.e., the patterns in a group are visible) or as black boxes (i.e., a symbol represents a pattern group and it is not possible to see the patterns inside it). The use of black boxes allows representing a PL at different detail levels, contributing to its understanding. 54% of the investigated PLs use groups to organize patterns. Only one of them use black boxes to organize the PL.

The investigated PLs use different types of models to represent their elements, providing different views to ease understanding and using the PL. Structural models present elements (e.g., pattern, patterns group) and their structural relations (e.g., dependency, composition). Process (or behavioral) models, in turn, present the possible paths to be followed to apply the patterns. 92% of the investigated studies present only one of these models (46% structural model and 46% process model). Only 8% of the PLs consider both views, half of them using a single (hybrid) model to address structural and process aspects, making it difficult for users to differentiate them.

Considering the panorama provided by the mapping study results, some gaps in the visual notations used to represent PLs can be pointed out: (i) lack of a standard visual notation; (ii) use of cognitively poor notations; (iii) lack of mechanisms to support patterns selection; and (iv) lack of a complete view of the PL, addressing both structural and process aspects.

3 Methodological Approach

For developing this work, we followed a Design Science methodological approach [10]. The addressed problem is: *How should we represent OPLs such that they become effective in guiding the use of related OPs to develop new ontologies?* The artifact to be designed is a *modeling language for representing OPLs*. A first representation already existed, the one used to represent the first versions of SP-OPL, E-OPL, ISP-OPL and M-OPL. This representation was an informal extension of the UML activity diagram (informal in the sense that there was not even a meta-model of it). Moreover, its application in each OPL was slightly different one from another. Thus, this work started by evaluating this representation to see whether it is effective in guiding the use of related OPs to develop new ontologies. A first experiment was accomplished using ISP-OPL [13] aiming at evaluating how the guidance provided by ISP-OPL affects the productivity of ontology engineers when developing domain ontologies for SE sub-domains, and the quality of the resulting ontologies. Besides, research questions regarding the visual notation used to represent ISP-OPL were also posed. The experiment took place

during the second semester of 2014, as part of the course "Ontologies for Software Engineering", an advanced course for graduate students in the Graduate Program in Informatics at Federal University of Espírito Santo, in Brazil. 19 graduate students with at least basic knowledge in conceptual modeling participated in the study. Concerning aspects related to ISP-OPL visual notation, most participants considered that it was easy to understand the OPL, but they pointed out problems related to some of the used symbols, and even more problematic, they had difficulties in following the paths in the OPL process, especially difficulties related to mandatory/optional paths.

Based on the results of this first experiment, we worked on changing some of the symbols in the visual notation, and on establishing a clearer way to use them. Those changes in the visual notation were applied in some of the OPLs existing up to that moment (E-OPL, ISP-OPL and M-OPL). The versions of such OPLs presented in [7] use this updated notation. Moreover, the updated visual notation was used to engineer S-OPL [14], and to evaluate it, we accomplished another experiment. In this study, S-OPL was used by 9 students to develop service ontologies for specific domains. The study took place as part of the course "Ontology Engineering", an advanced course for graduate students in the Graduate Program in Informatics at Federal University of Espírito Santo, in Brazil. After using S-OPL, the students were asked about benefits and difficulties of using it. As benefits they pointed out that the use of an OPL contributes to the to the quality of the resulting ontology and to the productivity of the development process. As main difficulties, they highlighted: (i) the lack of information about whether, or not, to follow the paths in the OPL process; (ii) difficulties to identify the OPL main flow; (iv) the lack of explicit flow final nodes, indicating where the process finishes; and (v) problems with variant patterns. In parallel with these studies, we developed the systematic mapping presented in Section 2.

Based on the findings of both the two experiments and the mapping, we decided to develop a modeling language for representing OPLs, called OPL-ML. In such endeavor, we started developing a meta-model, as its abstract syntax. Next, we relied on the principles of the Physics of Notations (PoN) [8] for designing a cognitively effective visual notation, and we followed the design process established in the PoN-Systematized (PoN-S) approach [9] to develop its concrete syntax. After developing OPL-ML, we used it to reengineer S-OPL and evaluated it through an experimental study. Aspects regarding the use of PoN-S to develop OPL-ML are discussed in [9], and are out of the scope of this paper. Here, our focus is on OPL-ML, which is presented next.

4 OPL-ML: Ontology Pattern Language Modeling Language

According to [15], a visual language consists of a set of graphical symbols (visual vocabulary), a set of compositional rules for forming valid expressions (visual grammar), and semantic definitions for each symbol (visual semantics). The set of symbols and compositional rules together form the visual (concrete) syntax. Graphical symbols are used to represent semantic constructs, typically defined by a meta-model. An expression in a visual notation is a diagram. Diagrams are composed of instances of graphical symbols, arranged according to the rules of the visual grammar. In this paper, we focus on the visual syntax of OPL-ML, and the meta-model that defines its constructs, which is shown in Figure 1.

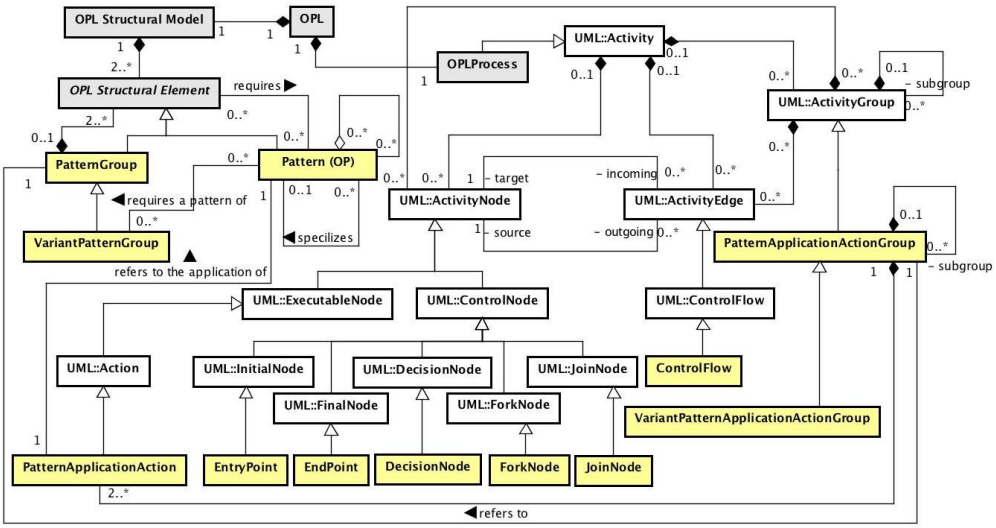


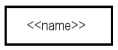
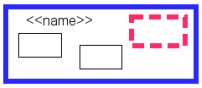
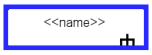
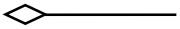
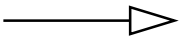
Fig. 1. OPL-ML Meta-model.

An OPL comprises both a structural model and a process. The *OPL Structural Model* focuses on patterns and pattern groups. It is composed of *OPL Structural Elements*, which can be of two types: *Pattern* and *Pattern Group*. A *Pattern* represents an OP, while a *Pattern Group* is a way of grouping related OPs and other pattern groups. Thus, a *Pattern Group* is composed of *OPL Structural Elements*. A special type of pattern group is the *Variant Pattern Group*, which is a set of patterns that solve the same problem, but each in a different way. Only one pattern from a *Variant Pattern Group* can be used at a time. *Patterns* that compose a *Variant Pattern Group* are variants of each other. Patterns can be composed of other patterns, and can specialize other patterns. Patterns may also depend on other patterns, i.e., for applying a pattern, another must be applied first. An OPL shall represent dependencies between patterns or between a *Pattern Group* and a *Pattern*. The *requires* relationship captures this dependency. Finally, a *Pattern* may require the application of a pattern from a *Variant Pattern Group*.

Table 1 shows the concrete syntax for representing the elements of the OPL structural model. It is important to say that meta-model elements shown in gray in Figure 1 do not require a symbol, and thus are not in Table 1. *Patterns* are represented by rectangles, which is the most common symbol used for representing patterns in SE pattern languages. *Pattern Groups* are represented by figures closed by blue straight solid lines (solid polygons). For representing *Variant Pattern Groups*, the same notion was applied, but now using red dashed lines. For allowing managing complexity in large diagrams, a black box representation is also provided for pattern groups. These alternative forms are represented by means of rectangles decorated by a rake-style icon (), indicating that this element is detailed in another diagram. This icon is used in UML to represent that an element represented by the decorated construct encapsulates further elements. Finally, regarding relationships, the dependency relations *requires* and *requires a pattern of*, both are represented by an arrow from the dependant to the dependee. For differentiating between them, arrows representing the *requires* association are

symbolized with solid lines, in contrast to the dashed lines for the *requires a pattern of* association. This decision is in line with the one of representing *Pattern Groups* using solid lines, and *Variant Pattern Groups* using dashed lines. Pattern composition and pattern specialization are represented by the same symbols used in UML for aggregation and specialization, respectively.

Table 1. Symbols of the visual notation for OPL structural models.

Element	Symbol
Pattern	
Pattern Group (expanded format)	
Pattern Group (black box format)	
Variant Pattern Group (expanded format)	
Variant Pattern Group (black box format)	
Relation “requires”	
Relation “requires a pattern of”	
Composition relation	
Specialization relation	

The OPL Process comprises a set of actions devoted to select and apply the patterns, and the control nodes that allow defining a workflow to navigate between the patterns. The part of OPL-ML meta-model describing the OPL Process is an extension of a subset of the UML’s meta-model regarding activity diagrams [16], and its concrete syntax is based on the UML notation for activity diagrams. This benefits users who are familiar with this notation. In Figure 1, classes from the UML’s meta-model are shown in white.

An *OPL Process* is a subtype of (UML::) *Activity* specifying a behavior regarding the application of OPs. This behavior is specified as a workflow connecting *Pattern Application Actions* by means of *Control Flows* to other *Pattern Application Actions* and *Control Nodes*. A *Pattern Application Action* is an (UML::) *Action* concerning a pattern application, i.e. this action node refers to the application of a specific OP. For allowing modeling the workflow of the *OPL Process*, the following *Control Nodes* can be used: *Entry Point*, *End Point*, *Decision Node*, *Fork Node*, and *Join Node*.

An *Entry Node* is a control node that acts as a starting point for executing the *OPL Process*. It is a subtype of (UML::) *Initial Node*, and as such, shall not have any incoming *Control Flows*. However, there is an important difference between *Entry Point* and (UML::) *Initial Node*: in a UML activity diagram, if an activity has more than one initial node, then invoking the activity starts multiple concurrent control flows, one for each initial node [OMG, 2015]; in an OPL, if its *OPL Process* has more than one *Entry Point*, this indicates that one and only one of the *Entry Points* should be selected as the starting

point. An *End Point*, like its super-type (UML::) *Final Node*, indicates a point at which the workflow stops. Thus, it shall not have outgoing *Control Flows*.

A *Decision Node* is a control node that chooses one between the outgoing *Control Flows*. It is a subtype of (UML::) *Decision Node*, but it presents a slightly different semantics. OPL-ML's *Decision Node* shall have at least one incoming *Control Flow*, and more than one outgoing *Control Flow*. Guard conditions shall be specified to help decide which *Control Flow* to follow. UML's *Decision Node*, in turn, does not admit multiple incoming *Control Flows*. We decided to admit multiple incoming *Control Flows* in OPL-ML to not need to include UML's *Merge Node* in OPL-ML meta-model. As a consequence, the diagrams built using OPL-ML tend to become simpler.

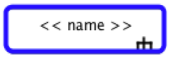
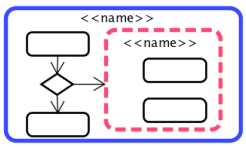
Fork Node and *Join Node* are subtypes of their homonymous counterparts in UML's meta-model, and preserve the same semantics. *Fork Node* is a control node that splits a flow into multiple concurrent flows. It shall have exactly one incoming *Control Flow* and multiple outgoing *Control Flows*, which must be followed. *Join Node* is a control node that synchronizes multiple flows. It shall have exactly one outgoing *Control Flow* but may have multiple incoming *Control Flows*.

Like patterns, *Patterns Application Actions* can be grouped. A *Pattern Application Action Group* groups a set of *Patterns Application Actions*, plus the *Control Flows* and *Control Nodes* establishing the workflow inside the action group. Thus *Pattern Application Action Group* is a subtype of (UML::) *Activity Group*, and as such, may be composed by other *Pattern Application Action Groups*. A *Pattern Application Action Group* should refer to a *Pattern Group* in the OPL Structural Model. When a *Pattern Application Action Group* refers to a *Variant Pattern Group*, then it is said a *Variant Pattern Application Action Group*. A *Variant Pattern Application Action Group* captures alternative *Pattern Application Actions* to be accomplished. Only one of them can be selected. Thus, *Variant Pattern Application Action Groups* do not represent workflows, and they do not admit *Control Flows* and *Control Nodes* inside them.

By developing the OPL Process meta-model as an extension of the UML's meta-model for activity diagrams, we could use the same concrete syntax for most of its elements, namely: Pattern Application Action (UML::Action), Control Flow (UML:: Control Flow), Entry Point (UML::Initial Node), End Point (UML::Final Node), Decision Node (UML::Decision Node), Fork Node (UML::Fork Node), and Join Node (UML::Join Node). See these notations in [OMG, 2015]. Only for groups we decided to make some extensions also in the concrete syntax.

Like in the case of the Structural Model, we decided to provide two ways to represent groups: one as a black box, and the other in an expanded format. In the black box representation, *Pattern Application Action Groups* are represented as UML::*Call Behavior Actions* with the adornment for calling activities, the rake-style symbol () [16]. Besides, we suggest setting blue to the border line as a redundant coding [8]. Variant Pattern Application Action Groups are represented by the same symbol, but with red dashed lines. In the expanded format, Pattern Application Action Groups are represented by means of closed regions delimited by blue straight lines, with rounded corners; while Variant Pattern Application Action Groups are represented by means of closed regions delimited by red dashed lines, with rounded corners. Table 2 shows the concrete syntax for representing (Variant) Pattern Application Action Groups.

Table 2. Symbols of the visual notation for groups of pattern application actions.

Element	Symbol
Pattern Application Action Group (black box format)	
Pattern Application Action Group (expanded format)	
Variant Pattern Application Action Group (black box format)	
Variant Application Action Group (expanded format)	

5 OPL-ML Evaluation

To preliminary evaluate OPL-ML, we used it to reengineer the Service OPL (S-OPL) [14]. Next, we discuss the main changes made during the S-OPL reengineering and present some S-OPL fragments. The full specifications of the first and the current version of S-OPL are available at *https://nemo.inf.ufes.br/projects/opl/s-opl/*.

(i) Separation of structural and behavior aspects: as previously discussed, different models should be used to address structural and behavioral aspects of an OPL. S-OPL was represented only by a process model. Thus, we created a structural model for S-OPL. Figure 2 shows a fragment of the defined structural model.

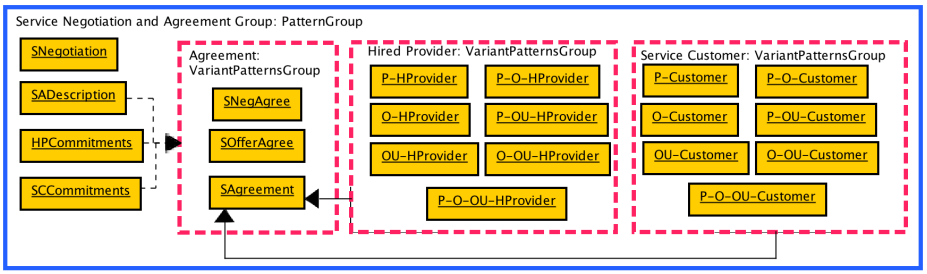


Fig. 2. Structural Model of the Service Negotiation and Agreement Group.

(ii) Complexity management: for managing complexity, we added to S-OPL a general behavior model (Figure 3) in which patterns groups are presented in the black box format, providing a clearer view of the whole process.

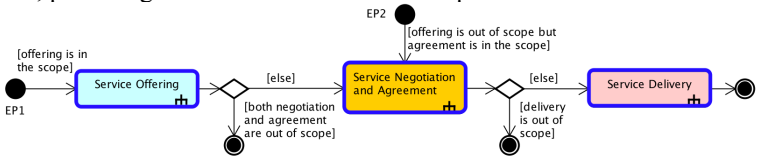


Fig. 3. – S-OPL Process Model (general view).

(iii) Creation of a main mandatory flow: in the S-OPL old version, the process flow was defined mainly by optional flows (only some flows were mandatory). Thus, when developing an ontology, the ontology engineer could stop following the process at any point (except when there were mandatory flows to follow). According to OPL-ML, a process must be followed from an entry point to an end point. Considering that, we reengineered the S-OPL process creating a main flow to be followed according to the ontology engineer decisions until s/he reaches an end point.

For creating the main flow, decision nodes were used to indicate optional flows. Decision nodes make it clear that a decision must be taken by the engineer and that one (and only one) of the output flows must be followed. Moreover, we changed the fork node semantics. In the S-OPL old version, outputs of fork nodes were optional flows. According to OPL-ML, outputs of fork nodes are mandatory flows, i.e., when following the input flow of a fork node, all its output flows must be followed. In order to not interrupt the process main flow, we used joint nodes to converge the output flows into a single one that take up the main flow. Finally, aiming to improve the process flow, we made some changes in patterns grouping. Figure 4 shows a fragment of the old version of the process model defined for the Service Negotiation and Agreement group. Figure 5 presents the corresponding reengineered model.

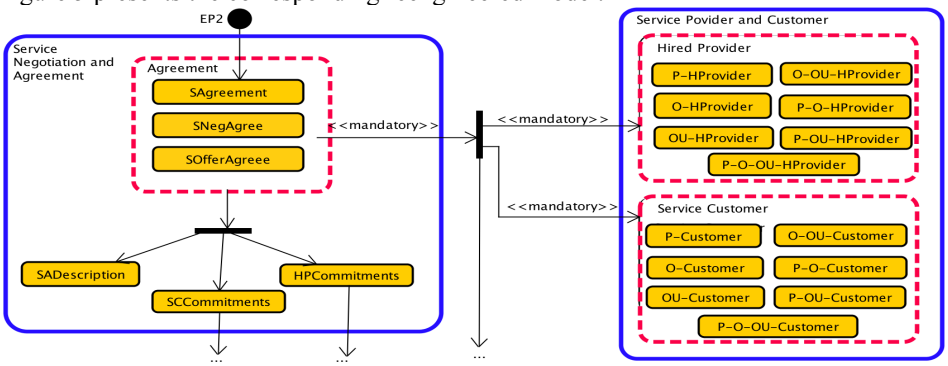


Fig. 4. Fragment of the old version of S-OPL process model.

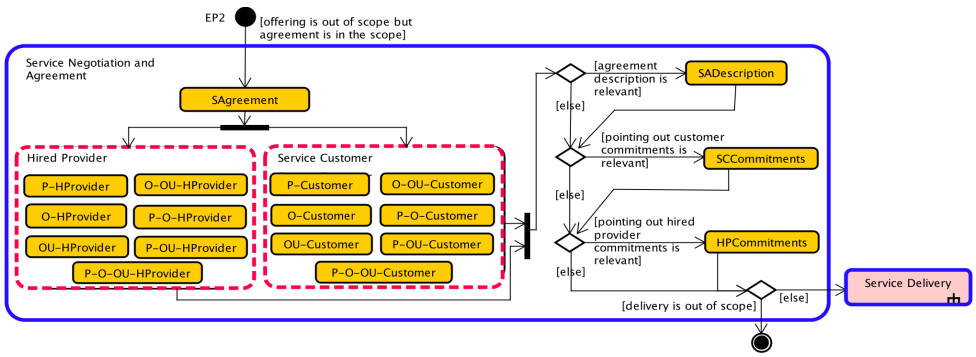


Fig. 5. Fragment of the reengineered S-OPL process model.

After reengineering S-OPL, aiming to evaluate its new version, we performed a study involving 6 of the 9 students that participated in the previous study with S-OPL.

The study goal was to evaluate: (i) if an OPL represented by using OPL-ML is easier to understand, and (ii) if the S-OPL new version is better than the previous one. The study procedure consisted of two steps. In the first step, the participants received the new specification of S-OPL and analyzed it. The specification contained the process and structural models, their descriptions, and the documentation of each pattern (the same of the previous version). In the second step, they provided feedback about the new version of S-OPL by filling a questionnaire containing two parts. The first one included questions about the difficulty for understanding the OPL, especially the different models representing structural and process aspects, and each one of the OPL symbols. The possible answers used a Likert Scale: very hard, hard, neutral, easy, and very easy. The second part concerned comparing the S-OPL new version with the previous one, considering process clarity and ease of understanding, and support provided by the structural model. Another questionnaire was applied to capture the participants' profile, analyzing their level of education, experience in conceptual modeling, experience in ontology development, and experience with OPLs. The group of participants has, mostly, medium experience (from one to three years) in conceptual modeling, medium/low experience in ontologies development and low experience (less than one year) with OPLs.

Concerning the OPL mechanism, one participant found it very easy, four found it easy, and one was neutral. As for the OPL symbols, except by one participant, who declared that it was difficult to understand variant patterns, all found easy or very easy to understand the symbols and reported that the notation is clear.

Comparing the S-OPL new version with the previous one, five participants said that in the new version the process is much clearer and one participant said that it is clearer. The participants commented that the main flow defined in the process made it easier to understand the process and follow it. Moreover, all participants found the new version better than the previous one. They pointed out that the different abstraction levels contribute to better understand the whole process, turning the process more intuitive and practical. Finally, five participants said that the structural model made it easier understand and use S-OPL. One participant said that the structural model made it much easier understanding and using S-OPL. Participants declared that the model served as a map to guide users through patterns relations, minimizing the effort spent searching the OPL textual specification.

The study results showed that an OPL modeled using OPL-ML is easier to understand. Moreover, the understanding about the OPL improved when compared with its previous representation. It is worthy noticing that the participants were not much experienced with ontologies and OPLs. Even so, they found it easy to understand the OPL. These results can be seen as initial evidences that OPL-ML is suitable for modeling OPLs. However, there are some limitations that do not allow us to generalize the results. As main limitations, we can point out: (i) the small number of participants in the study; (ii) the study was performed in an academic environment; (iv) when evaluating the new version of S-OPL, the participants had already used the S-OPL previous version, thus, the acquired knowledge about S-OPL might have influenced the understanding about its new version.

6 Related Works

As previously discussed, the use of OPLs is recent and the few works defining OPLs were done by the same research group, the Ontology and Conceptual Modeling Research Group (NEMO). This research group has proposed five OPLs [7]. In these OPLs, extensions of UML activity diagrams are used to represent the OPL processes. When comparing OPL-ML with the visual notation described in [7], the main similarity is that both represent the OPL process by means of extensions to the UML's activity diagram. As for differences, several advances are introduced by OPL-UML.

First, the visual notation described in [7] is limited, being not able to cover all aspects for properly representing an OPL. OPL-ML clearly defines the language abstract and concrete syntaxes. The abstract syntax is defined by a meta-model and part of it extends the UML meta-model for activity diagrams, including constructs not considered in [7] (e.g., end point). As for the concrete syntax, it was defined by following a systematic process applying the PoN principles, resulting in a cognitively rich visual notation.

Second, the OPLs presented in [7] are represented only by a process model. Structural aspects are not addressed, being difficult for users to identify relations between the patterns. OPL-ML proposes the use of two different models for representing an OPL: the structural model, dealing with the OPL structural elements and the relationships between them; and the process model, addressing the process to be followed for selecting and applying the patterns. Separating these views in two types of models contributes to better understand the OPL and the relationships between its elements.

Third, OPL-ML allows managing complexity by using models at different abstraction levels. A general model, using black box formats, can be created to represent a whole view of the OPL, while detailed models expand the black boxes and provide a detailed view of the OPL. By using this resource, large or complex OPLs can be easier understood. The visual notation described in [7] does not provide this facility.

Finally, by using OPL-ML, a main flow is defined in the OPL process, making it clear the paths (from beginning to end) to be followed by the ontology engineer according to his/her decisions. In the OPLs presented in [7], the process does not have a continuous flow and can be interrupted at any time.

As discussed in Section 2, the use of pattern languages (PLs) in Software Engineering (SE) are much more mature than in Ontology Engineering. This fact motivated us to use insight from SE PLs to define OPL-ML. However, despite of the use of PLs is already consolidated in SE, there are still problems regarding their visual representation. For instance, considering the investigated software-related PLs, only two of them use different models to address structural and behavioral aspects. In [17], Guerra et al. propose a PL for organizing the internal structure of metadata-based frameworks, which is represented by structural and navigation models. In [18], in turn, Zdun proposes a PL for designing aspect languages and aspect composition frameworks. Patterns and relationships between them are addressed by a relationship model, while a feature model helps patterns selection according to the features to be considered when designing frameworks. Concerning the structural model, Guerra et al. [17] address only dependency relationships, while in [18] the types of relationships vary and are not clear. As

for the process model, the navigation model described in [17] presents several sequences in which patterns can be applied. However, the visual notation is limited, being not possible, for example, to identify where the process starts or ends and which paths are mandatory. The feature model presented in [18], although can be helpful to patterns selection, does not provide a process to be followed. Thus, the sequence in which the patterns should be applied is not clear.

7 Final Considerations

As pointed by Blomqvist et al. [3], the works already done aiming to provide Ontology Pattern Languages (OPLs) are the first steps towards developing an ontology pattern representation language. In this paper, we take a step forward by providing a modeling language for representing OPLs: OPL-ML. Such language presents as striking features the following: (i) OPL-ML defines explicitly its abstract syntax by means of a metamodel; (ii) OPL-ML concrete syntax is designed following Moody's principles for designing cognitively effective visual notations [8]; (iii) OPL-ML is designed considering the results of a systematic mapping on visual notations for representing Software Engineering pattern languages, and from experimental studies involving the use of OPLs. We believe that the definition of a visual language for modeling OPLs amplifies the available resources for ontology engineers to develop OPLs, increasing productivity and contributing for advances in the area.

OPL-ML was used to reengineer S-OPL, a service OPL proposed in [14]. With S-OPL new version in hands, we performed a study to get perceptions from the OPL users. The findings indicate that OPL-ML use is viable and that it is capable of properly representing OPLs. However, the evaluation carried out by now is limited. Thus, new studies must be performed aiming to better evaluate OPL-ML. As future studies, we intend to reengineer the OPLs already developed at NEMO, develop new OPLs by using OPL-ML and evaluate the use of OPL-ML by other people, who can provide new feedback regarding OPL-ML use and effectiveness.

Besides, we have been working on the development of a supporting tool to aid ontology engineers in creating OPLs and using them to build ontologies.

Finally, although OPL-ML has been proposed to represent OPLs, we believe that it can be adapted and used to represent pattern languages in general. In this sense, we intend to investigate how to generalize OPL-ML, maybe including other elements typically used in other Software Engineering pattern languages.

Acknowledgements

This research is funded by the Brazilian Research Funding Agency CNPq (Processes 485368/2013-7 and 461777/2014-2) and FAPES (Process 69382549/2014).

References

1. Poveda-Villalón, M., Suárez-Figueroa, M.C., Gómez-Pérez, A., Reusing Ontology Design Patterns in a Context Ontology Network. In: Second Workshop on Ontology Patterns (WOP 2010), Shangai, China (2010).
2. Falbo, R.A., Guizzardi, G., Gangemi, A., and Presutti, V. Ontology patterns: clarifying concepts and terminology. In Proceedings of the 4th International Conference on Ontology and Semantic Web Patterns, Volume 1188, pages 14–26. CEUR-WS.org (2013).
3. Blomqvist, E., Hitzler, P., Janowicz, K., Krisnadhi, A., Narock, T., Solanki, M. Considerations regarding ontology design patterns. Semantic Web, 7(1):1–7 (2015).
4. Ruy, F., Guizzardi, G., Falbo, R.A., Reginato, C.C., Santos, V.A., From reference ontologies to ontology patterns and back, Data & Knowledge Engineering, March (2017).
5. Blomqvist, E., Gangemi, A., Presutti, V. Experiments on Pattern-based Ontology Design. In Proceedings of K-CAP 2009, pp. 41-48 (2009).
6. Falbo, R.A., Barcellos, M.P., Nardi, J.C., and Guizzardi, G. Organizing Ontology Design Patterns as Ontology Pattern Languages. Proceedings of the 10th Extended Semantic Web Conference - ESWC 2013, Montpellier, France (2013).
7. 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).
8. Moody, D.L.: The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE Transactions on Software Engineering. 35(6), 1–22 (2009).
9. Teixeira, M. G. S., Quirino, G. K., Gailly, F., Falbo, R. A., Guizzardi, G., Barcellos, M. P., PoN-S: A Systematic Approach for Applying the Physics of Notation (PoN). In 21st International Conference in Exploring Modelling Methods for Systems Analysis and Design (EMMSAD'2016), Ljubljana, Slovenia, pp. 432–447 (2016).
10. Wieringa, R.J., Design Science Methodology for Information Systems and Software Engineering, Springer-Verlag Berlin Heidelberg (2014).
11. Schmidt, D., Stal, M., Rohnert, H., Buschmann, F.: Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects. Wiley Publishing (2000).
12. Kitchenham, B., Charters, S. Guidelines for performing systematic literature reviews in software engineering - Version 2.3. EBSE Technical Report EBSE -2007-01 (2007).
13. Ruy, F.B., Falbo, R.A., Barcellos, M.P., Guizzardi, G., Quirino, G.K.S. An ISO-based software process ontology pattern language and its application for harmonizing standards. *SIGAPP Appl. Comput. Rev.* 15 (2), 27-40 (2015).
14. Falbo, R.A., Quirino, G.K., Nardi, J., Barcellos, M.P., Guizzardi, G., Guarino, N., Longo, A., Livieri, B., An Ontology Pattern Language for Service Modeling. In: 31st ACM/SIGAPP Symposium on Applied Computing. Pisa, Italy (2016).
15. Moody, D.L., Heymans, P., Matulevicius, R.,Visual syntax does matter: improving the cognitive effectiveness of the i* visual notation. Requirements Engineering, 15(2) (2010).
16. OMG. OMG Unified Modeling Language Version 2.5. (2015).
17. Guerra, E.; Alvez, F.; Kulesza, U.; Fernandes, C., A reference architecture for organizing the internal structure of metadata-based frameworks. Journal of Systems and Software, 86(5), pp. 1239–1256 (2013).
18. Zdun, U., Pattern language for the design of aspect languages and aspect composition frameworks. IEE Proceedings - Software 151(2), 67–83 (2004).