Organizing Ontology Design Patterns as Ontology Pattern Languages

Ricardo de Almeida Falbo, Monalessa Perini Barcellos, Julio Cesar Nardi, Giancarlo Guizzardi

Ontology and Conceptual Modeling Research Group (NEMO), Computer Science Department, Federal University of Espírito Santo, Vitória, Brazil
{falbo, monalessa, jnardi, gguizzardi}@inf.ufes.br

Abstract. Ontology design patterns have been pointed out as a promising approach for ontology engineering. The goal of this paper is twofold. Firstly, based on well-established works in Software Engineering, we revisit the notion of ontology patterns in Ontology Engineering to introduce the notion of ontology pattern language as a way to organize related ontology patterns. Secondly, we present an overview of a software process ontology pattern language.

Keywords: ontology design patterns, ontology pattern language, software process ontology

1 Introduction

Although nowadays ontology engineers are supported by a wide range of ontology engineering methods and tools, building ontologies is still a difficult task even for experts. In this context, reuse is pointed out as a promising approach for ontology engineering. Ontology reuse allows speeding up the ontology development process, saving time and money, and promoting the application of good practices [1]. However, ontology reuse in general is a hard research issue, and one of the most challenging and neglected areas of ontology engineering [2]. The problems of selecting the right ontologies for reuse, specializing them, and composing several ontology fragments are not properly addressed yet [3].

Ontology Design Patterns (ODPs) are an emerging approach that favors the reuse of encoded experiences and good practices. ODPs are modeling solutions to solve recurrent ontology development problems [4]. Experiments, such as the ones conducted by Blomqvist et al. [3], show that ontology engineers perceive ODPs as useful, and that the quality and usability of the resulting ontologies are improved. However, compared with Software Engineering, where patterns have been used for a long period, patterns in Ontology Engineering are still in infancy. The earliest works addressing the issue of patterns in Ontology Engineering are from the beginning of the 2000s (e.g. [5]), and only recently this approach has gained more attention in this area [1, 2, 3, 4] and in the Semantic Web area [6].

A striking feature of the current use of patterns in Ontology Engineering is that they are generally being applied as stand-alone entities. However, as pointed out by Alexander and colleagues in their pioneering work [7], each pattern can exist only to the extent that it is supported by other patterns. This is especially important to ontology patterns that are related to a specific domain.

Although many ODPs in the literature refer to others, most of these references fail to give more complete guidelines on how the patterns can be combined to form solutions to larger problems. Contexts and problem descriptions are usually stated as general as possible, so that each pattern can be applied in a wide variety of situations. In addition, solution descriptions tend to focus on applying the patterns in isolation, and do not properly address issues that arise when multiple patterns are applied in overlapping ways, such as the order in which they can be applied. This situation is problematic, since the features introduced by applying one pattern may be required by the next. A larger context is therefore needed to describe the 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* [8].

It is important to highlight that we borrowed the term "pattern language" from Software Engineering (SE), where patterns have been studied and applied for a long time. A pattern language, in a SE view, *is a network of interrelated patterns that defines a process for systematically solving coarse-grained software development problems* [8, 9]. Thus, we are not actually talking about a language properly speaking. In "pattern language", the use of 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. However, if we focus on a more general concept of a *representation system*, we can consider the constituent patterns as an alphabet of higher-granularity primitives. Moreover, in this case, we can consider the procedural rules prescribing how these primitives can be lawfully combined as defining a set of valid possible instantiations for that representation system.

That all said, perhaps a more appropriate name would be a *"Pattern System"*. In any case, since we intend to reuse notions well-established in SE to apply them in Ontology Engineering as well as connect to the tradition in that area, we decided to keep here the term "pattern language". Thus, we define Ontology Pattern Language (OPL) as a network of interrelated domain-related ontology patterns that provides holistic support for solving ontology development problems for a specific domain.

An OPL contains a set of interrelated domain-related ontology patterns, plus a process providing explicit guidance on what problems can arise in that domain, informing the order to address these problems, and suggesting one or more patterns to solve each specific problem. It is worthwhile to point out that, although an OPL provides a process describing how to use the patterns to address problems related to a specific domain, an OPL is not a method for building ontologies. It only deals with reuse in ontology development, and its guidance can be followed by ontology engineers using whatever ontology development method that considers ontology reuse as one of its activities.

According to Schmidt et al. [10], the trend in the SE patterns community is towards defining pattern languages, rather than stand-alone patterns. We advocate this should also be taken into account in Ontology Engineering, mainly for a class of ontologies called *Core Ontologies*. Core ontologies provide a precise definition of structural knowledge in a specific field that spans across different application domains in this field [11]. Thus, we argue that core ontologies are good candidates to be presented as ontology pattern languages.

In summary, the contribution of this paper is to incorporate ideas from patterns as used in Software Engineering to patterns in Ontology Engineering. Firstly, based on well-established works in Software Engineering, such as [9], we revisit the notion of ontology patterns in Ontology Engineering, and introduce the notion of Ontology Pattern Language as a way to organize domain-related ontology patterns. Secondly, we present a particular ontology pattern language in the Software Process domain.

This paper is organized as follows. In Section 2, we present pattern-related concepts, mainly as used in Software Engineering. In Section 3, we discuss ontologies focusing on their generality level. This discussion is important in the context of this paper to point out which is the generality level that we believe to be the most appropriate to build OPLs. In Section 4, we discuss ontology patterns and we introduce the notion of Ontology Pattern Language. In Section 5, we briefly present the Software Process Ontology Pattern Language (SP-OPL), and an example showing its use for building a fragment of a measurement process ontology. Section 6 discusses related works. Finally, in Section 7, we present the final considerations of the paper.

2 On Patterns and Pattern Languages

Patterns are vehicles for encapsulating knowledge. They are considered one of the most effective means for naming, organizing, and reasoning about design knowledge. "Design knowledge" here is employed in a general sense, meaning design in several different areas, such as Architecture and Software Engineering (SE). According to Buschmann et al. [9], "a pattern describes a particular recurring design problem that arises in specific design contexts and presents a well-proven solution for the problem. The solution is specified by describing the roles of its constituent participants, their responsibilities and relationships, and the ways in which they collaborate".

In SE, there are several types of patterns. The best known are analysis patterns, design patterns and idioms. An analysis pattern is a pattern that describes how to model a particular kind of problem in an application domain. A design pattern provides a scheme for refining elements of a software system or the relationships between them. An idiom is a pattern specific to a programming language or environment. An idiom describes how to implement particular behavior or structures in code using the features of the given language or environment [9].

Patterns are often considered and applied separately. However, no pattern is an island. Contrariwise, patterns are fond of company: sometimes with one pattern as an alternative to another, sometimes with one pattern as an adjunct to another, sometimes with a number of patterns bound together as a tightly-knit group. The manifold rela-

tionships that can exist between patterns help to strengthen and extend the power of an individual pattern beyond its specific focus [9].

A pattern language is a set of patterns and relationships among them that can be used to systematically solve coarse-grained problems [8]. A pattern language defines a process that aims to provide holistic support for using the patterns to address problems related to a specific technical or application domain. This holistic view should provide explicit guidance on what problems can arise in the domain, inform the order to address them, and suggest one or more patterns to solve each specific problem [9]. A pattern language should also provide guidelines showing how the patterns can be composed to form solutions to problems [8]. The patterns in a pattern language are usually designed to be used within the context of the language. Therefore, they tend to be tightly coupled, and it is difficult or even impossible to use them in isolation [8].

3 Ontologies and Their Generality Levels

There are different classifications of ontologies in the literature. In the context of this work, we are interested in the one that classifies ontologies according to their generality levels, discriminating between foundational, core and domain ontologies [11].

At the highest level of generality, there are the foundational ontologies. Foundational ontologies span across many fields and model the very basic and general concepts and relations that make up the world, such as object, event, parthood relation etc. [12, 13, 14]. Domain ontologies, in turn, describe the conceptualization related to a given domain, such as electrocardiogram in medicine [12]. With a level of generality between that of foundational and domain ontologies, there are core ontologies. Core ontologies provide a precise definition of structural knowledge in a specific field that spans across different application domains in this field. These ontologies are built based on foundational ontologies and provide a refinement to them by adding detailed concepts and relations in their specific field [11].

Guizzardi [15] makes an important distinction between ontologies as conceptual models, known as *reference ontologies*, and ontologies as coding artifacts, called here *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 [15]. On the other hand, once users have already agreed on a common conceptualization, operational versions of a reference ontology can be created. Contrary to reference ontologies, operational ontologies are designed with the focus on guaranteeing desirable computational properties.

Although we agree with Scherp et al.'s classification for ontologies [11], we perceive them as a continuum, ranging from pure foundational ontologies, such as DOLCE [13] and UFO (Parts A [14] and B [16]), to domain ontologies. In our view, there can be different levels of generality in ontologies that are classified as, for instance, core ontologies. In [11], for example, three core ontologies are presented: Event-Model-F provides a formal representation of the different aspects of events in which humans participate; The Core Ontology on Multimedia (COMM) describes

arbitrary digital media data; The Cross-Context Semantic Information Management Ontology (X-COSIMO) allows representing the communication taking place between different persons and systems and the information associated with it. Although all three are built based on DOLCE and classified as core ontologies, in our opinion, Event-Model-F is more general than COMM and X-COSIMO, since the last two address conceptualizations that are closer to a domain conceptualization (multimedia and personal information management, respectively) than the former (events for representing human experience).

We have experienced such situations when developing ontologies for the software process domain. Originally, we classified our Software Process Ontology (SPO) [16, 17] as a reference domain ontology. However, it has been used as basis for developing other reference domain ontologies related to specific software processes, such as the measurement process (Reference Software Measurement Ontology (RSMO) [18]). The latest version of SPO [17] is grounded in UFO-C, an ontology of social entities [16]. In [16], UFO-C is classified as a foundational ontology, but it builds on top of UFO-A (an ontology of endurants) and UFO-B (an ontology of events) to systematized social concepts such as action, goal, agent, commitment, among others.

In the light of the above, we see those categories of ontologies (foundational, core and domain ontologies) as regions in a spectrum with fuzzy boundaries between them. Figure 1 illustrates this continuous view using the aforementioned ontologies. DOLCE, UFO-A and UFO-B are genuine foundational ontologies. UFO-C and Event-Model-F are in the frontier between foundational and core ontologies. X-COSIMO, COMM and SPO are core ontologies, but the last is in the region closer to domain ontologies. Finally, RSMO is classified as a domain ontology.

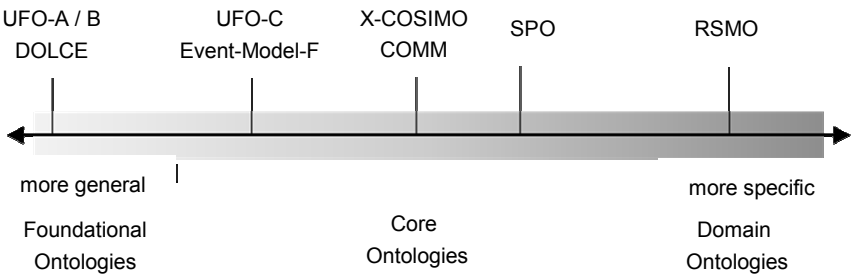

Fig. 1. Ontology level of generality as a continuum

In this paper we are interested in core ontologies, mainly those that are in a region closer to domain ontologies. Ontologies in this region, although general enough to be specialized when applied to more restrict domains, are still domain-related. We claim that these core ontologies should be presented as ontology pattern languages. Moreover, we are interested in patterns to support the development of reference domain ontologies [15], which are to be reused in the conceptualization phase. In the next section, we present a fuller argumentation defending our view that patterns defined in the level of Core Ontologies are the ones which can be most appropriately defined as a Pattern System or Pattern Language.

4 Ontology Design Patterns and Ontology Pattern Languages

According to Gangemi and Presutti [2], an Ontology Design Pattern (ODP) is a modeling solution to solve a recurrent ontology design problem. ODPs can be of different types, such as content, logical, architectural, and so on. Content Ontology Patterns (COPs) refer to small fragments of ontology conceptual models, and must be language-independent [2]. A COP can extract a fragment of either a foundational or a core ontology, which constitutes its background [19]. Thus, we consider two types of COPs: Foundational (FOPs) and Domain-related ontology patterns (DROPs).

Since FOPs are COPs extracted from foundational ontologies, they tend to be more generally applied. Although they certainly have dependencies with other patterns, these dependencies tend to be weaker, and the pattern is easily applied in isolation. Take the example discussed in [1] for the development of a context ontology network called mIO!. The reused patterns were selected among ODPs present in catalogues such as the one available in the `ontologydesignpatterns.org` portal. The reused patterns were related to general (formal) problems, such as taxonomical or part-whole relations, n-ary relations/participation. All the reused patterns are FOPs. None of the examples there are of DROPs.

In contrast, DROPs for a specific domain are very inter-related, and it is very difficult (if not impossible) to apply them in isolation. It is important to highlight, nonetheless, that as patterns move closer to a Domain ontology, they agglutinate to form a *stable model*, i.e., the constraints on how they can be inter-related become so strong that the very domain model is practically the only way they can appear together, thus, lacking the *potential for recurrence* which is part of the very definition of what a pattern is. That is why we advocate that DROPs occurring at the level of Core Ontologies are the best candidates for being organized as ontology pattern languages.

Regarding the way they are documented and communicated, COPs, in general, are comparable to design patterns in Software Engineering [2]. On the other hand, regarding their contents, DROPs are comparable to Software Engineering analysis patterns.

COPs should be encoded in a higher-order representation language [2]. OntoUML [14] is an example of an ontology representation language that is suitable for this purpose. OntoUML is a UML profile that enables modelers to make finer-grained modeling distinctions between different types of classes and relations according to ontological distinctions put forth by UFO-A. Thus, we advocate for the use of OntoUML as a modeling language for DROPs in an OPL. On the other hand, Gangemi and Presutti [2] state that "a (sample) representation in OWL is needed in order to (re)use the patterns as building blocks over the Semantic Web". We agree that an example in OWL could be useful, but it is not a requisite for DROPs. DROPs are to be reused in the conceptualization phase. If they have a counterpart implemented in some language (such as OWL), this operational version of the pattern can also be reused, amplifying the benefits of applying the pattern. However, we defend here that DROPs should be captured in a codification language independent manner. This allows for a modeling solution to be implemented in multiple codification languages.

A COP has to be small (typically two to ten classes with relations defined between them) [2]. Moreover, a COP can be an element in a partial order, where the ordering

relation requires that at least one of the classes or relations in the pattern is specialized [2]. These characteristics are essential for DROPs in an OPL. A user should be able to read the pattern, understand its applicability and decide if it is useful for the problem at hands or not. Once decided which DROPs to reuse, the user can specialize their concepts and relations.

A domain ontology typically results from the composition of several COPs, with appropriate dependencies between them, plus the necessary design expansion based on specific needs [3]. Making this knowledge explicit is essential for achieving the main benefits of reuse. Thus, organizing DROPs in catalogues is not a good choice. In a conventional catalog there is a lack of a strong sense of connection. We need something stronger than simply knowing that another pattern in the collection is related in some way. When collections are presented in conjunction with, for example, pattern sequences, we start to get a stronger sense of connection [9]. This is especially important for reusing DROPs.

An Ontology Pattern Language (OPL) aims to provide holistic support for using DROPs in ontology development for a specific application domain. It should provide explicit guidance on what problems can arise in that domain, inform the order to address these problems, and suggest one or more patterns to solve each specific problem. Thus, an OPL should support the explicit consideration of complementing or conflicting pattern combinations to solve a given problem, along with guidelines for integrating patterns into a concrete ontology conceptual model.

An OPL should indicate explicitly which referenced patterns address mandatory aspects and which ones address optional aspects. To ensure a stable and sound pattern application, referenced patterns should be presented in the suggested application order. Without this explicit procedural guidance, a representation that fits the basic network of the patterns might not provide a suitable process that helps to ensure a sufficiently complete and well-formed ontology.

OPLs are structured to support and encourage the application of one pattern at a time, in the order defined by the pattern sequences that result from the chosen paths through the language. This guideline ensures that the main property of piecemeal growth is preserved: the 'whole' always precedes its 'parts'. A pattern language is of little use if its audience loses the big picture. Conversely, the essential information of each individual pattern within the language must still be preserved [9].

In summary, an OPL should give concrete and thoughtful guidance for developing ontologies in a given domain, addressing at least the following issues: (i) What are the key problems to solve in the domain of interest? (ii) In what order should these problems be tackled? (iii) What alternatives exist for solving a given problem? (iv) How should dependencies between problems be handled? (v) How to resolve each individual problem most effectively in the presence of its surrounding problems?

Using the notion of OPLs, we can reorganize ontology pattern catalogues. We might provide an entry in a catalogue for each domain of interest. Each entry in the catalogue, in turn, can be viewed as a special purpose pattern language that advises developers how to construct a domain ontology with the help of DROPs.

For illustrating the ideas discussed above, in the next section, we present an OPL in the domain of Software Process. The patterns there were extracted from the Software

Process Ontology (SPO) presented in [17]. SPO has been developed since 1997, and it results from several revisions. The latest version was obtained as a result of a reengineering effort to ground it in UFO [17]. In the version presented here, we managed to advance further improvements, mainly regarding modularity, which directly affects reusability. For this reason, we decided to restructure SPO as an OPL.

5 An Ontology Pattern Language for the Software Process Domain (SP-OPL)

Figure 2 shows a UML activity diagram giving an overview of the SP-OPL. An activity diagram is one of the possible modeling notations comprising UML and it is the standard UML notation for representing temporal sequencing constraints between activity types and, hence, for specifying the possible order of execution between activities. In the model of Fig. 2, we use activities in an activity diagram (rounded rectangles) to represent specific patterns. Moreover, we use the activity ordering notation to represent the procedural rules governing the admissible sequences in which these patterns can be used. In that diagram, an extension to the original UML notation (dotted lines with arrows) was introduced to show variant patterns.

It is important to emphasize that we would have employed activity diagrams (or a language with similar representation capabilities) for that purpose regardless of the domain under study, i.e., the choice for using an activity-ordering language is related to the need for defining the permissible sequence of instantiation of the patterns. In particular, it bears no relation to the fact that, incidentally, the domain under study is about *Software Processes*.

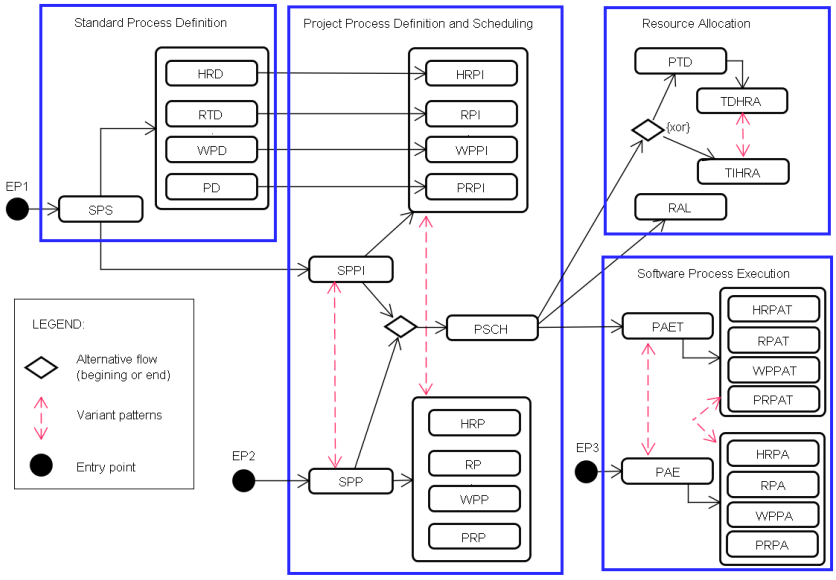

Fig. 2. Software Process Ontology Pattern Language (SP-OPL)

The main problem areas addressed by the SP-OPL are: *Standard Process Definition*, *Project Process Definition and Scheduling*, *Resource Allocation*, and *Software Process Execution*. Table 1 shows the patterns that compose the SP-OPL.

As shown in Fig. 2, SP-OPL has three entry points, depending on the focus of the ontology engineer. When the requirements for the domain ontology being developed include problems related to *Standard Process Definition*, the start point is EP1. In this case, first the ontology engineer should address problems related to how a standard process is structured in terms of standard sub-processes and activities (SPS). Following, he can optionally address problems related to the definition of human roles (HRD), types of resources (hardware and software) (RD), types of work products required (input) and produced (output) (WPD), and procedures (methods, techniques, guidelines etc.) (PD) that are required for performing each standard activity when it is instantiated in the scope of a project.

When the requirements for the ontology being developed include problems related to *Project Process Definition and Scheduling*, the start point is either EP2 or SPS. In this case, the ontology engineer has to first deal with problems related with the process planning in terms of project sub-processes and activities. If there is already defined a standard process, project process planning can be done by means of instantiating the standard process (SPI – Software Process Planning via Instantiation)[1], otherwise, the ontology engineer should consider planning the project process from scratch (SPP). Once defined the project processes and activities, he can treat modeling problems related to scheduling them (PSCH). Moreover, the ontology engineer can optionally treat modeling problems related to planning human roles (HRPI/HRP), types of resources (hardware and software) (RPI/RP), types of work products required (input) and produced (output) (WPPI/WPP), and procedures (methods, techniques, guidelines etc.) (PRPI/PRP) that are required for performing each project activity.

For dealing with problems related to *Resource Allocation*, it is necessary to have the project process planed and scheduled. Resource Allocation involves patterns regarding hardware and software resource allocation (RAL), project team definition (PTD), and human resource allocation. Human resource allocation problems can be solved considering constraints imposed by a project team (TDHRA) or not (TIHRA).

Finally, when there are requirements related to the *Software Process Execution*, the start point is either EP3 or PSCH. EP3 should be chosen when it is not a requirement for the ontology to address process planning and scheduling. In this case, the ontology engineer has to first deal with problems related to the execution of processes and activities (PAE). Then he can address problems related to resource (human and other) participation (HRPA and RPA), procedures adopted (PRPA), and work product inputs and outputs (WPPA). On the other hand, if the project process is already scheduled, it is possible to address problems related to process and activity execution and tracking, which involves the corresponding variant patterns PAET, HRPAT, RPAT, PRPAT

[1] The patterns SPPI, HRPI, RPI, WPPI, and PRPI shown in Fig. 2 are not listed in Table 1, due space limitations. Those patterns are variant patterns of SPP, HRP, RP, WPP and PRP, respectively, considering that they address the same problems, but considering the instantiation of a standard process or activity.

and WPPAT. These patterns, which are not shown in Table 1, address the same problems described above, but considering that it is possible to check if the execution of activities and processes conforms to their previous definition (process tracking).

Table 1. Domain-Related Ontology Patterns (DROPs) in the SP-OLP

Id	Name	Intent
Standard Process Definition		
SPS	Standard Process Structure	Represents how a standard software process is defined in terms of standard sub-processes and activities
HRD	Standard Activity Human Role Definition	Defines the human roles responsible for performing a standard activity in the projects that instantiate it
RTD	Standard Activity Resource Type Definition	Defines the types of resources (hardware and software) required for performing a standard activity
WPD	Standard Activity Work Product Definition	Defines the types of work products required (input) and produced (output) when performing a standard activity
PD	Standard Activity Procedure Definition	Defines the procedures (methods, techniques, guidelines etc.) to be applied when performing a standard activity
Project Process Definition and Scheduling		
SPP	Software Process Planning	Represents how a software process is planned in terms of sub-processes and activities
PSCH	Process Scheduling	Defines the time boundary for project processes and activities
HRP	Human Role Planning	Defines the human roles responsible for performing a project activity
RP	Resource Planning	Defines the types of resources (hardware and software) required for performing a project activity
WPP	Work Product Planning	Defines the types of work products required (input) and produced (output) when performing a project activity
PRP	Procedure Planning	Defines the procedures (methods, techniques, guidelines etc.) to be applied when performing a project activity
Resource Allocation		
PTD	Project Team Definition	Defines the human resources that are member of a project team
TDHRA	Team-dependent Human Resource Allocation	Allocates human resources to project activities, considering team allocation constraints
TIHRA	Team-independent Human Resource Allocation	Allocates human resources to project activities, when there is not a project team formally defined
RAL	Resource Allocation	Allocates resources (hardware equipments and software tools) to project activities
Software Process Execution		
PAE	Process and Activity Execution	Register the occurrences of processes and activities.
HRPA	Human Resource Participation	Registers the participation of Human Resources in an activity occurrence
RPA	Resource Participation	Registers the participation of Resources (hardware equipment or software tool) in an activity occurrence
WPPA	Work Product Participation	Register the participation of Work Products (as input or output) in an activity occurrence.
PRPA	Procedure Participation	Register the adoption of procedures by an activity occurrence

Figure 3 shows the conceptual model of the "Process and Activity Execution (PAE)" DROP. The intent of this pattern is to represent the occurrences of processes and activities in the context of a project, and their mereological structure. The following competency questions are addressed by this pattern: (CQ1) How is a process occurrence structured in terms of sub-processes and activities? (CQ2) When did a process/activity occurrence start and when did it end? (CQ3) From which activity occurrences does an activity occurrence depend on?

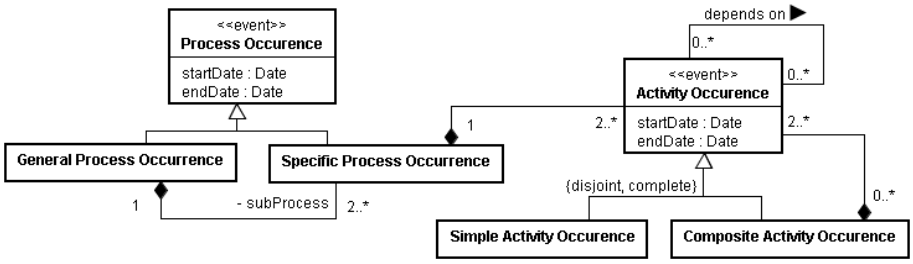

Fig. 3. The "Process and Activity Execution" (PAE) pattern

The foundations for the PAE pattern were given by UFO-B [16]. **Process Occurrences** and **Activity Occurrences** are complex events, and the whole-part relations between events are strict partial order. In the software process domain, there are two main kinds of **Process Occurrences**: **General Process Occurrence** and **Specific Process Occurrence**. A general process occurrence is the whole execution of a process. It is composed of specific process occurrences, allowing an organization to decompose a general process into sub-processes. A specific process occurrence, in turn, is decomposed into **Activity Occurrences**. Activity occurrences can be simple or composite. A composite activity occurrence is a complex event that is composed by other activity occurrences. A simple activity occurrence is not composed by other activity occurrences, but it is still a complex event in UFO-B, since it is composed by other events representing the participations of human resources, hardware and software resources, work products, and procedures in the activity occurrence.

The PAE pattern has some related patterns, with different types of relations holding between them. PAE has a variant pattern, the "Process and Activity Execution and Tracking (PAET)" pattern, which is an alternative to PAE when a project has a process previously defined and scheduled, allowing to track the execution against to what was previously planned. When PAE is used, its use can be followed by the use of patterns whose intent is to represent the participations of human resources (HRPA), software and hardware resources (RPA), procedures (PRPA), and work products (WPPA). Figure 4 presents the conceptual model of the WPPA pattern.

This pattern shows that an activity occurrence can have as its parts **Artifact Participations**, which are also events. An artifact participation is the participation of a single artifact. This is in line with UFO-B, which says that events are ontologically dependent entities in the sense that they existentially depend on objects in order to exist. **Artifact**, in turn, is a category in UFO-A [14], since it is a dispersive universal that aggregates essential properties (not shown in this pattern) that are common to

different subtypes of artifacts. Artifact participations can be of three types: (i) **Artifact Creation**, meaning that the artifact is created during the activity occurrence, and thus it is an output of this activity occurrence (the **/produces** derived relation); (ii) **Artifact Usage**, meaning that the artifact is only used during the activity occurrence, and thus it is only an input for the activity occurrence (the **/uses** derived relation); and (iii) **Artifact Change**, meaning that the artifact is changed during the activity occurrence, and thus it is both input and output of the activity occurrence. The foundations for this conceptualization are given by UFO-C [16], which defines four types of resource participations: creation, termination, usage and change. In the case of software processes, we consider that artifacts are not thrown away in activity occurrences, and thus there is not a case of termination participation in this domain.

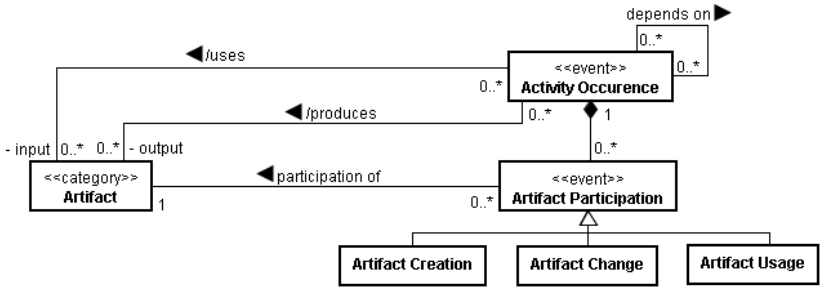

Fig. 4. The "Work Product Participation" (WPPA) pattern

SP-OPL was used for building a domain ontology about the software measurement process. Figure 5 shows a fragment of this domain ontology, considering the reuse of the two patterns presented before (PAE and WPPA). Concepts reused from the patterns are presented in grey.

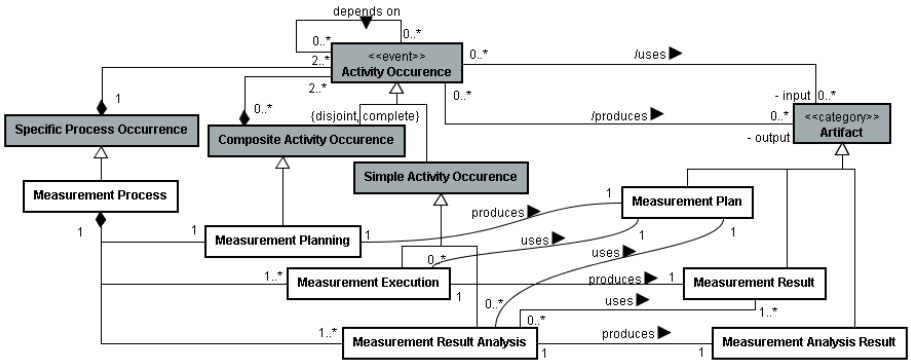

Fig. 5. A fragment of a Domain Ontology for the Software Measurement Process

As shown in Fig. 5, the **Measurement Process** is composed by activity occurrences of **Measurement Planning**, **Execution**, and **Result Analysis**. The first one is a composite activity occurrence, although, for simplicity, its parts are not shown in the

figure. The other two are simple activity occurrences. Measurement Planning produces a **Measurement Plan**, which is used by the other activity occurrences (Measurement Execution and Result Analysis occurrences). Measurement Execution produces **Measurement Results**, which are used by Measurement Result Analysis for producing **Measurement Analysis Results**.

6 Related Works

Our work is strongly inspired, on one side, by works on Ontology Design Patterns, especially those developed by Gangemi, Presutti and colleagues [2, 3, 19]; on the other side, by works on Pattern Languages in Software Engineering, especially those developed by Buschmann, Schmidt and colleagues [9, 10]. In fact, we believe that our main contribution in this paper is to introduce the idea of pattern languages, as used in Software Engineering, in the field of Ontology Design Patterns, which is especially important for Ontology Engineering and consequently for Semantic Web.

At the best of our knowledge, we are the first to organize domain-related ontology patterns as Ontology Patterns Languages (OPLs). However, it is important to reinforce that we borrowed the term "pattern language" from Software Engineering (SE), where it has a special meaning [8, 9]. A pattern language, in this context, is a network of interrelated patterns, plus a process for systematically solving software development problems [8, 9]. Highlighting the particular meaning that we associate to the term Pattern Language is particularly important in order to avoid confusion with existing literature. For instance, in [20], Noppens and Liebig seek to develop a language to encode OWL patterns in a declarative way. They did not use the term OPL in the sense we did.

Finally, although an OPL defines a process for traveling along the patterns, it is not a method for building ontologies. An OPL can be used jointly with several methods. For instance, the measurement process ontology partially presented in Section 5 was developed using the method SABiO [21], adapting one of its activities (Reusing Existing Ontologies) for using an OPL. In particular, the eXtreme Design (XD) method [4] is quite suitable to be used with an OPL, since it is a content pattern-oriented method. Tasks such as "Match Competency Questions to Generic Use Case", "Select Content Patterns (CPs) to Reuse", and "Reuse and Integrate Selected CPs" could be easily adapted to consider patterns in an OPL. In fact, an OPL has great potential to improve XD. Take the experiments done by Blomqvist et al. [3], which evaluate pattern-based ontology design using XD. As pointed by these authors, the participants of the experiments may be faster in using patterns if they are more familiar with them. Moreover, the particular set of CPs could have an impact on the time spent in the ontology development. In the reported experiments, most of the patterns were quite general. Regarding this, Blomqvist et al. suggest that more specific patterns could also improve this aspect. Based on those perceptions, we argue that an OPL could be used to improve XD. Firstly, the patterns in an OPL are domain-related patterns, and thus more specific ones. Secondly, the OPL gives a context for the patterns, and guides the ontology engineer in traveling along them.

7 Final Considerations

Nowadays, ontology design patterns are recognized as a beneficial approach for ontology development [2, 3]. Particularly in the case of Domain-related Ontology Patterns (DROPs), these benefits can increase if we organize them as a pattern language, as it has been shown to be the case in Software Engineering. In this paper we introduced the notion of Ontology Pattern Language (OPL) as a network of interrelated DROPs with procedural rules prescribing the order in which they can be combined. OPLs can then be used to systematically solve ontology modeling problems in a given (core) domain. We also briefly present an OPL for the Software Process domain (SP-OPL), which illustrates the approach.

We shall consider that, as pointed out by Buschmann et al. [9], useful pattern languages must be sufficiently complete and mature. In particular, they must be complete regarding the coverage of the problem and solution spaces for their subjects, and must be mature regarding the quality and interconnection of their constituent patterns. Quality and maturity cannot be produced casually and hastily, but require great care and much time to age gracefully. OPLs are not an exception. Moreover, we claim that OPLs must present some characteristics generally pointed as being present in "beautiful ontologies", such as [22]: satisfy relevant requirements, have a good coverage of the targeted domain, be often easily applicable in some context, be structurally well designed (either formally or according to desirable patterns), and their domains should introduce constraints that lead to modeling solutions that are non-trivial.

Finally, pattern languages should evolve in response to various events and insights. As new experiences are gained developing ontologies with reuse, it is certainly desirable to integrate these new experiences and patterns into related existing pattern languages to keep them up to date. Consequently, all pattern languages, from the rawest to the most mature, should always be considered as a work in progress that is subject to continuous revision, enhancement, refinement, completion, and sometimes even complete rewriting [9].

Acknowledgments. This research is funded by the Brazilian Research Funding Agencies FAPES (Process Number 52272362/11) and CNPq (Process Number 483383/2010-4).

References

1. Poveda-Villalón, M., Suárez-Figueroa, M.C., Gómez-Pérez, A.: Reusing Ontology Design Patterns in a Context Ontology Network. In: Proc. of the 2nd International Workshop on Ontology Patterns – WOP 2010, Shangai, China (2010)
2. Gangemi, A., Presutti, V.: Ontology Design Patterns. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies, Second edition, pp. 221 – 243, Springer (2009)
3. Blomqvist, E., Gangemi, A., Presutti, V.: Experiments on Pattern-based Ontology Design. In: Proc. of the Fifth International Conference on Knowledge Capture – K-CAP 2009, pp. 41-48, California, USA (2009)

4. Presutti, V., Daga, E., Gangemi, A., Blomqvist, E.: eXtreme Design with Content Ontology Design Patterns. In: Proc. Workshop on Ontology Patterns, Washington D.C., USA (2009)
5. Clark, P., Thompson, J., Porter, B.: Knowledge patterns. In: Proc. of the 7th International Conference on Principles of Knowledge Representation and Reasoning – KR 2000), pp. 591–600, San Francisco, USA (2000)
6. Svatek, V.: Design Patterns for Semantic Web Ontologies: Motivation and Discussion. In: Proc. of the 7th Conference on Business Information Systems, Poznan, Poland (2004)
7. Alexander, C., Ishikawa, S., Silverstein, M.: A Pattern Language. Oxford University Press, New York (1977)
8. Deutsch, P.: Models and Patterns, In: Greenfield, J., Short, K., Cook, S., Kent, S. (eds.) Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools, Wiley Publishing Inc., Indianapolis (2004)
9. Buschmann, F., Henney, K., Schmidt, D.C.: Pattern-Oriented Software Architecture, Volume 5: On Patterns and Pattern Languages, John Wiley & Sons Ltd (2007)
10. Schmidt, D., Stal, M., Rohnert, H., Buschmann, F.: Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects. Wiley Publishing (2000)
11. Scherp, A., Saathoff, C., Franz, T., Staab, S.: Designing core ontologies. Applied Ontology, vol. 6, n. 3, pp. 177-221, IOS Press (2011)
12. Guarino, N.: Formal Ontology and Information Systems. In: Guarino, N. (ed.) Formal Ontology and Information Systems, pp. 3–15, IOS Press, Amsterdam (1998)
13. Borgo, S., Masolo, C.: Foundational Choices in DOLCE. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies, Second edition, pp. 361 – 381, Springer (2009)
14. Guizzardi, G.: Ontological Foundations for Structural Conceptual Models, Universal Press, The Netherlands (2005)
15. 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)
16. 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. of the XI Iberoamerican Workshop on Requirements Engineering and Software Environments – IDEAS 2008, pp. 244-251, Recife, Brazil (2008)
17. 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)
18. Barcellos, M. P., Falbo, R. A., Dal Moro, R.: A Well-founded Software Measurement Ontology. In: Proc. of the 6th International Conference on Formal Ontology in Information Systems – FOIS'2010, p. 213-216, Toronto, Canada (2010)
19. Gangemi, A.: Ontology Design Patterns for Semantic Web Content, In: Proc. of the 4th International Semantic Web Conference – ISWC'2005, p. 262 – 272, Galway, Ireland (2005)
20. Noppens, O., Liebig, T.: Ontology Patterns and Beyond - Towards a Universal Pattern Language. In: Proc. Workshop on Ontology Patterns, Washington D.C., USA (2009)
21. Falbo, R.A.: Experiences in Using a Method for Building Domain Ontologies. In: Proc. of International Workshop on Ontology in Action. Banff, Canada (2004)
22. d'Aquin, M., Gangemi, A.: Is there beauty in ontologies? Applied Ontology, vol. 6, n.3, p. 165–175 (2011)