

## Chapter 7

---

# Ontology Pattern Languages

### **Ricardo Falbo**

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

### **Monalessa Barcellos**

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

### **Fabiano Ruy**

Ontology and Conceptual Modeling Research Group (NEMO), Computer Science Department, Federal University of Espírito Santo, Vitória; Informatics Department, Federal Institute of Espírito Santo, Campus Serra, Serra, Brazil

### **Giancarlo Guizzardi**

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

### **Renata Guizzardi**

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

Ontology design patterns are a promising approach for Ontology Engineering. In this chapter, we introduce the notion of Ontology Pattern Language (OPL) as a way to organize domain-related ontology patterns. This chapter is organized as follows: Section 7.1 presents the motivation for organizing Domain-Related

Ontology Patterns (DROPs) as OPLs. Section 7.2 discusses what an OPL is, and how OPLs are represented, showing an example of an OPL for the software process domain, based on ISO Standards (ISP-OPL). Section 7.3 discusses how to build OPLs from core ontologies, taking ISP-OPL as an example. Section 7.4 discusses how an OPL can be used for building a domain ontology. An example applying ISP-OPL for building a domain ontology about the Stakeholder Requirements Definition Process is presented. Section 7.5 presents an overview of existing OPLs. Finally, Section 7.6 presents our final remarks.

## 7.1. Motivation

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. Using OPs is an emerging approach that favors the reuse of encoded experiences and good practices. Different kinds of OPs support ontology engineers on distinct phases of the ontology development process [7]. For instance, a *Domain-related Ontology Pattern* (DROP) is a kind of *Content Ontology Design Pattern* that captures a reusable fragment extracted from a reference domain ontology, which may assist in building the conceptual model of a new domain ontology. A *Logical Ontology Design Pattern*, or an *Ontology Coding Pattern*<sup>1</sup>, in turn, supports ontology implementation, aiming at solving problems related to reasoning or related to shortcomings in the expressivity of a specific logical formalism.

This chapter focuses on the conceptual modeling phase of Ontology Engineering, thus dealing with DROPs. To emphasize this point, it is important that the reader understands that in this chapter, in an analogy to Software Engineering (SE), we take an *analysis* point of view, in which we build a model of the world, without any implementation concerns (e.g. tractability or efficiency). Typically, this conceptual model will be later refined and, finally, become an *ontology schema* after some adaptations required by a specific ontology modeling language/formalism.

As pointed out by Alexander and colleagues in their pioneering work [1], each pattern can exist only to the extent that it is supported by other patterns. According to Schmidt et al. [25], in the community of Software Engineering patterns, the trend is towards defining pattern languages, rather than stand-alone patterns. The term “pattern language” in SE refers to a *network of interrelated patterns that defines a process for systematically solving coarse-grained software development problems* [5] [3]. This approach can also be taken into account in Ontology Engineering, giving rise to **Ontology Pattern Languages** (OPLs).

Although many DROPs in the literature refer to other patterns, and explore relations such as subsumption and composition, most of these references fail to give more complete guidelines on how the patterns can be combined to form solutions to larger problems. Context 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

---

<sup>1</sup>Also referred as *Idioms*.

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 approach is insufficient for complex real-world situations, since the features introduced by applying one DROP may be required by the next. A larger context is therefore needed to describe larger problems that can be solved by combining patterns, and to address issues that arise when DROPs are used in combination [6].

It is important to highlight that Ontology Engineering is a complex task. It is of course important to take into consideration the need for speedy and easy development, and this is indeed another reason for motivating **reuse** in this area. However, an ontology engineer should also be aware that when building a sound ontology, dealing with *some complexity* is unavoidable. This is because an ontology is expected to be comprehensive and coherent, not leaving behind important concepts and relations, but also not including unnecessary ones. Otherwise, one might be just overestimating the power of the so-called ontology, in terms of semantic expressivity and interoperability. Taking this into account, we believe that using OPs and OPLs is paramount to help dealing with the aforementioned complexity. And we hope to convince the reader of this important point, by presenting some definitions and examples in the next sections of this chapter.

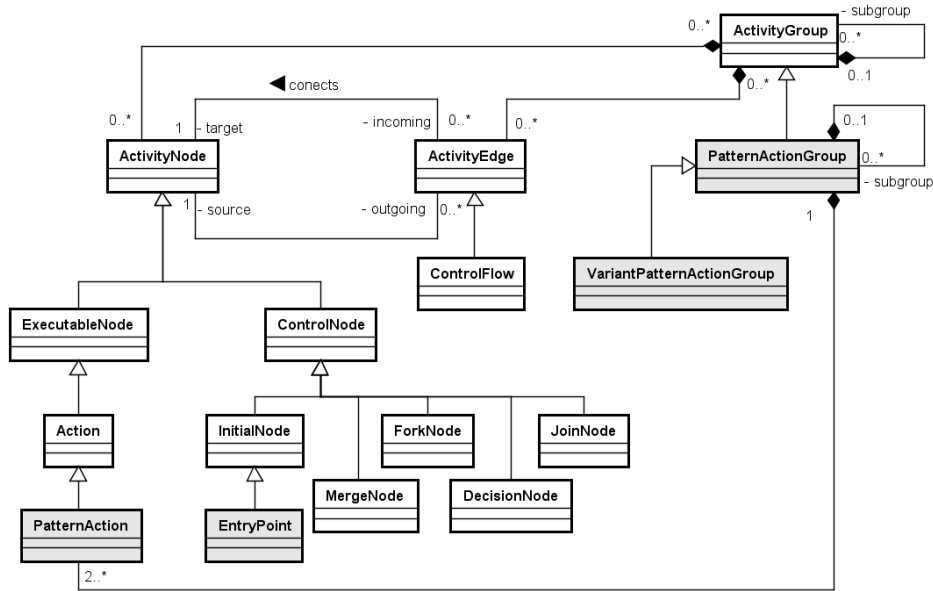
## 7.2. What an OPL is and how it is represented

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 interconnected 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 [6].

The notion of OPL provides a stronger sense of connection between DROPs, expressing different types of relationships among them [6]. For instance, to be applied, a pattern may require the previous application of other patterns (dependency); a larger pattern can be composed of smaller ones; or several patterns may solve the same problem in different ways (variant patterns). Those relationships impose constraints in the order in which patterns can be applied. Thus, an OPL provides explicit guidance on how to reuse and integrate related patterns into a concrete conceptual model of a new domain ontology. In this sense, an OPL is more than a catalogue of patterns. It includes, besides the patterns themselves, a rich description of their relationships, and a process guiding the order to apply them according to the problems to be modeled. OPLs encourage the application of one pattern at a time, following the order prescribed by paths chosen throughout the language. Consequently, by enabling the selective use of DROPs in a flexible way, an OPL releases the ontology engineer from the need to know upfront the whole set of concepts and relations addressed by the OPL. In this way, an OPL also aids managing complexity.

UML activity diagrams can be used for representing OPL process models, since they provide the main constructs for representing process models in general. Figure 7.1 shows an extension of a fragment of the UML meta-model for activity diagrams (version 2.5 [20]) that is adequate for representing OPLs. Modeling

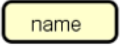

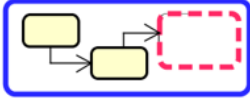


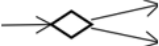
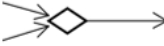


elements shown in grey are those extending the UML meta-model. Figure 7.2 shows the graphical notation for the modeling elements used to represent OPLs.



**Figure 7.1.** An extension of the UML meta-model for representing OPLs.

In this extended version of Activity Diagrams, a *Pattern Action* is an action in which a pattern is applied. Pattern Actions are represented as filled round-cornered rectangles. The name of the pattern is used to indicate the pattern to be applied. Pattern actions can be grouped in *Pattern Action Groups*. A Pattern Action Group is used to organize an OPL, aggregating pattern actions and other pattern action groups related to a more specific subject or sub-domain. Pattern Action Groups are represented as hollow round-cornered rectangles with wider borders. A special type of Pattern Action Group is the *Variant Pattern Action Group*, which aggregates variant pattern actions. Variant patterns groups aggregate patterns solving the same problem but in different and mutually exclusive ways. Thus, from a Variant Pattern Action Group only one of the patterns can be selected and applied. Variant Pattern Action Groups are represented as hollow round-cornered rectangles with wider dashed borders.

*Control Flows* represent the admissible sequences in which patterns in the language can be applied. A control flow is represented by an arrowed line connecting either: two pattern actions; a pattern action and a (variant) pattern action group; two (variant) pattern action groups. It is important to point out that, in an OPL, control flows indicate admissible paths along the OPL, i.e., they do not represent mandatory paths to be followed. The ontology engineer may decide to stop applying the patterns at any time. However, if he/she wants to proceed, the paths given by the control flows must be respected. When the application of a pattern requires applying the next one, the control flow linking the

Modeling Element	Notation
Pattern Action	
Control Flow	
Pattern Action Group	
Variant Pattern Action Group	
Entry Point	
Decision Node	
Merge Node	
Fork Node	
Join Node	

**Figure 7.2.** Graphical notation for representing OPLs.

two corresponding pattern actions is stereotyped as <<mandatory>>.

Besides the main modeling elements for representing pattern actions, groups of pattern actions and control flows between them, in order to allow representing more complex flows than sequential ones, control nodes are used. For representing OPLs, the following control nodes are used: entry point, decision and merge nodes, and fork and join nodes.

*Entry Points* indicate the patterns in the language that can be used without requiring the previous application of other patterns. There may be multiple entry points in an OPL, indicating different ways of using the OPL. When using an OPL, the ontology engineer must choose the entry point that better fits the requirements for the domain ontology being developed. Different from Initial Nodes in UML Activity Diagrams, in an OPL only one entry point is to be selected by the ontology engineer. Entry points (as UML initial nodes) are represented by solid circles.

As in the UML meta-model [20], a *Decision Node* is a control node that selects among alternative outgoing flows, while a *Merge Node* is a control node that

brings together multiple flows without synchronization. The notation for both Merge Nodes and Decision Nodes is a diamond-shaped symbol. The difference between them is that a Merge Node must have two or more incoming flows and a single outgoing flow, while a Decision Node must have a single incoming flow and multiple outgoing flows.

A *Fork Node* is a control node that splits a flow into multiple concurrent flows. A Fork Node has exactly one incoming control flow, though it may have multiple outgoing control flows. As discussed above for control flows, these outgoing flows may optionally be followed by the ontology engineer depending on the requirements for the ontology being developed. In other words, a fork node only indicates that the outgoing flows are admissible, but not that they are necessarily mandatory. If some of the outgoing flow is mandatory, the corresponding control flow must be stereotyped with <<mandatory>>. A *Join Node* is a control node that synchronizes multiple flows. A Join Node shall have exactly one outgoing control flow but may have multiple incoming flows. The notation for both Fork Nodes and Join Nodes is a thick line segment.

Figure 7.3 shows the process model of the ISO-based Software Process OPL (ISP-OPL) [22]. The purpose of ISP-OPL is to establish a common conceptualization about the software process domain, considering ISO Standards devoted to this subject, such as ISO/IEC 24744 [15], ISO/IEC 12207 [16] and ISO/IEC 15504 [14]. The main intended use for ISP-OPL is supporting ISO-Standard harmonization efforts. As Figure 7.3 shows, ISP-OPL is organized in three main groups of patterns: Work Units (WUs), Work Products (WPs) and Human Resources (HRs). ISP-OPL has three entry points. The ontology engineer should choose one of them, depending on the scope of the specific software process ontology being developed. The ontology engineer should choose EP1, when the requirements for the new ontology include the definition and planning of work units; he/she should choose EP2, if the scope of the ontology considers only the execution of work units (performed WUs); EP3 is to be chosen if the ontology engineer aims to model only the structure of work products.

Through entry point EP1, in order to model the structure of WUs, the ontology engineer needs to apply one (or both) of the following patterns: *WU Composition* and *WU Dependence*. These patterns are used to represent work units defined in an endeavor, without planning a time frame for them. The *WU Composition* pattern represents the mereological decomposition of work units, thus, specializing Work Unit into Process, Composite Task and Simple Task. The *WU Dependence* pattern deals with the dependence between work units. The *Project Process Definition* pattern captures the link between a Process and the Project for which it is defined. The *WU Scheduling* pattern is used to represent the time frame of a scheduled WU, defining its planned start and end dates.

After doing that, the ontology engineer can focus on modeling performed work units, i.e., work units already executed. Performed WUs, as past events, have actual start and end dates. The tracking of performed work units against defined work units is treated by the *Performed WU Tracking* pattern, which relates a Scheduled Work Unit to a Performed Work Unit caused by the former. The group encompassing the patterns *Performed WU Composition* and *Performed WU Dependence* is analogous to the group containing *WU Composition* and *WU*

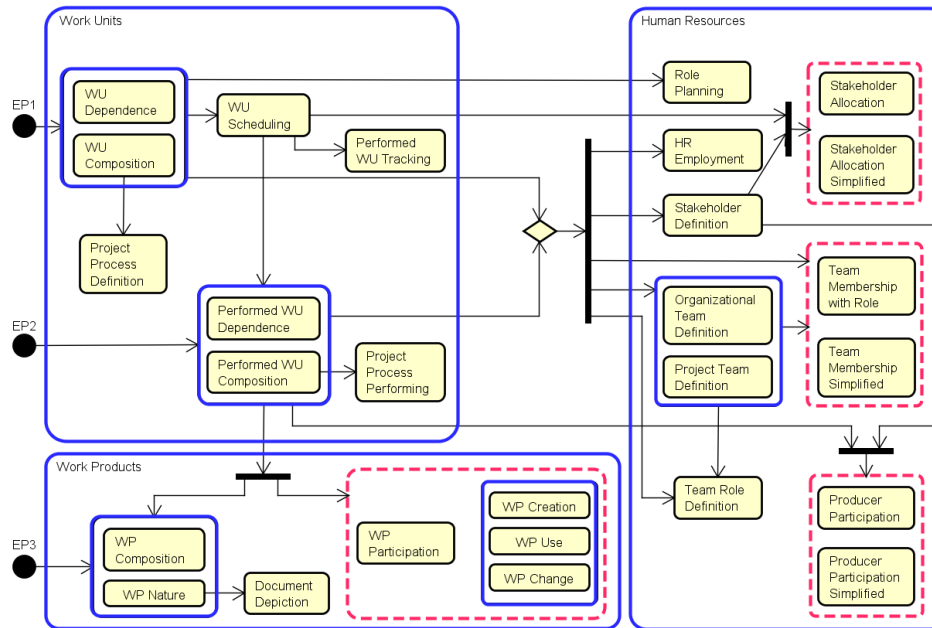


Figure 7.3. ISP-OPL process model.

*Dependence*. Additionally, the Project in the context of which a Process was performed can be modeled with the *Project Process Performing* pattern.

If the requirements for the ontology involve only performed work units, the entry point is EP2, allowing using, in this group, only the patterns *Performed WU Composition*, *Performed WU Dependence* and *Project Process Performing*.

After modeling aspects related to Work Units, the ontology engineer can address human resource related problems by applying the patterns of the Human Resource group. Some patterns of this group are adapted from patterns of the Enterprise OPL (E-OPL) [9] (see Section 7.5.1), such as *Human Resource Employment*, *Organizational Team Definition* and *Project Team Definition*.

The *Human Resource Employment* pattern establishes the employment relation between an Organization and a Person, which assumes the Human Resource role. The *Stakeholder Definition* pattern defines the concept of Stakeholder (someone involved in a Project). The *Organizational Team Definition* and *Project Team Definition* patterns are used to define organizational and project teams, respectively. The *Role Planning* pattern models the roles responsible for performing a defined work unit, while the *Team Role Definition* pattern can be applied to represent the roles a team can play.

In order to represent the membership relation between a team and its members (Persons), the ontology engineer can choose one of the alternative patterns *Team Membership Simplified* and *Team Membership with Role*. Then, one of these two alternative patterns can be used to represent the allocation of stakeholders to a scheduled work unit: *Stakeholder Allocation* and *Stakeholder Allocation Sim-*

*plified*. Finally, for dealing with the participation of stakeholders in performed work units, the ontology engineer can choose between the alternative patterns *Producer Participation* and *Producer Participation Simplified*.

The last group of patterns constituting ISP-OPL is the group related to Work Products (WPs). This group can be reached from the patterns related to Performed WU, but also through the entry point EP3. This group is to be chosen when the ontology engineer wants to represent only the structure of work products. The *WP Composition* pattern allows modeling the mereological structure of work products. *WP Nature* is related to types of work products (such as Document, Model and Information Item). Once the *WP Nature* pattern has been applied, the *Document Depiction* pattern can be used to model the fact that documents depict other work products.

Once the patterns for work unit execution have already been applied (either through EP1 or EP2), beyond the work product structure, the ontology engineer can also model work products handling. In this case, the *WP Participation* pattern sets the participation of work products in performed work units. In this pattern, the Work Product Participation is modeled as a concept with specializations for creation, change and usage participation. Alternatively, these three types of participations can be modeled only by means of a relation using the patterns *WP Creation*, *WP Change* and *WP Use*.

In addition to the process model, an OPL comprises the patterns themselves. The OPL specification includes the following items for each pattern:

- **Name:** the name of the pattern.
- **Intent:** describes the pattern purpose.
- **Rationale:** describes the rationale underlying the pattern.
- **Competency Questions:** describes the competency questions that the pattern aims to answer.
- **Conceptual Model:** depicts the conceptual model representing the pattern elements.
- **Axiomatization:** presents complementary formal axioms related to the diagrammatic form of the conceptual model. Those axioms typically capture constraints and other aspects of the pattern that cannot be directly represented by the diagrammatic form of the conceptual model.
- **Complementary Patterns:** list other ontology patterns that are related to the pattern being presented, but that are not part of the OPL to which the pattern being described belongs.

As an example, Table 7.1 shows the specification of the *Performed Work Unit Composition* pattern [22].

It is important to highlight that the term “pattern language” was borrowed from Software Engineering, where patterns have been studied and applied for a long time. 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. Moreover, although an



Performed WU Composition
<p><b>Name:</b> Performed Work Unit Composition (PWUC)</p> <p><b>Intent:</b> To represent the composition of <i>performed work units</i> in terms of other <i>performed work units</i>.</p> <p><b>Rationale:</b> <i>Performed Work Units</i> can be composed of other <i>performed work units</i>. From a mereological point of view, a <i>performed work unit</i> is simple, or composed of two or more parts. At the basic level, there are <i>Performed Simple Tasks</i> that can compose other <i>performed work units</i>, but which are not decomposable. <i>Performed Composite Tasks</i>, in turn, are composed of other performed tasks (<i>composite</i> or <i>simple performed tasks</i>). At the higher level, <i>Performed Processes</i> are also composed of performed tasks, but do not compose any other <i>performed work unit</i>.</p> <p><b>Competency Questions:</b></p> <ul style="list-style-type: none"> <li>Concerning their mereological structure, what are the possible types of <i>performed work units</i>?</li> <li>How is a <i>performed work unit</i> composed of other <i>performed work units</i>?</li> </ul> <p><b>Conceptual Model</b></p> <pre> classDiagram     class PerformedWorkUnit["&lt;&lt;event&gt;&gt; Performed Work Unit"] {         startDate         endDate     }     class PerformedCompositeWorkUnit["&lt;&lt;event&gt;&gt; Performed Composite Work Unit"] {     }     class PerformedProcess["&lt;&lt;event&gt;&gt; Performed Process"] {     }     class PerformedCompositeTask["&lt;&lt;event&gt;&gt; Performed Composite Task"] {     }     class PerformedSimpleTask["&lt;&lt;event&gt;&gt; Performed Simple Task"] {     }     PerformedCompositeWorkUnit &lt; -- PerformedProcess     PerformedCompositeWorkUnit &lt; -- PerformedCompositeTask     PerformedCompositeWorkUnit &lt; -- PerformedSimpleTask     PerformedCompositeWorkUnit &lt; -- PerformedWorkUnit     PerformedProcess "0..1" -- "2..*" PerformedWorkUnit     PerformedCompositeTask "2..*" -- "2..*" PerformedWorkUnit     PerformedSimpleTask "2..*" -- "2..*" PerformedWorkUnit     PerformedCompositeWorkUnit "1" -- "2..*" PerformedWorkUnit : mereological structure {disjoint, complete}   </pre>
<p><b>Axiomatization (partial)</b></p> <p><b>A1:</b> <math>\forall w: \neg \text{partOf}(w,w)</math> No individual (and, hence, no <i>Performed Work Unit</i>) can be part of itself.</p> <p><b>A2:</b> <math>\forall p: \text{PerformedProcess}(p) \rightarrow \neg \exists w \text{ PerformedWorkUnit}(w) \wedge \text{partOf}(p,w)</math> A <i>Performed Process</i> cannot be part of any <i>Performed Work Unit</i>.</p> <p><b>A3:</b> <math>\forall w1,w2: (\text{Event}(w1) \wedge \text{Event}(w2) \wedge \text{partOf}(w2,w1)) \rightarrow (\text{startDate}(w2) \geq \text{startDate}(w1)) \wedge (\text{endDate}(w2) \leq \text{endDate}(w1))</math> A <i>Performed Work Unit</i> that is part of another <i>Performed Work Unit</i> should occur within the time interval of the latter.</p>

**Table 7.1.** The Specification of the Performed WU Composition Pattern.

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. Instead, 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 [6].

### 7.3. Building OPLs from Core Ontologies

For building an OPL for some domain, we need to have a set of interconnected patterns for this domain. A good option to get these patterns is to extract them from ontologies developed for this domain, in particular core ontologies. A core ontology provides a precise definition of the structural knowledge in a specific field that spans across several application domains in that field. Core ontologies are conceived mainly aiming at reuse, and thus, a pattern-oriented design approach is appropriate for organizing them. By following a pattern-oriented design approach, core ontologies become more modular and extensible [24]. Moreover, by providing a network of patterns and rules on how they can be combined, an OPL improves the potential for reuse of a core ontology, by enabling the selective use of parts of the core ontology in a flexible way. This is very important due to pragmatic reasons, since ontology engineers developing specific ontologies for that domain might want to focus on selected aspects of the domain, disregarding others.

With a core ontology in hands, DROPs can be extracted from it through a fragmentation process. To illustrate the extraction of DROPs from core ontologies, consider the case of ISP-OPL. The patterns of ISP-OPL were extracted from a software process core ontology resulting from the ontological analysis of the ISO/IEC 24744 metamodel [15] (Software Engineering Metamodel for Development Methodologies - SEMDM) performed in [21], plus the application of some patterns of the Enterprise OPL (E-OPL) [9]. Figure 7.4 shows the conceptual model of the part of this core ontology dealing with Work Units. This conceptual model is fragmented in the patterns discussed in Section 7.2.

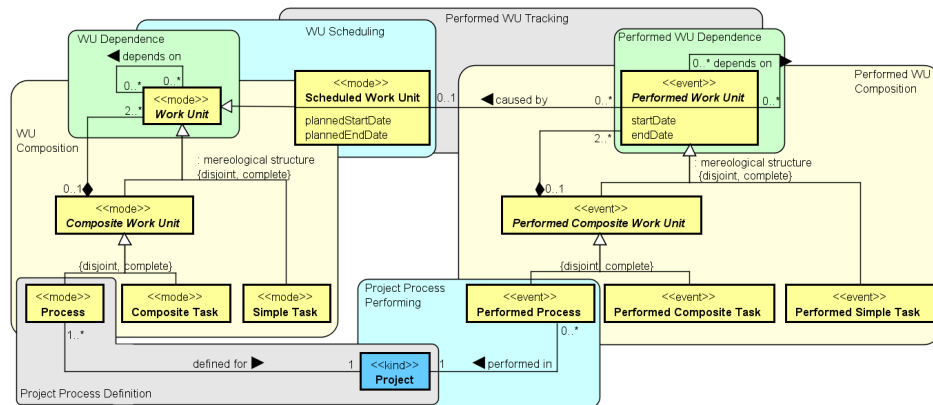


Figure 7.4. Patterns of the Work Unit Group.

DROP complexity can vary greatly depending on the domain fragment being represented. Sometimes a DROP contains only two related concepts. This is the case of the patterns *WU Dependence*, *WU Scheduling*, *Project Process Definition*, *Performed WU Tracking*, *Performed WU Dependence* and *Project Process Performing*. This fine-grained fragmentation is useful to prevent the ontology engineer from discarding parts of a pattern. In other situations, a DROP can

contain a complex combination of concepts and relations, such as in the case of *WU Composition* and *Performed WU Composition*. The *WU Composition* pattern represents the mereological decomposition of Composite Work Units into other Work Units. Composite Work Units are composed of at least two Work Units, and, according to the ISO Standards, they can be of two types: Processes or Composite Tasks. Work Units that are not composed by other Work Units are said to be Simple Tasks.

An important aspect to highlight is that, as pointed out by Scherp et al. [24], a core ontology should be precise. This is achieved by grounding the core ontology in a foundational ontology. Concepts and relations defined in a core ontology should be aligned to the basic categories of a foundational ontology [24]. By doing that, core ontologies incorporate a solid and semantically precise basis.

The ISO-based software process core ontology illustrated here is based on the Unified Foundational Ontology (UFO) [12] [13]. In fact, as previously mentioned, it is the outcome of an ontological analysis of the ISO/IEC 24744 meta-model [15] in terms of UFO. As a consequence, the resulting patterns extracted from this well-founded ontology are themselves well-founded.

#### 7.4. Using OPLs for building Domain Ontologies

In this section, we discuss how to build a domain ontology using an OPL, illustrating this by an example applying ISP-OPL for building a reference ontology for the Requirements Engineering (RE) process.

Before using an OPL, the ontology engineer needs to study the OPL process model and its patterns, so that he/she can better decide which entry point is the most suitable for the particular project at hand. Then, he/she has to apply the patterns following the admissible paths through the OPL until there are no more applicable patterns.

The RE Process Ontology presented here was derived from ISP-OPL according to the information extracted from selected ISO SC7 standards [22], namely: ISO/IEC 15288:2008 - System life cycle processes [17], ISO/IEC 12207:2008 - Software life cycle processes [16], and ISO/IEC/IEEE 29148:2011 - Requirements Engineering [18]. These standards define three requirements-related processes: *Stakeholder Requirements Definition*, *System Requirements Analysis*, and *Software Requirements Analysis*. In this chapter we present only the sub-ontology addressing the first process: *Stakeholder Requirements Definition* (Section 6.4.1 in ISO 12207 and ISO 15288, and Section 6.2 in ISO 29148).

We are interested in describing the execution of requirements processes, including the participations of human resources and work products, as it can be typically found in the case of organizations adopting these standards in their projects. Thus, we started using ISP-OPL by the entry point EP2. Figure 7.5 shows the chosen patterns and paths of the ISP-OPL process that we have followed for developing this ontology.

Figure 7.6 presents the *Stakeholder Requirements Definition Process* sub-ontology. On the top, the concepts with colored background are the ones defined as part of the ISP-OPL patterns. On the bottom, the concepts with white background are the specific ones from the RE Process Ontology. Relations in the RE

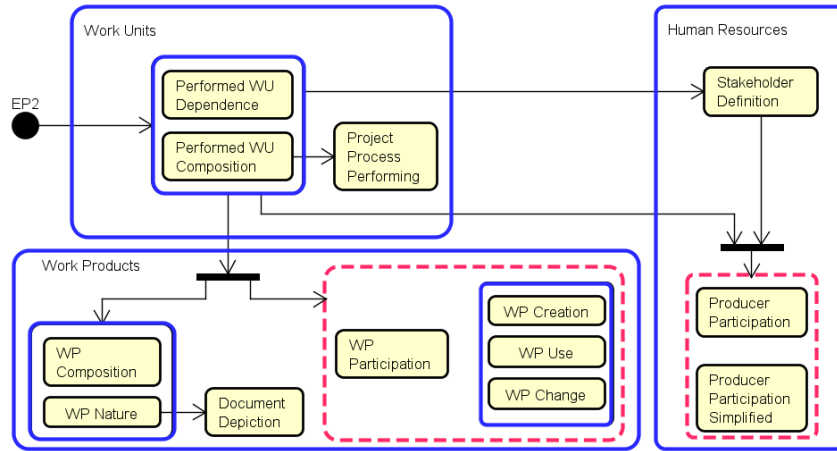


Figure 7.5. ISP-OPL Patterns used and Paths followed (adapted from [22]).

Process Ontology are specializations of the homonymous relations in the OPL. Cardinalities are omitted for the sake of legibility.

The *Stakeholder Requirements Definition* process is decomposed into activities, which, in turn, are decomposed into tasks. Thus, we started with the *Performed WU Composition* pattern, modeling the decomposition of performed work units. The *Stakeholder Requirements Definition Process* is a subtype of **Performed Process**. This specialized process is composed of five work units: *Stakeholder Identification*, *Requirements Identification*, *Requirements Evaluation*, *Requirements Agreement* and *Requirements Recording*. The first and fourth work units are **Performed Simple Tasks**, and the others are **Performed Composite Tasks**, which are themselves decomposed into simple tasks as shown in Figure 7.6.

Another pattern considered useful here is *Performed WU Dependence*, which defines dependencies between work units. Although the selected standards do not explicitly set dependencies between tasks, some of them can easily be inferred from the nature of work units and work products handled, as well as by considering the RE literature. Accordingly, we applied the *Performed WU Dependence* pattern to establish dependencies between the work units as shown in Figure 7.6. Still regarding work units, the last pattern applied was *Project Process Performing*, establishing the connection between the **Performed Process** and the **Project** wherein it is performed.

Once work units have been addressed, we could address human resources related problems. Due to the general nature of the standards, few information is given about human resources participating in work units. Thus, we have modeled only the stakeholder definition and its relation to work units. The first pattern applied was *Stakeholder Definition*, in order to establish the types of stakeholders to be considered. We considered only two types of stakeholders: *System Analyst*, and *Requirements Stakeholder*. Both of which are types of **Person Stakeholder** involved in a **Project**. Aiming at representing the participation

of stakeholders in work units, the *Producer Participation Simplified* pattern was used, specializing **Stakeholders** as **Producers** who participate in **Performed Work Units**.

The other path of ISP-OPL we followed is through the use of the work products patterns. Once there are different types of work products, it is useful to distinguish between them by applying the *WP Nature* pattern. Two subtypes of **Work Product** were considered: **Information Item** and **Document**. In the context of the *Stakeholder Requirements Definition Process*, we identified the following subtypes of **Information Item**: *Requirement* (specialized into *Stakeholder Requirement*), *Stakeholder List*, *Stakeholder Agreement*, and *Traceability Record*. Moreover, two sub-types of **Document** were considered: *Requirements Evaluation Doc*, and *Stakeholder Requirements Specification*. The *Stakeholder Requirements Specification* is the main result of this process and aggregates the *Stakeholder List* and the set of *Stakeholder Requirements*. Thus, using the *WP Composition* pattern, we establish *Stakeholder Requirements Specification* as a **Composite Work Product**, composed of *Stakeholder Requirements* and *Stakeholder List* (both of which are types of **Simple Work Product**). Additionally, by applying the *Document Depiction* pattern, we represented a *Stakeholder Requirements Specification*, which (as a document) represents the *Stakeholder Requirements*.

Finally, by using the patterns *WP Creation*, *WP Use* and *WP Change*, we established the relations of creation, usage and change between the work units of the *Stakeholder Requirements Definition Process* and their corresponding work products.

It is important to point out that reuse is not limited to the conceptual models, as discussed above. As Table 7.1 shows, the patterns' specification includes also competency questions (CQs) and axioms. Typically, these elements are also reused when a pattern in the OPL is selected. Once a DROP is chosen, its concepts and relations become part of the domain ontology (as Figure 7.6 shows), where they can be further extended. For CQs, a very similar approach holds: once a DROP is chosen, its CQs can be extended for the domain ontology. For example, the *Performed WU Composition* pattern (see Table 7.1) has the following CQs: (i) *Concerning their mereological structure, what are the possible types of performed work units?*; and (ii) *How is a performed work unit composed of other performed work units?* When this pattern was applied to the Software RE process, the following (extended) specific CQs were created: (i) *What are the possible types of performed work units in the RE process?*; and (ii) *How is the RE process decomposed?* This reuse helps with the definition of CQs, improving the productivity of the Ontology Engineering process [23]. The same applies to the case of reusing general axioms.

## 7.5. Existing OPLs

The use of OPLs is a recent initiative. There are still only few works defining OPLs, among them the following: Software Process OPL (SP-OPL) [6], ISO-based Software Process OPL (ISP-OPL) [22], Enterprise OPL (E-OPL) [9], Measurement OPL (M-OPL) [2], and Service OPL (S-OPL) [8]. ISP-OPL was pre-

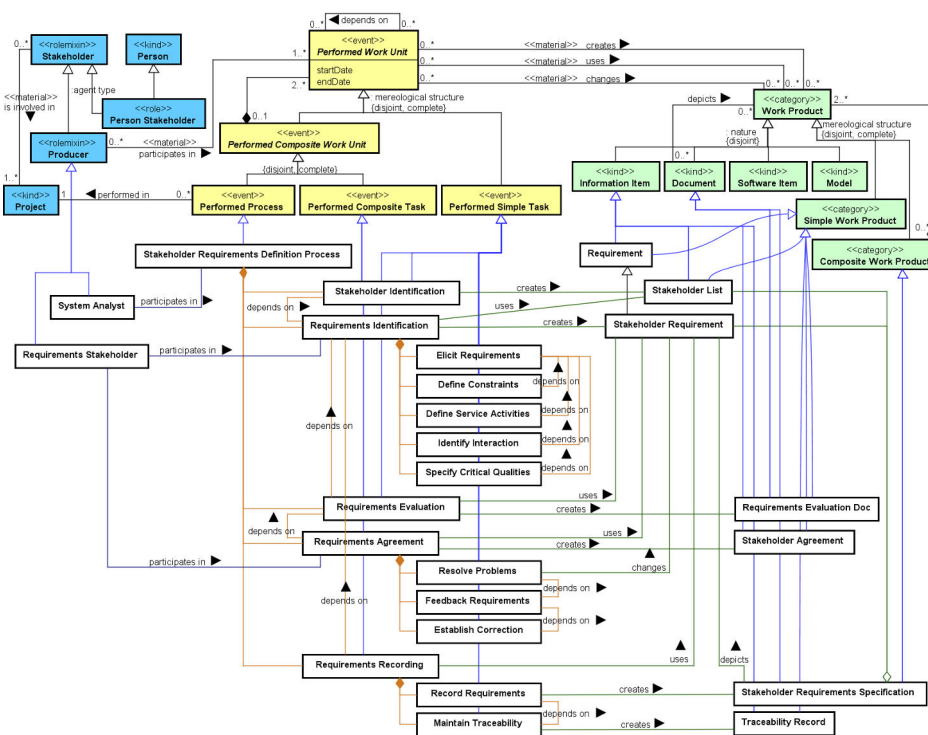


Figure 7.6. The Stakeholder Requirements Definition Process sub-ontology [22].

sented in the previous section. SP-OPL is related to the same domain (software processes), but it is more general than ISP-OPL since it is not devoted to ISO Standards. Thus, in this section, we briefly present the other three OPLs. Further information on these OPLs can be found at <http://nemo.inf.ufes.br/OPL>.

### 7.5.1. Enterprise OPL

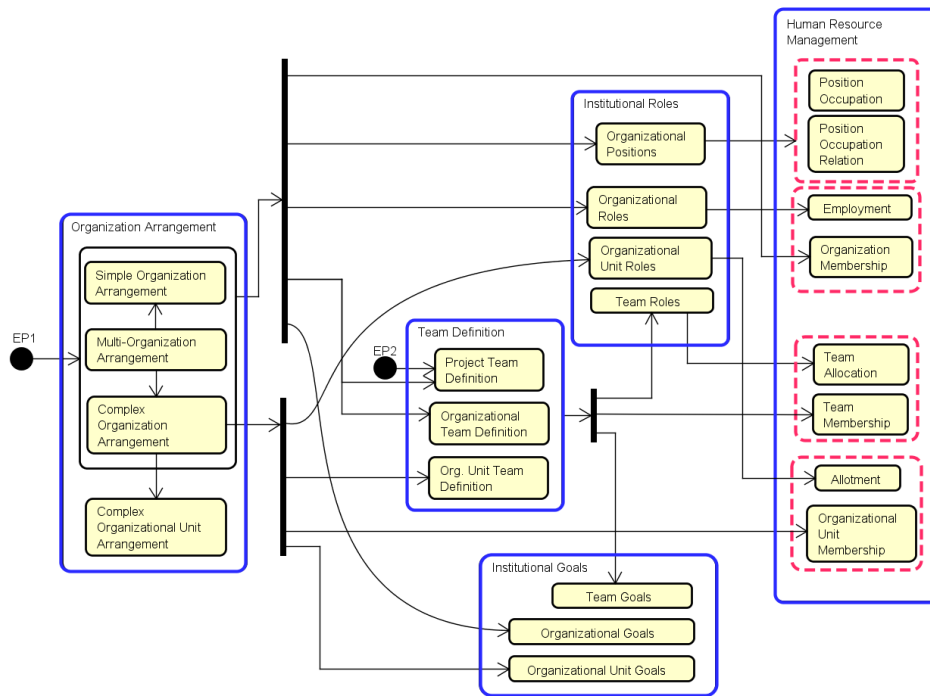
The Enterprise Ontology Pattern Language (E-OPL) [9] aims at providing patterns for enterprise ontology modeling. It is composed of 22 patterns addressing five aspects common to several enterprises:

- **Organization Arrangement** (4 patterns): includes patterns related to how multi-organizations are organized in terms of other organizations (*Multi-Organization Arrangement* pattern), how a complex organization is structured in terms of organizational units (*Complex Organization Arrangement* pattern), as well as how complex organizational units are structured in terms of other organizational units (*Complex Organizational Unit Arrangement* pattern). Finally, if organizations to be modeled are simple (not composed of other organizations or organizational units) or addressing organizations' structure is out of the scope of the domain ontology being developed, then there is the *Simple Organization Arrangement* pattern;

- **Team Definition** (3 patterns): deals with defining teams for projects (*Project Team Definition* pattern), organizations (*Organizational Team Definition* pattern) and organizational units (*Organizational Unit Team Definition* pattern);
- **Institutional Roles** (4 patterns): addresses the representation of roles and positions to be played by enterprise employees. This group contains the following patterns: the *Organizational Positions* pattern deals with positions defined in an organization; the *Organizational Roles* pattern addresses roles defined in an organization; the *Organizational Unit Roles* and *Team Roles* patterns concern informal roles defined by an organizational unit or a team, respectively;
- **Institutional Goals** (3 patterns): deals with institutional agents' goals, and there are three patterns available: *Organizational Goals*, *Organizational Unit Goals*, and *Team Goals* patterns;
- **Human Resource Management** (8 patterns): treats the following human resource relations in an enterprise: employment, allotment to an organizational unit, team allocation, and position occupation. For each one of these relations, two variant patterns are defined, one explicitly including a concept reifying the material relation (a relator), and another disregarding the relator and considering only the material relation.

Figure 7.7 shows the E-OPL process model. As this figure shows, E-OPL has two entry points. The ontology engineer should choose one of them, depending on the scope of the specific enterprise ontology being developed. When the requirements for the new enterprise ontology being developed include only problems related to the definition of project teams, the starting point is EP2. Otherwise, the starting point is EP1. In this case (EP1), first the ontology engineer should address problems related to how an organization is structured. One of the three following patterns has to be selected: *Simple Organization Arrangement* pattern, *Complex Organizational Unit Arrangement* pattern or *Multi-Organization Arrangement* pattern.

*Simple Organization Arrangement* pattern should be selected as the first pattern if the ontology engineer needs to represent only very simple standalone organizations, which are neither composed of other organizations nor composed of organizational units. The *Complex Organization Arrangement* pattern should be selected as the first pattern if the ontology engineer needs to represent only complex standalone organizations, which are not composed of other organizations, but which are composed of organizational units. When the *Complex Organization Arrangement* pattern is selected, the *Complex Organizational Unit Arrangement* pattern can be used in the sequel, if there is a need to represent complex organizational units, which are composed of other organizational units. Finally, the *Multi-Organization Arrangement* pattern should be selected as the first pattern if the ontology engineer needs to represent organizations that are composed of other organizations. When the *Multi-Organization Arrangement* pattern is used, the ontology engineer can also use the *Simple Organization Arrangement* and *Complex Organization Arrangement* patterns to address the organizational structure of the standalone organizations that compose a multi-organization.



**Figure 7.7.** E-OPL Process Model (adapted from [9]).

Once problems related to the organization arrangement are addressed, the ontology engineer can treat problems related to the definition of organizational teams, goals and roles, and some problems related to human resource management.

Concerning team definition, three types of teams are considered: organizational teams, organizational unit teams and project teams. The *Project Team Definition* pattern deals with teams that are defined with the specific purpose of performing a project. For this reason, this pattern does not require the problems related to organizational arrangement to be addressed prior to its use. For this reason, it takes EP2 to be its entry-point in E-OPL. The *Organizational Team Definition* and the *Organizational Unit Team Definition* patterns deal respectively with organizational and organizational unit teams.

Regarding goals, in a general view, in E-OPL, Institutional Agents (a generalization for Organizations, Organizational Units and Teams) may define Institutional Goals, and three patterns are available: *Organizational Goals*, *Organizational Unit Goals*, and *Team Goals*.

Concerning roles, Institutional Agents (Organizations, Organizational Units and Teams) may define Institutional Roles. Like the TOVE Ontology [10], E-OPL considers two main types of Institutional Roles: Positions and Human Resource Roles. A Position represents some formal position in the organization, such as “president”, “sales manager”, etc. A Human Resource Role defines a prototypical



function of a person in the scope of an Institutional Agent, such as “engineer” or “system analyst”. Moreover, E-OPL distinguishes between formal and informal roles. Formal Human Resource Roles are those recognized by the whole organization and its environment (partners and society in general). Informal Human Resource Roles are those recognized only in the scope of the corresponding institutional agent. Team Roles and Organizational Unit Roles are types of informal roles, recognized, respectively, by a Team and by an Organizational Unit. Organizational Roles can be formal or informal. Formal Organizational Roles are those considered when employments are created. Each employment is made for a specific formal role. On the other hand, a particular person, in the same employment, can assume several informal roles.

In order to deal with institutional roles, four patterns were defined. The *Organizational Positions* pattern deals with positions defined in an organization. The *Organizational Roles* pattern addresses both formal and informal roles defined in an organization. The *Organizational Unit Roles* pattern and *Team Roles* pattern address informal roles defined by an organizational unit or a team, respectively.

Finally, problems related to human resource management can be addressed. In E-OPL there are eight patterns treating four material relations that involve human resources, namely: employment, allotment, team allocation, and occupation.

According to UFO, material relations are relations that have material structure on their own. The relata of a material relation are mediated by individuals that are called relators. Relators are complex objectified relational properties [12] [11]. From a pragmatic point of view, depending on the requirements of the enterprise ontology being developed, the ontology engineer may be interested in showing only the material relation instead of showing the whole model, including also the relator and the mediation relations. Thus, each one of these material relations is addressed by two patterns: one explicitly including a concept representing the relator (*Position Occupation*, *Employment*, *Team Allocation* and *Allotment* patterns), and another disregarding the relator and considering only the material relation (*Position Occupation Relation*, *Organization Membership*, *Team Membership* and *Organizational Unit Membership* patterns, respectively). The patterns considering the relators are more complete. They allow capturing information about the relator (for instance, start date and end date of a position occupation, employment, team allocation, or allotment). However, if for a given context, such information is considered irrelevant, then the ontology engineer can choose simpler patterns, which disregard the respective relators, capturing only the fact that human resources are members of organizations, organizational units, and teams, or that they are capable holding positions.

For details regarding E-OPL and its patterns, see [9].

### 7.5.2. Service OPL

The Service Ontology Pattern Language (S-OPL) [8] aims at providing a network of interconnected ontology modeling patterns covering the core conceptualization of services. S-OPL builds on UFO-S, a commitment-based core ontology for services [19]. The S-OPL patterns support modeling types of customers and

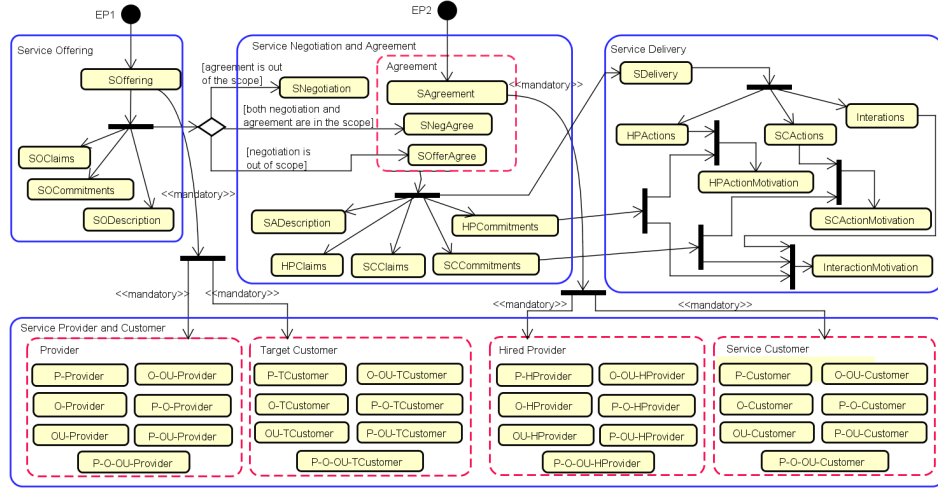
providers, as well as the main service life-cycle phases, namely: service offering, service negotiation/agreement, and service delivery. S-OPL is composed of 48 patterns, grouped in 4 groups, namely:

- **Service Offering** (4 patterns): includes patterns to deal with service offering (*SOffering* pattern), description of a service offering (*SODescription* pattern) as well as commitments and claims of the provider (*SOCCommitments* and *SOClaims* patterns);
- **Service Negotiation and Agreement** (9 patterns): includes patterns that deal with service negotiation and agreement (*SNegotiation*, *SAGreement*, *SNegAgree* and *SOfferAgree* patterns), description of a service agreement (*SADescription* pattern) as well as commitments and claims of both the hired provider and the service customer (*HPCommitments*, *HPClaims*, *SCCommitments* and *SCClaims* patterns);
- **Service Delivery** (7 patterns): includes patterns to deal with the service delivery (*SDelivery* pattern), the actions performed by both the hired provider and the service customer to deliver the service (*HPActions*, *SCActions* and *Interactions* patterns), as well as the motivation to perform these actions (*HPActionMotivation*, *SCActionMotivation* and *InteractionMotivation* patterns);
- **Service Provider and Customer** (28 patterns): deals with defining the types of agents (Person, Organization or Organizational Unit) that can act as Service Provider, Target Customer, Hired Provider and Service Customer.

Figure 7.8 shows the S-OPL process model. As this figure shows, S-OPL has two entry points (EP1 and EP2). The ontology engineer should choose one of them depending on the scope of the specific domain service ontology being developed. When the requirements for the ontology include describing the service offering, then the starting point is EP1. Otherwise, the starting point is EP2.

In the case EP1 is chosen, the ontology engineer should use first the *SOffering* pattern for modeling the service offering itself. Next, he/she must follow the mandatory path: the one that leads to the *Service Provider and Customer* group, which addresses the issue of modeling which types of providers and target customers are involved in the offering.

Providers and target customers can be people, organizations or organizational units. Therefore, the ontology engineer must select one of the patterns of the *Provider* sub-group, and one of the patterns of the *Target Customer* sub-group. These variant patterns are important, because depending on the nature of the service being modeled, only certain types of customers and providers are admissible. For instance, the passport issuing service is offered only to people. The car rental service, in turn, is offered to people, organizational units, and organizations. Thus, each pattern in the *Service Provider and Customer* group offers a different option for the ontology engineer to precisely decide what kinds of entities can play the roles of provider and customer in the service domain being modeled. For instance, in the *Target Customer* sub-group of variant patterns, *P-TCustomer* should be used when only people can play this role. *O-TCustomer* should be used



**Figure 7.8.** S-OPL Process Model (adapted from [8]).

when only organizations can play this role. *OU-TCustomer* should be used when exclusively organizational units can play this role. *O-OU-TCustomer* should be used when both organizations and organizational units can play this role. *P-O-TCustomer* should be used when both persons and organizations can play this role. *P-OU-TCustomer* should be used when both persons and organizational units can play this role. Finally, *P-O-OU-TCustomer* should be used when any of these kinds of entities can play this role.

Besides mandatorily modeling the types of providers and target customers, the ontology engineer can follow the several paths coming out of the fork node. Thus, he/she can use the patterns *SOClaims* and *SOCommitments*, in the cases in which he/she is interested in modeling offering claims and commitments, respectively. In addition, the ontology engineer can also choose the *SODescription* pattern, in case he/she is interested in describing the offering by means of a service offering description.

Once the service offering is modeled, the ontology engineer is able to address problems related to service negotiation and agreement. However, service offering may be out of the scope of the ontology. In this case, EP2 should be the entry point in the S-OPL process.

If the ontology engineer has already modeled the service offering, he/she must decide first whether he/she needs to represent service negotiation and/or service agreement. If he/she wants to model only the service negotiation, without modeling the agreement that could result from it (i.e., the agreement is out of scope), he/she should use the *SNegotiation* pattern. If he/she needs to model both the negotiation and the agreement, then he/she should use the *SNegAgree* pattern. Finally, if negotiation is out of the ontology scope, then he/she should use the *SOfferAgree* pattern, which represents an agreement in conformance to an offering.

If EP2 is the entry point in the process, the first pattern to be used is *SAgree-*

*ment*. In the sequel, the ontology engineer must select one of the patterns of the *Hired Provider* sub-group and one of the patterns of the *Service Customer* sub-group, in order to model the possible types of hired provider and service customer, respectively. The patterns in the *Hired Provider* and *Service Provider* sub-groups are analogous to the ones in the *Provider* and *Target Customer* sub-groups respectively. Note that defining the types of hired providers and service customers is necessary only if the chosen entry point is EP2, since in cases in which the entry point in the process is EP1, the types of providers and target customers would already have been modeled at that point.

Once the agreement is modeled, the following patterns can be optionally used: *HPCommitments* and *HPClaims*, depending whether the ontology engineer is interested in modeling the hired provider commitments and claims, respectively; *SCCommitments* and *SCClaims*, depending on whether he/she is interested in modeling service customer commitments and claims, respectively; and *SADescription*, in case he/she is interested in describing the service agreement by means of a description.

After modeling the agreement, the ontology engineer can model the service delivery. In this group, the first pattern to be used is *SDelivery*. In the sequel, if he/she wants to model the actions involved in a delivery, the following patterns must be applied: *HPActions*, for modeling actions performed by the hired provider; *SCActions*, for modeling actions performed by the service customer; and *Interactions*, for modeling actions performed by both the hired provider and service customer, in conjunction. After that, he/she can model the relationships between the actions and the commitments that motivate them. This can be done by using the following patterns: *HPActionMotivation*, *SCActionMotivation* and *InteractionMotivation*. Since these patterns establish links between commitments and actions, they require the patterns related to the former to be used prior to the patterns related to the latter.

For details regarding S-OPL and its patterns, see [8].

### 7.5.3. Measurement OPL

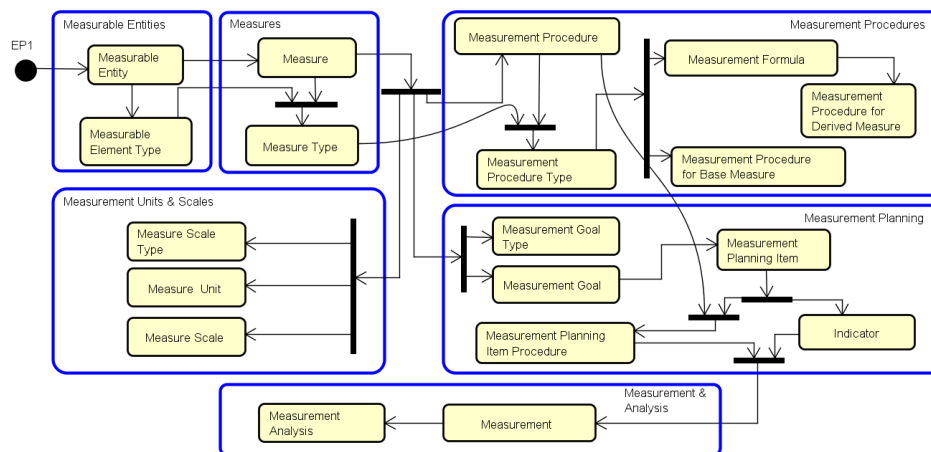
The Measurement Ontology Pattern Language (M-OPL) [2] aims at providing a network of ontology modeling patterns addressing the core conceptualization about measurement. It is composed of 19 patterns, organized in 6 groups, namely:

- **Measurable Entities** (2 patterns): includes patterns related to the entities and their properties that can be measured. There are two patterns: *Measurable Entity*, which deals with entities that can be measured, their types and their measurable properties; and *Measurable Element Type*, which is related to the types of measurable properties;
- **Measures** (2 patterns): includes patterns to deal with defining measures (*Measure*) and classifying them according to their dependence on others measures (*Measure Type*);
- **Measurement Units & Scales** (3 patterns): concerns the scales associated to measures and the measurement units used to partition scales. This group includes three patterns: *Measure Scale Type*, which addresses types

of scales; *Measure Unit*, dealing with measurement units; and *Measure Scale*, which is related to measures' scales;

- **Measurement Procedures** (5 patterns): addresses procedures needed to collect data for measures. Includes five patterns: *Measurement Procedure* deals with the data collection procedure; *Measurement Procedure Type* concerns types of measurement procedures; *Measurement Procedure for Base Measure*, which addresses measurement procedures established to base measures; *Measurement Procedure for Derived Measure*, which regards measurement procedures established to derived measures; and *Measurement Formula*, which deals with formulas used to obtain data for derived measures;
- **Measurement Planning** (5 patterns): treats the goals that drive measurement as well as the measures used to verify the achievement of goals. This group includes five patterns, namely: *Measurement Goal*, which deals with measurement goals and information needs to be satisfied by measurement; *Measurement Goal Type*, which is related to measurement goal decomposition; *Measurement Planning Item*, which addresses the basic planning for measurement, including the measurement goal to be monitored, the information need to be met and the measure to be used; *Measurement Planning Item Procedure*, which models measurement planning items adding the measurement procedure to be used; and *Indicator*, which is about identifying measures that act as indicators to goal monitoring;
- **Measurement & Analysis** (2 patterns): includes patterns related to the topics of data collection (*Measurement*) and data analysis (*Measurement Analysis*).

Figure 7.9 shows the M-OPL process model. As this figure shows, M-OPL has only one entry point: EP1.



**Figure 7.9.** M-OPL Process Model (adapted from [2]).

The first pattern to be used by the ontology engineer is *Measurable Entity*, which models entities that can be measured, their types and properties (i.e., measurable elements). After using this pattern, two patterns are applicable: *Measurable Element Type*, to distinguish direct from indirect measurable elements, and *Measures*, to model measures and relate them to measurable elements and measurable entity types. After the *Measure* pattern, it is possible to apply the *Measure Type* pattern, in order to model the distinction between base and derived measures. However, as shown in Figure 7.9, *Measure Type* can only be used after applying the *Measurable Element Type*, since the measure type is defined based on the type of the measurable element qualified by the measure.

From the *Measure* pattern, there are still three possible paths to follow. The first one goes to the *Measurement Units & Scales* group. In this group, if types of scales (e.g., interval, ordinal and rational) are relevant to the specific measurement domain ontology being developed, the ontology engineer must use the *Measure Scale Type* pattern. If the ontology engineer wants to address measurement units, he/she should use the *Measure Unit* pattern. If measures' scales are relevant to the ontology being developed, the *Measure Scale* pattern is to be used.

The second path from the *Measure* pattern goes to the *Measurement Procedures* group. *Measurement Procedure* should be used to model the procedures that guide data collection for measures. If the ontology engineer needs to address different types of measurement procedures, depending on the type of measures, he/she should use the *Measurement Procedure Type*. In this case, besides *Measure*, *Measure Type* should also be used in advance. Measurement procedures for base measures and derived measures are addressed respectively by *Measurement Procedure for Base Measure* and *Measurement Procedure for Derived Measure*. To model measurement procedures for derived measures, the engineer must first use the *Measurement Formula* pattern, which addresses measurement formulas used to calculate derived measures.

The third path from the *Measure* pattern goes to the *Measurement Planning* group. The ontology engineer should use the *Measurement Goal Type* pattern only if representing the decomposition of measurement goal is relevant. In order to address measurement goals and the information needs derived from them, the ontology engineer should use the *Measurement Goal* pattern. Then, if the ontology engineer wants to model measurement planning, addressing measurement goals, information needs and their relation with measures, he/she needs to use the *Measurement Planning Item* pattern. If it is relevant to indicate the measurement procedure in the measurement planning, then, he/she should also use *Measurement Planning Item Procedure*. It is important to notice that, in this case, as shown in Figure 7.9, the *Measurement Procedure* pattern must be used in advance. If the ontology engineer needs to model the relationship between measurement goals and the measures used to indicate their achievement, the *Indicator* pattern is to be used.

Finally, once the measurement planning is addressed, it is possible to model issues related to data collection and analysis. For dealing with data collection aspects, the ontology engineer should use the *Measurement* pattern. The *Measurement Analysis* pattern, in turn, should be used for the ontology engineer to model aspects related to data analysis.

For details regarding M-OPL and its patterns, see [2].

## 7.6. Final Remarks

This book deals with a range of themes related to ODPs, such as methodologies to use ontology patterns, ontology coding patterns, axiomatization of ODPs, among others. It is important to highlight that although the word “language” in Ontology Pattern Language (OPL) may be misleading, the work presented in this chapter actually concerns the support for domain ontology development. In this sense, OPLs provide guidelines for the development of the conceptual model of the ontology. Such conceptual model may eventually be implemented using a knowledge representation language, such as OWL, RDF(S), F-Logic, or others, a theme which falls outside the scope this chapter. In any case, the conceptual model representing the ontology has a value in itself, i.e., not only as an essential reference model for ontology implementation, but also supporting important tasks such as meaning negotiation and interoperability.

Summarizing the content of this chapter, we here discussed the notion of Ontology Pattern Languages (OPLs). Moreover, we exemplified how an ontology may be built with the support of an OPL. Finally, we presented existing OPLs on the domains of Software Process, Enterprises, Services and Measurement. The use of OPLs can guide the ontology engineer on selecting specific ontology patterns, depending on the problem being modeled. This may lead to gains in productivity, as well as to improvement in the quality of the resulting ontologies.

The pattern languages presented in this chapter must be seen as work in progress. In fact, OPLs in general should be expected to evolve as a result of new experiences gained throughout their application.

To summarize, an OPL provides concrete guidance, taking into consideration the following questions: (i) What are the main problems to be solved in the targeted domain? (ii) In which order should these problems be addressed? (iii) What alternatives are there to solve such problem? (iv) How should the dependencies among the problems be addressed? (v) How to solve each individual problem effectively, also considering the other problems related to it? In an OPL, the responses to these questions come as a network of ontology patterns that can be combined to create a domain ontology, and a process that guides the consistent application of such ontology patterns.

Completeness and maturity are paramount qualities of a good OPL. Moreover, we claim that OPLs must present some characteristics generally pointed as being present in “beautiful ontologies” [4]: 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.

## Bibliography

- [1] C. Alexander, S. Ishikawa, and M. Silverstein. *A pattern language: towns, buildings, construction*, volume 2. Oxford University Press, 1977.
- [2] M. Barcellos, R. Falbo, and V. Frauches. Towards a measurement ontology pattern language. In *ONTO.COM/ODISE FOIS*, 2014.
- [3] F. Buschmann, K. Henney, and D. Schimdt. *Pattern-oriented Software Architecture: On Patterns and Pattern Language*, volume 5. John Wiley & sons, 2007.
- [4] M. d’Aquin and A. Gangemi. Is there beauty in ontologies? *Applied Ontology*, 6(3):165–175, 2011.
- [5] P. Deutsch. Models and patterns. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, 2004.
- [6] R. Falbo, M. Barcellos, J. C. Nardi, and G. Guizzardi. Organizing ontology design patterns as ontology pattern languages. In *The Semantic Web: Semantics and Big Data*, pages 61–75. Springer, 2013.
- [7] R. Falbo, G. Guizzardi, A. Gangemi, and V. Presutti. 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.
- [8] R. Falbo, G. K. Quirino, J. C. Nardi, M. Barcellos, G. Guizzardi, N. Guarino, A. Longo, and B. Livieri. An ontology pattern language for service modeling. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. ACM, 2016.
- [9] R. Falbo, F. Ruy, G. Guizzardi, M. Barcellos, and J. P. Almeida. Towards an enterprise ontology pattern language. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 323–330. ACM, 2014.
- [10] M. S. Fox, M. Barbuceanu, M. Gruninger, and J. Lin. An organization ontology for enterprise modelling, 1997.
- [11] N. Guarino and G. Guizzardi. We need to discuss the relationship: Revisiting relationships as modeling constructs. In *Advanced Information Systems Engineering*, pages 279–294. Springer, 2015.
- [12] G. Guizzardi. *Ontological foundations for structural conceptual models*. CTIT, Centre for Telematics and Information Technology, 2005.
- [13] G. Guizzardi, R. Falbo, and R. Guizzardi. Grounding software domain ontologies in the unified foundational ontology (UFO): The case of the ode software process ontology. In *CIbSE*, pages 127–140, 2008.
- [14] ISO/IEC. ISO/IEC 15504. *Information Technology - Process Assessment. Part 1: Concepts and Vocabulary*, 2004.
- [15] ISO/IEC. ISO/IEC 24744. *Software Engineering - Metamodel for Development Methodologies*, 2007.
- [16] ISO/IEC. ISO/IEC 12207. *Systems and Software Engineering - Software Life Cycle Processes*, 2008.
- [17] ISO/IEC. ISO/IEC 15288. *Systems and Software Engineering - System Life Cycle Processes*, 2008.
- [18] ISO/IEC/IEEE. ISO/IEC/IEEE 29148. *Systems and software engineering - Life cycle processes - Requirements engineering*, 2008.



- [19] J. C. Nardi, R. Falbo, J. P. Almeida, G. Guizzardi, L. F. Pires, M. J. van Sinderen, N. Guarino, and C. Fonseca. A commitment-based reference ontology for services. *Information systems*, 54:263–288, 2015.
- [20] Object Management Group (OMG). Unified modeling language (UML), version 2.5.
- [21] F. Ruy, R. Falbo, M. Barcellos, and G. Guizzardi. An ontological analysis of the ISO/IEC 24744 metamodel. In *Proceedings of the 8th International Conference on Formal Ontology in Information Systems (FOIS'14)*, pages 330–343, 2014.
- [22] F. Ruy, R. Falbo, M. Barcellos, G. Guizzardi, and G. Quirino. An ISO-based software process ontology pattern language and its application for harmonizing standards. *ACM SIGAPP Applied Computing Review*, 15(2):27–40, 2015.
- [23] F. Ruy, C. Reginato, V. Santos, R. Falbo, and G. Guizzardi. Ontology engineering by combining ontology patterns. In *Proceeding of the 34th International Conference on Conceptual Modeling (ER'15)*, pages 173–186. Springer, 2015.
- [24] A. Scherp, C. Saathoff, T. Franz, and S. Staab. Designing core ontologies. *Applied Ontology*, 6(3):177–221, 2011.
- [25] D. C. Schmidt, M. Stal, H. Rohnert, F. Buschmann, and J. Wiley. *Pattern-oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*. Wiley, 2000.