PoN-S: A Systematic Approach for Applying the Physics of Notation (PoN)

Maria das Graças da Silva Teixeira[1], Glaice Kelly Quirino[1], Frederik Gailly[2],
Ricardo de Almeida Falbo[1], Giancarlo Guizzardi[1], Monalessa Perini Barcellos[1]

[1] Ontology and Conceptual Modeling Research Group (NEMO)
Federal University of Espírito Santo, Vitoria/ES, Brazil.
[2] Faculty of Economics and Business Administration. Ghent University, Ghent, Belgium.
`maria.teixeira@ufes.br, frederik.gailly@ugent.be, {falbo,`
`gksquirino, gguizzardi,monalessa}@inf.ufes.br`

Abstract. Visual Modeling Languages (VMLs) are important instruments of communication between modelers and stakeholders. Thus, it is important to provide guidelines for designing VMLs. The most widespread approach for analyzing and designing concrete syntaxes for VMLs is the so-called Physics of Notation (PoN). PoN has been successfully applied in the analysis of several VMLs. However, despite its popularity, the application of PoN principles for designing VMLs has been limited. This paper presents a systematic approach for applying PoN in the design of the concrete syntax of VMLs. We propose here a *design process* establishing activities to be performed, their connection to PoN principles, as well as criteria for grouping PoN principles that guide this process. Moreover, we present a case study in which a visual notation for representing Ontology Pattern Languages is designed.

Keywords: Concrete Syntax • Design Process • Visual Modeling Language • Physics of Notation, Ontology-Pattern Languages.

1 Introduction

Visual Modeling Languages (VMLs) are important instruments of communication between modelers and stakeholders. The quality of a VML influences the results of a modeling task [1]. Thus, it is relevant to provide guidelines for designing VMLs. Basically, a VML comprises an abstract syntax, which defines the modeling elements (constructs) of the language, and a concrete syntax, which defines the representational elements (symbols) of the language [2]. The concrete syntax can be constituted of one or more dialects, which are different symbol sets to represent the same abstract syntax. These different dialects reflect variations in the language users' profile and modeling task application [3]. Complementary to these alternative syntaxes, there are representation strategies for managing model complexity, which identify mechanisms to visualize large (or complex) models. Our focus is on the design of concrete syntaxes for VMLs. Thus, we are interested in, given an abstract syntax, how to design the correspondent concrete syntax, its complementary dialects and representation strategies to manage model complexity.

The most widespread work in the area of designing visual aspects of modeling languages is the *Physics of Notation (PoN)* [3]. PoN defines an approach that is supposed to be used for designing cognitively effective visual notations, i.e. notations that are optimized for being processed by the human mind. PoN consists of nine principles that are based on theories and empirical evidence from a wide range of fields [4]. However, PoN does not prescribe any method or process for systematically applying its principles [5][6].

In this paper, we present a systematic approach for bridging the theory and practice of PoN in the design of concrete syntaxes for VMLs. We term this approach *PoN-Systematized (PoN-S)*[1]. The process establishes an ordered set of tasks and suggests when to apply the PoN principles. It takes into account a way of grouping these principles. Also, we describe a case study applying the approach in the design of the concrete syntax of a visual language for modeling *Ontology Pattern Languages* [7].

This paper is structured as follow. Section 2 presents the foundations of PoN. Section 3 describes the proposed process for systematizing the application of PoN. Section 4 presents the case study. Section 5 discusses some related works. Finally, Section 6 presents our final considerations.

2 Fundamentals of the Physics of Notation (PoN)

PoN defines a set of principles for designing cognitively effective visual notations. The approach considers information visualization and pragmatic theories in order to improve the cognitive effectiveness of VMLs, which is defined as "the speed, ease, and accuracy with which a representation can be processed by the human mind" [3].

Following the tradition in the literature, PoN [3] considers the following elements as the ingredients of a visual notation: a set of graphical symbols (visual vocabulary), a set of compositional rules for forming valid expression (visual grammar), and semantic definitions for each symbol (visual semantics). The set of symbols and compositional rules form the visual (concrete) syntax. Graphical symbols are used to signify or symbolize (perceptually represent) semantic constructs, typically defined by a meta-model. An expression in a visual notation is called a visual sentence or diagram. Diagrams are composed of instances of graphical symbols arranged according to the rules of the visual grammar [8].

PoN identifies nine principles for designing cognitively effective visual notations, namely [3]: (i) *Semiotic Clarity*: "There should be a 1:1 correspondence between a meta-model construct and a graphical symbol"; (ii) *Semantic Transparency*: "Use symbols whose appearance suggests their meaning"; (iii) *Perceptual Discriminability*: "Symbols should be clearly distinguishable from one another"; (iv) *Complexity Management*: "Include explicit mechanisms for dealing with complexity"; (v) *Cognitive Integration*: "Include explicit mechanisms to support integration of information from different diagrams"; (vi) *Visual Expressiveness*: "Use the full range and capacities of visual variables"; (vii) *Graphic Economy*: "The number of different

---

[1]PONS is a region of the brainstem with neural pathways that carry sensory signals including those related to *eye movement*. Etymologically, the term from Latin also means *bridge*.

graphical symbols should be cognitively manageable"; (viii) *Dual Coding*: "Use text to complement graphics"; (ix) *Cognitive Fit*: "Use different visual dialects for different tasks and audiences".

There are in the literature a number of concrete cases of the application of PoN, for instance: (i) in [8], the authors describe the evaluation and redesign of a visual notation for the i* language. The publication describes a notation redesign effort and a description of the redesign process employed, which adds operational characteristics to PoN. This work is used as a study in [4], which analyzes the influence of model readers on the language concrete syntax; (ii) in [6], the authors evaluated the UCM visual notation and present not only the evaluation and redesign proposal, but their impressions concerning the application of the approach.

An important consideration is that the principles in PoN influence one another. Knowledge of these influence relations can be used to spot tradeoffs (where principles conflict with one another), as well as synergies (where principles complement or reinforce one another). In [3], Moody presents detailed descriptions of each principle and the influence relations between them. However, when a designer is applying these principles in a design process, s/he needs further design guidance. For instance, when should s/he apply a principle? In which sequence should principles be applied? Which principles should be applied in tandem? In order to overcome this limitation, we propose a systematic process for applying PoN.

3 PoN Systematized (PoN-S)

In this section, we present an approach for systematizing the application of PoN in the design of concrete syntax of VMLs. The methodology we followed is based on the Design Science approach discussed in [9]. The steps carried out here were: (i) we identified the design questions that a developer needs to answer; (ii) we established how the PoN principles are related to these design questions; (iii) in order to systematize the process, we described groups of PoN principles; (iv) we added ordering relations between the design tasks constituing the design process. Our approach is based not only on the foundations of PoN [3], but also on works that have applied PoN for analysis or (re)design of VMLs, as [4, 5, 6],[8].

Design questions: When designing the VML´s concrete syntax, we should deal with concerns at different levels. First, we need to decide whether different dialects for the same abstract syntax are needed. The motivation for creating more than one dialect should be clearly identified (e.g., the fact that the language must be suitable for more than one stakeholder profile, modeling task or problem domain characteristics). Second, at the language level, we need to determine the symbols to be used in the concrete syntax. Finally, at the instance level, we are concerned with the development of diagrams using the proposed concrete syntax. Table 1 presents the concerns for these different levels as design questions and identifies the PoN principles that can be applied to answer them. Answering these design questions helps the designer to understand the rationale behind the application of each principle, acting as initial guidance. However, this is not enough for guiding the design effort. To do so, we

propose a way of grouping the principles and a process for applying them when designing a concrete syntax.

Table 1. Answering to some basic design questions with PoN principles

Design Question	Related PoN Principles
Dialect set	
Do we need different dialects for the abstract syntax? If so, which dialects should we consider?	Cognitive Fit
For each dialect	
Language level	
Which symbol(s) do we need to create?	Semiotic Clarity
How to create each symbol?	Semantic Transparency
How to relate differents symbols? To what extent two or more symbols should be similar / different?	Perceptual Discriminability
How visual variables (such as shape, color and texture) and text should be applied in order to aid the identification of each representational element?	Visual Expressiveness Graphic Economy Dual Coding
Instance level	
Which procedures should we create to support the development of a(some) diagram(s)? (Depending on the answer to this question, it may be necessary to create new symbols, affecting decisions at the language level.)	Complexity Management Cognitive Integration

Grouping the Principles: Moody describes a number of influence relations between pairs of principles [3]. However, most of the time these principles act in group. This perception is fundamental to guide the VML design process. Thus, we suggest grouping PoN principles into the following groups:

- *Group 1 – Basic principles*. This group comprises three principles: Semiotic Clarity, Semantic Transparency and Perceptual Discriminability. These principles are considered basic principles, because they should be applied at some extent in the design of any concrete syntax. They are complementary in the sense that we need to create a symbol to each construct (Semiotic Clarity), and each of these symbols should be clearly identifiable (Semantic Transparency), and yet clearly distinguishable from other symbols in the language (Perceptual Discriminability). So, these principles should be applied together in the design of each dialect of the concrete syntax, and the level they should be in compliance with can vary in each dialect. Semiotic Clarity acts as a guarantee that the mapping between abstract and concrete syntaxes is complete, avoiding possible anomalies, i.e., that all necessary symbols are defined. Perceptual Discriminability is concerned with whether such symbols are adequately different from (or similar to, depending on the case) the others. Finally, Semantic Transparency is concerned with whether each symbol has its meaning easily inferred.

- *Group 2 – Information complexity management principles*. This group comprises two principles: Complexity Management and Cognitive Integration. These principles are commonly applied when dealing with large or complex diagrams. They are complementary, since the former deals with how to organize the information in a model (probably separating them in several diagrams), and the second refers to how to keep connection and traceability of the information spread in different diagrams. Thus, they should be applied together. Basically,

this group of principles will be applied at the level of individual diagrams, giving rise to representation strategies for managing model complexity. Ideally, the way of addressing information complexity management should be the same (or very similar) in any dialect of the concrete syntax. Finally, it is worth pointing out that the application of these two principles can demand the creation of new symbols, hence, affecting the language level (Group 1).

- *Group 3 – Supporting principles*. The principles in this group can somehow affect principles of groups 1 and 2. The support principles are: Visual Expressiveness, Graphic Economy and Dual Coding. Visual Expressiveness is connected to the other principles (except Dual Coding), in the sense that it provides the mechanisms (as visual variables) for implementing the other principles. Graphic Economy is also connected to the other principles, since it establishes a way to control them, trying to keep them as simple as possible. We consider here Dual Coding to refer only to redundant textual representational support.

- *Group 4 – Dialect set principle*. This group, in fact, contains only one principle: Cognitive Fit. This principle has an indirect connection to the other principles, because the other principles are applied to each dialect of the concrete syntax at a time, while Cognitive Fit is about defining the set of dialects.

The principles of a group can interact with principles of another group, as in Group 3 (which influences groups 1 and 2). Furthermore, the principles inside a group can interact with each other. Typically, this intra-group relationship is stronger than the inter-group relationships. This is a reason for grouping the principles in such way.

Design process: The design questions and groups of principles give us some guidance for designing the concrete syntax. However, to truly systematize the application of PoN, we need a *design process* for guiding this. Figures 1 to 4 present the proposed design process. This process is structured according to the concerns shown in Table 1, starting by the concern related to the dialect set, and in the sequel addressing concerns related to the language level and then the instance level, for each dialect. Each figure presents a part of the process, including inputs, outputs, tasks and decisions to be made. The process is represented by means of an extension the UML activity diagrams notation, introducing a new modeling element: *PoN principle.* A PoN principle can be seen as a guideline to perform an activity and it is represented by means of an ellipse, which is connected by a line to the activity that applies the principle at hand.

Fig. 1 depicts the whole design process, which comprises two main stages: *Define dialect set* and *Design dialect*. In the stage *Define dialect set*, the designer shall identify the requirements for the VML (modeling task, stakeholder profile, problem domain characteristcs) that help to *Define the size of the language dialect set*. Next, in the activity *Identify the dialect goal and directives for its design*, each dialect should be characterized, establishing its goal and directives for its design. In this task, the designer should take into account the influence relations (conflicts or synergies) that exist among PoN principles (see [3]). It is not possible to establish the same level of compliance to all principles. So, the designer should choose the principles to highlight

in each dialect. The stage *Define dialect set* should be performed considering language abstract syntax (as input) and the Cognitive Fit principle (as method).

In the second stage (*Design dialect*), each dialect identified should have its set of symbols (representational elements) defined in accordance to the goal and directives previously identified. This stage has two complex tasks: (i) *Define the dialect symbol set*, which is responsible for defining representational elements for the model elements identified by the abstract syntax; and (ii) *Identify ways to manage model complexity*, an optional task performed when the amount of elements requires managing model complexity. The input for these tasks are the language dialect set and the abstract syntax. The output is the concrete syntax, and optionally some representation strategies to deal with size and complexity of the models. These two complex tasks are further detailed in figures 2 and 4, respectively.
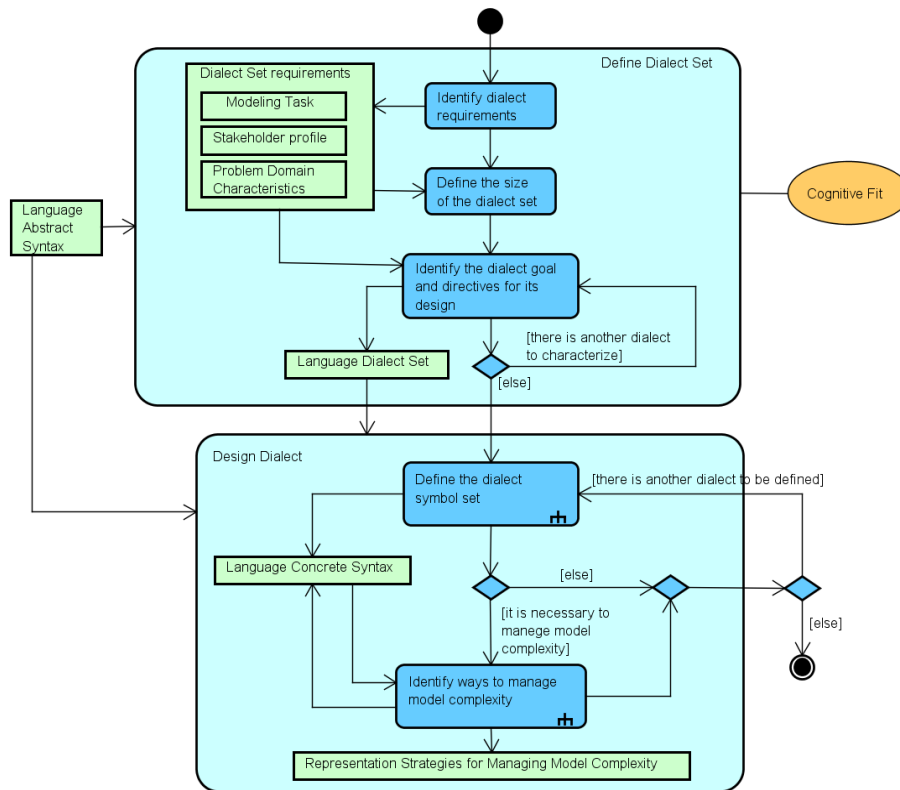

Fig. 1. PoN-S design process overview

Fig. 2 depicts the steps for defining the dialect symbol set for each dialect previously identified. This activity starts by choosing a model element to be represented. This task is guided by Semiotic Clarity principle to ensure that each model element will be represented by exactly one symbol, unless this situation is required due to the directives established for the dialect. Once the model element to be represented is chosen, we need to define a symbol for it (task *Define a symbol to the modeling element*). This activity is guided by the Semantic Transparency principle in order to

establish a clear meaning to the symbol. Also, we should relate the chosen symbol to the other symbols already defined in the concrete syntax, following the Perceptual Discriminability principle. This task aims at evaluating the visual distance between the new symbol and the other symbols already defined. These two tasks can be supported by the application of supporting principles (see Fig. 3). They are performed in a loop until all the representational elements of that dialect have been defined.

The *Apply supporting principles* activity depicted in Fig. 3 deals with the possible application of three supporting principles: Visual Expressiveness, Graphic Economy and Dual Coding. The designer can apply each principle as much as s/he deems necessary. There is no pre-defined order to be followed. The inputs are the language abstract and concrete syntaxes and the characteristics of the dialect. The output can be an update of the language concrete syntax or an update of some representation strategy for managing model complexity.
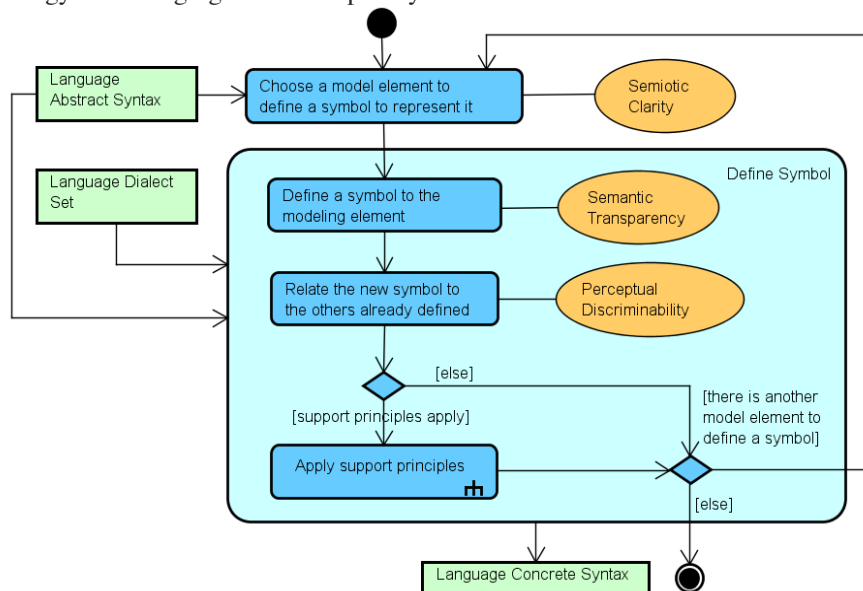

Fig. 2. "Define the dialect symbol set" activity

The *Improve the use of visual variables* activity is guided by the Visual Expressiveness principle. In this task, the designer shall review the symbol(s) (or strategies), possibly updating the visual variables values to maximize their expressiveness. The designer can do this individually (per symbol) or considering the whole symbol set. The *Simplify the symbol set* activity is guided by the Graphic Economy principle. In this task, the designer may also review the symbol(s) (or strategies), now with the goal of simplifying the dialect. Finally, in the *Define textual complement* activity, by applying the Dual Coding principle, the designer should evaluate when it is useful to introduce redundancy through the use of text. This can be necessary when the designer deems that the text will increase symbol expressiveness.

After defining the symbols of a dialect, the designer must decide if it is necessary to manage the model complexity in diagrams developed using this dialect. Therefore,

the *Identify ways to manage model complexity* activity is an optional activity whose importance increases as the language grows in size and complexity.

Fig. 4 details the complex activity *Identify ways to manage model complexity*. The inputs for this activity are the language abstract and concrete syntaxes as well as the characteristics of the dialect set. The outputs are the language concrete syntax (in case it suffers some update) and representation strategies for managing model complexity (as many as the designer deems necessary).

The first task is *Manage model complexity*, which is guided by the Complexity Management principle. In this task, representation strategies for managing the complexity of diagrams written in that dialect shall be established. An example is the use of modularization. As a complement to this task, there is the task *Integrate spread information*, which is guided by the Cognitive Integration principle. This task is responsible for establishing ways to trace information spread in several diagrams and strategies for connecting them. It is important to say that these two tasks can be applied in parallel, resulting in a single representation strategy that is in accordance with both aspects of complexity management (organization and integration of information). In fact, both tasks are applied in independent loops until deemed sufficient by the designer. Usually, each cycle results in a representation strategy for managing model complexity, which is complemented by new concrete syntax elements, when necessary.
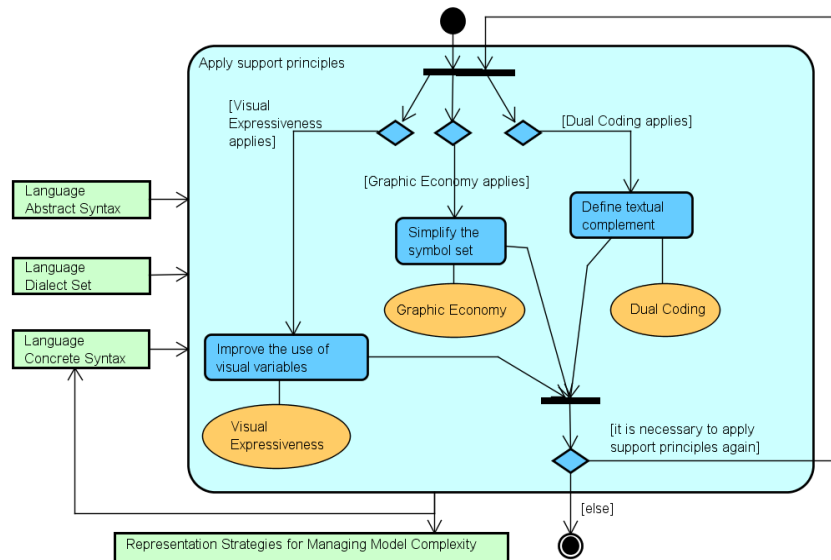

Fig. 3. "Apply support principles" activity

4 Applying PoN-S: A Case Study

In a preliminary evaluation of PoN-S, a case study was performed aiming at designing a visual notation for representing Ontology Pattern Languages (OPLs). An OPL is a

network of interrelated domain-related ontology patterns that provides holistic support for solving ontology development problems for a specific domain. It 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 in which these problems should be addressed, and suggesting one or more patterns to solve each specific problem [7][10]. For adequately representing OPLs, two types of models are necessary: a structural model, showing the patterns and the dependency relationships between them, and a process model, showing, among other things, the activities of applying the patterns, decision points, and entry and end points in the OPL process.
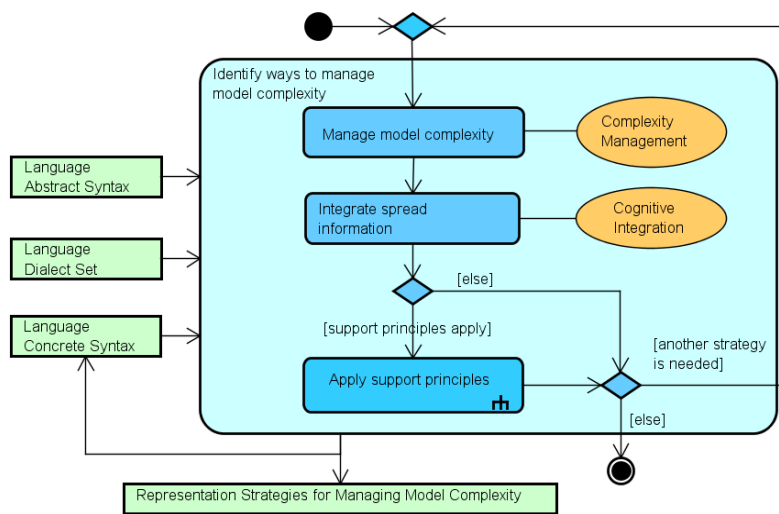

Fig. 4. "Identify ways to manage model complexity" activity

Regarding the process model, in a nutshell, its meta-model is an extension for representing OPLs of the meta-model of the UML activity diagram [9]. For this reason, its concrete syntax is based on the UML notation for activity diagrams. This has the advantage of benefiting users who are familiar with this notation. Due to space limitations, however, in this paper we do not discuss the design of the visual notation for the process model. Our focus here is on discussing the application of PoN-S, and thus we concentrate in the design of the visual notation for the structural model.

Fig. 5 shows the meta-model of the language concerning the OPL structural model. This model is composed of *OPL Structural Elements*. There are two types of *OPL Structural Elements*: *Pattern* and *Pattern Group*. A *Pattern* represents a domain-related ontology pattern, i.e., a small and reusable fragment of an ontology conceptual model, extracted from a reference ontology [11]. A *Pattern Group* is a way of grouping related patterns and other pattern groups. Thus, a *Pattern Group* is composed by *OPL Structural Elements*. A special type of pattern group is the *Variant Pattern Group*, which is a set of (variant) patterns that solve the same problem, but each in a different way. Only one pattern from a *Variant Pattern Group* can be used at a time. *Patterns* that compose a *Variant Pattern Group* are variants of each other, giving rise to the derived relationship *variantOf* between patterns.

Patterns may depend on other patterns, i.e., for applying a pattern *p2* another pattern *p1* has to be applied first. An OPL should be able to represent dependencies between patterns or between a *Pattern Group* and a *Pattern*. The *requires* relationship captures this dependency. In the case of a dependency between a *Pattern Group* and a *Pattern*, the following rule applies: If a pattern *p1* is part of a pattern group *pg* and *pg* requires a pattern *p2*, then *p1* requires *p2*. Finally, a *Pattern* may require the application of a pattern from a *Variant Pattern Group*.
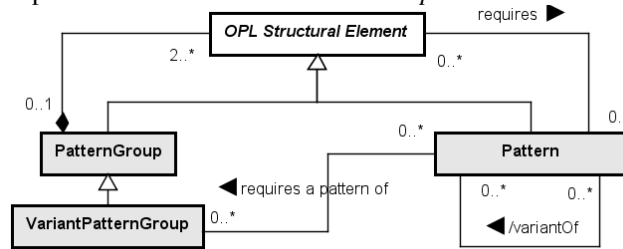

Fig. 5. OPL structural meta-model

As Fig. 1 shows, the design process started by identifying the dialect requirements, which includes: (i) *Domain characteristics:* the visual notation for representing OPLs. Each OPL can refer to a different domain. Thus, this is the case of a domain-independent language; (ii) *Stakeholder profile*: OPLs are typically used by ontology engineers (both beginners and experienced); (iii) *Modeling task*: developing domain ontologies by reusing domain-related ontology patterns.

Although there are stakeholders with different levels of experience, the OPL visual notation should be simple and intuitive for all kinds of stakeholders. Thus, the designer established that only one dialect is enough. The goal of this dialect is to provide a simple and intuitive visual notation for ontology engineers to develop domain ontologies by reusing ontology patterns [7][10]. The notation should contain symbols to represent all OPL constructs without ambiguity. Moreover, in case of the use of colors, it should be possible to print the diagrams in gray scale without denting their comprehensibility.

The next step is to define the dialect symbol set. A loop was performed, in which each model element was characterized and had a symbol defined for it. This loop was guided by the principles of Semiotic Clarity, Semantic Transparency, Perceptual Discriminability as well as the supporting principles. Initially, considering the abstract syntax defined by the meta-model shown in Fig. 5, and taking into account the Semiotic Clarity principle, a 1:1 correspondence between the meta-model constructs and graphical symbols was defined. This otherwise isomorphic mapping has two exceptions: the designer decided that it was not necessary to assign a symbol to the *OPL Structural Element* construct (an abstract modeling element), but only to its (concrete) subtypes (*Pattern* and *Pattern Group*). Moreover, symbols should be assigned to the relationships between these constructs, except for the *variantOf* relationship, since it is a derived association. Thus, symbols should only be assigned to the constructs shown in gray in Fig. 5 and to the regular associations between them.

The designer started assigning a symbol to the *Pattern* construct. Since s/he was dealing with a domain-independent language, s/he decided to represent patterns by rectangles (an abstract sign). This choice was done considering that this is a common

symbol used for representing patterns in Software Engineering Languages (e.g., UML class diagrams). Concerning Semantic Transparency, on one hand, this symbol is considered semantically opaque, since it does not inform its meaning directly [3]. However, on the other hand, it can be considered a good design decision, given that this symbol is easily recalled [3].

Pattern Groups are represented by figures closed by straight solid lines (solid polygons). For representing *Variant Pattern Groups*, the same notion was applied, but now using dashed lines. This decision was taken considering the Perceptual Discriminability principle, aiming at guaranteeing that symbols representing groups have a small visual distance. Furthermore, the visual variables *texture* and *color* were used to differentiate them. The lines of *Variant Pattern Groups* are dashed and red, while the lines of *Pattern Groups* are solid and blue.

For representing the relation between *Patterns* and *Pattern Groups*, the designer chose the notion of spatial containment: *Patterns* that are part of a *Pattern Group* represented as spatially enclosed by the symbol representing the latter. This choice affords the so-called *inferential free-rides* to the language, i.e., visual querying and reasoning operations of minimal cognitive costs [12]. Moreover, it is noteworthy that there is a visual variable that qualifies *Patterns* and *Pattern Groups*: size. The region that represents the group encompasses several patterns. Thus, the size of this region is greater than the rectangle representing the pattern.

Regarding the dependency relations *requires* and *requires a pattern of*, both are represented by an arrow from the dependant to the dependee. For differentiating between them, arrows representing the *requires* association are symbolized with solid lines, in contrast to the dashed lines for the *requires a pattern of* association. This decision is in line with the one of representing *Pattern Groups* using solid lines, and *Variant Pattern Groups* using dashed lines. Thus, it takes the Perceptual Discriminability principle into account. So, these symbols have small visual distances.

It is worthwhile to point out that supporting principles were also applied for making the aforementioned choices. Regarding the Visual Expressiveness principle, the proposed visual notation uses the following visual variables: shape, texture and size. Color values are used as a redundant encoding, because variation in color disappears when a diagram is printed in grayscale. The designer decided not to apply other visual variables, keeping the notation as simple as possible.

The Graphic Economy principle did not play a strong role in this case study. This is because PoN advocates the use of up to six elements in a dialect and the structural meta-model considered here has only four classes and three regular associations. Nevertheless, some decisions were taken aiming at making the language as simple as possible. In summary, no symbol was assigned to the following meta-model elements: *OPL Structural Element* construct, since it is an abstract class in the meta-model (i.e., it cannot be directly instantiated); whole-part relationship between *Pattern Group* and *Pattern*, since the notion of containment used to represent *Pattern Groups* also addresses this relation; and the derived association *variant of*, since it is also derived from the representation for *Pattern* and *Pattern Group*.

Finally, the Dual Coding principle, which deals with the use of text as an information supplement, was not applied. This is because, according to the designer: there´s a small amount of constructs to represent, their semantic are clear enough

without textual redundancy and use of textual values can be better applied to distinguish between instances (as instance labels).

After defining an initial version of the concrete syntax, it is time to evaluate if the language demands representation strategies for managing model complexity. If this is the case, we should apply the principles of Complexity Management and Cognitive Integration. The Complexity Management principle emphasizes the importance of managing the diagrammatic complexity, which is measured by the number of elements in a diagram, among others. In the case of this case study, the designer recognized the need for managing complexity. Although the proposed language for representing OPLs is simple, the models that may be built using it tend to be large. Thus, to increase the speed and accuracy of understanding the diagrams, the designer decided to introduce a symbol for representing *Pattern Groups* (including *Variant Pattern Groups*) that encapsulates the *Patterns* that comprise it. Following the Perceptual Discriminability principle, the designer chose to represent these alternative forms by means of rectangles decorated by the following icon (⛫), indicating that this element is detailed in another diagram[2].

Table 2 shows the final concrete syntax for representing OPL structural models.

Table 2. Symbols of the visual notation for OPL structural models

Structural Model	
Element	**Symbol**
Pattern	<<name>>
Pattern Group (expanded format)	<<name>>
Pattern Group (black box format)	<<name>>
Variant Pattern Group (expanded format)	<<name>>
Variant Pattern Group (black box format)	<<name>>
Relation "requires"	→
Relation "requires a pattern of"	--→

Fig. 6 shows an example of a structural model of an OPL: Service OPL (S-OPL). This OPL, which provides ontology patterns for service modeling, is discussed in details in [10]. As shown in this figure, S-OPL is organized in three groups: *Service Offering*, *Service Negotiation and Agreement* and *Service Delivery*. The *Service Offering Group* is composed by three patterns (*SOffering*, *SODescription* and *SOCommitments*) and two groups of variant patterns (*Provider Variant Group* and *Target Customer Variant Group*). The patterns *SODescription* and *SOCommitments* as well as the *Provider* and *Target Customer Variant Groups* require the pattern *SOffering*. *SOffering*, in turn,

---

[2]This icon is commonly used by UML to represent that an element represented by the decorated construct encapsulates further elements. A similar symbol is used by ARIS.

requires patterns of both *Provider* and *Target Customer Variant Groups*. *Provider* and *Target Customer Variant Groups* are both composed of seven variant patterns each. The *Service Negotiation and Agreement Group* is composed by four patterns (*SNegotiation*, *SADescription*, *HPCommitments* and *SCCommitments*) and three groups of variant patterns (*Agreement Variant Group*, *Hired Provider Variant Group* and *Service Customer Variant Group*). The *Agreement Variant Group* is composed by three patterns: *SNegAgree*, *SOfferAgree*, and *SAgreement*. The first two of these patterns as well as the SNegotiation pattern require *SOffering*. The patterns *SADescription*, *HPCommitments* and *SCCommitments* require a pattern of the *Agreement Variant Group*. The *SAgreement* pattern requires patterns of both *Hired Provider* and *Service Customer Variant Groups* (shown as black boxes in Fig. 6). These two variant groups, in turn, require the *SAgreement* pattern. Finally, the *Service Delivery Group* (shown as a black box in Fig. 6) requires the *Service Negotiation and Agreement Group*.
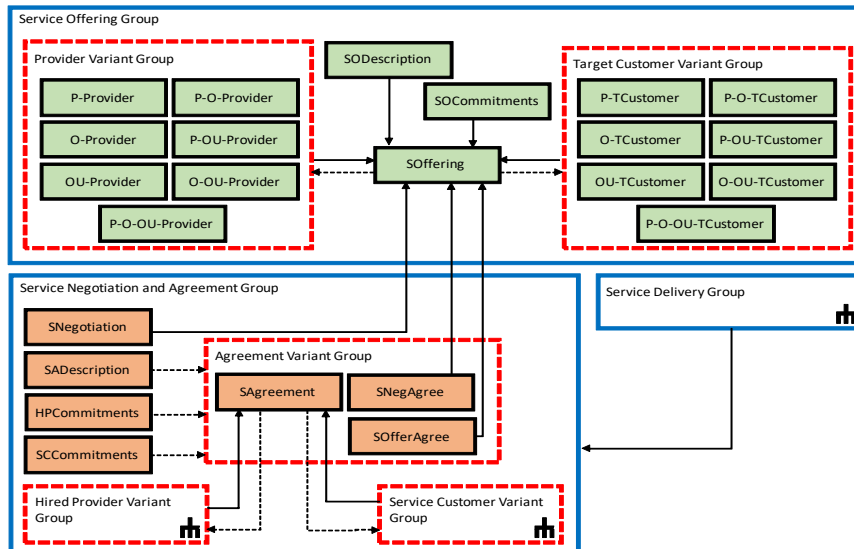

Fig. 6. S-OPL structural model

5   Related Work

In a brief literature review, executed to identify how concrete syntax of conceptual modeling languages have been evaluated and designed, we identified PoN as the most widespread approach for analysis and design of VML concrete syntax [5],[13]. Also, we noticed that studies discussing efforts in analyzing modeling languages (with associated redesign suggestions)(e.g., [6],[14]) are more common than those describing efforts in language design (e.g., [15]).

The need for improving the design process involving PoN has been identified by many researchers, including Moody himself. In [8], Moody *et al.* discuss operational

issues of PoN when presenting the analysis and redesign of i* (a language in the Requirements Engineering field). However, these issues are discussed individually for each principle, i.e., the authors do not define a process involving all principles. In [4], a work complementing the i* evaluation described in [8] , the authors added the idea of PoN operationalization, highlighting the importance of considering stakeholder profiles during language design. It is a clear contribution towards considering pragmatic issues for notation analysis and design. However, once more, they did not define a design process.

In [5], Storrle and Fish criticize PoN judging that it still needs improvements towards operationalization. In that article, the authors propose ways for operationalizing PoN focusing on the analysis task of modeling languages. Moreover, they established a series of measures that complement the PoN original proposal. However, they also do not propose a design process.

The work proposed in this paper contributes to this collective effort of proposing *operationalizable* techniques for the design of visual languages. In particular, PoN-S is a methodological contribution that supports language designers in the application of PoN through the definition of a design process, a gap that has been identified in the literature.

6   Final Considerations

This paper focused on the elaboration of PoN-S, a design process for applying the PoN principles in practice. The elements involved in this process model are: inputs, outputs, tasks, task ordering and procedures (the PoN principles). This process was applied in a case study aiming at developing a visual notation for Ontology Pattern Languages. The case study was the first validation of PoN-S. As a result of its execution, the design process was refined, for example, by identifying the need for a *Identify dialect requirements* task. This study also indicated that PoN-S is easy to follow. We are currently conducting an on going survey whose preliminary findings indicate that the OPLs' users approve the resulting concrete syntax.

An expected benefit of PoN-S is the establishment of a path that modeling language designers can follow. This is particularly helpful mainly for novice language designers. When defining a clear and simple path we are reducing the possibility of errors during the process. The need to reduce the effort of PoN application is a recognized problem [6]. Also, a systematic process aids in the standardization of the language design, which facilitates future maintenance tasks and facilitates teamwork.

The establishment of the tasks constituting PoN-S take into account: (i) the PoN principles, assuring that every principle is considered; (ii) visual aspects of a VML (symbol set, dialects, representation strategies for manage model complexity). A current limitation of the process is the level of details in which some tasks have been defined. For example, in *Identify ways to manage model complexity* task, we state that strategies should be defined, but we do not identify how to create these strategies.

We are planning to extend the design process of PoN-S to provide more directed and complete guidelines for the language designer. In particular, ontological theories such as the ones discussed in [12] are the basis for such future extensions of PoN-S.

References

1. Krogstie, J., Solvberg, A.: Information Systems Engineering: Conceptual Modeling in a Quality Perspective. Draft of Book, Information Systems Groups, NTNU, Trondheim, Norway (2000).
2. Ruiz, M., Costal, D., España, S., Franch, X., Pastor, Ó.: Integrating the Goal and Business Process Perspectives in Information System Analysis. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., and Horkoff, J. (eds.) Advanced Information Systems Engineering: 26th International Conference, CAiSE 2014, Thessaloniki, Greece, June 16-20, 2014. Proceedings. pp. 332–346. Springer International Publishing, Cham (2014).
3. Moody, D.L.: The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE TSE. 35(6), 1–22 (2009).
4. Caire, P., Genon, N., Heymans, P., Moody, D.L.: Visual notation design 2.0: Towards user comprehensible requirements engineering notations. Requirements Engineering Conference (RE). pp. 115–124. IEEE Computer Society (2013).
5. Storrle, H., Fish, A.: Towards an Operationalization of the "Physics of Notations" for the Analysis of Visual Languages. In: Moreira, A., Schatz, B., Gray, J., Vallecillo, A., and Clarke, P.J. (eds.) International Conference on Model Driven Engineering Languages and Systems (MoDELS). pp. 104–120. Springer (2013).
6. Genon, N., Amyot, D., Heymans, P.: Analysing the cognitive effectiveness of the UCM visual notation. 6th International Conference on System Analysis and Modeling (SAM'10). , Oslo, Norway (2010).
7. Falbo, R. de A., Barcellos, M.P., Nardi, J.C., Guizzardi, G.: Organizing Ontology Design Patterns as Ontology Pattern Language. 10th Extended Semantic Web Conference - ESWC 2013-a. , Montpellier, France (2013).
8. Moody, D.L., Heymans, P., Matulevicius, R.: Visual syntax does matter: improving the cognitive effectiveness of the i* visual notation. Requir. Eng. 15, 141–175 (2010).
9. Wieringa, R.J.: Design Science Methodology for Information Systems and Software Engineering. Springer Verlag, London (2014).
10. Falbo, R. de A., Quirino, G.K., Nardi, J.C., Barcellos, M.P., Guizzardi, G., Guarino, N., Longo, A., Livieri, B.: An Ontology Pattern Language for Service Modeling. Proceedings of the 31th Annual ACM Symposium on Applied Computing - ACM-SAC 2016 (2016).
11. Falbo, R.D.A., Guizzardi, G., Gangemi, A., Presutti, V.: Ontology Patterns: Clarifying Concepts and Terminology. Proceedings of the 4th Workshop on Ontology and Semantic Web Patterns (2013).
12. Guizzardi, G.: Ontology-Based Evaluation and Design of Visual Conceptual Modeling Languages. In: Reinhartz-Berger, I.; Sturm, A.; Clark, T.; Cohen, S.; Bettin, J. (ed.) Domain Engineering. Product Lines, Languages and Conceptual Models. p. 345. Springer, New York, USA (2013).
13. Genon, N., Heymans, P., Amyot, D.: Analysing the Cognitive Effectiveness of the BPMN 2.0 Visual Notation. Proceedings of the Third International Conference on Software Language Engineering. pp. 377–396. Springer-Verlag, Berlin, Heidelberg (2011).
14. Figl, K., Derntl, M.: The Impact of Perceived Cognitive Effectiveness on Perceived Usefulness of Visual Conceptual Modeling Languages. In: Jeusfeld, M.A., Delcambre, L.M.L., and Ling, T.W. (eds.) International Conference on Conceptual Modeling (ER). pp. 78–91. Springer (2011).
15. Miske, C., Rothenberger, M.A., Peffers, K.: Towards a More Cognitively Effective Business Process Notation for Requirements Engineering. In: Tremblay, M.C., VanderMeer, D.E., Rothenberger, M.A., Gupta, A., and Yoon, V.Y. (eds.) Advancing the Impact of Design Science: Moving from Theory to Practice - 9th International Conference, (DESRIST). pp. 360–367. Springer (2014).