A UML-based Process Meta-Model Integrating a Rigorous Process Patterns Definition

Hanh Nhi Tran[1], Bernard Coulette[1], Bich Thuy Dong[2]

[1] University of Toulouse 2 -GRIMM
5 allées A. Machado F-31058 Toulouse, France
{tran,coulette}@univ-tlse2.fr
[2] University of Natural Sciences, VNUHCM
227 Nguyen Van Cu, Q5, HoChiMinh Ville, Vietnam
thuy@hcmuns.edu.vn

Abstract. Process Pattern is an emergent approach to reuse process knowledge. However, in practice this concept still remains difficult to be exploited due to the lack of formalization and supporting methodology. In this paper, we propose a way to formalize the process pattern concept by introducing it into a process meta-model. We provide a general definition to cover various kinds of process-related patterns in different domains. We define rigorously process concepts and their relations to allow representing processes based on process patterns and to facilitate the development of supporting tools. By distinguishing process patterns at different abstraction levels, we aim to develop a systematic approach to define and apply process patterns.

1 Introduction

Recently, process patterns approach has been adopted by process communities to capture and reuse proven processes. The expected interests of reduced development time and improved process quality make process patterns approach attractive. However, up to now works on process pattern have not been enough developed to permit applying this concept efficiently. Firstly, it lacks a definition that can cover the diversity of process patterns (c.f. [8][9] for more detailed discussions on process patterns definitions and taxonomy). Secondly, the representation of process patterns is not adequately formal to be directly reused in process modeling. Finally, a process patterns based methodology for developing processes is still needed to guide process designers applying process patterns systematically.

To cope with these issues, we believe that the first step is to formalize process patterns. This objective can be achieved by introducing the process pattern concept into a process meta-model. We present in this paper a UML-based meta-model integrating the concept of process pattern into the description of software processes. Our meta-model provides rigorous concepts to describe process patterns and processes based on process patterns. The meta-model will be presented in the following section and exemplified in section 3. Related works and contributions are summed up in the final section.

2 A meta-model for representing Process Patterns

Our objective is to define process patterns as patterns for modeling processes. Thus, we provide a wide definition to cover all the related notions.

We introduce multi-abstraction levels[1] into process representation to promote process reuse. The relations between process models at different abstraction levels are defined in order to clarify the way of defining process models (and then process patterns which capture them). To allow process modeling based on patterns and facilitate patterns organization, we also define explicitly the aspects of process pattern concept as well as the relationships among patterns, between process patterns and process models. To attaint the requirement on standardization, initially we aimed at integrating the process pattern concept into the software process meta-model SPEM 1.1[4]. However, SPEM 1.1 has some deficiencies and is progressing towards a next version [5]. So, we decided to develop a meta-model which is strongly inspired by SPEM 1.1 for process description, but based directly on UML[2].

To present our meta-model, we use UML class diagrams[3] for the abstract syntax. The semantics is expressed in natural language and reinforced by OCL expressions[4].

2.1 ProcessModel

This concept is used to describe (part of) a process. It is composed of process tasks (*Task*), the required products (*Product*) and the participating roles (*Role*) of these tasks (Figure 1a).

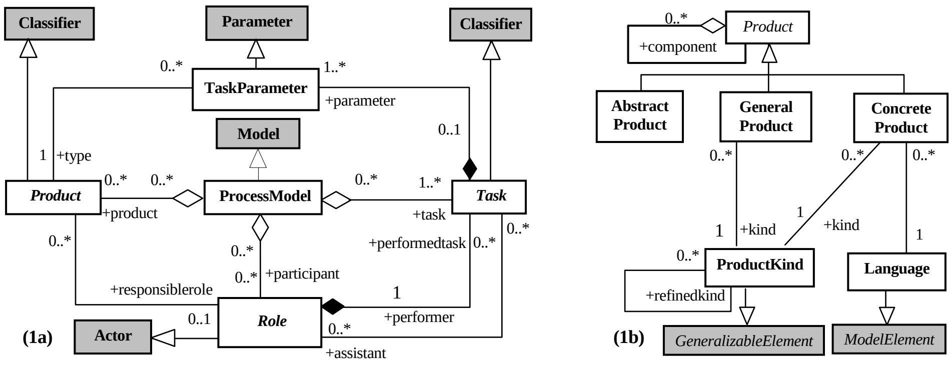

Fig. 1. Meta-model for *ProcessModel* and *Product*

Product is an artifact created, consumed or modified during the development. A *Task* is a unit of work realized to create or modify products. Necessary products for the

[1] The abstraction level of a process model reflects how detailed its content is described.

[2] In our meta-model, the part describing processes (which is inspired by SPEM 1.1) is defined separately with the part introducing process patterns. Thus, in the future, we just need to change the first part to conform our meta-model to the stable version of SPEM.

[3] Light grayed classes represent ones from UML; white classes represent new defined concepts.

[4] Due to the space constraint, we cannot present here the OCL well-formedness rules.

execution of a task are described explicitly as its parameters (*TaskParameter*). A *Role* is an abstract concept describing a set of competences of development using to realize or assist a task.

Typology of Products: Products help to specify the precise meaning of a task. An *AbstractProduct* describes a certain product without any exact meaning. A *GeneralProduct* is specified by a product kind (*ProductKind*) which can be specialized further for more specific goals. A *Concrete Product* belongs to a product kind and is represented by a concrete formalism (*Language*) (Figure 1b).

Description of Tasks: Figure 2 shows the detailed description of a task with the conditions to begin or finish (*PreCondition* and *PostCondition*) and sub-tasks.

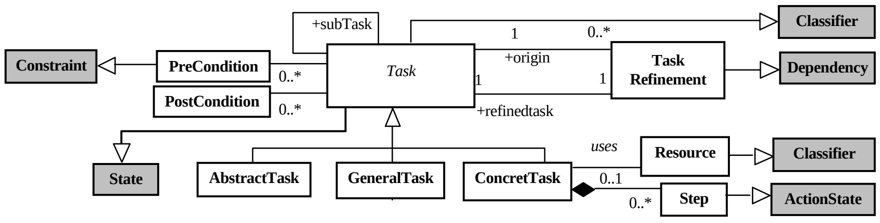

Fig. 2. Meta-model for *Task*

We propose a categorization of tasks abstraction levels based on abstraction levels of the products that they manipulate. An *Abstract Task* doesn't have any associated semantic action and is unexecutable. A *General Task* creates or modifies one or several general products. It has an associated semantic action but is not ready to be executed because the meaning of its actions depends on incompletely specified products. A *Concrete Task* creates or modifies concrete products. It can be decomposed into elementary actions (*Step*) that have a precise semantics. Actions of a concrete task are described completely in terms of resources (*Resource*) used to accomplish it. A concrete task is therefore ready to be executed.

If a task works on several products having different abstraction levels, the abstraction level of the task is deduced from the highest abstraction level of its products. The relation *«has subTask»* permits to decompose a task into sub-tasks which have to be realized together to accomplish the parent task. We also highlight the dependencies between tasks at different abstraction levels (*«TaskRefinement»*). This relation allows refining a task to obtain more specific tasks by specifying more details on its semantic action and its manipulated products.

2.2 ProcessPattern

A *Process Pattern* captures a *Process Model* that can be reused to resolve a recurrent process development *Problem* in a given *Context* (Figure 3). A *Problem* expresses the intention of a pattern and can be associated to a development task through the relation *«is applied for»*. It is possible to have several process patterns addressing the same problem. A *Process Model* represents the solution of a process pattern. A *Context* characterizes conditions in which a process pattern can be applied (*Initiation*), results that must be achieved after applying the pattern (*Resulting*) and situation recommended for reusing it (*Reuse Situation*).

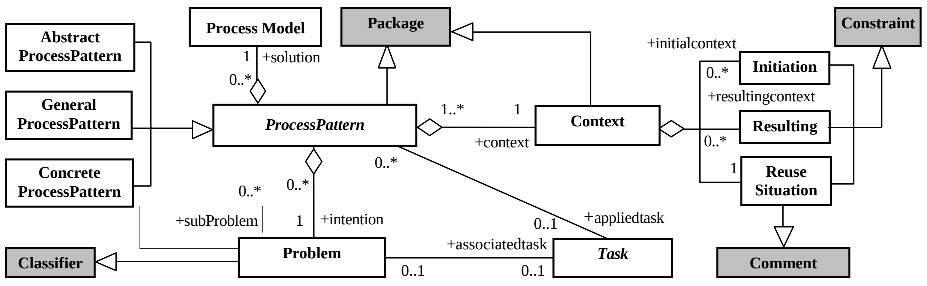

Fig. 3. Meta-model for *ProcessPattern*

We distinguish several types of process patterns according to abstraction level of their captured process model. An *AbstractProcessPattern* captures a recurrent generic structure for modeling or organizing processes. A *GeneralProcessPattern* captures a process model that can be applied to realize a general task and a *ConcreteProcessPattern* captures a process model that can be applied to realize a concrete task.

Our meta-model reflects four important relations between process patterns: *Sequence, Use, PatternRefinement* and *PatternAlternative* (Figure 4).

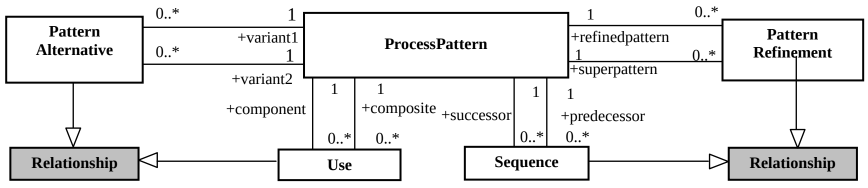

Fig. 4. Process Patterns Relationships

A *Sequence* relationship links two patterns if the predecessor produces all products required by the successor. A pattern "*uses*" another if the latter can be applied for one (or more) task in the solution of the composite pattern. A process pattern is *"refined"* from another if its associated task has the relation "*TaskRefinement*" with the associated task of the super pattern. Two process patterns are related by the relationship *PatternAlternative* if they solve the same problem with different solutions.

3 Example of process patterns

In this example, we will show how a process pattern is represented and how it can be applied in process modeling.

Fagan Inspection Process [1] is one of well-known processes used for detecting defects of software products. This process can be applied on different types of products (e.g. requirements, design, code test plans/cases and user documentation), therefore we define the general pattern "FaganInspectionPattern" to capture it[5] (Figure 5a).

[5] For the sake of simplicity, here we just represent the principal elements of this process.

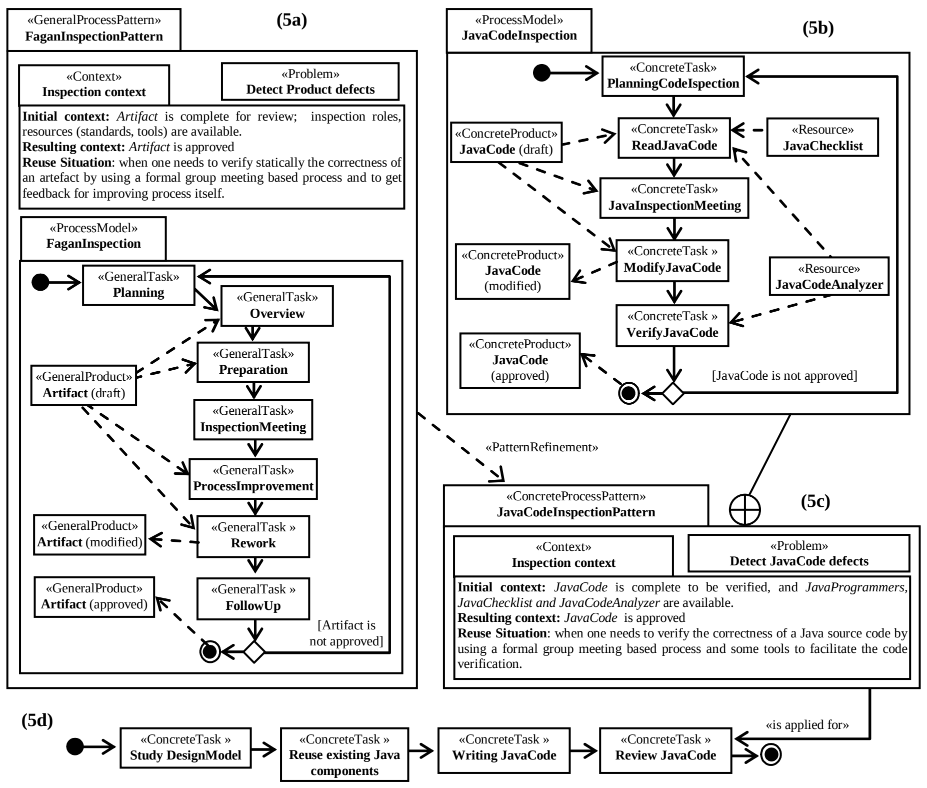
(5a)
«GeneralProcessPattern» FaganInspectionPattern
«Context» Inspection context
«Problem» Detect Product defects
Initial context: Artifact is complete for review; inspection roles, resources (standards, tools) are available.
Resulting context: Artifact is approved
Reuse Situation: when one needs to verify statically the correctness of an artefact by using a formal group meeting based process and to get feedback for improving process itself.
«ProcessModel» FaganInspection
«GeneralTask» Planning
«GeneralTask» Overview
«GeneralTask» Preparation
«GeneralTask» InspectionMeeting
«GeneralTask» ProcessImprovement
«GeneralTask» Rework
«GeneralTask» FollowUp
«GeneralProduct» Artifact (draft)
«GeneralProduct» Artifact (modified)
«GeneralProduct» Artifact (approved)
[Artifact is not approved]
(5b)
«ProcessModel» JavaCodeInspection
«ConcreteTask» PlanningCodeIspection
«ConcreteTask» ReadJavaCode
«ConcreteTask» JavaInspectionMeeting
«ConcreteTask» ModifyJavaCode
«ConcreteTask» VerifyJavaCode
«ConcreteProduct» JavaCode (draft)
«ConcreteProduct» JavaCode (modified)
«ConcreteProduct» JavaCode (approved)
«Resource» JavaChecklist
«Resource» JavaCodeAnalyzer
[JavaCode is not approved]
«PatternRefinement»
(5c)
«ConcreteProcessPattern» JavaCodeInspectionPattern
«Context» Inspection context
«Problem» Detect JavaCode defects
Initial context: JavaCode is complete to be verified, and JavaProgrammers, JavaChecklist and JavaCodeAnalyzer are available.
Resulting context: JavaCode is approved
Reuse Situation: when one needs to verify the correctness of a Java source code by using a formal group meeting based process and some tools to facilitate the code verification.
(5d)
«ConcreteTask» Study DesignModel
«ConcreteTask» Reuse existing Java components
«ConcreteTask» Writing JavaCode
«ConcreteTask» Review JavaCode
«is applied for»

Fig. 5. Patterns for software artifacts inspection and their applications

When a process designer wants to define a group meeting-based process to review a Java source code, he can reuse the FaganInspection process by applying the pattern in Figure 5. To do so, he instantiates the captured process model of the pattern, then refines the general tasks by concrete tasks with their necessary resources (e.g. the concrete task *ReadJavaCode* which is refined from the general task *Preparation* uses the *JavaChecklist* and *JavaCodeAnalyser* as particular resources), and replaces general products by concrete products (e.g. the general product *Artifact* is replaced by the concrete product *JavaCode*). He can also modify the process model by choosing just the pertinent tasks to be refined (e.g. the task *Overview* and *ProcessImprovement* are omitted). Figure 5b shows the modified process model. After certain successfully applications of process described in Figure 5b, the process designer can capture this process model to define a new concrete process pattern "JavaCodeInspection" (Figure 5c). From now on, this concrete pattern can be applied over and over again in any process having a task of JavaCode verification. For example, Figure 5d shows a process for developing Java programs. In this process, the designer want to reuse exactly the process model captured in "JavaCodeInspectionPattern", thus he applies this concrete pattern directly for the task "ReviewJavaCode" by making a link from the task to the pattern. This reference mechanism is permitted in process modeling thanks to the relationship *"is applied for"* described in our meta-model (c.f. Figure 3).

4 Related works and Conclusion

In regard to process patterns formalization, works are still modest. Gnatz et al. proposed a process framework[2] to describe software processes and introduced the notion of process pattern as a modular way for the documentation of development knowledge. Hagen et al. developed a the UML-based language PROPEL[3] to describe explicitly process patterns and their relationships. The process modeling language PROMENADE[7] defines the process pattern concept as a parameterized process template and proposes a set of high-operators for manipulating process models. In the field of method engineering, Rolland et al.[6] have developed a meta-model to capture "way of working" – a closed concept to process pattern. All of these meta-models are not based on a standardized process meta-model and just concentrate on general software process patterns, thus cannot cover other kinds and abstraction levels of process patterns. Compared with the above works, our approach aims at integrating process pattern to a widely accepted process meta-model, i.e. SPEM of OMG. We introduced a more general definition which covers a large variety of process patterns. Especially, our meta-model allows representing process models and process patterns at different abstraction levels. Furthermore, besides a semi-formal description, a set of OCL constraints are also defined to provide a more rigorous semantics for the proposed meta-model. In the first draft submission for SPEM 2.0[5], the process pattern concept is proposed as a Capability Pattern which describes reusable clusters of activities in common process areas. However, the internal structure of this concept as well as relationships between patterns is not defined. Moreover, in contrast to our approach, SPEM 2.0 does not pay attention to the different abstraction levels of patterns. Our work thus still will be useful on integrating and adapting to SPEM 2.0 when it is stabilized.

References

1. Fagan, M.E.: Advances in Software Inspections. IEEE Transactions on Software Engineering, Vol. SE-12, No. 7, Page 744-751(1986)
2. Gnatz, M. Marschall, F., Popp, G., Rausch, A., Schwerin, W.: The Living Software Development Process. Journal Software Quality Professional, Volume 5, Issue 3 (2003)
3. Hagen, M., Gruhn, V.: Process Patterns - a Means to Describe Processes in a Flexible Way. ProSim04, Edinburgh, United Kingdom (2004)
4. OMG: Software Process Engineering Metamodel (SPEM v1.1) Specification: OMG Document formal/05-01-06 (2005)
5. OMG: SPEM 2.0 Draft Adopted Specification. http://www.omg.org/docs/ad/05-06-05.pdf
6. Ralyté J., Rolland C.: An Assembly Process Model for Method Engineering. CAISE'01, Interlaken, Switzerland (2001)
7. Ribó J. M; Franch X.: Supporting Process Reuse in PROMENADE. Research Report LSI-02-14-R, Dept. LSI, Politechnical University of Catalonia (2002)
8. Tran H.N., Coulette B., Dong T.B.T. Towards a better understanding of Process Patterns. SERP 2005 (2005)
9. Tran H.N., Coulette B., Dong T.B.T. A Classification of Process Patterns. SWDC-REK 2005. Reykjavik, Island (2005)