Modeling Process Patterns and Their Application

Hanh Nhi Tran, Bernard Coulette,
MACAO-IRIT, University of Toulouse 2
5 allées A. Machado, 31058 Toulouse, France
{tran,coulette}@univ-tlse2.fr

Bich Thuy Dong
University of Natural Sciences
227 Nguyen Van Cu, Q5 HoChiMinh, Vietnam
thuy@hcmuns.edu.vn

Abstract

Process pattern is an emerging approach for process reuse. Representing process models based on process patterns to explicit process solutions and factor recurrent process constituents is useful for process understanding as well as process modeling. This subject, however, is still a challenge for the software process technology community. In this paper, we present a UML-based process meta-model that allows explicit representation of process patterns in process models. The novel of our proposition is to enable the application of different process knowledge kinds not only for building but also for improving process models.

1. Introduction

The idea behind process patterns is to capture and reuse process knowledge for recurrent development problems. Inspired by the success of software product patterns, the process community aims at applying process patterns firstly to propagate process best-practices, secondly to reduce process modeling time. While the first objective can be achieved even with informally represented process patterns as in [1][3][6] [9][14][15], the latter requires an adequate formalization of process patterns to enable process modeling tools that supports patterns reuse.

By 'adequate formalization of process patterns', we mean that: (1) the process pattern concept must be defined as a model element together with suitable modeling constructs to allow an explicit representation of process patterns applications in process models; (2) the proposed constructs should reflect the variety of process patterns and their applications; (3) a formally defined syntax and semantics should be provided, so that the proposed constructs are computable.

There are several attempts to formalize process patterns [4][7][8][12][17], but none of them can fulfill all the above requirements. Some works propose just notations for process patterns [17], others [7][8] define the applications of process patterns in too simplistic ways – i.e. exact reuse. Only the formalisms in [4][12] provide rather complete and flexible constructs to representing process patterns applications with customization. All of these works focus only on capturing and reusing recurrent development tasks (e.g. *TechnicalReview*[1]) as building blocks for constructing new processes.

However, we argue that process patterns could capture more abstract process knowledge that can be reused as templates structure for modeling processes (e.g. the patterns proposed in [10] define pragmatic and abstract building blocks for modeling recurrent situations in collaborative works; other examples can also be found in [14][15][21]). This kind of process patterns can be useful for both process construction and process improvement, but it is currently ignored in process pattern formalization and application.

We present in this paper another process patterns formalization solution that helps to overcome the shortcomings of the existing approaches. We propose a more general definition covering different kinds of process patterns, and identify potential applications which would be highly beneficial to process modeling. These propositions are rigorously defined in a UML-based process meta-model to help us satisfy all the requirements (1), (2) and (3).

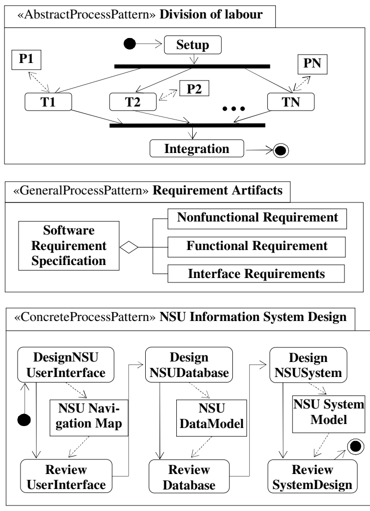
2. Process Patterns in Our Eyes

The notion of process pattern has been used with terminological and conceptual confusion. Thus, we want to clarify what is exactly a process pattern, and in what ways process modeling can benefit from it.

2.1. Process Pattern Definition

As we discussed in the introduction, currently, the process community considers only reusable development tasks as process patterns and ignores patterns capturing more abstract process knowledge. We think that such a limited view leads to an inadequate definition and formalization of process patterns. In our opinion, process patterns are *patterns for modeling processes*. More precisely, a process pattern captures a (fragment of) *process model* representing a solution that can be applied in a given context to resolve a recurrent problem of process modeling. Such problems can span from defining the content of a specific process element (i.e. task, role or product) to describing general process organizations, relations between process elements. Consequently, solutions provided by process patterns could be represented at different levels of abstraction.

We classify process patterns into three types according to the abstraction level of their captured process knowledge [19]: (a) *Abstract process patterns* capture generic process structures which are applicable to any process, such as templates for modeling different process workflows[21]; (b) *General process patterns* capture process elements of proven development methods or techniques that can be applied for different application domains, such as the inspection technique for verifying software artifacts [2]; (c) *Concrete process patterns* capture concrete hands-on experiences whiwh are applicable to a particular process type for producing a specific kind of product, such as a process for designing information systems adopted by a specific team. Fig. 1 shows an example of these three types of process patterns in our definition.



(1a) a generic process structure enabling parallel collaborative works of N task T_i on N products P_i (adapted from [10]).

(1b) a model defining the common artefacts of requirement processes. The representation concrete of these products is not specified.

(1c) the Design Process of the NSU development teams having 1 designer and 1 tester. In this process, the tasks are specified into detailed steps; the products are specified with concrete templates.

Figure 1. Examples of three types of process patterns

2.2. Process Pattern Applications

After examining many issues in process modeling [16], we distinguished two types of problems: *defining a process element* and *organizing process elements*.

The first problem concerns the question of *how to define the details of a process element*. A process pattern addressing such a question captures a model describing the content of the required process element. For example, the pattern in Fig.1b defines the content of the product *Software Requirement Specification*; or the pattern in Fig.1c describes the steps of the task *Information System Design*. This type of problem is ordinary but frequent. The question for the second problem is *how to organize a group of process elements* to satisfy a special situation or certain constraints. Such problems focus on how to achieve high quality process models and effective executions of those models. A pattern dealing with this type of problem provides a model for structuring a (fragment of) process, i.e. for establishing certain relationships between process elements. For instance, in [10] Lonchamp described the pattern *Division of labour* as a solution for modeling collaborative tasks performed by several actors on subparts of a document to build the whole document in a parallel way (Fig.1a). Although this type of problem is currently overlooked in process patterns formalization, it exists and was discussed in many works [14][15][21].

Based on the two above types of process modeling problems, we propose two main uses of process patterns as follows.

First, process patterns can be used to *generate the content of a process element*. Often, in the process definition phase, process designers need to elaborate a detail description for a process element. In such cases, process patterns can be used as building blocks to construct new processes quickly. This is the most popular use, and to our knowledge, it is the unique use of process patterns proposed by the process community for process modeling.

Secondly, process patterns can be used *to (re)organize a group of existing process elements*. For process elements organizing problems, another way to apply process patterns is required. The general situation in such cases is that process designers want to apply new relations on a group of existing process elements, or to add new information to that group. Process patterns for such problems can be used as process structure templates to (re)organize processes. A similar idea has been discussed for design patterns on *models refactoring* [5], however, this kind of process pattern reuse has not been exploited yet in practical process modeling.

3. A Meta-Model for Process Patterns

In this section, we present a meta-model formally defining the propositions about process patterns presented in the previous section. This meta-model provides the necessary concepts to describe: (1) a process pattern and its internal components; (2) the reuse of process patterns in process models. Our meta-model is developed as an extension of UML 2.0 [13] to profit from the standardization and expressiveness of this language as well as its modeling tools.

3.1. Representing Process Patterns

A process pattern (*ProcessPattern*) is composed of a problem (*PatternProblem*), a solution (*ProcessModel*) and a context (*PatternContext*) (Fig. 2).

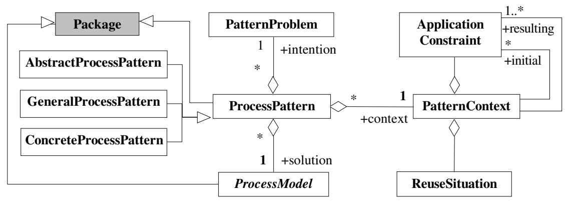


Figure 2. Meta-model for Process Pattern

A problem describes a recurrent issue of process modeling and expresses the intention of a pattern. A context characterizes required conditions concerning the application of a pattern. The *ProcessModel* concept is used to describe a (fragment of) process. Inspired by SPEM 1.1[11], we describe a process with three primary process elements: task, role and product. A *Task* is a unit of controlled work realized by a *Role* to create or modify *Products* (Fig.4). As we distinguish three types of process patterns (c.f. section 2.1), we need to represent their solutions (i.e. captured process models) at different levels of abstraction. Thus, we also define process elements that compose process models at three abstraction levels: abstract, general and concrete.

An *abstract element* is used to refer to an unspecified element without defining its precise semantic (e.g. a task *T*, a product *P*). A *general element* is partly defined but can be specified further (e.g. a *code* can be *functional* or *object-oriented* with distinguished characteristics; a *review task* can work on *code* or *design document* with different details). A *concrete element* is completely specified for a specific purpose (e.g. a *JavaCodeReview* task). For simplicity, here we just represent the process elements by their concrete meta-classes, e.g. *AbstractTask* (c.f. [20] for the definition of the hierarchy of process elements).

A process model is represented in the form of a diagram composed of a set of process elements (*Node*) and relations among them (*Edge*). We propose two types of models, *structural* and *behavioral*, to represent respectively static and dynamic aspects of processes. Our structural model is based on the UML class diagram and the behavioral model develops from the UML activity diagram. In contrast UML, we use explicit elements for representing these diagrams in order to permit defining operations for manipulating process models at meta-model level.

A structural model reflects the composition of a process element or organizational relations between process elements (Fig.3). Structural edges shows the static relations defined between two process elements.

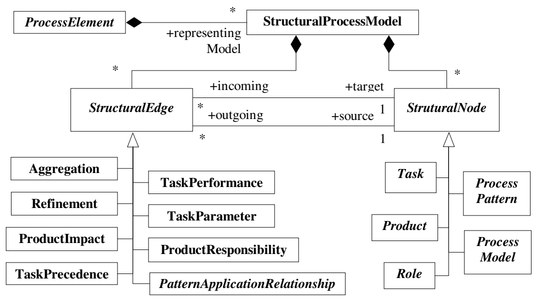


Figure 3. Syntax of a structural model

Fig.4 shows three main process relations: *TaskParameter* allows representing the manipulated products (input, output) of a task; *TaskPerformance* specifies the roles participating in the execution of a task as performer or assistant; *ProductResponsibility* links a role to the products that it is responsible for.

Besides these relations, we also define the relations *Aggregation* and *Refinement* between two process elements of the same type to represent the decomposition and the specialization of process elements, respectively. The dependency *TaskPrecedence* expresses the constraint on the execution order of two tasks while *ProductImpact* shows the dependence of a product on another.

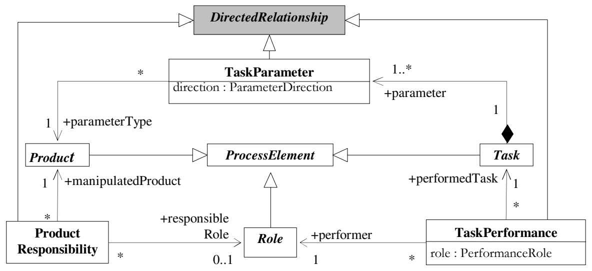


Figure 4. Relations between process elements

We use behavioral models to represent the workflow of actions in a task or the task workflow of a process. We base our behavioral model on the UML 2.0 *Activity* concept[13], which allows representing nested and reusable behaviors. More precisely, to compose a behavioral model in our system, we reuse the syntax abstract of UML activity diagram, then align the concepts of our meta-model to the corresponding UML concepts (the grey elements in Fig.5).

Being a specialization of UML *Activity*, a *Task* can be defined by a workflow of *Steps* (a specialization of UML *Action*), which in turn can invoke other *Tasks* (thanks to the subtype *CallBehaviorAction*), etc. The *ControlFlows* established between *ExecutableNodes* (*Task* or *Step*) and *ControlNodes* express the sequence of actions. The *Objectflows* link *ObjectNodes* (i.e. a *Product* in our meta-model) to *Tasks* or *Steps* to express the input and output of actions.

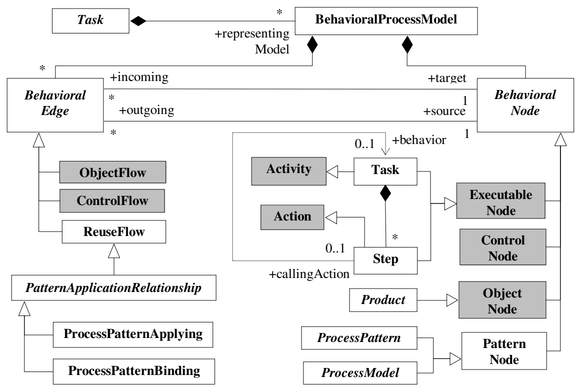


Figure 5. Syntax of a behavioral model

To allow the appearance of process pattern applications in process models, first we define the concepts *ProcessPattern* and *ProcessModel* as specializations of both *StructuralNode* and *BehavioralNode*. Then we define the new concept *PatternApplicationRelationship* to represent the applications of process patterns in the form of the relations between *ProcessPatterns* and *ProcessElements* or *ProcessModels*. The process pattern application relations can play the role of both *StructuralEdge* and *BehavioralEdge* (c.f. Fig.3, Fig.5).

3.2. Representing Process Patterns Applications

One of the key points of our meta-model is representing process patterns at different levels of abstractions and enabling the process patterns reuse by abstraction mechanism, i.e. we define process patterns as parameterized element [18]. This helps to promote the applications of process patterns. Then, we define two relations, *PatternBinding* and *PatternApplying* (c.f. Fig.5), to represent respectively the applications of process patterns proposed in section 2.2.

In addition to the abstract syntax, we propose a concrete syntax based on UML notations to represent the new concepts. A set of OCL constraints was also defined to formalize the semantic of the process patterns application relations and facilitate a potential process model transformation in the future. Due to the space limit, we cannot present these constraints here.

3.2.1. Process Pattern as a Template. Fig. 6a shows the definition of a process pattern as a parameterized model element. A *ProcessPattern* may have a signature that specifies a set of formal parameters for the pattern. A *ProcessPatternParameter* refers to an element that may be substituted in a pattern application relation.

3.2.2. ProcessPatternBinding Relation. This relation represents a relationship between a process element (source) and a process pattern (target). This relation allows reusing a process pattern as a template to generate the content of a bound element (Fig.6b). The presence of a *ProcessPatternBinding* relation implies that the contents of the model captured in the target process pattern will be copied into the bound process element, substituting elements exposed as formal template parameters by the corresponding elements specified as actual parameters in this binding. Because this relation is used to represent the generation of process elements' content from patterns, it is not required that actual parameters of a *binding* must be existing elements.

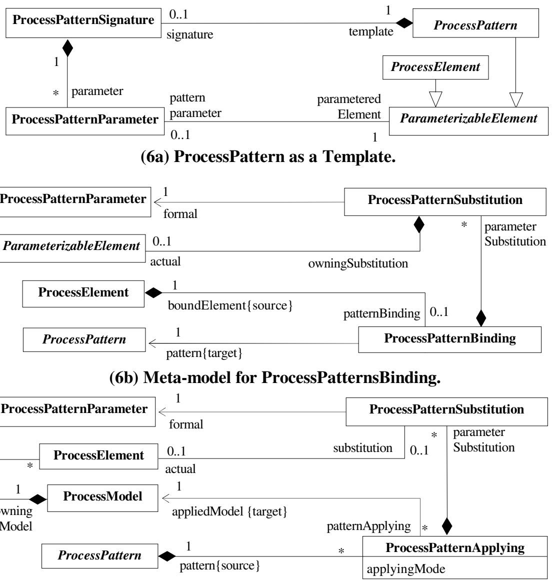


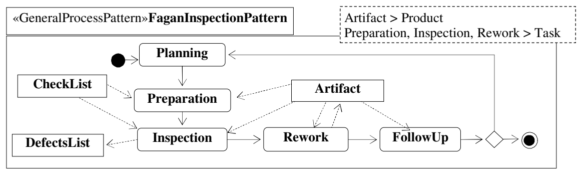
Figure 6. Meta-model for Process Patterns Application

3.2.3.ProcessPatternApplying Relation. This relation represents a relationship between a source process pattern and a group of target process model containing several process elements. This relation allows reusing a process pattern to reorganize or enrich a fragment of process model (Fig.6c). A *ProcessPatternApplying* relation maps each pattern's element exposed as formal parameters to a specific existing element in the target model. The presence of a *ProcessPatternApplying* relation implies that the structure of the process model captured in a source pattern will be applied to the target model. The specific way this application is realized depends on the value of the attribute *applyingMode* of the relation. If the mode c*hange* is chosen, all relations (if exist) between target model's elements will be removed, only new relations and dependencies (together with associated elements) defined among the elements of the source pattern will be applied to the corresponding elements in the target model. If the mode *replace* is chosen, the content of the source pattern will override the existing one of the target model. More precisely, in the case the target model contains elements and relations other than the source pattern, these existing elements will be conserved in the resulting model only if they do not conflict with the ones defined in the source pattern, otherwise they will be replaced by the pattern's elements. The mode *extend* indicates that the content of the source pattern will be merged with the existing one of the target model. However, the existing elements are intact and the new ones will be ignored if there are conflicts [1] between them.

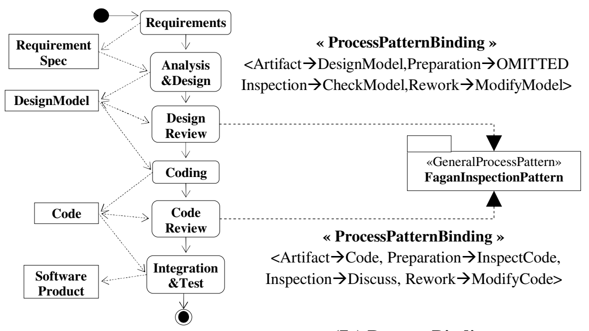
4. Illustrating Examples

We present in this section some examples to illustrate how process patterns applications are explicitly represented in process models.

First, we show in Fig.7 an example of using process patterns to generate process elements' contents through the relation *ProcessPatternBinding*.



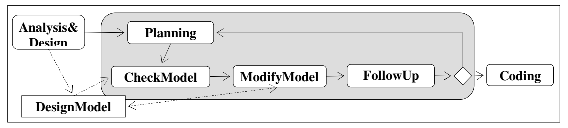
(7a) A ProcessPattern captures a reusable definition for a review



Requirements
Requirement Spec
Analysis &Design
DesignModel
Design Review
Coding
Code
Code Review
Integration &Test
Software Product
« ProcessPatternBinding »
<Artifact→DesignModel,Preparation→OMITTED
Inspection→CheckModel,Rework→ModifyModel>
«GeneralProcessPattern»
FaganInspectionPattern
« ProcessPatternBinding »
<Artifact→Code, Preparation→InspectCode,
Inspection→Discuss, Rework→ModifyCode>

(7b) Baseline Process Model with unspecified process elements

(7c) Pattern Bindings generate the contents of source elements



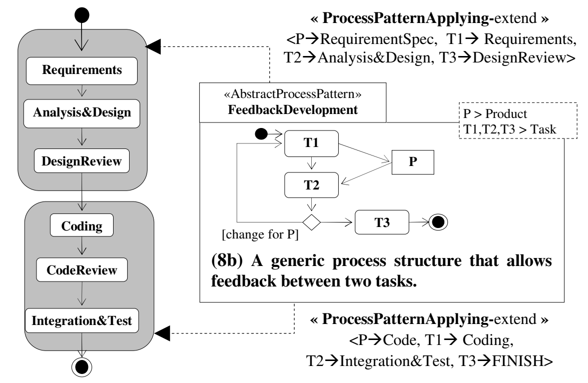
(7d) Resulting model of the binding relation on *DesginReview*

Figure 7. An example of process pattern application for generating a process element's content

Fig.7b shows the model of the NSU baseline development process which contains unspecified process elements. When refining this baseline process, the process designer decided to apply the Fagan process [2] to review different software artifacts. So he reused the process pattern *FaganInspectionPattern* (Fig.7a) to generate the content of the tasks *DesignReview* and *CodeReview*. Fig.7c shows a double use of *FaganInspectionPattern* to generate the contents of the tasks *DesignReview* and *CodeReview*. These applications are represented through the relations *ProcessPatternBinding* with the actual parameters that substitute the corresponding pattern formal parameters. The unfolded model for the task *DesignReview* after the pattern application is shown in Fig.7d.

We present in Fig.8 the use of *ProcessPatternApplying* relation for representing the application of process patterns to reorganize the structure of the example process model.

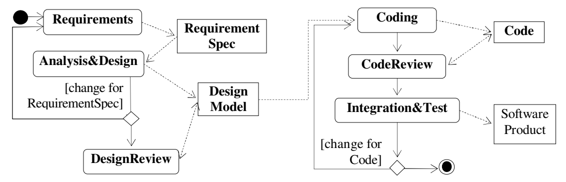
With the simple sequence lifecycle model of the baseline process (Fig.8a), it is difficult to accommodate changes after the process is underway. Thus, the process designer wants to adopt another lifecycle model that allows feedbacks. However, he must keep the original process elements of the company's baseline process which are fully specified. For that, the process designer applies the abstract pattern *FeedbackDevelopment* (Fig.8b) which captures a generic process structure representing a feedback control flow between two tasks to reorganize the existing process elements in Fig.8a.



(8b) A generic process structure that allows feedback between two tasks.

(8a) Current baseline process with a simple sequence lifecyce

(8c) ProcessPatternApplying for restructuring process model



(8d) Resulting model of the applying relation on the baseline process

Figure 8. An example of process pattern application for reorganizing process elements

First, the *FeedbackDevelopment* pattern is applied in mode *extend* to the process model containing the following process elements: *RequirementSpec*, *Requirements*, *Analysis&Design* and *DesignReview* (Figure 7b). These elements play respectively the roles of

[1] Conflicts may come from inconsistent dependencies between tasks' execution orders, products' impacts; or from the incoherent relations between tasks and manipulated products, between tasks and participants roles, etc. The detailed analysis for such conflicts is out of scope of this paper.

P, T1, T2, and T3 in the pattern. The new control flow between T2 and T1 is added to the target model (Fig.8d). The second application of the pattern *FeedbackDevelopment* to the process model containing *Code*, *Coding*, *CodeReview*, *Integration&Test* and FINISH is a merging applying. The target model in this case contains another element, the tasks *CodeReview*. This additional element is conserved in the resulting model because the new added control flow between *Coding* and *Integration&Test* does not conflict with the existing relations between *CodeReview* and the others.

5. Related Works and Conclusion

In [17] Storrle specified a presentation scheme for process patterns using UML notations, but did not define a supporting meta-model for it. Gnatz et al. introduced the notion of process pattern as a modular way for documenting development knowledge [7] but did not define the process pattern structure. Hagen et al. developed a UML-based language [8] to describe explicitly process patterns and their interrelationships. Franch and Ribo [4] developed the language PROMENADE which defines process pattern as a process template and provides a set of relations supporting process reuse and harvesting mechanisms to manipulate process models. In the draft submission for SPEM 2.0 [12], process patterns are defined as reusable clusters of activities in common process areas. SPEM 2.0 supports the application of a pattern through the Activity Use mechanisms, which provides some predefined relationship kinds to allow the adaptation of reuse activities.

All the above works, adopt a rather limited definition for process pattern concept, i.e. just as reusable development tasks used as building blocks for constructing new processes. Thus, they only support the relation "binding" to applying process patterns in process models. The diversity of process patterns abstraction levels is ignored in these works, apart in [4] where it is implicitly distinguished. Consequently, they do not allow representing and exploiting patterns addressing the organizing process elements issues.

Compared with related works, our approach emphasizes process patterns reuse by abstraction mechanism. By distinguishing process elements at different abstraction levels, our meta-model allows representing various kinds of process knowledge encapsulated in process models. By using parameterization to define process pattern concept and its reuse relations, we promote the use of process patterns, i.e. allow using process pattern on a high abstraction level to generate or reorganize process elements on lower levels of abstraction. Especially, the two proposed relations, *binding* and *applying*, provide complementary ways to exploit process patterns for different process modeling needs, which were ignored in previous works.

To validate our work, we are now implementing the proposed meta-model as a UML profile by using the *Objecteering ProfileBuilder*. We are also defining a meta-process describing a systematic way to build process models by reusing process patterns. This meta-process will be integrated into a process modeling tool supporting the proposed meta-model to facilitate the pattern-based process constructing and tailoring.

7. References

[1] S.W. Ambler, *Process Patterns: Building Large-Scale Systems Using Object Technology*, SIGS Books/Cambridge University Press, New York, 1998.

[2] M.E. Fagan, "Advances in Software Inspections". *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 7, 1986

[3] D. Firesmith, B. Henderson-Sellers, *The OPEN Process Framework. An Introduction*. Addison-Wesley, 2001.

[4] X. Franch, J. M. Ribó, "A UML-Based Approach to Enhance Reuse within Process Technology", *EWSPT 2003*, Findland, 2003.

[5] R. France, S. Ghosh, E. Song and DY. Kim, "A metamodeling approach to pattern-based model refactoring". *IEEE Software*, September 2003, pages 52-58.

[6] M. Gnatz, F. Marschall, G. Popp, A. Rausch, M. Rodenberg-Ruiz, W. Schwerin (Eds.). *Proceedings of the 1st Workshop on Software Development Process Patterns (SDPP'02), OOPSLA' 02*, 2002.

[7] M. Gnatz, F. Marschall, G. Popp, A. Rausch, W. Schwerin, "The Living Software Development Process", *Journal Software Quality Professional*, Volume 5, Issue 3, 2003.

[8] M. Hagen, V. Gruhn, "Process Patterns - a Means to Describe Processes in a Flexible Way". *ProSim04*, Scotland, 2004.

[9] I. Jacobson, G. Booch, J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.

[10] J. Lonchamp, "Process Model Patterns for Collaborative Work", *Telecoop'98*, Austria. 1998.

[11] OMG, *Software Process Engineering Metamodel v1.1*, 2005.

[12] OMG, *SPEM 2.0 Final Adopted Specification*, 2007.

[13] OMG, *UML2.0 Superstructure Final Adopted Specification*, 2003

[14] V. Pavlov and D. Malenko, "Mining MSF for Process Patterns: a SPEM-based Approach", *Viking PLoP 2004*, Uppsala , 2004.

[15] M. Penker, H.E. Eriksson, *Business Modeling With UML: Business Patterns at Work*. John Wiley & Sons, 2000.

[16] D. E. Perry, "Practical Issues in Process Reuse". *10th Int'l. Software Process Workshop*, France, 1996.

[17] H. Storrle, "Describing Process Patterns with UML", *EWSPT2001*

[18] DT. Tran, HN. Tran, B. Coulette, X. Crégut, BT. Dong, "Topological properties for characterizing well-formedness of Process Components", *Software Process: Improvement and Practice*, Wiley Interscience, V.10 N. 2, 2005, p. 217-247.

[19] HN. Tran, B. Coulette, BT. Dong, "A classification of Process Patterns". *SWDC-REK 2005*. Reykjavik, Island , 2005.

[20] HN. Tran, B. Coulette, BT. Dong, "A UML based process meta-model integrating a rigorous process patterns definition", *PROFES 2006*, Amsterdam, 2006.

[21] W.M.P. Van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros, "Workflow Patterns", *Distributed and Parallel Databases*, 14(3), pages 5-51, July, 2003.