Reflections on How to Improve Software Process Patterns Capitalization and Reuse

Nahla Jlaiel¹, Mohamed Ben Ahmed²

¹ RIADI Laboratory, National School of Computer Science, University of La Manouba, La Manouba 2010, Tunisia
² Nahla.Jlaiel@riadi.rnu.tn
³ Mohamed.Benahmed@riadi.rnu.tn

Abstract—Process Patterns is an emergent approach and a valuable means for capitalization, reuse and management of process experiences and best practices. In the context of software engineering process, many formalisms and languages have been proposed to describe software process patterns. This multiplicity makes capitalization and/or reuse of process patterns difficult to be achieved. This paper presents a comparative study of process pattern formalisms proposed in the literature and addresses some reflections to deal with problems arising from this survey, in order to propose a new framework for process patterns’ capitalization and reuse.

Keywords—Software process patterns, process patterns’ description languages, patterns’ unification ontology, pattern warehousing, pattern reuse, pattern capitalization, pattern mining.

I. INTRODUCTION

Process patterns are widely used by the software engineering community as an excellent mechanism for communicating software development knowledge (experiences and best practices) that has proven to be effective in practice. Furthermore, process patterns are the reusable building blocks from which an organization may tailor a mature software process [1]. As a consequence, several related works have been carried out dealing with this process patterns. This paper is structured as follows. In section II, we provide an overview of the process pattern’s definition and it emergence’s context. In section III, we present a theoretical survey of several related research works found in the literature. In section IV we present our work in progress, proposing some key ideas in order to improve software process patterns’ capitalization and reuse within the software engineering community. Finally, we conclude this paper in section V.

II. SOFTWARE PROCESS PATTERNS

To define what a process pattern is, we should define what a pattern is in general. In the literature, Patterns were introduced in 1979 by Alexander [2] as an architectural concept to designate a general solution to a common problem or issue, one from which a specific solution may be derived.

Inspired by Alexander’s patterns, Beck and Cuningham [3] introduced in 1987, the concept of patterns in software engineering for the first time. Years later, this concept became widespread thanks to object-oriented design patterns published in [4].

Consequently, patterns grow to be widely used as proven solutions to recurring problems. They are generally represented by a triplet (problem, context, solution). Patterns are not restricted to a particular domain to be applied in or to emerge of. Instead, they have been developed for several domains like Architecture, Software Engineering, Organization, Pedagogy and Human Computer Interaction.

Within the software engineering community, the pattern’s concept has been increasingly recognized as an effective method to reuse knowledge and best practices gained during the whole software lifecycle. Thus, several kinds of patterns have been proposed in the literature.

According to [5], these could be classified into three main axes: first, type of knowledge that pattern focuses on, second pattern’s coverage and third development’s phase supported by the pattern. For the first axis, a pattern can be either result pattern that capitalizes specifications or implementations of a goal, or a process pattern that capitalizes specifications or implementations of an approach to be followed to achieve the result. For the second axis, a pattern can be generic or geared towards a specific area. For the third axis, patterns are classified according to lifecycle’s phase they support, e.g. analysis pattern, design pattern, implementation pattern, test pattern.

This paper focuses on the first classification’s category (result versus process pattern). In fact, result patterns are widely used in several works among which, the most significant are: analysis patterns, design patterns, architectural patterns and implementation patterns. Regarding process patterns, they still represent an emerging topic that is not fully enhanced and investigated.

Result patterns describe how the solution for the problem looks like (the solution is the result) whereas process patterns describe which process leads to the desired result (the solution is the process).

Regardless the application domain and the pattern’s type, the main benefits of patterns consist in proven and helpful knowledge representation, communication and understanding [6] as well as best practices capitalization and enhancement.
III. PROCESS PATTERNS DESCRIPTION MODELS AND LANGUAGES: A THEORETICAL SURVEY

This section deals with the study that we carried out in order to assess process patterns representation and reuse within software development communities. First, principal works found in the literature, are cited. Second, proposed evaluation criteria are described to achieve this study. Third, comparative tables are presented showing the study’s results. Finally, main open issues are addressed discussing our survey’s findings.

A. Survey’s Focus

Different works have been carried out concerning process pattern’s modeling and formalization. These works could be divided into two main categories, namely:

1) Process Patterns’ Description Models: This category regroups works that provide a terminology for process patterns description. Main process pattern description models include:
   - AMBLER (1998) [1]
   - RHODES (2000) [7]
   - GNATZ (2001) [8]
   - P-SIGMA (2001) [5]
   - STÖRRLE (2001) [9]

2) Process Patterns’ Description Languages: This category regroups works that provide a language for process patterns description. Main process pattern description languages include:
   - PROMENADE (2002) [10]
   - PLMLx (2004) [12]

Because of space limitations, we will not present details about the studied works. References are provided to give further information concerning these works.

B. Survey’s Criteria

In order to discuss an eventual improvement of process patterns capitalization, management and reuse during software development processes, we propose the following criteria to evaluate the aforementioned works.

1) Pattern’s Formalization Degree: It aims to classify a given work depending on the formalization level used for presenting pattern’s knowledge.

2) Pattern’s Context Scope: It identifies the different types of context covered by the pattern, i.e. if the pattern provides a description of initial context (preconditions) and/or resulting context (post conditions).

3) Pattern’s Coverage: It serves to identify the software development phase (s) concerned by the pattern, namely: analysis, design, implementation and testing.

4) Pattern’s Domain: It identifies the domain addressed by the pattern.

5) Pattern’s Artifact Referencing: It indicates if the described process pattern provides any reference to artifact(s) developed and / or used during pattern’s application. Pattern’s Referencing. It identifies whether a pattern maintains relationships with one or several pattern (s) and if it is the case, the relationship type.

6) Experience’s Referencing: It identifies if the process pattern refers to cases in which the pattern was applied.

7) Pattern’s Guidance Level: It identifies the assistance level provided by the pattern’s formalism by the offered guideline’s form e.g. examples of use or application, adaptation parameters, adaptation scenarios.

8) Role’s Implication: It specifies if the pattern’s formalism takes into consideration roles participating in a pattern.

9) Work Purpose: It identifies the context in which the pattern model or language appeared (i.e. if the work focuses on patterns’ formalization and management issues or other issues related to or using the concept of patterns).

10) Pattern’s Meta-modeling: It identifies if the proposed format is supported by a meta-model.

11) Pattern’s Support Tool: It identifies if the proposed pattern’s format is supported by a process pattern based prototype, environment, tool or system.

C. Survey’s Results

Table 2 and table 3 show the comparative study’s results of the presented works according to the agreed evaluation’s criteria. Table 1 presents the comparative tables’ legend.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.</td>
<td>Initial context</td>
</tr>
<tr>
<td>R.</td>
<td>Resulting context</td>
</tr>
<tr>
<td>I. and R.</td>
<td>Initial and Resulting context</td>
</tr>
<tr>
<td>S.D.</td>
<td>Software Development</td>
</tr>
<tr>
<td>H.C.I.</td>
<td>Human Computer Interaction</td>
</tr>
<tr>
<td>S.E.</td>
<td>Software Engineering</td>
</tr>
<tr>
<td>I.S.E.</td>
<td>Information System Engineering</td>
</tr>
<tr>
<td>+</td>
<td>Very well supported criterion</td>
</tr>
<tr>
<td>--</td>
<td>Not supported criterion</td>
</tr>
<tr>
<td>+-</td>
<td>Not very well supported criterion</td>
</tr>
</tbody>
</table>
**Table II**
Evaluation results for process patterns’ description models

<table>
<thead>
<tr>
<th>Criterion</th>
<th>AMBLER</th>
<th>RHODES</th>
<th>P-SIGMA</th>
<th>GNATZ</th>
<th>STÖRRLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formalization</td>
<td>Informal</td>
<td>Formal</td>
<td>Semi formal</td>
<td>Semi formal</td>
<td>Semi formal</td>
</tr>
<tr>
<td>Coverage</td>
<td>Software life cycle</td>
<td>Software life cycle</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Domain</td>
<td>S.D.</td>
<td>S.D.</td>
<td>I.S.E.</td>
<td>S.D.</td>
<td>S.D.</td>
</tr>
<tr>
<td>Artifact</td>
<td>--</td>
<td>--</td>
<td>+</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Relationships</td>
<td>--</td>
<td>--</td>
<td>Informal</td>
<td>Informal</td>
<td>Informal</td>
</tr>
<tr>
<td>Experience</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Guidance</td>
<td>+</td>
<td>+</td>
<td>--</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Roles</td>
<td>+</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>+</td>
</tr>
<tr>
<td>Purpose</td>
<td>Process patterns description</td>
<td>Process modeling</td>
<td>Process patterns management and reuse</td>
<td>Software development process management</td>
<td>Process patterns reuse</td>
</tr>
<tr>
<td>Meta-modeling</td>
<td>--</td>
<td>--</td>
<td>+</td>
<td>+-</td>
<td>--</td>
</tr>
<tr>
<td>Support tool</td>
<td>--</td>
<td>RHODES</td>
<td>AGAP</td>
<td>LISA-PRO</td>
<td>--</td>
</tr>
</tbody>
</table>

**Table III**
Evaluation results for process patterns’ description languages

<table>
<thead>
<tr>
<th>Criterion</th>
<th>PROMENADE</th>
<th>PPDL</th>
<th>PROPEL</th>
<th>PLMLx</th>
<th>UML-PP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formalization</td>
<td>Semi formal</td>
<td>Semi-formal</td>
<td>Semi formal</td>
<td>Informal</td>
<td>Formal</td>
</tr>
<tr>
<td>Coverage</td>
<td>--</td>
<td>Software life cycle</td>
<td>--</td>
<td>Design, Implementation</td>
<td>Software life cycle</td>
</tr>
<tr>
<td>Domain</td>
<td>S.D.</td>
<td>S.D.</td>
<td>S.D.</td>
<td>H.C.I.</td>
<td>S.D.</td>
</tr>
<tr>
<td>Artifact</td>
<td>--</td>
<td>+</td>
<td>--</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Relationships</td>
<td>--</td>
<td>Semi-formal</td>
<td>Semi-formal</td>
<td>Semi-formal</td>
<td>Formal</td>
</tr>
<tr>
<td>Experience</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>+</td>
<td>--</td>
</tr>
<tr>
<td>Guidance</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+-</td>
</tr>
<tr>
<td>Roles</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>--</td>
<td>+</td>
</tr>
<tr>
<td>Purpose</td>
<td>Process modeling and reuse</td>
<td>Process patterns’ description and management</td>
<td>Process patterns management</td>
<td>Pattern languages description</td>
<td>Software process modeling</td>
</tr>
<tr>
<td>Meta-modeling</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>--</td>
<td>+</td>
</tr>
<tr>
<td>Support tool</td>
<td>--</td>
<td>--</td>
<td>Process Pattern Workbench</td>
<td>PLML PET</td>
<td>PATPRO-MOD</td>
</tr>
</tbody>
</table>
D. Survey Results’ Discussion

The theoretical study that we carried out reveals the following main issues:

1) Lack of Formalization: RHODES (using PBOOL [14]) and UML-PP (using OCL 2.0 constraints [15]) are the only works that tried to formalize the process patterns’ description. The formalization of relevant parts in a process pattern description (e.g. problem, context, solution and patterns’ relationships) will, indeed, improve its identification, selection and consequently its reuse. We do not reject informal (texts in natural language) or semi formal (e.g. UML diagrams) description but we encourage the three description levels.

2) Weak Pattern’s Consideration and Apprehension: Among the studied works, AMBLER, P-SIGMA, STÖRRLE, PPDL, PROPEL and PLMLx are the only ones that are dedicated to pattern’s matters i.e. they directly and explicitly deal with process patterns. The other ones use this concept to tackle other issues related to processes.

3) Lack of Process Pattern Meta Modeling: Among the studied works, few of them are supported by a process pattern meta model, namely: PROMENADE, P-SIGMA, PPDL, PROPEL and UML-PP. GNATZ does not emphasize on process patterns as it principally focuses on process meta modeling using the pattern’s concept.

4) Guidance Insufficiency: Only four works (GNATZ, STÖRRLE, PROPEL and PLMLx) accord importance and support means for pattern comprehension, application and reuse by giving examples, remarks, guidelines, application constraints, adaptation parameters, and known uses.

5) Lack of Software Life Cycle Coverage: Only four works consider the importance of patterns during the whole software life cycle, namely: AMBLER and RHODES among the process pattern description models and PPDL as well as UML-PP among the process pattern description languages (cf. Table 2 and Table 3). We assume that patterns be of different types covering the different phases, steps and activities of a software development process. This principle aims to provide a pattern support for the whole software life cycle.

6) Lack of Terminological Consent: With the exception of the pattern’s triplet (problem, context and solution) for the majority’s works, most of them suffer from synonymy’s problem which refers to a case where different terms are used to refer to the same meaning as well as polysemy’s problem which refers to a case where a same term is used for multiple meanings. For examples, to refer to the pattern’s intention (goal), STÖRRLE, GNATZ and PROMENADE use the term “intent”, PLMLx and P-SIGMA use the term “force”. Although, in RHODES, the term “intention” refers to the “problem” resolved by the pattern. Furthermore, to describe the pattern’s context, some works (i.e. AMBLER, PPDL, PROPEL, UML-PP) consider both “initial context” and “resulting context” (for PROMENADE, “result context”), while other works use “context” to refer to the initial context and “consequences” (GNATZ) or “resulting context” (PLMLx) for the resulting context, others use only “context” and do not distinguish between the two types (i.e. RHODES).

7) Lack of Architectural Consent: In fact, different process pattern description formats have been proposed; each one has its proper architecture that best fit the model designer’s needs. Consequently, some works added new facets and included new concepts in their models such as roles implied in the pattern (or participant) and artifacts used or produced by the pattern, pattern management parameters, and many other features.

8) Missing Process Patterns’ Support Approaches and Tools: Even though, some works as P-SIGMA, PROPEL, PLMLx and UML-PP deal with this issue, some important challenges still remain such as: unified pattern’s description, pattern’s semantic annotation, pattern’s classification, search and mining. Indeed, the majority of the studied works provide process support methods and tools based on patterns (e.g. RHODES, GNATZ, cf. Table 2) and do not focus entirely on patterns’ support improvement.

As a conclusion, we notice that different heterogeneity’s levels emerged from this study, namely:

- The architectural level which refers to patterns’ structure.
- The terminological level which refers to terms used as labels to describe process patterns.
- The knowledge level which refers to the different facets (e.g. guidance, roles, artifacts, management, organization, evaluation) that are considered by the process patterns.

These levels imply another one which is the “semantic level” due to different interpretations and considerations of the process pattern concept.

IV. WORK IN PROGRESS

To overcome these deficiencies and to benefit from large and different process patterns’ collections, unification and mediation efforts are needed.

In fact, we think that if we make abstraction of the above mentioned disparities, we could learn more from different process patterns and as a consequence capitalize and reuse more knowledge.

To reach this goal, we are now focusing on the construction of a meta pattern’s ontology providing two unification levels, as shown in Fig. 1:

A. Architectural Unification

The architectural unification level is ensured by means of an Architectural Core (i.e. a native pattern) which contains a set of normalized Terms ($N_T$) concerning common concepts supported by the studied works. Each $N_T$ is annotated by a set of terms matching those used in the studied works.

In addition to the Core, we include Derived Architectures (i.e. derived patterns) containing specific concepts used by the different proposed works. Furthermore, Shared concepts are added corresponding to concepts that are supported by some
works but not all of them. The whole is semantically and/or hierarchically linked to form an architectural mediation graph.

This unification aims to normalize the description of any given process pattern so as to better reuse it later.

**B. Semantic Unification**

This level concerns the content of different process patterns’ fields. This unification consists in a text mining process, extracting terms and/or concepts that are most representative for the different patterns’ fields’ contents. These latter (terms, concepts) are weighted according to their occurrence number in the content and then, sorted according to their weight representing thus, a semantic annotation for the concerned features. The fields’ labels might also be enriched by synonyms extracted from the WordNet’s ontology, contributing thus, to the enrichment of these semantic annotations.

The objective of our work is to provide a framework for software process patterns’ capitalization and reuse. So, to achieve this, we adopt a Process Pattern Warehousing and Pattern Mining approach to well manage best practices and process implementation traces that are embedded in process patterns.

Process Pattern Warehousing consists in the integration of different process patterns’ collections in a Process Pattern Warehouse via a unification schema. This unification is performed by the ontology described above, consisting in concepts and their semantic annotations, relations, as well as the respective unification axioms.

On the other hand, Process Patterns’ Mining consists in the reasoning process on the unified patterns in the warehouse based on the proposed ontology. Consequently, we could better search similar patterns for a given problem by clustering them according to their similarity’s level to the problem. Hence, the process patterns’ mining process will improve process pattern’s capitalization and process reuse quality.

As a consequence to these two unification levels, different patterns would be capitalized in a unique and general format in the warehouse, used then, for mining and finally if necessary, converted to the desired format for reuse during the whole software development process.

**V. CONCLUSION**

Although a considerable number of works focused on process patterns, some important challenges for the research community still remain. This paper provides some reflections on issues arising from the study that we carried out concerning process patterns’ description models and languages. These reflections consist in the need of software process patterns’ capitalization to improve patterns’ reuse during software development processes. To do this, a meta pattern’s ontology is being constructed providing an architectural and semantic mediation for different process pattern descriptions.

---

**Fig. 1** Proposed approach for process patterns capitalization and reuse
REFERENCES


