Knowledge Representation in Pattern Management

Pankaj Kamthan
Concordia University, Canada

Hsueh-Ieng Pai
Concordia University, Canada

INTRODUCTION

The reliance on past experience and expertise is critical to any development. Patterns are a reusable form of knowledge gained by experts in solving problems that occur repeatedly in a domain. The concept of a pattern was introduced by Christopher Alexander in the 1970s (Alexander, 1979).

Patterns have found widespread use since their inception. Applications of patterns originated in the civil engineering and urban planning domains as an approach to design buildings, roadways, and towns (Alexander, 1979). Since then, patterns have been applied to various areas including use cases (Adolph, Bramble, Cockburn, & Pols, 2002), software design (Gamma, Helm, Johnson, & Vlissides, 1995), human-computer interaction (Borchers, 2001), electronic business (Adams, Koushik, Vasudeva, & Galambos, 2001), and configuration management (Berczuk & Appleton, 2003), to name a few in computing.

As the number of patterns grows and its user community broadens, the need for an effective management of patterns arises. If not addressed, patterns may, for example, fail to communicate their purpose to the user community, could be misused, or be virtually inaccessible when called upon. This can adversely affect further acceptance and evolution of patterns as means for solving frequently occurring problems.

This article discusses issues in pattern management as they relate to knowledge representation (KR). Although deemed important (Vlissides, 1998), there has been little attention in this area and one of the goals of this work is to fill that void. The article is organized as follows. We first provide a brief background on patterns within the context of pattern management. Next, requirements for representing patterns are given and different ways of representing patterns, along with an analysis of each approach, are discussed in detail. This is followed by an outline of some future directions and trends that representation of pattern knowledge is likely to take. Finally, we present concluding remarks.

BACKGROUND

Patterns are important, as they are time- and technology-independent abstractions that suggest proven, reusable solutions to common problems in a domain. In a software context, patterns represent knowledge and experience that underlies many redesigns and reengineering efforts of engineers that have struggled to achieve greater reuse and flexibility in their software (Devedzic, 2002).

Patterns are related to, but different from other forms of knowledge such as principles, guidelines, and frameworks: Patterns are at a lower level of abstraction with respect to principles, and in fact, patterns often rely on principles for the quality and acceptance of their solutions; patterns are more concrete compared to guidelines, which often tend to be prescriptive, vague, and targeted more toward an expert rather than a novice; patterns are at a higher level of abstraction with respect to frameworks, although frameworks often make use of patterns to provide reusable code components.

An abstract model of patterns can be given as follows: A pattern P consists of a finite set of *elements*, that is, $P = \{E_1, E_2, \ldots, E_i\}$, where each element is part of pattern description as given. These elements form an anatomy of a pattern where they act as placeholders for pattern content and enable the pattern to be described completely for practical use. Typical elements are:

- **Name:** The unique, often metaphoric, name of the pattern by which it is known in the community.
- **Context:** The environment, situation, or interrelated conditions within the scope of which the PROBLEM recurs, and for which the SOLUTION is desirable.
- **Problem:** An issue that needs to be investigated and resolved, and is typically constrained by the CONTEXT in which it occurs.
- **Forces:** These describe relevant assumptions and constraints of the PROBLEM and how they interact/conflict with one another. These help determining the kinds of tradeoffs that must be considered to arrive at a SOLUTION.

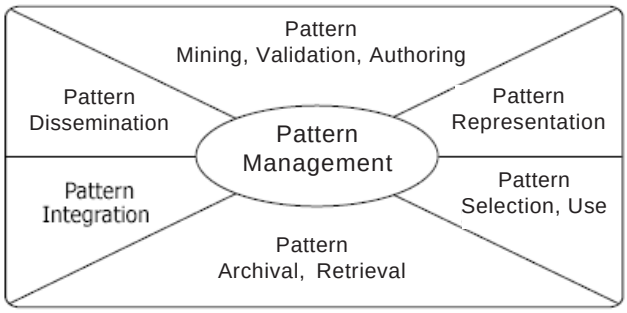
- **Solution:** The response to the PROBLEM in a CONTEXT that helps resolve the issue(s).
- **Rationale:** The reasoning behind and suitability of the pattern as a justified choice toward solving the PROBLEM.
- **Resulting Context:** The state of the entity (such as a software process or a software product) after applying the pattern, including both the positive and negative consequences.
- **Related Patterns:** The patterns that may be applicable as a result of the new context in which the entity finds itself.
- **Example:** The instance(s) of "real-world" situation(s) where the specified pattern has been applied.

This list could be extended to include other elements, for example, ANTI-PATTERNS (non-examples of the use of the pattern) or include elements of metadata (author information, version control, and so forth). The choice of both elements and element names varies in the community. Also, some of these elements (such as EXAMPLE) can be repeated while some of the elements (such as RATIONALE) are considered as optional.

A collection of patterns along with their (often implicit) context-driven relationships gives rise to a *pattern language*. A pattern language PL consists of a finite set of patterns $P_1, P_2, \ldots, P_j$ and a finite set of non-reflexive relationship types $R_1, R_2, \ldots, R_k$, that is, PL = $\{P_1, P_2, \ldots, P_j; R_1, R_2, \ldots, R_k$ such that $P_u R_t P_v$ for some u, $v \in \{1, 2, \ldots, j\}$, $u \neq v$, and $t \in \{1, 2, \ldots, k\}\}$. From a graph drawing perspective, a pattern language could be viewed as a directed acyclic graph (DAG) where $P_1, P_2, \ldots, P_j$ are the nodes and $R_1, R_2, \ldots, R_k$ are the vertices. Pattern languages differ from each other with respect to the domain that they correspond to, but need not be mutually exclusive (unrelated). For example, a pattern language for high-level user interface design of software is considered to be different from that for low-level source code design. However, the choices of patterns at the higher level could influence (restrict) the possibilities for selecting patterns at the lower level.

Pattern management involves various, not necessarily mutually exclusive, activities (Figure 1) that include pattern integration (combining patterns from a pattern language), pattern dissemination (making patterns available on a medium such as on an electronic network), pattern mining (pattern elicitation/discovery), pattern validation (designating potential candidate patterns and tracking them to qualification), pattern authoring (documenting and editing), pattern representation (explicitly specifying the syntactical and semantical knowledge inherent in a pattern), pattern selection and use (from a given set of options, choosing a pattern suitable for a problem and applying it), pattern archival (storing patterns expressed in a representation for future use), and pattern retrieval (searching and finding a pattern from a given archive). Tasks involved in these activities have been shown (May & Taylor, 2003) to fit in the Nonaka SECI model of knowledge (Nonaka & Takeuchi, 1995).

Figure 1. Activities in pattern management



The issue of adequate representation of patterns is central to pattern management as it directly or indirectly impacts other activities, including authoring, selection, integration, dissemination, archival, and retrieval. According to the COCOMO II cost estimation model (Boehm et al., 2001), reuse comes with a cost of adaptation to new contexts and a proper representation is crucial in this respect. Therefore, we next focus primarily on pattern representation from the perspective of knowledge management.

REPRESENTATION OF PATTERNS

There is a need for pattern representation at three levels: representing knowledge inherent in an individual pattern, representing relationships of patterns within a pattern language, and representing relationships across pattern languages. In this regard, technical requirements that we consider necessary for a language to be suitable for pattern representation, in no particular order of priority, are:

- **R1:** A representation must be able to accurately reflect the elements that constitute a pattern and their properties. To do that, a representation must have both a [R1.1] syntactical and [R1.2] semantical basis. This is necessary to be able to clearly express the syntax and semantics of relations among patterns and between pattern languages. It also makes it possible to check the validity of a pattern description with respect to its representation language.
- **R2:** A representation must allow means for unique identification of the pattern and its elements. This