UML-Based Conceptual Modeling of Pattern-Bases*

Stefano Rizzi

DEIS - University of Bologna
Viale Risorgimento, 2
40136 Bologna - Italy
srizzi@deis.unibo.it

Abstract. The concept of pattern, meant as an interesting knowledge artifact extracted from data, is considered to be a an effective answer to the advanced analysis requirements emerging in complex industrial and scientific applications. Overall, it appears that designing and building large pattern-bases will probably be a hot issue for future applications. In this paper we focus on conceptual design of pattern-bases, by discussing how UML could be used and extended to this end. In particular we address the main issues in static modeling, including the representation of relationships between patterns, and we briefly present some issues related to functional and dynamic modeling.

1 Introduction

Conceptual modeling is a key issue during the building of applications, since it provides a solid foundation for the design and implementation phases, it allows user requirements to be intuitively expressed and validated, and it provides expressive a-posteriori documentation for the design process. In particular, conceptual modeling of databases has been long studied and practiced; while through the past 20 years the Entity/Relationship model [3] has undoubtedly established itself as the most widely used formalism to this end, during the last few years UML has been gradually superseeding Entity/Relationship by becoming a standard for modeling a very wide range of applications and domains, including databases [2].

Recently, the research literature is emphasizing that databases may no longer be an effective answer to the advanced analysis requirements emerging in complex industrial and scientific applications. The concept of *pattern*, meant as an interesting knowledge artifact extracted from data (for instance a cluster, an association rule, a time series, etc.), is considered to be a good candidate to fill this gap. Among the main pattern modeling efforts we cite PMML [1], that uses XML to represent data mining models, SQL/MM [8], where the supported mining models are represented as SQL types, and the Pattern Query Language [7],

* This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-33058 PANDA project (2001-2004)

that is is an SQL-like query language for patterns. In *inductive databases*, data and patterns are represented together to be uniformly retrieved and manipulated [6]. In [4] a logical framework for modeling patterns is proposed; differently from the previously mentioned approached, here a strong emphasis is given to the generality, extensibility, and reusability issues.

Overall, it appears that designing and building large *pattern-bases* will probably be a hot issue for future applications. In particular, we believe that conceptual modeling will have a critical role in ensuring that the resulting pattern-base is well-designed, robust, and capable of effectively and flexibly supporting the analysis requirements of specialized users, especially since the conceptual schema constitutes a privileged platform for capturing and expressing complex relationships between patterns. Unfortunately, the peculiar nature of patterns as compared to data makes the design techniques developed for data mostly useless for pattern design.

In this paper we focus on conceptual modeling of pattern-bases, by discussing how UML could be used and extended to this end. In general, modeling should be carried out along three coordinates: *static*, aimed at describing the pattern-base from a structural point of view, *functional*, meant to give an insight on the processes that transform data in the pattern-base, and *dynamic*, that focuses on representing the evolution in time of the state of the pattern-base. While the discussion is mainly focused on static modeling, also some issues in functional and dynamic modeling are considered in the paper. The reference logical model adopted is the one proposed in [4].

The paper outline is as follows. In Section 2 we summarize the main features of the reference logical model. In Section 3 we present some notable examples of patterns which will be used throughout the paper to demonstrate the modeling solutions proposed. Sections 4, 5, and 6 describe our proposal for static, functional, and dynamic modeling of pattern-bases, respectively. Finally, Section 7 draws the conclusions.

2 Background on Pattern-Bases

In this section we summarize the main features of the logical model for patterns proposed in [4].

2.1 Pattern Types, Patterns, Classes

Patterns can be regarded as artifacts which effectively describe subsets of raw data by isolating and emphasizing some interesting properties. Thus, we may informally say that *a pattern is a compact and rich in semantics representation of raw data*. While in most cases a pattern is interesting to the end-users because it describes a recurrent behaviour (e.g., in market segmentation, stock exchange analysis, etc.), sometimes it is relevant just because it is related to some singular, unexpected event (e.g., in failure monitoring).

A pattern type represents the intensional form of patterns, giving a formal description of their structure and relationship with source data. Given a set of *base types* and a set of *type constructors*, in the following we will refer to a set T of types including all the base types together with all the types recursively defined by applying a type constructor to one or more other types; types are applied to *attributes*.

Definition 1 (Pattern type). *A* pattern type *pt is a quintuple*

$$pt = (n, ss, ds, ms, f)$$

where:

1. *n is the* name *of the pattern type;*
2. *ss (*structure schema*) is a type in T that defines the pattern space by describing the structure of the patterns instances of the pattern type;*
3. *ds (*source schema*) is a type in T that defines the related raw data space by describing the dataset from which patterns are constructed;*
4. *ms (*measure schema*) is a type in T describing the measures which quantify the quality of the source data representation achieved by the pattern;*
5. *f is a* formula*, referring to attributes appearing in the source and in the structure schemas, that describes the relationship between the source space and the pattern space, thus carrying the semantics of the pattern.*

Let raw data be stored in a number of databases and/or files. A *dataset* is any subset of these data, which we assume to be wrapped under a type of our typing system (*dataset type*).

Definition 2 (Pattern). *Let $pt = (n, ss, ds, ms, f)$ be a pattern type. A* pattern *p instance of pt is a quintuple $p = (pid, s, d, m, e)$ where: pid (*pattern identifier*) is a unique identifier for p; s (*structure*) is a value for type ss; d (*source*) is a dataset whose type conforms to type ds; m (*measure*) is a value for type ms; e is an expression denoting the region of the source space that is related to p.*

A class is a set of semantically related patterns and constitutes the key concept in defining the pattern query language. A class is defined for a given pattern type and contains only patterns of that type. Moreover, each pattern must belong to at least one class.

Definition 3 (Class). *A* class *c is a triple $c = (cid, pt, pc)$ where cid (*class identifier*) is a unique identifier for c, pt is a pattern type, and pc is a collection of patterns of type pt.*

2.2 Relationships Between Patterns

Three different relationships between patterns were identified in [4]: specialization, composition, refinement.

Definition 4 (Specialization). *Pattern type* pt_1 specializes *pattern type* pt_2 *when the structure schema, the source schema, and the measure schema of* pt_1 *specialize the structure schema, the source schema, and the measure schema of* pt_2*, respectively.*

Composition and refinement relationships arise from the possibility of extending the set of base types with pattern types, thus giving the user the possibility of declaring *complex types* to be used either in the structure schema or in the source schema.

Definition 5 (Composition). *Pattern type* pt_1 is part of *pattern type* pt_2 *when the structure schema of* pt_2 *is a complex type including* pt_1*.*

Definition 6 (Refinement). *Pattern type* pt_1 refines *pattern type* pt_2 *when the source schema of* pt_2 *is a complex type including* pt_1*.*

While composition enables the definition of patterns recursively containing other patterns, refinement allows for supporting the modeling of patterns obtained by mining other existing patterns.

3 Working Examples

This section includes some working examples we will use in the following to demonstrate the modeling solutions proposed.

Example 1 (Association Rule). Given a domain D of values and a set of transactions, each including a subset of D, an *association rule* takes the form $A \rightarrow B$ where $A \subset D$, $B \subset D$, $A \cap B = \emptyset$. A is often called the *head* of the rule, while B is its *body* [5]. A possible pattern type for modeling association rules over strings is the following:

$$
\begin{aligned}
n &: \mathsf{AssociationRule} \\
ss &: \mathsf{TUPLE(head:\ SET(STRING),\ body:\ SET(STRING))} \\
ds &: \mathsf{BAG(transaction:\ SET(STRING))} \\
ms &: \mathsf{TUPLE(confidence:\ REAL,\ support:\ REAL)} \\
f &: \forall x(x \in \mathsf{head} \vee x \in \mathsf{body} \Rightarrow x \in \mathsf{transaction})
\end{aligned}
$$

The structure schema is a tuple modeling the head and the body. The source schema specifies that association rules are constructed from a bag of transactions, each defined as a set of strings. The measure schema includes two common measures used to assess the relevance of a rule: its confidence and its support. Finally, the formula of the constraint calculus represents the pattern/dataset relationship by associating each rule with the set of transactions which support it.

Example 2 (Cluster and clustering). Let pattern type Cluster represent any cluster defined by enumerating a set of elements in a space and represented by an element of that space:

$$\begin{aligned} n &: \mathsf{Cluster} \\ ss &: \mathsf{representative}{:}\bot \\ ds &: \mathsf{space}{:}\ \mathsf{SET}(\mathsf{item}{:}\ \bot) \\ ms &: \mathsf{cardinality}{:}\ \mathsf{INTEGER} \\ f &: \exists S\ {:}\mathsf{item} \in S \end{aligned}$$

where $\bot$ denotes the root type of the typing system; the formula defines the relationship between the source space and the pattern space by trivially enumerating all the elements belonging to the cluster. Consider now that a clustering is a set of clusters: intuitively, also clustering is a pattern, whose structure is modeled by a complex type which aggregates a set of clusters:

$$\begin{aligned} n &: \mathsf{Clustering} \\ ss &: \mathsf{clusters}{:}\ \mathsf{SET}(\mathsf{Cluster}) \\ ds &: \mathsf{space}{:}\ \mathsf{SET}(\mathsf{item}{:}\ \bot) \\ ms &: \mathsf{clusteringValidity}{:}\ \mathsf{REAL} \\ f &: \bigvee_{c \in \mathsf{clusters}} c.f \end{aligned}$$

Thus, there is a composition relationship between Clustering and Cluster.

Example 3 (Cluster of association rules). Let pattern type ClusterOfRules describe a cluster of association rules: the source schema here represents the space of association rules, and the structure models one cluster-representative rule. Assuming that each cluster trivially includes all the rules sharing the same head, it is:

$$\begin{aligned} n &: \mathsf{ClusterOfRules} \\ ss &: \mathsf{representative}{:}\ \mathsf{AssociationRule} \\ ds &: \mathsf{space}{:}\ \mathsf{SET}(\mathsf{item}{:}\ \mathsf{AssociationRule}) \\ ms &: \mathsf{TUPLE}(\mathsf{deviationOnConfidence}{:}\ \mathsf{REAL},\ \mathsf{deviationOnSupport}{:}\ \mathsf{REAL}) \\ f &: \mathsf{item.ss.head} = \mathsf{representative.ss.head} \end{aligned}$$

where a standard dot notation is adopted to address the components of pattern types. Obviously, ClusterOfRules specializes Cluster; besides, there are both a refinement relationship and a composition relationship between ClusterOfRules and AssociationRule.

4 Static Modeling

When facing the problem of devising a formalism for conceptual modeling of pattern-bases, the first obvious step is to verify whether a widespread standard

like UML can effectively be used for this task. In this section we outline the requirements for static modeling of pattern-bases as emerging from the logical framework proposed in [4], then we discuss to what extent each of them could be accommodated in UML.

The main requirements for static modeling of pattern-bases can be summarized as follows:

♯1 The different components of pattern types should be explicitly modeled in order to emphasize their different semantic role in describing patterns.
♯2 Data sources should be modeled.
♯3 Both the intensional and the extensional aspects of data should be modeled.
♯4 Specialization relationships between patterns should be modeled.
♯5 Composition relationships between patterns should be modeled.
♯6 Refinement relationships between patterns should be modeled.

As to requirement ♯1, we observe that UML represents classes by distinguishing three compartments, respectively reserved to the class name, the class attributes (structure), and the class operations (behavior). Modeling pattern types in UML requires (1) to define a ≪`pattern type`≫ stereotype; (2) to include in the attribute compartment two fixed attributes, `ss` for the structure schema and `ms` for the measure schema; (3) to include in the operation compartment one fixed operation `f` for the formula (see Figure 1). As to modeling the source schema, two different approaches are feasible:

– When little emphasis is given to raw data, the source schema may be represented by adding one more fixed attribute `ds` to the pattern type, as shown in Figure 1, top.
– When the designer wishes to give more relevance to raw data, according to requirement ♯2, the source schema may be represented by a class on its own, possibly stereotyped, connected to the pattern type by a ≪`refines`≫ dependency (see Figure 1, bottom). The reason for choosing a UML dependency is that it expresses the fact that one element of the model (the pattern type) needs another element (the raw data schema) in order to work properly; the ≪`refines`≫ stereotype expresses the fact that the pattern type gives a more abstract representation of raw data.

As to requirement ♯3, in UML it is possible to distinguish objects from classes by underlining their names; besides, a ≪`type`≫ stereotpye can be used to model abstract data types (intensional) as opposed to classes (extensional). In much the same way, pattern types, patterns, and classes could be modeled as shown in Figure 2. However, we wish to emphasize that object diagrams are seldom used in UML; similarly, we expect that specific patterns will be modeled only when the designer assumes that providing some notable example is useful for understanding.

Requirement ♯4 is already supported by the specialization hierarchies of UML, whose semantics accomodate inheritance of attributes, operations, and other properties, plus the possibility of extending the specialized class with new,

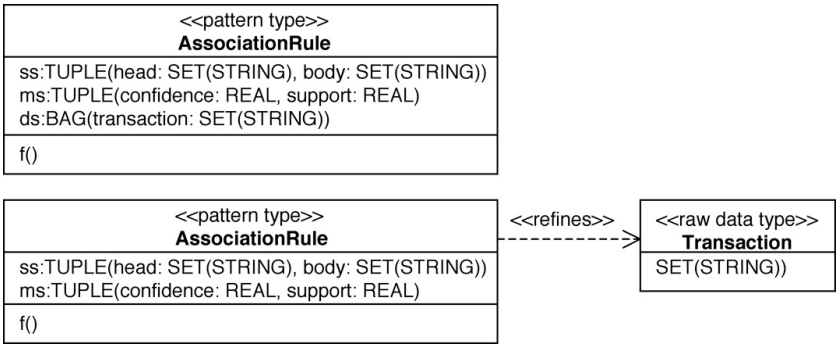

Fig. 1. Modeling of pattern types, giving different relevance to raw data

Fig. 2. Classes, patterns, and pattern types

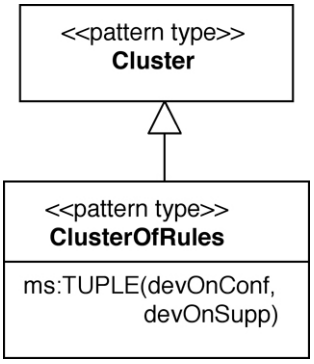

Fig. 3. Pattern specialization

specific structure and behaviour. Figure 3 shows the specialization relationship between ClusterOfRules and Cluster. Besides, the composition semantics provided by UML directly supports requirement ♯5 (see Figure 4). As a matter of fact, we prefer using the aggregation syntax of UML (a white diamond) in order to emphasize that the component type is generally not a weak one.

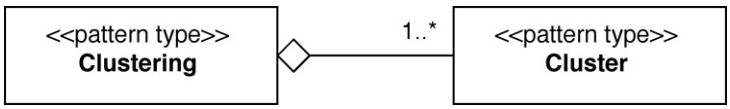

Fig. 4. Pattern composition

Finally, requirement ♯6 could be satisfied through stereotyping, by using the same ≪`refines`≫ dependency adopted for source schemas, as depicted in Figure 5.

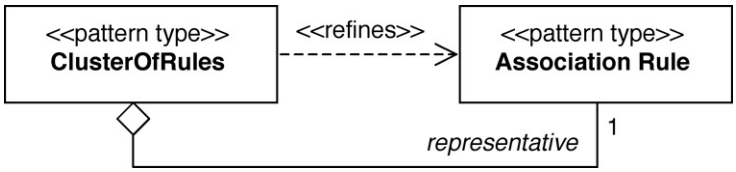

Fig. 5. Pattern refinement and composition

Currently, the logical model proposed in [4] does not consider the possibility of defining associations between patterns; however, these would be easily modeled in UML. This is equally true for the behavioural component of patterns, that could be modeled by properly defining class operations.

5 Functional Modeling

Functional modeling of a pattern-base mainly requires to represent the processes used to establish the relationship between data and patterns: in most cases, these will be mining or extraction algorithms, but they could as well be sets of rules used to enforce constraints on the dataset. Different functional formalisms could be adopted:

1. DFDs, that give an overview of the processes, actors, and archives involved emphasizing the data flows between them.
2. UML use case diagrams, that describe the processes from the point of view of users and of the utility they carry.

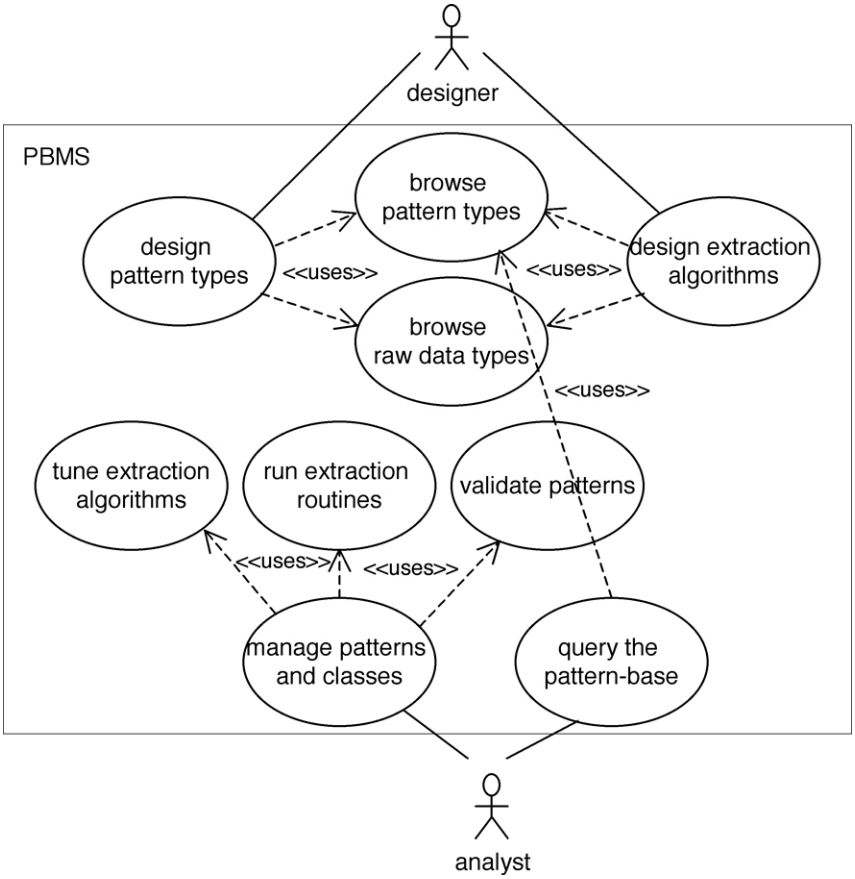

Fig. 6. Use cases for a Pattern-Base Management System

3. Flow charts or UML activity diagrams, that sketch the control flow within an algorithm (thus, they could be used to specify a DFD process or a use case).

An example is shown in Figure 6, were a simple use case diagram is used to represent the main activities supported by a Pattern-Base Management System.

6 Dynamic Modeling

Dynamic modeling of a pattern-base mainly requires to describe how patterns and raw data are kept in sync. Different dynamic formalisms could be adopted to this end:

1. State charts, that represent how the state of the pattern-base evolves in response to events.

2. UML sequence diagrams, that describe the sequences of messages exchanged by patterns and raw data during some relevant scenario.

Figure 7 shows a sequence diagram modeling the asynchronous approach chosen to trigger the execution of an extraction algorithm when some changes in the raw data occur.

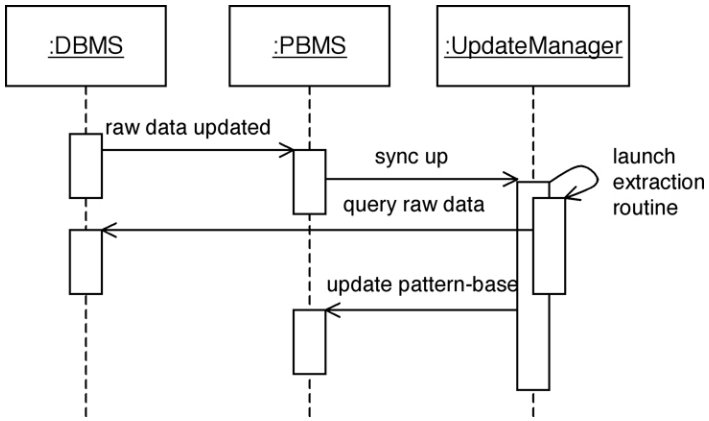

Fig. 7. Sequence diagram for updating the pattern-base

7 Conclusion

In this paper we have shown how it would be possible to conceptually model a pattern-base from the static, functional, and dynamic points of view by properly extending UML through the stereotyping mechanism. Though a new, ad hoc formalism for modeling pattern-bases could actually be devised in order to better express the semantics of patterns and their components, we believe that adopting UML is still preferrable since it is a standard *de facto* for most software engineering applications. Besides, as seen in Sections 4, 5, and 6, all the main issues related to pattern-bases can be expressively represented without giving up the usual syntax and semantics of UML. This is especially true for complex inter-pattern relationships, whose accurate representation is necessary to achieve high modeling expressivity.

References

1. Predictive Model Markup Language (PMML). `http://www.dmg.org/pmmlspecs_v2/pmml_v2_0.html`, 2003.
2. G. Booch, J. Rumbaugh, and I. Jacobson. *The UML user guide G. Booch, J. Rumbaugh, I. Jacobson. The UML user guide. Addison Wesley, 1999.* Addison Wesley, 1999.

3. P. Chen. The entity-relationship model - toward a unified view of data. *ACM TODS*, 1(1):9–36, 1976.
4. S. Rizzi et al. Towards a logical model for patterns. In *Proc. ER Conference*, pages 77–90, Chicago, 2003.
5. J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Academic Press, 2001.
6. T. Imielinski and H. Mannila. A Database Perspective on Knowledge Discovery. *Communications of the ACM*, 39(11):58–64, 1996.
7. Information Discovery Data Mining Suite. `http://www.patternwarehouse.com /dmsuite.htm`, 2002.
8. ISO SQL/MM Part 6. `http://www.sql-99.org/SC32/WG4/Progression_Documents/ FCD/fcd-datamining-2001-05.pdf`, 2001.