PAtterns for Next-generation DAtabase systems: preliminary results of the PANDA project*

Ilaria Bartolini[1], Elisa Bertino[2], Barbara Catania[3], Paolo Ciaccia[1], Matteo Golfarelli[1], Marco Patella[1], and Stefano Rizzi[1]

[1] DEIS, Univ. of Bologna, Italy
[2] Dept. of Computer Science, Univ. of Milan, Italy
[3] Dept. of Computer and Information Science, Univ. of Genoa, Italy

Abstract. Nowadays, the vast volume of collected digital data obliges us to employ processing methods like pattern recognition and data mining in order to reduce the complexity of data management. The output of these techniques are knowledge artifacts, heterogeneous in both structure and semantics. We claim that the concept of *pattern* is a good candidate for generic representation of these novel information types. The PANDA project is aimed at studying the main issues related to pattern handling. In this paper we present the preliminary results obtained: we outline the architecture of Pattern-Base Management Systems, we provide the foundations of the logical framework, and we present the preliminary issues related to processing queries on patterns.

1 Introduction and motivation

The increasing opportunity of quickly collecting and cheaply storing large volumes of data, and the need for extracting concise information to be efficiently manipulated and intuitively analysed have determined in the last decade the spreading of data warehousing systems. On the other hand, the limited analysis power of OLAP interfaces represents an obstacle when huge quantity of data are involved, and non-standard analysis techniques are requested. Hence, sophisticated data processing tools (based for instance on data mining, pattern recognition, and knowledge extraction techniques) were devised in order to reduce, as far as possible, the user intervention in the process of extracting interesting knowledge artifacts (e.g., clusters, association rules, time series) from raw data [7]. We claim that the term *pattern* is a good candidate to generally denote these novel information types, characterized by a high degree of diversity and complexity. DBMSs are not powerful and flexible enough to handle this new kind of knowledge, therefore a specific management system capable of modeling and storing patterns is required. We will call PBMS (*Pattern-Base Management System*) such a system, that is characterized by the following properties:

- *Abstraction.* Within a PBMS, patterns are made first-class citizens thus providing the user with a meaningful abstraction of raw data to be directly analyzed and manipulated.

* This work was supported by the PANDA IST Thematic Network.

– *Efficiency.* Introducing an architectural separation between the PBMS and the DBMS improves the efficiency of both traditional transactions on the DBMS and advanced processing on patterns.
– *Flexible Querying.* The PBMS provides an expressive language for querying the pattern-base in order to retrieve and compare patterns.

In this context, the purposes of the *PANDA* (*PAtterns for Next-generation DAtabase systems* [1]) project of the European Community are: (1) to lay the foundations for pattern modeling; (2) to investigate the main issues involved in managing and querying a pattern-base; and (3) to outline the requirements for building a PBMS. Differently from other proposals, usually dealing with association rules [8, 9, 12] or string patterns [13], no predefined pattern types are considered in PANDA.

In this paper we present the preliminary results of the PANDA project: starting from the concept of PBMS, we provide the logical foundations of a general framework based on the notions of pattern types and pattern classes, then we introduce the main issues in querying patterns.

2 A management system for patterns

Raw data are recorded from various sources in the real world, often by collecting measurements from various instruments or devices (e.g., cellular phones, environment measurements, monitoring of computer systems, etc.). The determining property of raw data is the vastness of their volume; moreover, a significant degree of heterogeneity may be present.

Data in such huge volumes do not constitute knowledge *per se*, i.e. little useful information can be deduced simply by their observation, so they hardly can be directly exploited by human beings. Thus, more elaborate techniques are required in order to extract the hidden knowledge and make these data valuable for end-users. The common characteristic of these techniques is that large portions of the available data are abstracted and effectively represented by a small number of knowledge-carrying representatives, which we call *patterns*. Hence, in general, one pattern is related to many data items; on the other hand, several patterns (possibly of different types) can be associated to the same data item (e.g., due to the application of different algorithms). Patterns can be regarded as artifacts which effectively describe subsets of raw data (they are *compact*) by isolating and emphasizing some interesting properties (they are *rich in semantics*).

Differently from other existing proposals [8, 13], we claim that differences in the raw data and in the extraction algorithms induce a strong heterogeneity in the pattern structures that requires an ad-hoc management system to be efficiently handled. A Pattern-Base Management System is thus defined as follows.

Definition 1 (PBMS). *A* Pattern-Base Management System *(PBMS) is a system for handling (storing/processing/retrieving) patterns defined over raw data in order to efficiently support pattern matching and to exploit pattern-related*

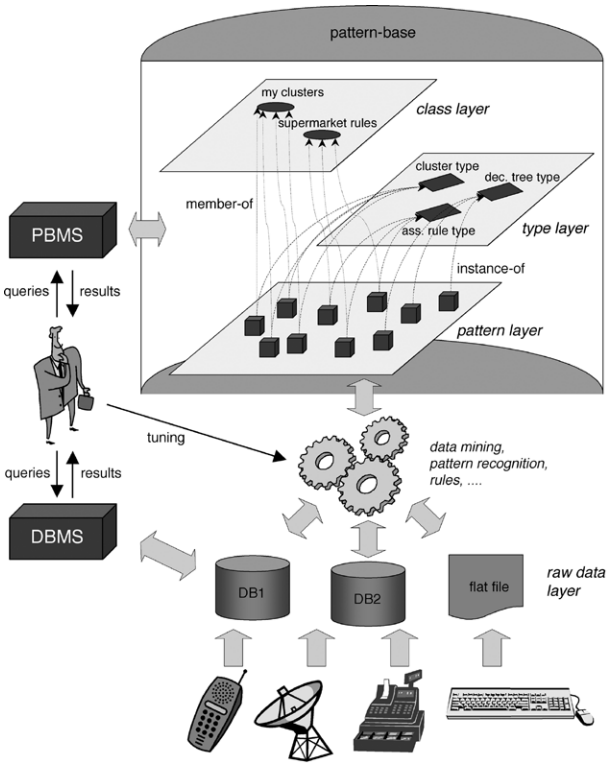

Fig. 1. The PBMS architecture

operations generating intensional information. The set of patterns managed by a PBMS is called pattern-base*.*

The reference architecture for a PBMS is depicted in Figure 1. On the bottom layer, a set of devices produce data, which are then organized and stored within databases or files to be typically, but not necessarily, managed by a DBMS. Knowledge discovery algorithms are applied over these data and generate patterns to be fed into the PBMS.

Within the PBMS, we distinguish three different layers. The *pattern layer* is populated with patterns. Patterns with similar structural characteristics have the same type (either built-in or user-defined) that is described in the *type layer*. The *class layer* holds definitions of pattern classes, i.e., collections of semantically related patterns. Classes play the role of collections in the object-oriented context and are the key concept in the definition of a pattern query language.

Besides using the DBMS, end-users may directly interact with the PBMS: to this end, the PBMS adopts ad-hoc techniques not only for representing and storing patterns, but also for posing and processing queries and for efficiently retrieving patterns.

3 The logical modeling framework

In this section we formalize our proposal of a logical framework by characterizing pattern types, their instances, and the classes which collect them.

3.1 Pattern types

Though our approach is parametric on the typing system adopted, the examples provided in this paper we will be based on a specific, very common typing system similar to those adopted in OODBMSs. Assuming there is a set of *base types* and a set of *type constructors*, the set T of types includes 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*.

A pattern type represents the intensional form of patterns, giving a formal description of their structure and relationship with source data. Thus, pattern types play the same role than abstract data types in the object-oriented model.

Definition 2 (Pattern type). *A* pattern type *pt is a quintuple $pt = (n, ss, ds, ms, f)$ where n is the* name *of the pattern type; ss, ds, and ms (called respectively* structure schema*,* source schema*, and* measure schema*) are types in T; f is a* formula*, written in a given language, which refers to attributes appearing in the source and in the structure schemas.*

This definition is intended to satisfy three basic requirements: *generality* (i.e. all the specific requirements posed in different application domains for different kinds of patterns should be met), *extensibility* (i.e. it should be possible to accommodate new kinds of patterns introduced by novel and challenging applications) and *reusability* (i.e. the reuse of what has already been defined is encouraged). In particular, the first component of a pattern type has an obvious meaning; the remaining four have the following roles:

- The structure schema ss defines the pattern space by describing the structure of the patterns instances of the pattern type. The achievable complexity of the pattern space (hence, the flexibility of pattern representation) depends on the expressivity of the typing system. Note that, by extending the set of base types with pattern types, it is possible to recursively define *complex types*; for istance, a clustering is a set of clusters, thus the structure schema of a clustering pattern type is defined in terms of cluster pattern types.
- The source schema ds defines the related source space by describing the dataset from which patterns, instances of the pattern type being defined, are constructed. Characterizing the source schema is fundamental for every operation which involves both the pattern space and the source space (e.g., when applying some technique to extract patterns from raw data or when checking for the validity of a pattern on a dataset).
- The measure schema ms describes the measures which quantify the quality of the source data representation achieved by the pattern, thus enabling the user to evaluate how accurate and significant for a given application each

pattern is. Besides, the different semantics of the measure component with reference to the structure can be exploited in order to define more effective functions for evaluating the distance between two patterns [6].
– The formula f describes the relationship between the source space and the pattern space, thus carrying the semantics of the pattern. Inside f, attributes are interpreted as free variables ranging over the components of either the source or the pattern space. Note that, though in some particular domains f may exactly express the inter-space relationship (at most, by allowing all raw data related to the pattern to be enumerated), in most cases it will describe it only approximatively.

Though our approach to pattern modeling is parametric on the language adopted for formulas, the achievable semantics for patterns strongly depends on its expressivity. For the examples reported in this paper, we use a constraint calculus based on a polynomial constraint theory which seems suitable for several classes of patterns [10].

Example 1. 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* [7]. An example of a pattern type for modeling association rules over strings representing products 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 products. The measure schema includes two common measures used to assess the relevance of a rule: its confidence (what percentage of the transactions including the head also include the body) and its support (what percentage of the whole set of transactions include both the head and the body). Finally, the formula of the constraint calculus represents (exactly, in this case) the pattern/dataset relationship by associating each rule with the set of transactions which support it.

3.2 Patterns

Definition 3 (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 related to pattern p.*

According to this definition, a pattern is characterized by (1) a pattern identifier (which plays the same role of OIDs in the object model), (2) a structure that positions the pattern within the pattern space defined by its pattern type, (3) a source that identifies the specific dataset the pattern relates to, (4) a measure that estimates the quality of the raw data representation achieved by the pattern, (5) an expression which relates the pattern to the source data. A pattern instance of a complex pattern type is said to be a *complex pattern*, and the patterns it includes are called its *components*.

Example 2. Consider again pattern type AssociationRule defined in Example 1, and suppose that raw data include a relational database containing a table sales which stores data related to the sales transactions in a sport shop: sales (transactionId, article, quantity). Using an extended SQL syntax to denote the dataset, an example of an instance of AssociationRule is:

$$
\begin{aligned}
&pid : 512\\
&\quad s : (\mathsf{head} = \{\text{'Boots'}\}, \mathsf{body} = \{\text{'Socks'}, \text{'Hat'}\})\\
&\quad d : \text{'SELECT SETOF(article) AS transaction}\\
&\qquad\quad \text{FROM sales GROUP BY transactionId'}\\
&\quad m : (\mathsf{confidence} = 0.75, \mathsf{support} = 0.55)\\
&\quad e : \{\mathsf{transaction} : \forall x(x \in \{\text{'Boots'}, \text{'Socks'}, \text{'Hat'}\} \Rightarrow x \in \mathsf{transaction})\}
\end{aligned}
$$

In the expression, transaction ranges over the source space; the values given to head and body within the structure are used to bind variables head and body in the formula of pattern type AssociationRule.

3.3 Classes

A class is a set of semantically related patterns; it is defined for a given pattern type and contains only patterns of that type.

Definition 4 (Class). *A class c is a triple c=(cid,pt,pc) where cid is a unique identifier for c, pt is a pattern type, and pc is a collection of patterns of type pt.*

Example 3. The *Apriori* algorithm described in [2] could be used to generate association rules from the dataset presented in Example 2. The generated patterns could be inserted in a class called *SaleRules* for pattern type AssociationRule defined in Example 1. The collection of patterns associated with the class can be later extended to include also rules generated from a different dataset, representing for instance the sales transactions recorded in a different store.

4 Querying Issues

Apart from the issues related to the physical storage of patterns, important challenges are posed when considering how a pattern base can be queried.[1]

[1] Note that, under the proposed architecture, pattern generation is a PBMS data manipulation operation, differently from other tightly coupled approaches [8, 9, 12], where pattern generation is a query language issue.

Besides pattern retrieval operations, cross-over operations [9], between patterns and raw data, have to be supported. Under the proposed architecture, they can be implemented similarly to the drill-through operation in data warehousing systems [5], rewriting parts of the queries taking into account the pattern source.

A basic operation between patterns is that of comparison: two patterns of the same type can be compared to compute a score s, $s \in [0, 1]$, assessing their mutual similarity. The similarity between two patterns is computed as a function of the similarity between both the structure and the measure components: $sim(p_1, p_2) = f(sim_{struct}(p_1.s, p_2.s), sim_{meas}(p_1.m, p_2.m))$ where $p.s$ and $p.m$ denote the structure and the measure for pattern p, respectively. If the two patterns have the same structure, then $sim_{struct}(p_1.s, p_2.s) = 1$ and the similarity measure naturally corresponds to a comparison of the patterns' measures, e.g. by aggregating differences between each measure [6]. In the general case, however, the patterns to be compared have different structures, thus a preliminary step is needed to reconcile the two structures to make them comparable.

Particularly challenging is the comparison between complex patterns; in this case, the measure of the complex pattern may include values defined as expressions of the component patterns. The similarity score s between two complex patterns of the same type is computed starting from the similarity between component patterns, then the obtained scores are aggregated, using an aggregation logic, to determine the overall similarity [6]. The aggregation logic may be very simple, just an expression combining numerical values, or a more complex one, if constraints and/or transformation costs are to be considered. For example, a suitable "matching" between components patterns might be needed: this is the case for region-based image retrieval systems, where each image (the complex pattern) is composed by several regions (base patterns) and the similarity at the image level is computed by combining the similarity of matched regions [3]. The aggregation logic can also involve some transformations, each with an associated cost, and the overall similarity score is obtained as the maximum score obtained by applying all the possible transformations to the component patterns. Consider for example a comparison between time series, modeled as complex patterns whose structure is a list of patterns representing data samples and the structure of each sample includes the sample instant and the sample value. In this case, the aggregation logic should be aware of the order of succession of samples; thus a simple logic that matches each sample of one series with the closer sample of the other is not correct and more complex aggregation logics, such as a Dynamic Time Warping algorithm [11], are needed. Possible transformations include time scaling, vertical shifting, amplitude scaling, linear drifting, stuttering, and so on.

Efficient evaluation of general aggregation algorithms, however, seems to be a formidable challenge, since the application of transformations and aggregation logics could significantly alter the complexity of matching. In the simpler case, assessment of similarity can be performed by exploiting an index structure, like the M-tree [4], provided that similarity between component patterns is computed by way of a metric distance. Moreover, to speed-up the matching, one can consider the use of approximate strategies.

5 Conclusions and future work

In this paper we presented a general framework for manipulating patterns describing large volumes of data. The framework relies on a specific management system (PBMS) for storing and querying patterns, modeled according to an ad-hoc data model, general enough to cover a broad range of real-world cases. Besides, we discussed the preliminary issues related to pattern query processing.

Though the fundamentals of pattern modeling have been addressed, several important issues still need be investigated. Future research includes both theoretical aspects as well as implementation-specific issues. The theoretical aspects include: (1) defining a pattern manipulation language, including pattern generation operations; (2) studying the possibility of posing constraints on patterns; (3) defining the common operations on patterns; (4) evaluating and comparing the expressivity of different languages to formulate expression templates; (4) devising a flexible query language for retrieving and comparing patterns. Concerning pattern generation, we believe that extensions of already proposed operators can be used to this purpose [9, 12]. Implementation issues involve primarily the construction of ad-hoc storage management and query processing modules for the efficient management of patterns.

References

1. The PANDA Project. `http://dke.cti.gr/panda/`, 2002.
2. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th VLDB*, 1994.
3. I. Bartolini, P. Ciaccia, and P. Patella. A sound algorithm for region-based image retrieval using an index. In *Proc. QPMIDS'00*, pages 930–934, Greenwich, UK, 2000.
4. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. 23rd VLDB*, pages 426–435, Athens, Greece, 1997.
5. Y. Cui and J. Widom. Lineage tracing in a data warehousing system. In *Proc. 16th Int. Conf. on Data Engineering*, San Diego, CA, 2000.
6. V. Ganti, R. Ramakrishnan, J. Gehrke, and W.-Y. Loh. A framework for measuring distances in data characteristics. *PODS*, 1999.
7. J. Han and M. Kamber. *Data mining: concepts and techniques*. Academic Press, 2001.
8. T. Imielinski and H. Mannila. A Database Perspective on Knowledge Discovery. *Communications of the ACM*, 39(11):58–64, 1996.
9. T. Imielinski and A. Virmani. MSQL: A Query Language for Database Mining. *Data Mining and Knowledge Discovery*, 3:373–408, 1999.
10. P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):25–52, 1995.
11. E. Keogh. Exact indexing of dynamic time warping. In *Proc. 28th VLDB 2002*, pages 406–417, Hong Kong, 2002.
12. R. Meo, G. Psaila, and S. Ceri. A new sql-like operator for mining association rules. In *Proc. 22nd VLDB Conf.*, Bombay, India, 1996.
13. L. De Raedt. A Perspective on Inductive Databases. *SIGKDD Explorations*, 4(2), 2002.