Towards a Logical Model for Patterns[⋆]

Stefano Rizzi[1], Elisa Bertino[2], Barbara Catania[3], Matteo Golfarelli[1], Maria Halkidi[4], Manolis Terrovitis[5], Panos Vassiliadis[6], Michalis Vazirgiannis[4], and Euripides Vrachnos[4]

[1] DEIS, Univ. of Bologna, Italy
[2] DICO, Univ. of Milan, Italy
[3] DISI, Univ. of Genoa, Italy
[4] Athens Univ. of Economics & Business, Greece
[5] Dept. of Electrical and Computer Engineering, Nat. Tech. Univ. of Athens, Greece
[6] Dept. of Computer Science, University of Ioannina, Greece

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. In this paper, we present the architecture and the logical foundations for the management of the produced knowledge artifacts, which we call *patterns*. To this end, we first introduce the concept of Pattern-Base Management System; then, we provide the logical foundations of a general framework based on the notions of pattern types and pattern classes, which stand for the intensional and extensional description of pattern instances, respectively. The framework is general and extensible enough to cover a broad range of real-world patterns, each of which is characterized by its structure, the related underlying data, an expression that carries the semantics of the pattern, and measurements of how successful the representation of raw data is. Finally, some remarkable types of relationships between patterns are discussed.

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, are posing new requirements for DBMSs in both industrial and scientific applications. In this direction, during the last decade we witnessed the progressive spreading and success of data warehousing systems, built on top of operational databases in order to provide managers and knowledge workers with ready-at-hand summary data to be used for decision support. In these systems, the transactional view of data is replaced by a multidimensional view, relying on an *ad-hoc* logical model (the *multidimensional*

[⋆] 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

Table 1. Some examples of patterns.

Application	*Raw data*	*Type of pattern*
market-basket analysis	sales transactions	item association rules
signal processing	complex signals	recurrent waveforms
mobile objects monitoring	measured trajectories	equations
information retrieval	documents	keyword frequencies
image recognition	image database	image features
market segmentation	user profiles	user clusters
music retrieval	music scores, audio files	rhythm, melody, harmony
system monitoring	system output stream	failure patterns
financial brokerage	trading records	stock trends
click-stream analysis	web-server logs	sequences of clicks
epidemiology	clinical records	symptom-diagnosis correlations
risk evaluation	customer records	decision trees

model [17]) whose key concepts are made first-class citizens, meaning that they can be directly stored, queried, and manipulated.

On the other hand, the limited analysis power provided by OLAP interfaces proved to be insufficient for advanced applications, in which the huge quantity of data stored necessarily requires semi-automated processing techniques, and the peculiarity of the user requirements calls for non-standard analysis techniques. Thus, 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 [8,11,13]. 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. Some examples of patterns in different application domains are reported in Table 1.

Differently from the case of data warehousing systems, the problem of directly storing and querying pattern-bases has received very limited attention so far in the commercial world, and probably no attention at all from the database community. On the other hand, we claim that end-users from both industrial and scientific domains would greatly benefit from adopting a *Pattern-Base Management System* (PBMS) capable of modeling and storing patterns, for the following main reasons:

- *Abstraction.* Within a PBMS, patterns would be made first-class citizens thus providing the user with a meaningful abstraction of raw data to be directly analyzed and manipulated.
- *Efficiency.* Introducing an architectural separation between the PBMS and the DBMS would improve the efficiency of both traditional transactions on the DBMS and advanced processing on patterns.
- *Querying.* The PBMS would provide 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* [4]) 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.

In this paper we propose a logical framework for modeling patterns, aimed at satisfying three basic requirements:

– *Generality.* The model must be general enough to meet the specific requirements posed in different application domains for different kinds of patterns.
– *Extensibility.* The model must be extensible to accomodate new kinds of patterns introduced by novel and challenging applications.
– *Reusability.* The model must include constructs encouraging the reuse of what has already been defined.

Informally, a pattern can be thought of as a *compact* and *rich in semantics* representation of raw data. In our approach, a pattern is modeled by its *structure*, *measure*, *source*, and *expression*. The structure component *qualifies* the pattern by locating it within a pattern space. The measure component *quantifies* the pattern by measuring the quality of the raw data representation achieved by the pattern itself. The source component describes the raw data which the pattern relates to. The expression component describes the (approximate) mapping between the raw data space and the pattern space.

The paper is structured as follows. In Section 2 we deliver an informal definition of patterns and outline a reference architecture in which a PBMS could be framed. In Section 3 we formally describe our proposal of a logical framework for modeling patterns, while Section 4 discusses some remarkable types of relationships between patterns. In Section 5 we survey some related approaches. Finally, in Section 6 the conclusions are drawn and the future work is outlined.

2 From DBMSs to PBMSs

2.1 Raw Data vs. 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.

Clearly, 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 all 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

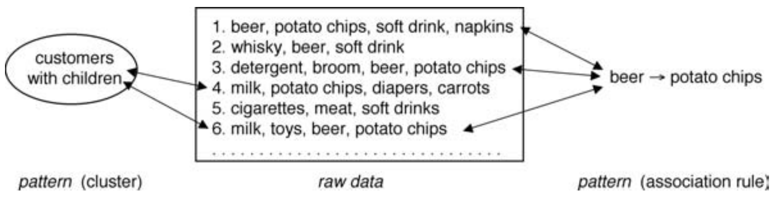
customers with children
1. beer, potato chips, soft drink, napkins
2. whisky, beer, soft drink
3. detergent, broom, beer, potato chips
4. milk, potato chips, diapers, carrots
5. cigarettes, meat, soft drinks
6. milk, toys, beer, potato chips
beer → potato chips
pattern (cluster)
raw data
pattern (association rule)

Fig. 1. Patterns and raw data in the supermarket example.

patterns. Thus, 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).

Example 1. Consider a supermarket database which records the items purchased by each customer within each sales transaction. While this large volume of data is not providing the supermarket management with any clear indication about the buying habits of customers, some knowledge discovery algorithm can be applied to come up with relevant knowledge. In Figure 1 both a clustering technique [15] and an algorithm for extracting association rules [7] have been applied. In the first case, patterns are clusters of customers which share some categories of products. In the second, patterns come in the form of association rules which relate sets of items frequently bought together by customers; the relevance of each rule is typically expressed by statistical measures which quantify its support and confidence. Note that each pattern, besides providing the end-user with some hidden knowledge over the underlying data, can be mapped to the subset of data it is related to.

Patterns, thus, can be regarded as artifacts which effectively describe subsets of raw data (thus, they are compact) by isolating and emphasizing some interesting properties (thus, they are rich in semantics). 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). Note that all the kinds of patterns, besides being somehow related to raw data, also imply some processing in order to either generate them through some learning algorithm or to check/map them against raw data.

We are now ready to give a preliminary, informal, definition for patterns; a formal definition will be given in Section 3.1.

Definition 1 (Pattern). *A pattern is a compact and rich in semantics representation of raw data.*

2.2 Architecture

Patterns can be managed by using a Pattern-Base Management System exactly as database records are managed by a database management system. A Pattern-Base Management System is thus defined as follows.

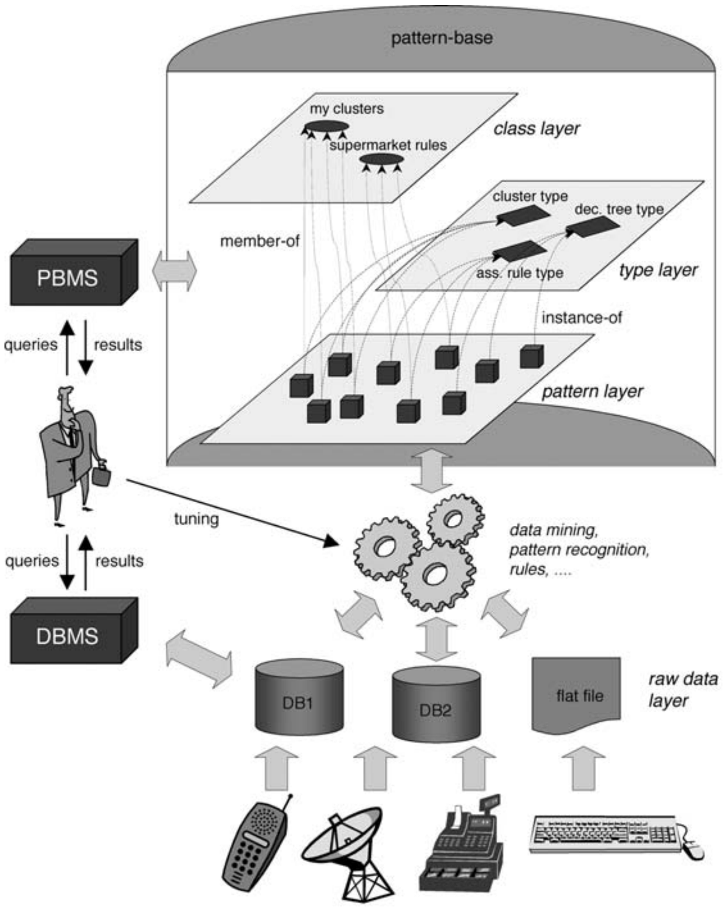

Fig. 2. The PBMS architecture.

Definition 2 (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 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 2. 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; note that, in our approach, these algorithms are loosely coupled with the PBMS. Within the PBMS, it is worth to distinguish three different layers:

1. The *pattern layer* is populated with patterns.
2. The *type layer* holds built-in and user-defined types for patterns. Patterns of the same type share similar structural characteristics.
3. 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 for modeling patterns 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 will be based on a specific, very common typing system. Assuming there is a set of *base types* (including the *root type* $\perp$) 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*.

Let base types include integers, reals, Booleans, strings, and timestamps; let type constructors include list, set, bag, array, and tuple. Using an obvious syntax, some examples of type declarations are (we use uppercase for base types and type constructors, lowercase for attributes):

– salary: REAL
– SET(INTEGER)
– TUPLE(x: INTEGER, y: INTEGER)
– personnel: LIST(TUPLE(age: INTEGER, salary: INTEGER))

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 of abstract data types in the object-oriented model.

Definition 3 (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.*

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.
– 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. The role of this component is to enable 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 [12].
– 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 try a constraint calculus based on polynomial constraints which seems suitable for several types of patterns [16]; still, a full exploration of the most suitable language is outside the scope of the paper.

Example 2. 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* [13]. A possible pattern type for modeling association rules over strings representing products is the following:

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

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.

Example 3. An example of a mathematical pattern is a straight line which interpolates a set of samples. In this case, the source schema models the samples, the structure schema includes the two coefficients necessary to determine a line, while the measure schema includes, for instance, a fitting quantifier. The formula which establishes the approximate correspondence between the pattern and the source data is the equation of the line.

$$\begin{aligned} n &: \mathsf{InterpolatingLine} \\ ss &: \mathsf{TUPLE(a{:}\ REAL,\ b{:}REAL)} \\ ds &: \mathsf{SET(sample{:}\ TUPLE(x{:}\ REAL,\ y{:}\ REAL))} \\ ms &: \mathsf{fitting{:}\ REAL} \\ f &: \mathsf{y = a \cdot x + b} \end{aligned}$$

3.2 Patterns

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 4 (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.*

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. In particular, the expression is obtained by the formula f in the pattern type by (1) instantiating each attribute appearing in ss with the corresponding value specified in s, and (2) letting the attributes appearing in ds range over the source space. Note that further information could be associated to each pattern, specifying for instance the mining session which produced it, which algorithm was used, which parameter values rule the algorithm, etc.

Example 4. Consider again pattern type AssociationRule defined in Example 2, 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:

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

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.

Example 5. Let raw data be an array of real values corresponding to samples periodically taken from a signal, and let each pattern represent a recurrent waveshape together with the position where it appears within the dataset and its amplitude shift and gain:

$$\begin{aligned}
n &: \text{TimeSeries}\\
ss &: \text{TUPLE(curve: ARRAY[1..5](REAL), position: INTEGER,}\\
&\quad \text{shift: REAL, gain: REAL)}\\
ds &: \text{samples: ARRAY[1..100](REAL)}\\
ms &: \text{similarity: REAL}\\
f &: \mathsf{samples}[\mathsf{position} + i - 1] = \mathsf{shift} + \mathsf{gain} \times \mathsf{curve}[i],\\
&\quad \forall i : 1 \leq i \leq 5
\end{aligned}$$

Measure similarity expresses how well waveshape curve approximates the source signal in that position. The formula approximatively maps curve onto the data space in position. A possible pattern, extracted from a dataset which records the hourly-detected levels of the Colorado river, is as follows:

$$\begin{aligned}
pid &: 456\\
s &: (\mathsf{curve} = (\mathsf{y} = 0, \mathsf{y} = 0.8, \mathsf{y} = 1, \mathsf{y} = 0.8, \mathsf{y} = 0),\\
&\quad \mathsf{position} = 12, \mathsf{shift} = 2.0, \mathsf{gain} = 1.5)\\
d &: \text{'colorado.txt'}\\
m &: \mathsf{similarity} = 0.83\\
e &: \{\mathsf{samples}[12] = 2.0, \mathsf{samples}[13] = 3.2, \mathsf{samples}[14] = 3.5,\\
&\quad \mathsf{samples}[15] = 3.2, \mathsf{samples}[16] = 2.0\}
\end{aligned}$$

3.3 Classes

A class is a set of semantically related patterns and constitutes the key concept in defining a 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. Formally, a class is defined as follows.

Definition 5 (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.*

Example 6. The *Apriori* algorithm described in [7] could be used to generate relevant association rules from the dataset presented in Example 4. All the generated patterns could be inserted in a class called *SaleRules* for pattern type AssociationRule defined in Example 2. The collection of patterns associated with the class can be later extended to include rules generated from a different dataset, representing for instance the sales transaction recorded in a different store.

4 Relationships between Patterns

In this section we introduce some interesting relationships between patterns aimed at increasing the modeling expressivity of our logical framework, but which also improve reusability and extensibility and impact the querying flexibility. For space reasons we will propose here an informal presentation; see [9] for formal details.

4.1 Specialization

Abstraction by specialization (the so-called *IS-A* relationship) is widely used in most modeling approaches, and the associated inheritance mechanism significatively addresses the extensibility and reusability issues by allowing new entities to be cheaply derived from existing ones.

Specialization between pattern types can be defined by first introducing a standard notion of subtyping between base types (e.g., integer is a subtype of real). Subtyping can then be inductively extended to deal with types containing type constructors: t_1 specializes t_2 if the outermost constructors in t_1 and t_2 coincide and each component in t_2 is specialized by one component in t_1. Finally, pattern type pt_1 specializes pattern type pt_2 if 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.

Note that, if pt_1 specializes pt_2 and class c is defined for pt_2, also the instances of pt_1 can be part of c.

Example 7. Given a set S of points, a clustering is a set of clusters, each being a subset of S, such that the points in a cluster are more similar to each other than points in different clusters [13]. The components for pattern type Cluster, representing circular clusters defined on a 2-dimensional space, are:

$$\begin{aligned} n &: \mathsf{Cluster} \\ ss &: \mathsf{TUPLE(radius{:}\ \bot,\ center{:}\ TUPLE(cx{:}\ \bot,\ cy{:}\ \bot))} \\ ds &: \mathsf{SET(x{:}\ \bot,\ y{:}\ \bot)} \\ ms &: \emptyset \\ f &: (\mathsf{x}-\mathsf{cx})^2 + (\mathsf{y}-\mathsf{cy})^2 \leq \mathsf{radius}^2 \end{aligned}$$

where f gives an approximate evaluation of the region of the source space represented by each cluster. While in Cluster the 2-dimensional source schema is generically defined, clusters on any specific source space will be easily defined by specialization. Thus, Cluster could be for instance specialized into a new pattern type ClusterOfIntegers where cx, cy, x, and y are specialized to integers, radius is specialized to reals, and a new measure avgIntraClusterDistance of type real is added.

4.2 Composition and Refinement

A nice feature of the object model is the possibility of creating complex objects, i.e. objects which consist of other objects. In our pattern framework, this can be achieved by extending the set of base types with pattern types, thus giving the user the possibility of declaring *complex types*. This technique may have two different impacts on modeling.

Firstly, it is possible to declare the structure schema as a complex type, in order to create patterns recursively containing other patterns thus defining, from the conceptual point of view, a *part-of* hierarchy. We will call *composition* this relationship.

Secondly, a complex type may appear within the source schema: this allows for supporting the modeling of patterns obtained by mining other existing patterns. Since in general a pattern is a compact representation of its source data, we may call *refinement* this relationship in order to emphasize that moving from a pattern type to the pattern type(s) that appear in its source entails increasing the level of detail in representing knowledge.

Example 8. Let pattern type ClusterOfRules describe a mono-dimensional 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:

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

where a standard dot notation is adopted to address the components of pattern types. Thus, there is a refinement relationship between ClusterOfRules and AssociationRule. 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. Thus, there would be a composition relationship between pattern types Clustering and ClusterOfRules.

5 Related Approaches

The most popular efforts for modeling patterns is the Predictive Model Markup Language [6], that uses XML to represent data mining models. Though PMML enables the exchange of patterns between heterogeneous pattern-bases, it does not provide any general model for the representation of different pattern types; besides, the problem of mapping patterns against raw data is not considered.

Among the other approaches, we mention the SQL/MM standard [2]; here, the supported mining models are represented as SQL types and made accessible through the SQL:1999 base syntax. A framework for metadata representation is proposed by the Common Warehouse Model [1], whose main purpose is to enable easy interchange of warehouse and business intelligence metadata between various heterogeneous repositories, and not the effective manipulation of these metadata. The Java Data Mining API [5] addresses the need for procedural support of all the existing and evolving data mining standards; in particular, it supports the building of data mining models as well as the creation, storage, access, and maintenance of data and metadata that represent data mining results. Finally, the Pattern Query Language is an SQL-like query language for patterns [3], assumed to be stored like traditional data in relational tables.

Overall, the listed approaches seem inadequate to represent and handle different classes of patterns in a flexible, effective, and coherent way: in fact, a given list of predefined pattern types is considered and no general approach to pattern modeling is proposed. In contrast, in our framework, the definition of a general model and the possibility of constructing new pattern types by inheritance from a root type allow uniform manipulation of all patterns.

A specific mention is deserved by *inductive databases*, where data and patterns, in the form of rules inducted by data, are represented together to be uniformly retrieved and manipulated [14]. Our approach differs from the inductive database one in different ways:

– While only association rules and string patterns are usually considered there and no attempt is made towards a general pattern model, in our approach no predefined pattern types are considered and the main focus lies in devising a general and extensible model for patterns.
– Patterns are far more complex than the raw data they represent and, we argue, cannot be effectively stored in a relational manner.
– The difference in semantics between patterns and raw data discourage from adopting the same query language for both, rather call for defining a *pattern*

query language capable of capitalizing on the peculiar semantics of each pattern component.
- Differently from [14], we claim that the peculiarities of patterns in terms of structure and behaviour, together with the characteristic of the expected workload on them, call for a logical separation between the database and the pattern-base in order to ensure efficient handling of both raw data and patterns through dedicated management systems.

We close this section by observing that, though our approach shares some similarities with the object-oriented model, there are several specific requirements calling for an ad-hoc model and for a dedicated management system:

- In terms of the logical framework, the discriminating feature is the requirement for a semantically rich representation of patterns, achieved by separating structure and measure on the one hand, by introducing the expression component on the other.
- From a conceptual point of view, the modeling framework adopted entails the refinement relationship between patterns, not directly supported by object-oriented models.
- From the functional point of view, instead of pointer-chasing object-oriented queries, novel querying requirements will presumably arise, including (a) ad hoc operations over the source and pattern spaces and their mapping; (b) pattern matching tests, strengthened by the separation of structure and measure; (c) reasoning facilities based on the expression component of patterns.
- In terms of system architecture, the relevance of queries aimed at evaluating similarity between patterns and the request for efficiency call for alternative storage and query optimization techniques.

6 Conclusions and Future Work

In this paper, we have dealt with the introduction of the architecture and the logical foundations for pattern management. First, we introduced Pattern-Base Management Systems and their architecture. Then, we provided the logical foundations of a general framework, as the basis for PBMS management. Our logical framework is based on the principles of generality, extensibility and reusability. To address the generality goal we introduced a simple yet powerful modeling framework, able to cover a broad range of real-world patterns. Thus, the most general definition of a pattern specifies its structure, the underlying data that correspond to it, an expression which is rich in semantics so as to characterize what the pattern stands for, and measurements of how successful the raw data abstraction is. To address the extensibility and usability goals, we introduced type hierarchies, that provide the PBMS with the flexibility of smoothly incorporating novel pattern types, as well as mechanisms for constructing composite patterns and for refining patterns.

Though the fundamentals of pattern modeling have been addressed, several important issues still need to be investigated. Future research includes both theoretical aspects as well as implementation-specific issues. Implementation issues involve primarily the construction of ad-hoc storage management and query processing modules for the efficient management of patterns. The theoretical aspects include the evaluation and comparison of the expressivity of different languages to express formulas, and the study of a flexible query language for retrieving and comparing complex patterns. In particular, as to query languages, a basic operation is that of comparison: two patterns of the same type can be compared to compute a score assessing their mutual similarity as a function of the similarity between both the structure and the measure components. Particularly challenging is the comparison between complex patterns: in this case, the similarity score is computed starting from the similarity between component patterns, then the obtained scores are aggregated, using an aggregation logic, to determine the overall similarity [10].

References

1. Common Warehouse Metamodel (CWM). `http://www.omg.org/cwm`, 2001.
2. ISO SQL/MM Part 6. `http://www.sql-99.org/SC32/WG4/Progression_Documents/FCD/fcd-datamining-2001-05.pdf`, 2001.
3. Information Discovery Data Mining Suite. `http://www.patternwarehouse.com/dmsuite.htm`, 2002.
4. The PANDA Project. `http://dke.cti.gr/panda/`, 2002.
5. Java Data Mining API. `http://www.jcp.org/jsr/detail/73.prt`, 2003.
6. Predictive Model Markup Language (PMML). `http://www.dmg.org/pmmlspecs_v2/pmml_v2_0.html`, 2003.
7. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th VLDB*, 1994.
8. M. Berry and G. Linoff. *Data mining techniques: for marketing, sales, and customer support.* John Wiley, 1996.
9. E. Bertino et al. A preliminary proposal for the PANDA logical model. Technical Report TR-2003-02, PANDA, 2003.
10. I. Bartolini et al. PAtterns for Next-generation DAtabase systems: preliminary results of the PANDA project. In *Proc. 11th SEBD*, Cetraro, Italy, 2003.
11. U. Fayyad, G. Piatesky-Shapiro, and P. Smyth. From data mining to knowledge discovery: an overview. In *Advances in Knowledge Discovery and Data Mining*, pages 1–34. AAAI Press and the MIT Press, 1996.
12. V. Ganti, R. Ramakrishnan, J. Gehrke, and W.-Y. Loh. A framework for measuring distances in data characteristics. *PODS*, 1999.
13. J. Han and M. Kamber. *Data mining: concepts and techniques.* Academic Press, 2001.
14. T. Imielinski and H. Mannila. A Database Perspective on Knowledge Discovery. *Communications of the ACM*, 39(11):58–64, 1996.
15. A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a survey. *ACM Computing Surveys*, 31:264–323, 1999.
16. P. Kanellakis, G. Kuper, and P. Revesz. Constraint Query Languages. *Journal of Computer and System Sciences*, 51(1):25–52, 1995.
17. P. Vassiliadis and T. Sellis. A survey of logical models for OLAP databases. *SIGMOD Record*, 28(4):64–69, 1999.