# A Framework for Data Mining Pattern Management

Barbara Catania[1], Anna Maddalena[1], Maurizio Mazza[1], Elisa Bertino[2], and Stefano Rizzi[3]

[1] University of Genova (Italy)
{catania,maddalena,mazza}@disi.unige.it
[2] Purdue University (IL)
bertino@cerias.purdue.edu
[3] University of Bologna (Italy)
srizzi@deis.unibo.it

**Abstract.** To represent and manage data mining patterns, several aspects have to be taken into account: (i) patterns are heterogeneous in nature; (ii) patterns can be extracted from raw data by using data mining tools (a-posteriori patterns) but also defined by the users and used for example to check how well they represent some input data source (a-priori patterns); (iii) since source data change frequently, issues concerning pattern validity and synchronization are very important; (iv) patterns have to be manipulated and queried according to specific languages. Several approaches have been proposed so far to deal with patterns, however all of them lack some of the previous characteristics. The aim of this paper is to present an overall framework to cope with all these features.

## 1 Introduction

In many different modern contexts, a huge quantity of raw data is collected. An usual approach to analyze such data is to generate some compact knowledge artifacts (i.e., clusters, association rules, frequent itemsets, etc.) through data processing methods, to make them manageable from humans while preserving as much as possible their hidden information or discovering new interesting correlations. Those knowledge artifacts, which can be very heterogeneous and complex, are also called *patterns*. Although a large variety of techniques for pattern mining exist, we still miss comprehensive environments supporting the development of *knowledge intensive* applications. Such an environment goes much beyond the use of pattern mining techniques; it has to provide support for combining heterogeneous patterns, for characterizing their temporal behavior, and for querying and manipulating them. In what follows we elaborate on these requirements.

**Heterogeneity**. There are many different application contexts from which various types of patterns can be generated and need to be managed. For example, in the market-basket analysis, common patterns are association rules, which identify sets of items usually sold together, or clusters, used to realize a market segmentation analysis. Moreover, we may be interested not only in patterns generated from raw data by using some data mining tools (a-posteriori patterns)

J.-F. Boulicaut et al. (Eds.): PKDD 2004, LNAI 3202, pp. 87–98, 2004.

but also in patterns known by the users and used for example to check how well some data source is represented by them (a-priori patterns).

**Temporal information.** Since source data change with high frequency, an important issue consists in determining whether existing patterns, after a certain time, still represent the data source from which they have been generated, possibly being able to change pattern information when the quality of the representation changes. Two different time information can be considered: (i) *transaction time*, i.e. the time the pattern "starts to live" in the system. For a-priori patterns, it is the instant when the user inserts the pattern in the system; for a-posteriori patterns, it is the instant when the pattern is extracted from raw data and inserted in the system; (ii) *validity period*, i.e., the time interval in which the pattern is assumed to be reliable with respect to its data source. The validity period can be either assigned by the user or by the system, depending on the quality of raw data representation (*semantic validity*) achieved by the pattern.

**Pattern languages.** Patterns should be manipulated (e.g. extracted, synchronized, deleted) and queried through a Pattern Manipulation Language (PML) and a Pattern Query Language (PQL). PML must support the management of a-posteriori and a-priori patterns. PQL must support both operations against patterns and operations combining patterns with raw data (cross-over queries).

Several approaches have been proposed so far to deal with patterns, however all of them lack some of the previous characteristics. Most of them deal with specific types of a-posteriori patterns, often stored together with raw data, and do not consider temporal information [6, 8, 12–15]. However, as it has been recognized [5], due to the quite different characteristics of raw data and patterns, to ensure an efficient handling of both, it could be better to use two dedicated systems: a traditional Data Base Management System (DBMS) for raw data and a specific Pattern Based Management System (PBMS) for patterns.

In this paper we propose a comprehensive framework to deal with patterns within a PBMS, addressing the above requirements, and we develop in details some key notions of the framework, such as: (i) a temporal pattern representation model, allowing one to associate time and validity with patterns; (ii) a temporal pattern manipulation language (TPML) and a temporal pattern query language (TPQL), supporting specialized predicates and operators to deal with temporal information. To the best of our knowledge this is the first proposal dealing with temporal aspects of pattern representation and management.

The remainder of the paper is organized as follows. In Section 2, the basic architecture and the pattern model are introduced. In Sections 3 and 4, the TPML and TPQL are discussed, respectively. Related work is then discussed in Section 5. Finally, Section 6 presents some conclusions and outlines future work.

## 2 The Pattern Model

**Pattern-Base Management System.** A *Pattern-Base Management System* (PBMS), first introduced in the context of the PANDA project [5], is a system for handling patterns defined over raw data.

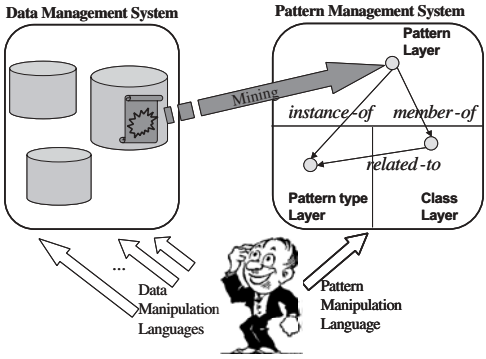


**Fig. 1.** PBMS Architecture

The overall architecture of the system is shown in Fig. 1. The *Data Management System* on the left-hand side of the figure deals with data collections, whereas the *Pattern Management System*, on the right-hand side, deals with patterns. The user may interact with both systems by mean of dedicated manipulation languages. Within the PBMS, we distinguish three different layers: (i) the *pattern layer*, which is populated with patterns (pattern-base); (ii) the *pattern type layer*, which holds built-in and user-defined types for patterns; (iii) the *class layer*, which holds definitions of pattern classes, i.e., collections of patterns. 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 querying patterns or recalculating them from raw data.

**Basic Model Concepts.** Based on the proposed architecture, the concepts at the basis of the pattern model are: pattern types, patterns, and classes (for additional details, see [15]). A *pattern type* is the intensional form of patterns, giving a formal description of their structure and relationship with source data. It is a record with five elements: (i) the *pattern name* $n$; (ii) the *structure schema* $s$, which defines the pattern space by describing the structure of the patterns instances of the pattern type; (iii) the *source schema* $d$, which defines the related source space by describing the dataset from which patterns are constructed; (iv) the *measure schema* $m$, which is a tuple describing the measures which quantify the quality of the source data representation achieved by the pattern; (v) the *formula* $f$, which describes the relationship between the source space and the pattern space, thus representing 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, in most cases it will describe it only approximatively.

Given a pattern type $pt$, a *mining function* $\mu$ for $pt$ takes as input a data source, applies a certain computation to it, and returns a set of patterns, instances of $pt$. We then call *measure function* the function computing the measures

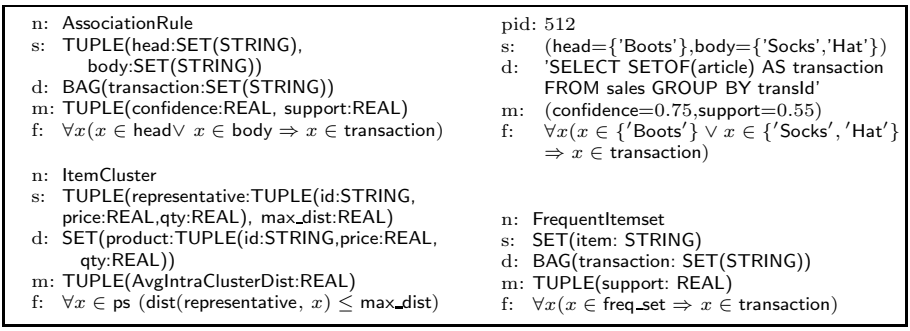


**Fig. 2.** Examples of pattern types and patterns

of patterns over a certain dataset. We store such information in some catalog of the pattern layer.

*Patterns* are instances of a specific pattern type containing: (i) a pattern identifier *pid*; (ii) a structure that positions the pattern within the pattern space; (iii) a source that identifies the specific dataset the pattern relates to[1]; (iv) a measure that estimates the quality of the raw data representation achieved by the pattern; (v) an instantiated formula, obtained from the one in the pattern type by instantiating each attribute appearing in $s$ with the corresponding value, and letting the attributes appearing in $d$ range over the source space. Dot notation and path expressions can be used to denote pattern components.

A *class* is a set of semantically related patterns of a certain pattern type and constitutes the key concept in defining a pattern query language.

*Example 1.* Consider the following scenario. A commercial vendor traces shop transactions and he applies data mining techniques to determine how he can further increase his sales. To this purpose, the vendor deals with several kinds of patterns: (i) *association rules*, representing correlations between items sold; (ii) *clusters of products*, grouping sold products with respect to their price and sold quantity; (iii) *frequent itemsets*, recording items most frequently sold together.

As an example, consider the pattern type for association rules in Fig. 2. The structure schema is a tuple modeling the head and the body as sets of strings representing products. 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 to assess the rule relevance: its confidence (what percentage of 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). The formula represents (exactly, in this case) the pattern/source data relationship by associating each rule with the set of transactions which support it. Now suppose that data related to sales transactions are stored in a relational table sales(transId, article, qty). Using an extended SQL

[1] When no otherwise stated, data sources are intensionally described as queries over the raw data.

syntax to denote the dataset, an example of an instance of AssociationRule (generated, for instance, by using the *Apriori* algorithm [10]) is presented in Fig. 2.

Other examples of pattern types are presented in Fig. 2. For instance, a cluster of products (represented by some numeric features) can be modeled by defining its representative element and the allowed maximal distance between each element in the cluster and its representative, whereas, a frequent itemset is just characterized by the set of items it represents. □

**Pattern Validity.** We extend the model proposed in [15] to deal with semantic and temporal validity issues. To this end, we assume that no temporal information is available from raw data and we associate each pattern with a *transaction time* and a *validity period*. *Transaction time* is automatically computed by the PBMS and points out when a pattern has been inserted in the system. This information cannot be changed by the user, thus it is just recorded in system catalogs. On the other hand, the *validity period* is the interval $[StartTime, EndTime)$ in which the pattern can be considered reliable with respect to raw data, and therefore usable. The validity period can be queried by the user, thus it must be inserted in the model. Moreover, we suppose that the validity period can be either assigned and managed by the user or by the system, depending on the operations performed over patterns (see Section 3). For the sake of simplicity, we deal with a fixed time granularity $tg$, chosen by the PBMS administrator. Thus, the validity period schema is always of type $[StartTime\colon tg, EndTime\colon tg)$ where $tg$ is fixed. Each pattern is then extended with a new component $vt$, representing the actual pattern validity period according to the chosen granularity.

Temporal validity specifies that the pattern is *assumed* to be valid in that period. However, since raw data change with a high frequency, the pattern, in its validity period, may not correctly represent raw data it is associated with. To this end, we also introduce the concept of *semantic validity* with respect to a data source and the notion of *safety*, for patterns that are both temporally and semantically valid in a certain instant of time. Note that semantic validity (and thus safety) can only be checked at an instant-by-instant base, since we do not know how the data source will change in the future and how it was in the past.

**Definition 1.** *Let $p$ be a pattern, with $p.m =< m_1 : v_1, ..., m_n : v_n >$, and $t$ an instant of time. Suppose each measure $m_i$ is associated with a boolean operator $\theta_i$ such that $v_1 \theta_i v_2$ means that $v_1$ is "better than" $v_2$. $p$ is temporally valid at $t$ if $t \in p.vt$. $p$ is semantically valid at $t$ with respect to a data source $D$ with thresholds $v_1, ..., v_n$ if and only if, $D \models_t p$ and $p.m.m_i\ \theta_i\ v_i$, $i = 1, ..., n$. $D \models_t p$ means that $p$ can be extracted from $D$ at time $t$. $p$ is safe at $t$ with respect to a data source $D$ with thresholds $v_1, ..., v_n$ if it is both temporally and semantically valid at $t$ with respect to $D$ and $v_1, ..., v_n$.* □

Semantic validity can be seen as a function of time. By checking semantic validity periodically, we may plot how measures change in the time.

*Example 2.* Consider the pattern in Fig. 2 (say $p$). Suppose $p.vt$ = [1-APR-04,31-MAR-05), thus $p$ is valid from 1-APR-04 to 31-MAR-05. For instance, $p$ is

temporally valid on 22-MAY-04. However, since raw data is continuously changing, it may be possible that, on 22-MAY-04, no transaction in the $p$ data source (say $D$) contains both 'Hat' and 'Boots'. Thus, the support and the confidence of $p$ in $D$ on 22-MAY-04 are 0. Thus, on 22-MAY-04, $p$ is not semantically valid with respect to $D$, for any threshold values, and it is not safe. □

## 3 Temporal Pattern Manipulation Language (TPML)

TPML must support primitives to generate patterns from raw data, to insert them in the PBMS, to delete, and to update patterns. These operations are defined by taking into account validity issues and differences between a-posteriori and a-priori patterns.

**Insertion Operations.** To cope with both a-posteriori and a-priori patterns, three different types of insertions are supported: extraction, direct insertion, and recomputation. Insertion operators must set the validity period of the interested patterns, which is an additional parameter of the operator. If the user does not specify any validity period, it is set by default to $[Current\_time, +\infty)$.

*Extraction* $\mathcal{E}(pt, d, cond, \mu, pr)$ extracts patterns of pattern type $pt$ from a raw dataset $d$ by applying mining function $\mu$. To these (a-posteriori) patterns the validity period $pr$ is assigned and they are inserted in the PBMS if they satisfy condition *cond*, defined by using predicates that will be presented in Section 4.1.

*Direct Insertion* $\mathcal{I}(pt, d, s, m, pr)$ allows the user to insert in the PBMS patterns from scratch (a-priori patterns) by taking as input a pattern type $pt$, a source $d$, a structure $s$, a tuple of measure values $m$, and a validity period $pr$.

*Recomputation* $\mathcal{R}(pt, cond, d, \mu_m, pr)$ generates new patterns from old ones, by recomputing their measures over a given raw dataset. More precisely, given the instances of a pattern type $pt$ satisfying a given condition *cond*, the measures of those patterns over a raw dataset $D$ are computed, accordingly to some input measure function $\mu_m$. New patterns are created and inserted into the system.

*Example 3.* Consider Example 1. At the end of every month, the vendor mines his transaction data to extract association rules and frequent sold item sets. A validity period is assigned to each extracted pattern, from the first day till the last day of the month. In order to generate such patterns, the extraction operator can be used (Fig. 3 lines 2-3 and 5) against the relational view $JuneSales$, storing information concerning sales in June, with mining functions $\mu_{aPriori}$ and $\mu_{FreqSeq}$. The extracted patterns are then inserted in classes $MinedAssociationRules$ and $UsedFrequentItemsets$, respectively (Fig. 3 lines 4 and 6) (see below for class-based operators). Furthermore, to specialize his advertising campaign, he mines clusters of products, based on numerical information concerning price and sold quantity of each product. Since the vendor does not know the expiration time of those clusters, he assumes they are always valid. To implement this behavior, the extraction operator is used against the relational view $Products$, storing information concerning sold products (Fig. 3 line 7), by using mining function $\mu_{SLink}$. Such patterns are then inserted in class $SoldItemClusters$ (Fig. 3 lines 8). □

```
1: /* Pattern generation */
2:  AR = E(AssociationRules, JuneSales, ⟨confidence ≥ 0.30, support ≥ 0.25⟩,
3:        μaPriori, [1-JUN-04, 30-JUN-04 ))
4:  FORALL i ∈ AR DO IC(i, MinedAssociationRules)
5:  FS = E(FrequentItemset, JuneSales, support ≥ 0.30, μFreqSeq, [1-JUN-04, 30-JUN-04 ))
6:  FORALL j ∈ FS DO IC(j, UsedFrequentItemsets)
7:  CS = E(ItemCluster, Products, AvgIntraClusterDist ≤ 0.20, μSLink)
8:  FORALL k ∈ CS DO IC(k, SoldItemClusters)
9:/* Pattern deletion */
10: δ(AssociationRule, support ≤ 0.50)
11:/* A-priori pattern management */
12: v = I (FrequentItemset, DS, {A, B}, < 0 >)
13: IC(v, UsedFrequentItemset)
14: S(v, true, μFreqSeq)
15: /* Restoring temporal validity */
16: DC(Current_time after vt, c)
17: /* Promotion of a new product */
18: W = π(<representative>,<>) (σf(P) (SoldItemClusters))
19: W ⋈s.representative∈s.head,cf (MinedAssociationRules)
20: S(AssociationRule, true, μaPriori)
21: S(FrequentItemset, true, μFreqSeq)
22: S(ItemCluster, true, μSLink)
```

**Fig. 3.** Pattern manipulation session

**Deletion Operator.** $\delta(pt, cond)$ removes the instances of pattern type $pt$ satisfying condition $cond$ from the pattern layer if they belong to no class.

*Example 4.* Consider Example 1. Suppose the vendor is no more interested in association rules with support lower than 0.50. Thus he performs a deletion (Fig. 3 line 10). Note that association rules extracted at lines 2-3 are not deleted since they have already been inserted in a class. □

**Update Operators.** We assume that only measures and the validity period can be changed, by using the following operators.
*New_Period* $\mathcal{N}_P(pt, cond, pr)$ updates a set of patterns, instances of a pattern type $pt$ and satisfying a condition $cond$, by setting the validity period to $pr$. Note that this operator does not recompute the measure values of the patterns.
*Synchronize* $\mathcal{S}(pt, cond, \mu_m)$ makes patterns safe *without changing* the validity period. More precisely, it re-computes the measure values associated with temporally valid patterns, instances of a pattern type $pt$ and satisfying a condition $cond$, to reflect data source modifications, by using the input measure function $\mu_m$. Only temporally valid patterns are synchronized since all the others, by definition of safety, cannot become safe.
*Validate* $\mathcal{V}(pt, cond, \mu_m)$ makes patterns safe *by changing* their validity period. More precisely, it first recomputes the measure values associated with temporally valid patterns, instances of a pattern type $pt$ and satisfying a condition $cond$. If such measure values are better than the ones associated with patterns before validation, patterns are semantically valid. Thus, similarly to synchronization, measures are modified, and the validity period is left unchanged. On the other hand, if measures are worst than before, the validity period of the pattern is changed, setting the end time to *Current_time* and a new pattern is created, with the same structure and dataset than the previous one, but with the new measures

and validity period $[Current\_time, +\infty)$. Since, after validation, patterns are semantically valid at the starting and ending points of their validity period, it is possible to use the validity period as an approximation of the periods in which a pattern is semantically valid.

*Example 5.* Consider Example 1. After a certain period of time, the vendor receives information about sale transactions of a new shop (suppose those data are stored in a dataset $DS$). Suppose he wants to trace information concerning how often a product $A$ (e.g. milk) and another product $B$ (e.g. cookies) are sold together in this shop. To this purpose, he first inserts the pattern representing such itemset in the system with measure value equal to 0. To this purpose, he uses a direct insertion operation (Fig. 3 lines 12-13) and then he synchronizes the pattern with raw data to get the right frequency (Fig. 3 line 14). □

**Operators for Classes.** According to our model, a pattern must be inserted in at least one class in order to be queried. Thus, two TPML operations are provided: insertion, $\mathcal{I}_{\mathcal{C}}(p, c)$, of a pattern $p$ into class $c$, and deletion, $\mathcal{D}_{\mathcal{C}}(cond, c)$, of all the patterns in class $c$ satisfying condition *cond*. Note that the deletion operator just removes patterns from a class but leaves them in the system.

*Example 6.* Consider Example 1. In general, the user may be interested in restoring the temporal validity of a certain class $c$, i.e. he may want to delete from $c$ all patterns that are not temporally valid at the current time. Such behavior can be achieved by using the $\mathcal{D}_{\mathcal{C}}$ operator, by using a temporal predicate in its condition (Fig. 3 line 16). □

## 4 Temporal Pattern Query Language (TPQL)

TPQL supports the retrieval of patterns from the PBMS, taking temporal issues into account. Each operator of TPQL takes classes as input and returns a set of patterns as output. Moreover, *cross-over operators*, binding patterns with raw data, are provided. In the following, before presenting the TPQL operators, some useful predicates are identified.

### 4.1 Pattern Predicates

**Predicates over Pattern Components.** Let $p_1$ and $p_2$ be two patterns. The general forms of a predicate over pattern components are $t_1 \theta t_2$ and $t_1 \theta o$, where $t_1$ and $t_2$ are path expressions that denote components of patterns $p_1$ and $p_2$, of *compatible* type, $o$ is a constant suitable for the type of $t_1$, and $\theta$ is an operator, suitable for the type of $t_1$, $t_2$, and $o$. We consider the following special cases:

- If $t_1$ and $t_2$ are data sources, then $\theta \in \{=^i, \subseteq^i, =^e, \subseteq^e\}$. Constants $o$ in this case are queries characterizing a dataset. $=^i$ stands for equivalence and $\subseteq^i$ for containment between intensional data source descriptions (i.e., between queries). These predicates do not require accessing raw data and can be

checked by using results obtained in the literature for queries. On the other hand, $=^e$ and $\subseteq^e$ are checked by accessing raw data (thus, they are cross-over predicates). More precisely, $t_1 =^e t_2$ if and only if $\forall x\ (x \in t_1 \Leftrightarrow x \in t_2)$ and $t_1 \subseteq^e t_2$ if and only if $\forall x\ (x \in t_1 \Rightarrow x \in t_2)$.
- If $t_1$ and $t_2$ are pattern formulas, then $\theta \in \{\equiv, \preceq\}$. $t_1 \equiv t_2$ is true if and only if $t_1$ and $t_2$ are equivalent formulas; $t_1 \preceq t_2$ is true if and only if $t_1$ logically implies $t_2$. Given a tuple $o$, containing one value for each free variable in $t_1$, $t_1(o)$ is true if and only if $t_1$ instantiated with the values in $o$ is true.
- If $t_1$ and $t_2$ are validity periods, then $\theta \in \{equals, before, meets, overlaps, during, starts, finishes\}$. The meaning of such predicates is defined in [16]. $o$ in this case is a temporal value, according to the chosen granularity.

**Predicates over Patterns.** In the following, $p_1$ and $p_2$ are patterns.

- *Identity* $(=)$. $p_1 = p_2$ if $p_1.pid = p_2.pid$.
- *Shallow equality* $(=^s)$. $p_1 =^s p_2$ if their corresponding components, except for *pid* and the validity period $v$, are equal. For the data source, we consider intensional equality.
- *Intensional subsumption* $(\preceq^i)$. $p_1 \preceq^i p_2$ if they have the same structure but $p_1$ represents a smaller set of raw data, i.e. $p_1.s = p_2.s$, $p_1.d \subseteq^i p_2.d$ and $p_1.f \preceq p_2.f$.
- *Extensional subsumption* $(\preceq^e)$. $p_1 \preceq^e p_2$ if they have the same structure but $p_1$ represents a smaller set of raw data through the considered formula, i.e. $p_1.s = p_2.s$ and $p_1.d_{\lceil p_1.f} \subseteq p_2.d_{\lceil p_2.f}$, where $d_{\lceil f}$ represents the set of source data items satisfying the formula.
- *Goodness* $(\nearrow)$. $p_1 \nearrow p_2$ if they have the same pattern type, $p_1 \preceq^e p_2$, and $p_1$ measures are better than $p_2$ measures, i.e., assuming that $pt.m = \langle m_1, ..., m_n \rangle$, $p_1.m_i \theta_i p_2.m_i$, $i = 1, ..., n$ [2].
- *Temporal validity* $(\omega_T)$. Given a pattern $p_1$ and a temporal value $t$, $\omega_T(p_1, t)$ is true if and only if $p_1$ is temporally valid at time $t$.
- *Semantic validity* $(\omega_S)$. Given a pattern $p$ of type $pt$, a data source $D$, a measure function $\mu_m$ for $pt$, and some thresholds $v_1, ..., v_n$, $\omega_S(p, D, \mu_m, < v_1, ..., v_n >)$ is true if and only if $p$ is semantically valid with respect to $D$ and $v_1, ..., v_n$, assuming to compute measure values by using $\mu_m$.

Note that $\preceq^e$, $\nearrow$, and $\omega_S$ are cross-over predicates.

### 4.2 Query Operators

**Basic Operators.** In the PBMS framework, queries are executed against classes. Besides typical relational operators (such as renaming, set-based operators), several other query operators are proposed (see Table 1). For example, projection is revisited to project out structure and measure components. The selection operator allows one to select patterns belonging to a certain class

[2] According to Def. 1, $\theta_i$ is a predicate expressing that $p_1.m_i$ is "better than" $p_2.m_i$.

**Table 1.** TPQL basic operators

| Name | Operator | Description |
|---|---|---|
| Projection | $\pi_{(l\_s, l\_m)}(c)$ where:<br>$c$ is a pattern class, $l\_s$ is a non empty list of attributes appearing the pattern structure, and $l\_m$ a list of attributes appearing in the pattern measure | it reduces the structure and the measures of the patterns in $c$ by projecting out components not appearing in $l\_s$ and $l\_m$ |
| Selection | $\sigma_F(c)$ where:<br>$c$ is a class and $F$ is a selection predicate | it selects the patterns in $c$ satisfying $F$ |
| Join | $c_1 \bowtie_{F,cf} c_2$ where:<br>$c_1$ and $c_2$ are two classes,<br>$F$: join predicate , and $cf$: composition function | it combines patterns belonging to $c_1$ and $c_2$, if they satisfy the join predicate $F$; each new pattern is generated by using the composition function $cf$ |

satisfying a certain condition, using any predicate introduced in Section 4.1. When using cross-over predicates, it becomes a cross-over operator. Finally, the join operator combines patterns belonging to two different classes, with possibly different pattern types. It requires the specification of a join predicate and a composition function, which defines the pattern type of the result.

**Temporal Operators.** Since we deal with temporal information associated with patterns, the need arises of querying such information. By using the proposed query operators (especially selection and join) and the predicates defined over validity periods (see Section 4), several interesting temporal queries can be specified. For instance, the user may be interested in retrieving from a certain class $c$, at a fixed instant of time (e.g. 'now'), all safe patterns. To this purpose, selection can be used as follows: $\sigma_{\omega_S(p,p.d,\mu_m,v)\wedge\omega_T(p,'now')}(c)$ [3]. As another example, retrieval of the patterns belonging to a certain class $c$, which are temporally valid in a given interval of time (e.g. a certain year), can be specified as follows: $\sigma_{vt\ during\ [01-JAN-03,31-DEC-03)}(c)$.

**Cross-over Operators.** They correlate patterns with raw data, providing a way for navigating from the pattern layer to the raw data layer and vice versa.
*Drill-Through* $\gamma$. It allows one to retrieve the subset of source data associated with at least one pattern in a class $c$, satisfying condition *cond*:

$\gamma(c, cond) = \{x | \exists p \in c, cond(c) = true, x \in p.d\}$.

*Data Covering* $\theta_d$. Let $p$ be a pattern of type $pt$, $D$ a data source, $\mu$ a mining function for patterns of type $pt$, and $v =< v_1, ..., v_n >$ some user-specified thresholds. Data covering allows us to determine the subset of source data represented by at least one pattern in the class. To this purpose, the formula is used:

$\theta_d(c, D, \mu) = \{x | x \in D, \exists p \in c, p.f(x) = true\}$.

*Cross-over selection.* When using a cross-over predicate within a selection, we need to access raw data to execute the query. For example, suppose that $c$ is a class of association rules and $D$ a dataset suitable for patterns in $c$. The query $\sigma_{D \subseteq^e d \wedge support > 0.6}(c)$ returns all rules in $c$ representing a superset of $D$, with a support greater than 0.6.

[3] $p$ denotes a generic pattern in $c$.

*Example 7.* Consider Example 1. Suppose that from April 2005, the vendor will start to sell a certain product $P$ and he wants to know how $P$ can be promoted. To do that, he looks for a correlation between $P$ and some other items sold. With such information, he may activate an advertising campaign to promote some other product he already sells in order to stimulate the demand for $P$. In this way, when he starts to sell $P$, probably customers will start to buy it without the need for a dedicated advertising campaign. A possible approach could be the following. First of all it determines in which cluster of products $P$ belongs and gets the representative $R$, by using a selection and a projection operator (Fig. 3 line 18). Then, he determines which products stimulate the sale of $R$ by considering bodies of association rules having $R$ in their head. This result can be achieved by performing a join operation (Fig. 3 line 19) between patterns just retrieved and association rules already mined. We assume that the used composition function returns patterns representing the bodies of the selected association rules. Products in association rule bodies are such that whenever a customer buys one of them, with a high probability he buys also $R$. Since $P$ is in the cluster represented by $R$, $P$ and $R$ are similar with respect to customer preferences, thus it is most likely that when the vendor starts to sell $P$, customers will behave as for $R$. When, on April 1 2005, the vendor starts to sell $P$, new data are collected and patterns previously extracted may become unreliable. Thus, a synchronization is required between data and patterns (Fig. 3 lines 20-22). □

## 5 Related Work

Several approaches have been proposed to model patterns. Among standardization efforts for modeling patterns, we recall the Predictive Model Markup Language (PMML) [4], the ISO SQL/MM standard [3], and the Common Warehouse Model (CWM) framework [2]. Although these approaches represent a wide range of data mining results, they do not provide a generic model to handle arbitrary pattern types. Furthermore, their main purpose is to enable an easy interchange of metadata not their effective manipulation.

In inductive databases, data and patterns are stored and queried together [1, 7, 8, 12]. They rely on specific (but extensible) types of patterns and are primarily focused on a-posteriori patterns. Moreover, validity is not considered. Within this framework, the entire knowledge discovering process is a querying process [12–14]. However, new SQL-based operators do not allow the user to specify specific mining functions [9]. Moreover, none of the proposed languages deals with pattern validity and synchronization aspects.

In [11], the authors propose a unified algebraic framework for multi-step knowledge discovery. Similarly to our approach, they model different types of patterns and maintain data and patterns separated. However, temporal information is not considered.

Previous work strictly related to the work presented here has been reported in two previous papers by us and other authors [6, 15], where a model for patterns and a pattern query language have been proposed. The major differences between that work and the one presented here is the extension with temporal features. We

believe that this is relevant extension from both a theoretical and architectural point of view.

## 6 Concluding Remarks

In this paper, we presented a general framework for patterns representation and management, taking into account validity information, a-priori and a-posteriori patterns. The resulting framework seems general enough to cope with real data mining applications. We are currently working on the development of a prototype of the proposed framework. Future work includes the definition of a pattern calculus, equivalent to the proposed algebra, the analysis of their expressive power and complexity, and the comparison of the expressive power of existing approaches to deal with patterns. We also plan to further investigate semantic validity to extend temporal analysis capabilities for patterns.

## References

1. The CINQ project. `http://www.cinq-project.org`, 1998-2002
2. Common Warehouse Metamodel (CWM). `http://www.omg.org/cwm`, 2001.
3. ISO SQL/MM Part 6. `http://www.sql-99.org/SC32/WG4/Progression_Documents/FCD/fcd-datamining-2001-05.pdf`, 2001.
4. Predictive Model Markup Language (PMML). `http://www.dmg.org/pmmlspecs_v2/pmml_v2_0.html`, 2003.
5. The PANDA Project. `http://dke.cti.gr/panda/`, 2002.
6. E. Bertino, B. Catania, and A. Maddalena. Towards a Language for Pattern Manipulation and Querying. In *Proc. of the 1st Int. Workshop on Pattern Representation and Management (PaRMa'04)*, 2004.
7. J. F. Boulicaut, M. Klemettinen, and H. Mannila. Modeling KDD Processes within the Inductive Database Framework. In *Proc. of the Data Warehousing and Knowledge Discovery*, pages 293–302, 1999.
8. L. De Raedt. A Perspective on Inductive Databases. *ACM SIGKDD Explorations Newsletter*, 4(2):69–77, 2002.
9. B. Goethals and J. Van den Bussche. A Priori versus a Posteriori Filtering of Association Rules. In *Proc. of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1999.
10. J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Academic Press, 2001.
11. T. Johnson, L.V.S Lakshmanan, and R.T. Ng. *The 3W Model and Algebra for Unified Data Mining*. In *Proc. of the 26th Int. Conf. on Very Large Data Bases*, 2000.
12. T. Imielinski and H. Mannila. A Database Perspective on Knowledge Discovery. *Communications of the ACM*, 39(11):58–64, 1996.
13. T. Imielinski and A. Virmani. MSQL: A Query Language for Database Mining. *Data Mining and Knowledge Discovery*, 2(4):373–408, 1999.
14. R. Meo, G. Psaila, and S. Ceri. An Extension to SQL for Mining Association Rules. *Data Mining and Knowledge Discovery*, 2(2):195–224, 1999.
15. S. Rizzi et Al. Towards a Logical Model for Patterns. In *Proc. of the 22nd Int. Conf. on Conceptual Modeling (ER 2003)*, 2003.
16. R. T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer, 1995.