Modeling and Language Support for the Management of Pattern-Bases [⋆]

Manolis Terrovitis [a,∗], Panos Vassiliadis [b], Spiros Skiadopoulos [c], Elisa Bertino [d], Barbara Catania [e], Anna Maddalena [e], and Stefano Rizzi [f]

[a]*School of Electrical and Computer Eng., Nat'l Tech. Univ. of Athens, Hellas*

[b]*Dept. of Computer Science, Univ. of Ioannina, Hellas*

[c]*Dept. of Computer Science, Univ. of Peloponnese, Hellas*

[d]*Dept. of Computer Science, Purdue Univ., USA*

[e]*Dept. of Computer and Information Science, Univ. of Genoa, Italy*

[f]*Dept. of Electronics, Computer Science and Systems, Univ. of Bologna, Italy*

Abstract

Information overloading is today a serious concern that may hinder the potential of modern web-based information systems. A promising approach to deal with this problem is represented by knowledge extraction methods able to produce artifacts (also called patterns) that concisely represent data. Patterns are usually quite heterogeneous and voluminous. So far, little emphasis has been posed on developing an overall integrated environment for uniformly representing and querying different types of patterns. In this paper we consider the larger problem of modelling, storing, and querying patterns, in a database-like setting and use a Pattern-Base Management System (PBMS) for this purpose. Specifically, (a) we formally define the logical foundations for the global setting of pattern management through a model that covers data, patterns, and their intermediate mappings; (b) we present a formalism for pattern specification along with safety restrictions; and (c) we introduce predicates for comparing patterns and query operators.

Key words: Pattern Bases, Pattern-Base Management Systems, Data modelling, Knowledge warehousing

[⋆] This work was partially funded by the European Commission and the Hellenic Ministry of Education through EPEAEK II and the PANDA IST Thematic Network.

[∗] Corresponding author. Tel.: +30-210-7721402, Email: mter@dbnet.ece.ntua.gr

1 Introduction

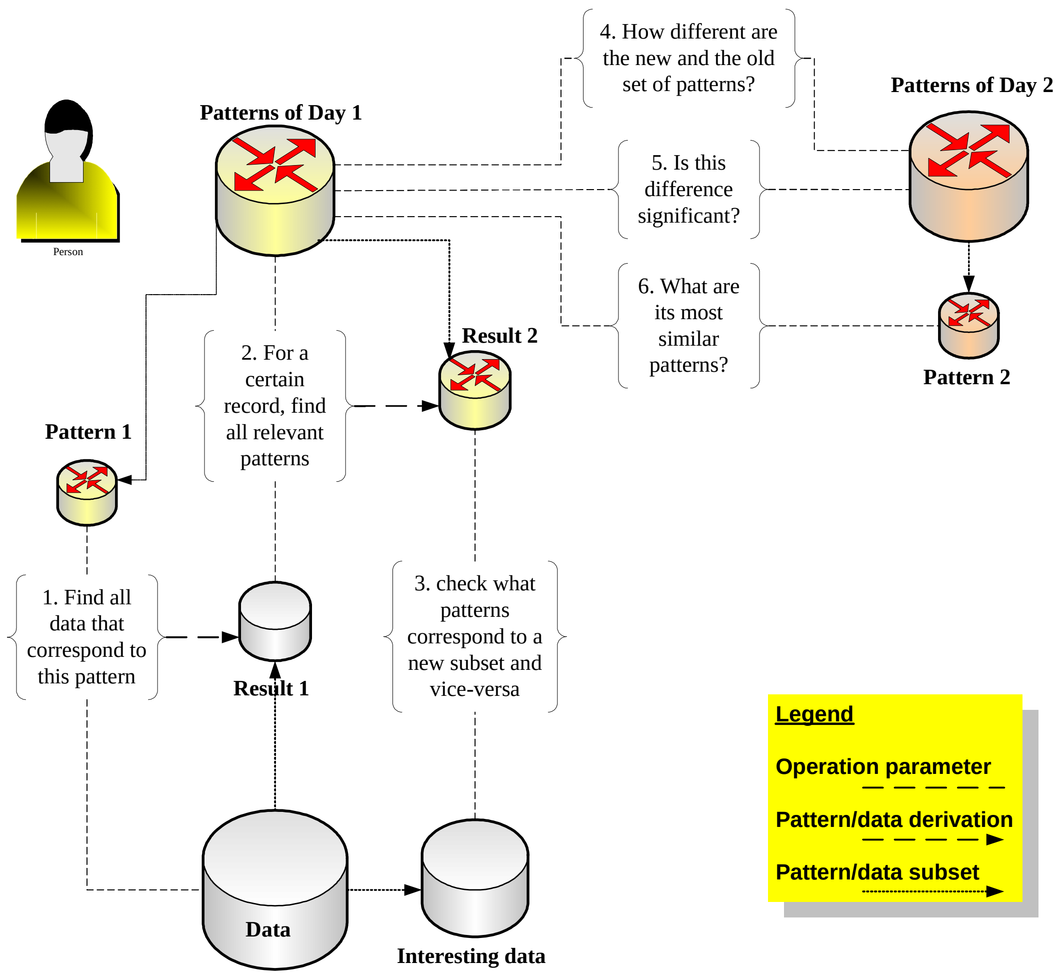
Nowadays, we are experiencing a phenomenon of information overload, which escalates beyond any of our traditional beliefs. As a recent survey states [1], the world produces between 1 and 2 exabytes of unique information per year, 90% of which is digital and with a 50% annual growth rate. Clearly, this sheer volume of collected data in digital form calls for novel information extraction, management, and querying techniques, thus posing the need for novel Database Management Systems (DBMSs). Still, even novel DBMS architectures are insufficient to cover the gap between the exponential growth of data and the slow growth of our understanding [2], due to our methodological bottlenecks and simple human limitations. To compensate for these shortcomings, we reduce the available data to *knowledge artifacts* (e.g., clusters, association rules) through data processing methods (pattern recognition, data mining, knowledge extraction) that reduce their number and size (so that they are manageable by humans), while preserving as much as possible from their hidden/interesting/available information. Again, the volume, diversity, and complexity of these knowledge artifacts make their management by a DBMS-like environment imperative. In the remainder of this document, we will refer to all these knowledge artifacts as *patterns*.

So far, patterns have not been adequately treated as *persistent objects* that can be stored, retrieved, and queried. Thus, the challenge of integration between patterns and data seems to be achievable by designing fundamental approaches for providing database support to pattern management. In particular, since patterns represent relevant knowledge, often very large in size, it is important that such knowledge is handled as *first-class* citizen. This means that *patterns should be modelled, stored, processed, and queried in a similar way to data in traditional DBMSs.*

Motivating example. To show the benefits of treating patterns as first class citizens, we briefly discuss a scenario involving a manager interacting with a set of stored patterns that he can query through the facilities offered by a *Pattern Base Management System* (PBMS) [3] (Fig. 1). We will assume the existence of data for the customers and the transactions of two different supermarket branches organized in an object-relational database like the following:

```
CustBranch1(id:int,name:varchar,age:int,income:int,sex:int)
CustBranch2(id:int,name:varchar,age:int,income:int,sex:int)
TransBranch1(id:long int,customerID:int,totalSum:euro,items:{int})
TransBranch2(id:long int,customerID:int,totalSum:euro,items:{int})
```

The relations `CustBranch1`, `CustBranch2` hold data for customers from two different branches of the supermarket and the relations `TransBranch1`, `TransBranch2` hold data for the transactions performed at the respective branches. The sce-



4. How different are the new and the old set of patterns?
Patterns of Day 2
Patterns of Day 1
Person
5. Is this difference significant?
6. What are its most similar patterns?
Result 2
2. For a certain record, find all relevant patterns
Pattern 2
Pattern 1
1. Find all data that correspond to this pattern
Result 1
3. check what patterns correspond to a new subset and vice-versa
Legend
Operation parameter
Pattern/data derivation
Pattern/data subset
Data
Interesting data

Fig. 1. An example of pattern generation scenario

nario takes place during two different days.

Day 1: A data mining algorithm is executed over the underlying data and the results are stored in the pattern base. For example, the data mining algorithm involves the identification of frequent itemsets. Possible patterns stored in the pattern base involve sets of items that are frequently bought together; e.g., {`bread,butter`}. The user is notified on this event and starts interacting with the stored data and patterns. During the early stages of this interaction the user is navigating back and forth from the pattern to data space.

Which are the data represented by a pattern? The user selects an interesting pattern and decides to find out what are the underlying data that relate to it.

Which are the patterns representing a data item? Out of a vast number of data that the user receives as an answer, he picks an interesting record and decides to see which other patterns relate to it (so that he can relate the originating pattern with these ones, if this is possible).

Does a pattern represent adequately a data set? This process of cross-over operations goes on for a while, as the user gets accustomed to the data. Then, he decides to proceed to some more elaborate analysis, consisting of "what-if" operations. Therefore, he arbitrarily chooses another dataset (on the basis of some assumption) and checks which of the available patterns can adequately represent it. Then, on the basis of the results, he selects a certain pattern and tries to determine which sets of data correspond to this pattern. This interaction with the stored data and patterns goes on for some time.

Could we derive new patterns from knowledge in the pattern base? The user may exploit patterns stored in the pattern base in order to derive or even to conjecture the existence of new patterns representing existing data without applying any time consuming mining technique but using available query operators.

Day 2: Data in the source databases change, therefore, the original data mining algorithms are executed on a regular basis (e.g., weekly) in order to capture different trends on the data. Once a second version of the patterns is computed (for the new data this time), the analyst is interested in determining the changes that have taken place.

How different are two pattern classes? First, the user wants to find out which are the new patterns and which patterns are missing from the new pattern set. This involves a simple qualitative criterion: presence or absence of a pattern.

Is the difference of two pattern classes significant? Then, the user wants some quantitative picture of the pattern set: which patterns have significant differences in their measures compared to their previous version?

How similar are two patterns? Finally, the user wants to see trends: how could he predict that something is about to happen? So, he picks a new pattern and tries to find its most similar pattern in the previous pattern set, through a similarity operator.

Why is the problem hard to solve with the current state-of-practice?
Assuming that someone would like to treat patterns as first-class citizens (i.e., store and query them), a straightforward approach would be to employ standard object-relational or XML database technology to store patterns and exploit its query facilities to navigate between the two spaces and pose queries. Although it is feasible to come up with an implementation of this kind, several problems occur. A preliminary object-relational implementation was pursued in the PANDA project [28] and we briefly list its findings here. First, we need to address the problem of having a generic structure able to accommodate all kinds of patterns; unfortunately, standard relational, and even object-relational technology results in a schema which is too complicated. As typically happens with relational technology, such a complex structure has a direct overhead in the formulation of user queries, which become too lengthy

and complicated for the user to construct. An XML database could probably handle the heterogeneity problem gracefully; nevertheless, the main strength of XML databases lies in the efficient answering of path queries. Unfortunately, the particularities of the (pointer-like) navigation between the data and the pattern space as well as the novel query operations concerning collections of patterns are not compatible with this nature of XML path queries.

Based on all the above, the situation calls for usage of customized techniques. In particular, we need both a logical model for the concise definition of a pattern base and an appropriate query language for the efficient formulation of queries. As we shall see, in this paper we treat the problem from a blank sheet of paper and discuss the management of patterns as a sui-generis problem. It is also interesting to point out that having an appropriate logical model, allows its underlying support by an object-relational or an XML storage engine in a way that is transparent to the user.

Why is the problem novel? Both the standardization and the research community have made efforts close to the main problem that we investigate; nevertheless, to the best of our knowledge there is no approach that successfully covers all the parameters of the problem in a coherent setting.

So far, *standardization efforts* [4,5,6] have focused towards providing a common format for describing patterns with the aim of interchangeability. The storage and eventual querying of such patterns has not been taken into consideration in the design of these approaches. A much deeper approach, similar to our own, has been performed in the field of *inductive databases*: a particular characteristic of inductive databases is that the querying process treats data and patterns equally. Nevertheless, the inductive database community avoids the provision of a generic model, capable of handling any kind of patterns and focuses, instead, on specific categories of them. Therefore, our approaches are complementary since the inductive database community follows a bottom-up approach, with individual cases of patterns considered separately, whereas we follow a top-down approach with a generic model that aims to integrate any kind of patterns. Finally, it is interesting to point out the relationship to the *data warehousing* approach, where data are aggregated and stored in application-oriented data marts. We follow a similar, still broader, approach, in the sense that although data warehousing aggregation is one possible way to provide semantically rich information to the user in a concise way, it is not the only one. As already mentioned, we try to pursue this goal in an extensible fashion, for a significantly broader range of patterns.

Problem and main idea. *The problem that we address is focused towards providing a generic and extensible model for patterns, along with the necessary languages for their definition and manipulation.* The issues that arise in this context are specifically: (a) the accommodation of any kind of patterns through a generic model, (b) the imposing of a logical structure to the vast space of patterns and the organization of patterns in semantically rich, easy to query collections, (c) the ability to navigate between the data and the pattern space and the ability to interrelate the corresponding objects of the two worlds, and, (d) the ability to perform novel types queries both over data, and, most importantly, over collections of patterns.

The importance of a logical model is paramount in this effort. As usual, a logical model provides the intermediate mathematical formulation that gracefully bridges the subjective, conceptual user view to the actual physical implementation. Moreover, apart from the formal view on the internal structure of data that the logical model imposes, it also forms the basis upon which a query language can be defined.

Contributions. The main idea of this paper is the organization of patterns and data in a *Pattern Warehouse* environment that (a) is based on the formal foundations of a well specified logical model and (b) provides the necessary mechanisms for the querying of patterns and data. Specifically, we make the following contributions:

- First, we formally define the foundations for the global setting of pattern management through a logical model that covers data, patterns and intermediate mappings. The model covers typing issues, both for data and patterns, with the later provision aiming towards being able to define reusable pattern types that are customized per case. Moreover, the model allows the database designer to organize semantically similar patterns in the appropriate collections that can be subsequently queried, just like relations in traditional databases. A distinguishing provision of our model is the ability to relate patterns and relevant data either explicitly, through intermediate, pointer-based mappings, or approximately, through declarative expressions.
- Second, we discuss language issues in the context of pattern definition and management. In particular, we present a declarative formalism for defining the approximate relation between the patterns and the data space. The approximate formulae that perform this approximate mapping are complementary to the precise mappings (e.g., for the case that the latter are absent). Safety issues are also discussed in this context.
- Third, we provide a characterization of pattern relationships (disjointness, equivalence, subsumption, similarity and equality) and we discuss their completeness and minimality. This fundamental set of relations is later exploited in the formation of the appropriate predicates on patterns, which are part of the query language that we introduce.
- Fourth, we introduce query operators for patterns and discuss their usage and semantics. Our query operators involve (a) the selection of patterns satisfying some criteria, (b) the reconstruction or combination of existing patterns to form new ones and (c) the navigation from the data to the

pattern space and vice versa.
- Finally, we describe PSYCHO (*Pattern base management SYstem arCHitecture prOtotype*) [7], a PBMS prototype implementation, developed in order to assess the usability of the proposed pattern model and languages.

A short version of this paper appears in [8]. In this paper, we extend [8] as follows: (a) we provide a fundamental classification of both precise pattern relationships, upon which we define appropriate predicates; (b) a proof of completeness for this classification; (c) a similar classification, along with the appropriate predicates and completeness proofs for approximate pattern relationships, (d) a novel family of similarity operators for patterns, (e) a thorough discussion and exemplification of the components of the Pattern Warehouse throughout all the paper, with particular emphasis on the formal definition and discussion of the operators of the query language, and finally, (f) a brief presentation of a prototype implementation.

The rest of this paper is organized as follows. In Section 2, we introduce the notion of Pattern-Base Management system. Section 3 presents a general model for the coexistence of data and patterns. In Section 4, we present the formalism that describes the closed form of the relation between data and patterns. In Section 5, we explain how patterns can be queried and introduce predicates for comparing patterns and query operators and in Section 6, we discuss the related work. Finally, in Section 7 we provide a brief description of a prototype implementation, while Section 8 offers conclusions and topics of future work.

2 Patterns and pattern-base management systems

By observing the motivating example of Section 1, we can see that the overall structure of the environment and the operations performed by the user abide by some fundamental principles:

- There exist a *data space* and a *pattern space.*
- Patterns serve as artifacts, which describe (a subset of) data with similar properties and/or behavior, providing a compact and rich in semantics representation of data. Frequent itemsets in the case of our motivating example are compact (in terms of size) and rich in terms of information conveyed to the user.
- *Relationships* exist among the members of the data space and the members of the pattern space. In general, these relationships can be of cardinality *many-to-many*, i.e., a pattern can correspond to more than one data item and vice versa. We say that the data that are represented by a pattern form the *image of the pattern* in the data space.

- *The user navigates from the pattern to the data space*, back and forth. Patterns are related to their data images and data are mapped to patterns corresponding to them.
- To avoid performing costly operations over large data spaces, the user exploits their concise pattern representations, by *applying query operators over collections of patterns, as well as individual patterns.* Therefore, patterns are selected by value, compared to each other, matched for similarity or equivalence, just like data records are. At the same time, whole classes of patterns are compared to provide differences and similarities among their members, again, exactly as collections of data records (e.g., relational tables, object-oriented classes, etc.) would be handled.

To serve the above purposes, we envision an environment where the patterns are managed by a *Pattern-Base Management System* (PBMS) exactly as database records are managed by a database management system. The reference architecture for a PBMS is depicted in Fig. 2 and consists of three major layers of information organization. In the bottom of Fig. 2, we depict the *data space* consisting of data stores that contain data (*data layer*). Data can be either managed by a DBMS or stored in files, streams, or other physical means managed outside a DBMS. At the top of Fig. 2, we depict the PBMS repository that models the *pattern space* and contains patterns. Finally, in the middle of Fig. 2, we can observe the *intermediate mappings* that relate patterns to their corresponding data, forming the *intermediate data layer.* Intermediate mappings facilitate the justification of any knowledge inferred at the PBMS with respect to the data; for example, they could be used to retrieve the rows that produced the association rules of Fig. 2. The overall architecture is called *integrated pattern-base management architecture*, or simply *Pattern Warehouse.*

Next, we present a brief description of all the entities that appear in this abstract description of the PBMS:

- *Data layer.* The data from which the patterns are created reside in this layer. Data can be physically managed outside the PBMS, in a different repository. The motivation for the introduction of the original data in the Pattern Warehouse is dual: (a) there are several queries that require data as their answers, and, (b) from a modeling point of view, data are the measure of correctness for any approximations that occur in the PBMS setting. Naturally, in a practical setting, data can be absent, or not available to the PBMS; this can prohibit the answering of several questions, but cannot block the entire functionality of the PBMS. For reasons of simplicity and uniformity, we assume that the data are organized in an object-relational database.
- *Intermediate mappings layer.* This layer manages information on the relations between the patterns and the underlying data, involving specialized

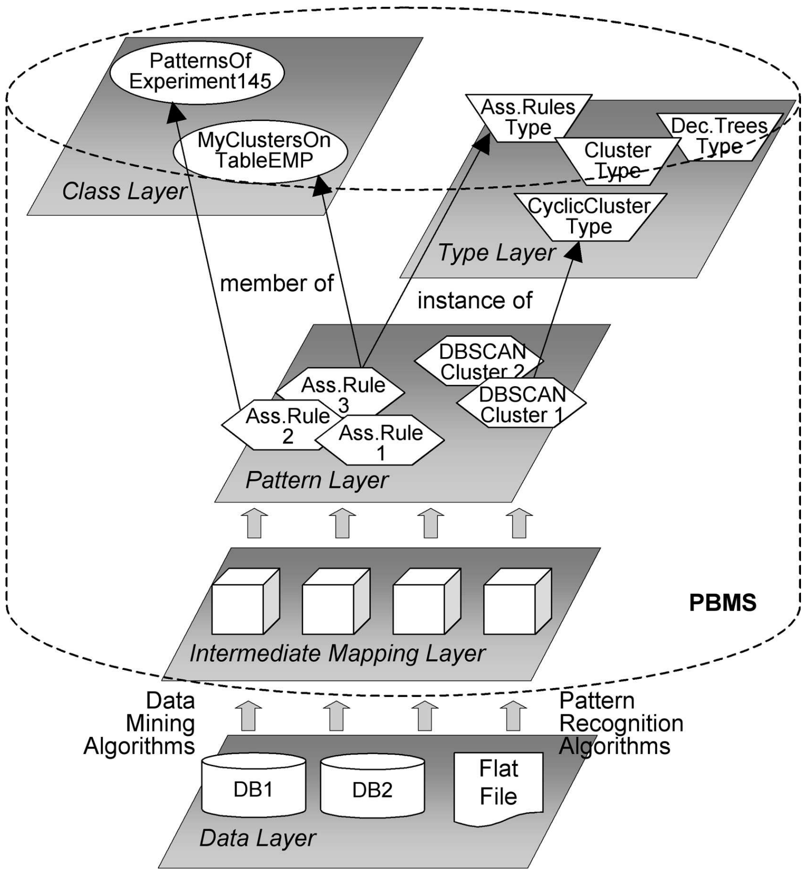


Fig. 2. Reference architecture for the Pattern-Base Management System

storage and indexing structures. Keeping intermediate mappings is clearly motivated by the existence of cross-over queries, where the user navigates back and forth between the pattern and the data space. The information kept here comprises catalogs linking each pattern with the data it represents. In general, one can imagine that these catalogs are huge and might not be available for every pattern, or they might be very expensive to be queried often. To address this problem, we consider two approaches for the representation of information about the pattern-data relation: (a) *exact*, where intermediate mappings are described by catalogs; and (b) *approximate*, where a formula is used to describe the pattern in the data space. In the latter case, one can employ different variations of these approximations through data reduction techniques (e.g., wavelets), summaries, or even on-line data mining algorithms. We emphasize that the exact intermediate layer can also be unavailable to the PBMS; the latter can still operate due to the approximations of the patterns' formulae.

- *Pattern layer*. Patterns are the actual information stored in the PBMS. They are concise and rich in semantics representations of the underlying data. Patterns are the stored information through which a user interacts with the system and performs queries and serve exactly as records serve in a traditional database. Some -but not obligatorily all- patterns are related to the underlying data through intermediate mappings and/or are characterized by an approximation formula.

- *Type layer.* The PBMS pattern types describe the intensional definition, i.e., the syntax of the patterns. Patterns of same type share similar characteristics, therefore pattern types play the role of data types in traditional DBMS's or object types in OODBMS's. Normally, we anticipate the PBMS to include a predefined set of built-in, popular pattern types (e.g., association rules and clusters). Still, the type layer must be extensible, simply because the set of pattern types that it incorporates must be extensible.
- *Class layer.* The PBMS classes are collections of patterns which share some semantic similarity. Patterns that are members of the same class are obligatorily required to belong to the same type. Classes are used to store patterns with predefined semantics given by the designer; by doing so, the designer makes it easier for users to work on classes. For example, a class may comprise patterns that resulted from the same experiments, like the association rules in Fig. 2. In other words, classes have the same role as tables in relational databases and object classes in object-oriented databases: they are logical-level entrypoints, collecting semantically similar data and easying the formulation of queries for the user.

3 Modeling data and patterns

In this section, we give the formal foundations for the treatment of data and patterns within the unifying framework of a pattern warehouse. First, we define the data space according to the complex value model of [9], further assuming that data are organized as an object-relational database. Then, we formally define pattern types, pattern classes, patterns, and the intermediate mappings between data and patterns. Finally, we define pattern bases and pattern warehouses.

3.1 The data space

Whereas in the general case, data can be organized under many different models without being incompatible to our framework, we need to assume a certain form of organization in order to give the formal logical definitions for the internal structure of a PBMS. Our choice for the data model is the complex value model, which can be considered as a quite broad variant of the object-relational model [9]. This model generalizes the relational model by allowing set and tuple constructors to be recursively applied. We believe that this model is flexible and, at the same time, elegantly extended by user defined predicates and functions. In this section we present, sometimes verbatim, some basic definitions from [9] to make the paper clearer and more self contained.

Data types, in the complex value model adopted, are structured types that use domain names, *set* and *tuple* constructors, and attributes. For reasons of space and simplicity, throughout the rest of this section, our examples consider only integer and real numbers, as well as strings.

Definition 1 *Data types (or simply,* types*) are defined as follows:*

- *If $\widehat{D}$ is a domain name, then $\widehat{D}$ is an* atomic *type.*
- *If $T_1, \ldots, T_n$ are types and $A_1, \ldots, A_n$ are distinct attribute names then $[A_1{:}T_1, \ldots, A_n{:}T_n]$ is an unnamed* tuple *type.*
- *If T is a type then $\{T\}$ is an unnamed* set *type.*
- *If T is an unnamed type and A is an attribute name, then $A{:}T$ is a* named *type.*

The *values* of a specific type are defined in the natural way. For atomic types, we assume an infinite and countable set of values as their domain, which we call **dom**(T). The domain of set types is defined as the powerset of the domain of the composing type. The domain of tuple types is defined as the product of their constituent types.

Example 1 *Consider the following types.*

$$T_1 = [X{:}real,\ Y{:}real,\ Z{:}integer]$$
$$T_2 = \Big\{\ [Id{:}integer,\ Disk{:}\{\ [Center{:}[X{:}real, Y{:}real],\ Rad{:}real]\ \}\ ]\ \Big\}$$

The expressions

$$[X{:}4.1,\ Y{:}5.2,\ Z{:}3] \text{ and } \Big\{\ [Id{:}7,\ Disk{:}\{\ [Center{:}[X{:}2.0, Y{:}3.0], Rad{:}4.1]\ \}]\ \Big\}$$

are values of types T_1 and T_2 respectively.

Relations in our setting are sets of tuples defined over a certain composite data type, as in the nested relational model. This definition is more restrictive than the one presented in [9], which allows for relations that are sets of sets, but here, we adopt it for the sake of simplicity. Also, for each relation, we assume an implicit, system-generated row identifier *RID* that uniquely identifies each tuple. We will not explicitly refer to *RID*'s in the sequel, unless necessary. In this context, a database and a database schema are defined as follows [9]:

Definition 2 *A database schema is a pair $\widehat{DB} = \langle[\widehat{D_1}, \ldots, \widehat{D_k}], [\widehat{R_1}{:}T_1, \ldots, \widehat{R_n}{:}T_n]\rangle$ where $T_1, \ldots, T_n$ are set types involving only the domains $\widehat{D_1}, \ldots, \widehat{D_k}$ and $\widehat{R_1}, \ldots, \widehat{R_n}$ are relation names.*

Definition 3 *An instance of $\widehat{DB}$, i.e., a database, is a pair $DB = \langle[D_1, \ldots, D_k],$*

$[R_1, \ldots, R_n]\rangle$, where R_i's are relations and D_i's are domains.

We also refer to $\widehat{DB}$ as the *database type*, and to DB as the *database value*.

As we will see in the following, we need to be able to define patterns over joins, projections, and selections over database relations. To this end, we extend Definition 2 with a set of *materialized views* $V_1, \ldots, V_m$ defined over relations $R_1, \ldots, R_n$ using the relational algebra. Throughout the rest of the paper, we address views as relations, unless explicitly specified otherwise.

3.2 The pattern space

In Section 2, we have described patterns as compact and rich representation of data and presented their functionalities and relations with the underlying data. In this section, we refine their description and give the logical definition of the entities that reside in pattern, pattern type and pattern class layers. As already mentioned, pattern types are templates for the actual patterns and pattern classes are groups of semantically similar patterns.

Patterns as previously described are formally translated as quintuples. We intuitively discuss the components of a pattern here and give the definition right next.

- First, a pattern is uniquely identified by a *Pattern Id* (*PID*).
- Second, a pattern has a *structure*. For example, an association rule comprises a *head* and a *body*, and a circular cluster comprises a *center* and a *radius*.
- Third, a pattern is typically generated by applying a data mining algorithm over some underlying data. By overriding some traditional database terminology, we call this data collection the *active domain* of the pattern. Note that the domain deals with a broader collection of data than the ones that precisely map to the pattern; therefore, the relationship deals with the context of validity of the pattern and not with its exact image in the data space.
- Fourth, a pattern informs the user about its quality, i.e., how closely it approximates reality compared to the underlying data, through a set of statistical *measures*. For example, an association rule is characterized by *confidence* and *support*.
- Finally, a *mapping formula* provides the semantics of the pattern. This component provides a possibly simplified representation of the relation between data represented by the pattern and the pattern structure. In Section 4, we present the formalism that we have adopted for expressing this relation.

A *pattern type* represents the intensional description of a pattern, the same way data types represent the intentional description of data in the case of object-relational data. In other words, a pattern type acts as a template for the generation of patterns. Each pattern is an *instance* of a specific pattern type. There are four major components that a pattern type specifies:

- First, the pattern type dictates the structure of its instances through a *structure schema*. For example, it specifies that association rules must comprise a head and a body.
- Moreover, a pattern type specifies the *domain*. The domain describes the schema of the underlying data that is used to generate the patterns. Practically, the domain is the schema of the relations that can be used as input for the pattern detection algorithm.
- Third, it provides a *measure schema*, i.e., a set of statistical measures that quantify the quality of the approximation that is employed by the instances of the pattern type.
- Finally, it provides the mapping formula in terms of data types to provide the general guidelines that connect the underlying data with the pattern to be instantiated by the pattern type. This formula is instantiated in each pattern.

Definition 4 *A* pattern type *is a quintuple* $[Name,\ SS,\ D,\ MS,\ MF]$ *where (a) $Name$ is a identifier unique among pattern types; (b) SS, the* Structure Schema*, is a distinct complex type; (c) D, the* Domain*, is a set type; (d) MS, the* Measure Schema*, is a tuple of atomic types; and (e) MF, the* Mapping Formula*, is a predicate over SS and D.*

Definition 5 *A* pattern (instance) *p over a pattern type PT is a quintuple $[PID,\ S,\ AD,\ M,\ MF]$ such that (a) PID is a unique identifier among all patterns, (b) S, the* Structure*, and M,* Measure*, are valid values of the respective structure and measure schemata of PT, (c) AD, the* Active Domain*, is a relation, which instantiates the set type of the Domain, and (d) MF, the* mapping formula*, is a predicate expressed in terms of AD and S, instantiating the respective mapping formula of PT.*

In order to define the mapping formula in the pattern type, we need to be able to reference the subelements of the complex value type appearing in the domain field. Since a pattern type is a generic construct not particularly bounded to a specific data set with specified attribute names, we need to employ auxiliary names for the complex value type, that will be overridden in the instantiation procedure.

Let us consider the instantiation procedure that generates patterns based on pattern types. Assume that a certain pattern type PT is instantiated in a new pattern p. Then:

- The data types specified in the Structure Schema SS and the Measure Schema MS of PT are instantiated by valid values in p.

- The auxiliary relation and attribute names in the domain D of PT are replaced by regular relation and attribute names from an underlying database, thus specifying the active domain of the pattern.
- Both the previous instantiations also apply for the Mapping Formula MF. The attributes of the Structure Schema are instantiated to values and the auxiliary names used in the definition of the domain are replaced by the actual relations and attribute names appearing in the active domain.

Having defined the data space and the pattern entities, we are ready to define the notions of *pattern class* and *pattern base (PB)*. Our final goal is to introduce the global framework, called *pattern warehouse*, as a unified environment in the context of which data- and pattern-bases coexist.

A pattern class over a pattern type is a collection of semantically related patterns, which are instances of this particular pattern type. Pattern classes play the role of pattern collections, just like relations are collections of tuples in the relational model.

Definition 6 *A* pattern class *is a triplet* $[Name, PT, Extension]$ *such that:(a) Name is a unique identifier among all classes; (b) PT is a pattern type; and (c) Extension is a finite set of patterns with pattern type PT.*

Now, we can introduce pattern bases as finite collections of classes defined over a set of pattern types and containing pattern instances.

Definition 7 *A* Pattern Base Schema *defined over a database schema* $\widehat{DD}$ *is defined as* $\widehat{PB} = \langle [\widehat{D_1}, \ldots, \widehat{D_n}], [\widehat{PC_1}{:}PT_1, \ldots, \widehat{PC_m}{:}PT_m] \rangle$*, where* PT_i*'s are pattern types involving the domains* $\widehat{D_1}, \ldots, \widehat{D_n}$ *and* $\widehat{PC_i}$*'s are pattern class names.*

Definition 8 *An instance of* $\widehat{PB}$*, i.e., a* pattern base*, over a database DB is defined as* $PB = \langle [PT_1, \ldots, PT_k], [PC_1, \ldots, PC_m] \rangle$*, where* PC_i*'s are pattern classes defined over pattern types* PT_i*, with patterns whose data range over the data in DB.*

3.3 Motivating example revisited

Coming back to our motivating example, assume now that the manager of the company wants to analyze the data about the employees and the products sold and employs a clustering algorithm on the `age` and `income` attributes to find groups of employees that have similar age and salaries. The clustering algorithm returns the clusters depicted in Table 1. The tuples with `id` $\in \{346, 733, 289, 923\}$ are represented by Cluster 1, the tuples with `id` $\in \{533, 657, 135, 014\}$ are represented by Cluster 2, and so on. These sets

CustBranch1(id,name,age,income,sex)		CustBranch2(id,name,age,income,sex)	
Cluster 1	Cluster 2	Cluster 3	Cluster 4
[346,A,30,33,1]	[533,E,43,60,1]	[532,I,31,34,1]	[012,O,41,59,1]
[733,B,31,31,2]	[657,F,47,60,2]	[322,J,32,31,2]	[230,N,45,59,1]
[289,C,29,29,1]	[135,G,45,59,2]	[315,K,30,29,0]	[292,M,43,58,1]
[923,D,30,27,2]	[014,I,49,61,2]	[943,H,31,28,1]	[971,L,43,60,1]

Table 1
Clustering in relations `CustBranch1` and `CustBranch2`

of tuples form the explicit images of the respective patterns.

Since the clustering algorithm is known to the system or the analyst, the abstract form (i.e., the pattern type) of the result is known a priori. In the case of the clusters of our reference example, the result is a 2D-disk represented as follows:

Name	Cluster
Structure Schema	`disk:[Center:[X:real,Y:real],Rad:real]`
Domain	`rel:{[A1:integer, A2:integer]}`
Measure Schema	Precision: `real`
Mapping Formula	$(\texttt{rel.A1} - \texttt{disk.Center.X})^2 + (\texttt{rel.A1} - \texttt{disk.Center.}Y)^2 \leq \texttt{disk.Rad}^2$

The actual clusters for the data set `CustBranch1` are `Cluster1` and `Cluster2`. Since the clustering algorithm is applied only over the attributes `age` and `income` of `CustBranch1`, only these attributes appear in the Active Domain. Technically this can be achieved by using a simple view on `CustBranch1`.

Pid	339 (Cluster 1)
Structure	`disk:[Center:[X:30,Y:30],Rad:3]`
Active Domain	`CustBranch1:{[age,income]}`
Measure	Precision: `1`
Mapping Formula	$(\texttt{CustBranch1.age} - 30)^2 + (\texttt{CustBranch1.income} - 30)^2 \leq 3^2$

Pid	340 (Cluster 2)
Structure	`disk:[Center:[X:45,Y:60],Rad:2]`
Active Domain	`CustBranch1:{[age,income]}`
Measure	Precision: `0.75`
Mapping Formula	$(\texttt{CustBranch1.age} - 45)^2 + (\texttt{CustBranch1.income} - 60)^2 \leq 2^2$

In the case of the products sold, the manager wants to detect products that are often sold together, so he uses a data mining algorithm that detects frequent itemsets and among others he discovers the frequent itemsets depicted in Table 2. Even in this case, the pattern type is known in advance:

TransBranch1:{bread,butter}

id	items
193	bread, butter, milk
189	bread, butter
124	bread, beer, butter
133	bread, knife, butter

TransBranch2:{bread,butter,milk}

id	items
293	bread, butter, milk, candies
289	bread, butter, milk
224	bread, beer, butter, milk
233	bread, knife, butter, milk

Table 2
Frequent Itemsets in relations TransBranch1 and TransBranch2

Name	Frequent Itemset
Structure Schema	fitems:{item: string}
Domain	rel:{[tid: integer, items: {string}]}
Measure Schema	Precision: real
Mapping Formula	fitems $\subseteq$ rel.items

3.4 The pattern warehouse

Having defined the data and the pattern space, we are ready to introduce the global framework within which data- and pattern-bases coexist. To this end, we formally define the relation between data and patterns and the overall context of patterns, data, and their mappings.

When a pattern is created by using a data mining algorithm, a subset of the data set used as input for the data mining algorithm is mapped to the pattern. For example, a clustering algorithm maps several data from a data set to the label of one cluster. We say that these data are *explicitly represented* by the pattern and we keep a catalog linking all these data with the pattern in the intermediate mappings. Thus, the intermediate mappings are nothing but an extensional form of the relation between the patterns and the underlying data. The PBMS offers functions to handle them, although the creation and deletion of the intermediate mappings are not its focus. Formally, the intermediate mappings between data and patterns can be defined as follows:

Definition 9 *Let $\mathcal{P}$ be the the set of all patterns in the pattern space and $\mathcal{D}$ be the set of all data appearing in the data space. An* intermediate mapping*, denoted as Φ, is a partial function: $\mathcal{P} \times \mathcal{D} \rightarrow \{true, false\}$.*

Using this definition, we can formally define the explicit representation as follows.

Definition 10 *A data item d is* explicitly represented *by a pattern p, denoted as $d \hookrightarrow p$, iff $\Phi(p, d) = true$.*

Finally, the explicit image of a pattern can be defined as follows.

Definition 11 *An* explicit image *of a pattern p, denoted as $\mathcal{I}_e(p)$, is the finite set of the data values explicitly represented by the pattern p, i.e., $\mathcal{I}_e(p) = \{x \mid x \hookrightarrow p\}$.*

To avoid any possible confusion regarding the relation between the image and the active domain of a pattern, it should be stated that the active domain concerns the data set which was used as input to the data mining algorithm, whereas the image describes the subset of the active domain that the data mining algorithm attributed to the pattern.

Remember that patterns are not linked to the data only by the intermediate mappings. Patterns are also *approximately* linked to the data through the mapping formula. The formula of each pattern is an effort to summarize the information carried by the intermediate mappings, describing as accurately as possible a criterion for selecting the data that are represented by the pattern from the original data set. In this sense, *the formula of each pattern is an approximation of the mapping* Φ. This approximate link actually carries a more generalized notion of the relation between the pattern and the data space; each pattern is mapped to a whole *region* of the data space, independently of the actual data records that exist. All points of this region are *approximately represented* by the pattern. More formally, assuming that the formula is actually a predicate called *mapping predicate*, the approximate representation is defined as follows:

Definition 12 *Let p be a pattern whose represented data range in the domain D. An element $x \in D$ is* approximately represented *(or simply,* represented*) by pattern p, denoted as $x \rightsquigarrow p$, iff $mp(x) = true$, where mp is a predicate called the* mapping predicate *of p.*

Note that D in the above definition does not refer to the active domain of the data of a certain pattern, but rather, D refers to all the possible data records, that qualify for its mapping predicate. Note also that throughout the paper, by *represent* we refer to the relationship *approximately represents*. When the semantics of Definition 10 is used we indicate the relationship as explicit representation.

Similarly to the explicit image, we can now define the *approximate image* of a pattern, or simply the image of the pattern:

Definition 13 *An* approximate image *of a pattern p, denoted by $\mathcal{I}(p)$, is the set of the data values approximately represented by the pattern p, i.e., $\mathcal{I}(p) = \{x \mid x \rightsquigarrow p\}$*

The above definitions do not specify (directly or indirectly) any relation be-

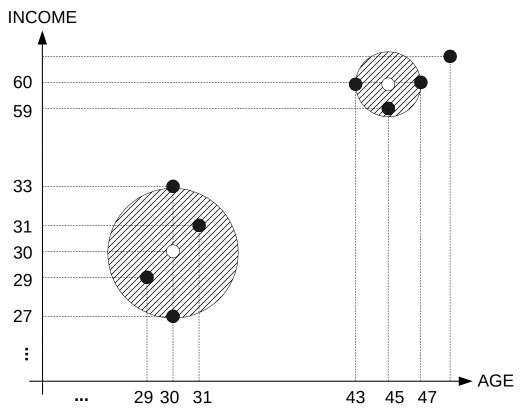


Fig. 3. The explicit and approximate image of Clusters 1 and 2

tween the approximate image $\mathcal{I}(p)$ and the explicit image $\mathcal{I}_e(p)$ of a pattern p. It is a matter of the pattern base designer to decide if some constraint (like $\mathcal{I}_e(p) \subseteq \mathcal{I}(p)$) has to be imposed. In principle, it is an implementation and administration issue whether the explicit image $\mathcal{I}_e(p)$ is saved (with the storage and maintenance cost that this implies) or all the operations are executed by using the approximate image $\mathcal{I}(p)$ (with the respective approximation error), which requires knowing only the mapping predicate. In Figure 3, the images of Cluster 1 and Cluster 2 (defined in Table 1) are depicted. The black dots are the points included in the two explicit images while the two grey areas represent the approximate images of the two clusters.

Since the relations between the pattern and the data space are so important for the aims of the PBMS, we need to define a logical entity that encompasses both. To this end, we introduce the notion of *pattern warehouse* which incorporates the underlying database, the pattern base, and the intermediate mappings. Although we separate patterns from data, we need the full environment in order to answer interesting queries and to support interactive user sessions navigating from the pattern to the data space and vice versa.

Definition 14 *A* pattern warehouse schema *is a pair* $\langle \widehat{DB}, \widehat{PB} \rangle$, *where* $\widehat{PB}$ *is a pattern base schema defined over the database schema* $\widehat{DB}$.

Definition 15 *A* pattern warehouse *is an instance of a pattern warehouse schema defined as a triplet:* $\langle DB, PB, \Phi \rangle$, *where* DB *is a database instance,* PB *is a pattern base instance, and* Φ *is an intermediate pattern-data mapping over* DB *and* PB.

4 The mapping formula

The mapping formula is the intensional description of the relation between the pattern and the data space. The mapping formula must provide a strictly defined *logical criterion* enabling the identification of the data that are represented by a pattern and vice versa. In this section, we present a formalism for defining mapping formulae and we focus on the following aspects: (a) the requirements for the formalism; (b) the language that we adopt to express it; (c) the safe incorporation of the introduced formalism in a query language; and (d) the extensibility of this formalism in terms of incorporating user-defined functions and predicates.

Requirements. Given the complexity of the relation that may exist between the data and the patterns, as well as the various complicated structures of data we may face, several requirements arise for the mapping formula.

- The mapping formula must be *informative* to the user, i.e., the user must be able to intuitively understand the abstraction provided by the pattern.
- A basic requirement is the ability to handle the *complex data values and types* that appear in the structure and domain fields.
- The mapping formula must be able to express a great variety of relations between patterns and data, and it should allow simple and easy extensions.
- The information described in the mapping formula should be in a form that can be easily used by queries. This means that the mapping formula must be expressed in a generic language that facilitates reasoning methods.

From a more technical point of view, at the pattern type level, the mapping formula relates the *structure schema* with the respective *domain*. At the pattern level, the mapping formula relates the *structure*, which is an instantiation of the structure schema, with the *active domain* of the pattern, i.e., with the set value (relation in our examples) appearing in the active domain component. Intuitively, the mapping formula at the pattern type level correlates regions of the pattern space with regions of the data space, whereas in the pattern level it correlates a single region of the data space with a specific value (the structure) in the pattern space. In the general case, the mapping formula does not have to be stored at the pattern level since it can be derived from the respective field of the pattern type and from the structure and active domain values of the pattern.

Now, we can formally define what a well-formed *mapping formula* is for a *pattern-type.*

Definition 16 *A pattern type mapping formula takes the form:*

$$mp(\overline{dv}, \overline{pv}) \tag{1}$$

where mp is a predicate called the mapping predicate*, $\overline{dv}$ are variable names appearing in the auxiliary relation in domain D and $\overline{pv}$ are variable names that appear in the structure schema SS.*

At instantiation time, $\overline{pv}$ is assigned values of the structure component and $\overline{dv}$ is mapped to the relation appearing in the active domain component. Similarly, we can formally define well-formed *mapping formula* for patterns.

Definition 17 *A pattern mapping formula takes the form:*

$$mp(\overline{dv}) \tag{2}$$

where mp is a predicate called the mapping predicate *and $\overline{dv}$ are variables appearing in the active domain component AD.*

Syntax. In the previous definitions, we choose to define the mapping formula as a predicate. This predicate is true for all the values in the domain (and of course in the active domain) of the pattern that are approximately represented by the pattern. This predicate is defined using first order logic, the logical connectives ($\neg$, $\vee$, $\wedge$ etc), and at least two constructor functions: the tuple [ ] and the set { } constructors, in order to be able to handle complex values, as they were presented in Section 3.1.

Extensibility. We can increase the expressive power of the mapping predicate by introducing more functions and predicates in the language we have adopted. In this context, we have *interpreted functions* and *predicates* that allow the formula to be informative to the user and offer reasoning abilities.

Safety in the use of the mapping predicate. The mapping predicate by itself is not a query, thus, there is no issue of safety. Still, we want to be able to use it in queries. For example, we would like to be able to use the mapping predicate to construct a query that would return all data from a specific dataset that can be approximately represented by the pattern. Being able to answer this query on the data, without any further knowledge, implies that all the free variables of the predicate take values from the active domain of the pattern. This is ensured by range-restricting all variables appearing in the formula predicate to constants or to the finite datasets appearing in the active domain. This results in a *safe* query. Still, the presence of even very simple functions makes the guarantee of safety, in terms of the classical domain independence, impossible. There is a need for a notion of safety broad enough to deal with functions and predicates. We opted to define safety for the PBMS queries in terms of the n-depth domain independence proposed in [9] that allows a finite number of application of functions. In this sense, safe queries are domain-independent in the active domain is closed under n applications of functions. In other words, any query can be safely evaluated and its results can be computed and returned to the user, provided that there is an upper bound (n) to the number of times that functions can be applied in it. Similar generalized forms of domain independence have been proposed in [10,11].

5 Querying the pattern warehouse

Having defined all the components of a pattern warehouse, we can now proceed in investigating how we can exploit the information stored in it. The pattern warehouse is a larger environment than the pattern base, as it incorporates patterns, data, and intermediate mappings. Therefore, the user can potentially pose queries over data, patterns, and their intermediate relationships. In this section, we will address the following fundamental query operations over a pattern warehouse:

- What are the possible relationships between two individual patterns?
- Given two patterns, can we combine them to produce a new one, or can we compare them to find out how similar they are?
- Given a set of patterns, i.e., a pattern class, or possibly, a set of classes, what kind of information can we extract by querying them?
- How can we possibly navigate back and forth between the pattern base and the raw data? What kind of interesting queries can we pose over the pattern-data interrelationships?

In this section, first we discuss the issue of managing individual patterns. We exploit results already available from the field of spatial data management and we prove that we can provide a complete set of operators that characterize all the possible relationships between patterns. Also, we provide constructors that provide the union, difference and similarity of two patterns. Next, we discuss the management of whole classes of patterns. We provide operators similar to the relational ones for the management of pattern classes and investigate their particularities when applied to collections of patterns. Finally, we provide three operators for navigating between the pattern and the data space.

5.1 Predicates on patterns

In this section, we investigate relations between patterns, using both their extensional and their intensional representation, and propose a complete set of predicates that characterize all possible situations between two patterns. To this end, we treat patterns as point sets in the data space. In fact, as defined in Section 3.4, patterns actually correspond to two possibly different point-sets in the data space: (a) a finite set comprising all data which are explicitly represented by the pattern, i.e., there exist intermediate mappings linking them to the pattern, and (b) a possibly infinite set described by the formula predicate, i.e., all points of this set are approximately represented by the pattern. Naturally, the former refers to the explicit image of the pattern and the latter to the approximate image of the pattern.

In all cases, we will define our binary pattern operators in two flavors:

- *Explicit relationships.* Explicit relationships are defined in terms of the specific data that patterns represent, i.e., with respect to their explicit image, as obtained through intermediate mappings.
- *Approximate relationships.* Approximate relationships are defined in terms of the approximately represented values of the data space, i.e., with respect to the approximate images of the involved patterns.

We introduce three fundamental pattern operators, with semantics similar (but not identical) to the respective set-theoretic ones. Two given patterns p_A and p_B can satisfy exactly one of the following:

- p_A *subsumes* p_B (denoted as $p_A \succ p_B$), meaning that the image of p_A is a superset of the one of p_B,
- p_A and p_B are *disjoint* (denoted as $p_A \pitchfork p_B$), meaning that their images in the data space have no common points, or
- p_A and p_B *intersect* (denoted as $p_A \cap p_B$), meaning that their images in the data space have at least one common and one not common point.

Before specifying the above operators, we define the notion of *data compatible patterns* that clarifies when it is meaningful to compare two patterns with respect to the underlying data.

Definition 18 *Two pattern types are* data compatible *if the data types described in their data schemata describe the same domain. Two patterns are data compatible if they are instances of pattern types that are data compatible.*

In the sequel, first we introduce our pattern operators and then we prove that they are complete and minimal.

5.1.1 Intersecting patterns

The first test that we can pose on two patterns concerns the possibility of these two patterns having common data in the data space. The notion of pattern intersection is slightly different from the well-known, set-theoretic intersection. In our case, we require that (a) at least a common data item exists in the explicit images of both patterns involved in the comparison (as usual), (b) there exists at least one data item that corresponds to the first pattern, but does not correspond to the second one (i.e., their difference is not empty) and

(c) the previous relationship also holds the other way. Our deviation from the traditional definition practically identifies patterns that are neither identical, nor disjoint. This is necessary for providing a minimal and complete set of comparison operators for two patterns and it will be made evident in the sequel (Theorem 1).

As already mentioned, we provide two variants for every binary operator, so we define an explicit and an approximate variant of intersection.

Definition 19 *Two data compatible patterns p_A and p_B* explicitly intersect*, denoted as $p_A \cap_e p_B$, iff $(\exists x)(x \hookrightarrow p_A \wedge x \hookrightarrow p_B) \wedge (\exists y, z)(y \hookrightarrow p_A \wedge z \hookrightarrow p_A \wedge \neg(y \hookrightarrow p_B) \wedge \neg(z \hookrightarrow p_B))$.*

Definition 20 *Two data compatible patterns p_A and p_B* approximately intersect*, denoted as $p_A \cap p_B$, iff $(\exists x)(x \rightsquigarrow p_A \wedge x \rightsquigarrow p_B) \wedge (\exists y, z)(y \rightsquigarrow p_A \wedge z \rightsquigarrow p_B \wedge \neg(y \rightsquigarrow p_B) \wedge \neg(z \rightsquigarrow p_A))$.*

5.1.2 Disjoint patterns

As usual, two patterns are disjoint if they do not share any common item in the data space. With respect to their formulae, two patterns intersect when they are defined over the same domain and the conjunction of their formula predicates is satisfied. If the formula cannot be satisfied in the domain D then they are disjoint. The *disjoint* relationship is given in the following two definitions (for the explicit and the approximate variant).

Definition 21 *Two data compatible patterns p_A and p_B are* explicitly disjoint*, denoted as $p_A \pitchfork_e p_B$, iff $(\nexists x)(x \hookrightarrow p_A \wedge x \hookrightarrow p_B)$.*

Definition 22 *Two data compatible patterns p_A and p_B are* approximately disjoint *(or simply,* disjoint*), denoted as $p_A \pitchfork p_B$, iff $(\nexists x)(x \rightsquigarrow p_A \wedge x \rightsquigarrow p_B)$.*

5.1.3 Pattern equivalence

A pattern p_A is equivalent to another pattern p_B when all data records represented by p_B are also represented by p_A and vice versa. The explicit and the approximate variant of the equivalence relationship is defined as follows.

Definition 23 *Let p_A and p_B be two data compatible patterns. Pattern p_A and p_B are* explicitly equivalent*, denoted as $p_A \equiv_e p_B$, iff $(\forall x)\big((x \hookrightarrow p_B) \Leftrightarrow (x \hookrightarrow p_A)\big)$.*

Definition 24 *Let p_A and p_B be two data compatible patterns. Pattern p_A and p_B are* approximately equivalent *(or simply,* equivalent*), denoted as $p_A \equiv p_B$,* *iff* $(\forall x)\Big((x \rightsquigarrow p_B) \Leftrightarrow (x \rightsquigarrow p_A)\Big)$.

We also use $p_A \not\equiv p_B$ (respectively $p_A \not\equiv_e p_B$) to denote that p_A and p_B are *not* equivalent (respectively explicit equivalent).

5.1.4 Pattern subsumption

A pattern p_A subsumes another pattern p_B when (a) p_A and p_B are not equivalent and (b) all data records represented by p_B are also represented by p_A. For example, if the formula predicate of p_A is $x > 5$ and the formula predicate of p_B is $6 < x < 8$, and x is defined over the same domain D, then, we can say that p_A approximately subsumes p_B. The explicit and the approximate variant of the subsume relationship is defined as follows.

Definition 25 *Let p_A and p_B be two data compatible patterns. Pattern p_A* explicitly subsumes *p_B, denoted as $p_A \succ_e p_B$, iff (a) $p_A \not\equiv_e p_B$ and (b) $(\forall x)\Big((x \hookrightarrow p_B) \Rightarrow (x \hookrightarrow p_A)\Big)$.*

Definition 26 *Let p_A and p_B be two data compatible patterns. Pattern p_A* approximately subsumes *(or simply,* subsumes*) p_B, denoted as $p_A \succ p_B$, iff (a) $p_A \not\equiv p_B$ and (b) $(\forall x)\Big((x \rightsquigarrow p_B) \Rightarrow (x \rightsquigarrow p_A)\Big)$.*

5.1.5 A complete set of binary pattern relations

Given two data compatible patterns p_A and p_B, we can easily prove that, with respect to their images, they have exactly one of the previously introduced relationships: (a) they are disjoint (i.e., they do not have any common data item in the data space), (b) they are equivalent, (c) one subsumes the other (i.e., all data of one pattern belong to the corresponding data of the other), or (d) they have some, but clearly not all data in common. This property holds for both the explicit and the approximate images of data. In other words, the introduced set of relationships is *complete.* We can easily prove that the introduced set is *minimal.* These properties are formalized in the following theorems.

Theorem 1 *Every two data compatible patterns p_A and p_B satisfy exactly one of the relations $\{p_A \cap_e p_B,\ p_A \pitchfork_e p_B,\ p_A \equiv_e p_B,\ p_A \succ_e p_B,\ p_B \succ_e p_A\}$ with respect to their explicit images.*

Proof: The proof that the above set of relations are mutually exclusive is trivial and we omit it for lack of space. Given that they are mutually exclusive it is also trivial to see that they are minimal: we only need to find one example for each different relation.

To prove that the above set of relations is complete, we will exploit the tautology $\alpha \vee (\neg\alpha) \Leftrightarrow true$. Specifically, for any data compatible patterns p_A and p_B the following hold:

(1) $(\exists x)(x \in \mathcal{I}_e(p_A) \wedge x \in \mathcal{I}_e(p_B)) \vee \neg(\exists x)(x \in \mathcal{I}_e(p_A) \wedge x \in \mathcal{I}_e(p_B)) \Leftrightarrow$

(2) $(\exists x)(x \in \mathcal{I}_e(p_A) \wedge x \in \mathcal{I}_e(p_B)) \vee \neg(\exists x)(x \hookrightarrow p_A \wedge x \hookrightarrow p_B) \Leftrightarrow$

(3) $(\exists x)(x \in \mathcal{I}_e(p_A) \wedge x \in \mathcal{I}_e(p_A)) \vee p_A \pitchfork_e p_B \Leftrightarrow$

(4) $[(\exists x)(x \in \mathcal{I}_e(p_A) \wedge x \in \mathcal{I}_e(p_B)) \wedge$
$[(\forall x)(x \in \mathcal{I}_e(p_A) \Rightarrow x \in \mathcal{I}_e(p_B)) \vee \neg(\forall x)(x \in \mathcal{I}_e(p_A) \Rightarrow x \in \mathcal{I}_e(p_B))] \wedge$
$[(\forall x)(x \in \mathcal{I}_e(p_B) \Rightarrow x \in \mathcal{I}_e(p_A)) \vee \neg(\forall x)(x \in \mathcal{I}_e(p_B) \Rightarrow x \in \mathcal{I}_e(p_A))]]$
$\vee p_A \pitchfork_e p_B \Leftrightarrow$

(5) $[(\exists x)(x \in \mathcal{I}_e(p_A) \wedge x \in \mathcal{I}_e(p_B)) \wedge$
$[(\forall x)(x \in \mathcal{I}_e(p_A) \Rightarrow x \in \mathcal{I}_e(p_B)) \vee (\exists x)(x \in \mathcal{I}_e(p_A) \wedge \neg(x \in \mathcal{I}_e(p_B)))] \wedge$
$[(\forall x)(x \in \mathcal{I}_e(p_B) \Rightarrow x \in \mathcal{I}_e(p_A)) \vee (\exists x)(x \in \mathcal{I}_e(p_B) \wedge \neg(x \in \mathcal{I}_e(p_A)))]]$
$\vee p_A \pitchfork_e p_B \Leftrightarrow$

(6) $[(\exists x)(x \in \mathcal{I}_e(p_A) \wedge x \in \mathcal{I}_e(p_B)) \wedge$
$[(\forall x)(x \in \mathcal{I}_e(p_A) \Rightarrow x \in \mathcal{I}_e(p_B)) \wedge (\forall x)(x \in \mathcal{I}_e(p_B) \Rightarrow x \in \mathcal{I}_e(p_A)) \vee$
$(\forall x)(x \in \mathcal{I}_e(p_A) \Rightarrow x \in \mathcal{I}_e(p_B)) \wedge (\exists x)(x \in \mathcal{I}_e(p_B) \wedge \neg(x \in \mathcal{I}_e(p_A))) \vee$
$(\exists x)(x \in \mathcal{I}_e(p_A) \wedge \neg(x \in \mathcal{I}_e(p_B))) \wedge (\forall x)(x \in \mathcal{I}_e(p_B) \Rightarrow x \in \mathcal{I}_e(p_A)) \vee$
$(\exists x)(x \in \mathcal{I}_e(p_A) \wedge \neg(x \in \mathcal{I}_e(p_B))) \wedge (\exists x)(x \in \mathcal{I}_e(p_B) \wedge \neg(x \in \mathcal{I}_e(p_A)))]]$
$\vee p_A \pitchfork_e p_B \Leftrightarrow$

(7) $[(\exists x)(x \in \mathcal{I}_e(p_A) \wedge x \in \mathcal{I}_e(p_B)) \wedge (\forall x)(x \in \mathcal{I}_e(p_A) \Rightarrow x \in \mathcal{I}_e(p_B)) \wedge (\forall x)(x \in \mathcal{I}_e(p_B) \Rightarrow x \in \mathcal{I}_e(p_A))] \vee$
$[(\exists x)(x \in \mathcal{I}_e(p_A) \wedge x \in \mathcal{I}_e(p_B)) \wedge (\forall x)(x \in \mathcal{I}_e(p_A) \Rightarrow x \in \mathcal{I}_e(p_B)) \wedge (\exists x)(x \in \mathcal{I}_e(p_B) \wedge \neg(x \in \mathcal{I}_e(p_A)))] \vee$
$[(\exists x)(x \in \mathcal{I}_e(p_A) \wedge x \in \mathcal{I}_e(p_B)) \wedge (\exists x)(x \in \mathcal{I}_e(p_A) \wedge \neg(x \in \mathcal{I}_e(p_B))) \wedge (\forall x)(x \in \mathcal{I}_e(p_B) \Rightarrow x \in \mathcal{I}_e(p_A))] \vee$
$[(\exists x)(x \in \mathcal{I}_e(p_A) \wedge x \in \mathcal{I}_e(p_B)) \wedge (\exists x)(x \in \mathcal{I}_e(p_A) \wedge \neg(x \in \mathcal{I}_e(p_B)) \wedge (\exists x)(x \in \mathcal{I}_e(p_B) \wedge \neg(x \in \mathcal{I}_e(p_A)))] \vee$
$p_A \pitchfork_e p_B \Leftrightarrow$

(8) $[(\exists x)(x \hookrightarrow p_A \wedge x \hookrightarrow p_B) \wedge (\forall x)(x \hookrightarrow p_A \Rightarrow x \hookrightarrow p_B) \wedge (\forall x)(x \hookrightarrow p_B \Rightarrow x \hookrightarrow p_A)] \vee$
$[(\exists x)(x \hookrightarrow p_A \wedge x \hookrightarrow p_B) \wedge (\forall x)(x \hookrightarrow p_A \Rightarrow x \hookrightarrow p_B) \wedge (\exists x)(x \hookrightarrow p_B \wedge \neg(x \hookrightarrow p_A))] \vee$
$[(\exists x)(x \hookrightarrow p_A \wedge x \hookrightarrow p_B) \wedge (\exists x)(x \hookrightarrow p_A \wedge \neg(x \hookrightarrow p_B)) \wedge (\forall x)(x \hookrightarrow p_B \Rightarrow x \hookrightarrow p_A)] \vee$
$[(\exists x)(x \hookrightarrow p_A \wedge x \hookrightarrow p_B) \wedge (\exists x)(x \hookrightarrow p_A \wedge \neg(x \hookrightarrow p_B) \wedge (\exists x)(x \hookrightarrow p_B \wedge \neg(x \hookrightarrow p_A)] \vee$
$p_A \pitchfork_e p_B \Leftrightarrow$

(9) $[(\exists x)(x \hookrightarrow p_A \wedge x \hookrightarrow p_B) \wedge (\forall x)(x \hookrightarrow p_A \Rightarrow x \hookrightarrow p_B) \wedge (\forall x)(x \hookrightarrow p_B \Rightarrow x \hookrightarrow p_A)] \vee$
$p_A \succ_e p_B \vee$

$p_B \succ_e p_A \vee$
$p_A \cap_e p_B \vee$
$p_A \pitchfork_e p_B \Leftrightarrow$

(10) $p_A \equiv_e p_B \vee$
$p_A \succ_e p_B \vee$
$p_B \succ_e p_A \vee$
$p_A \cap_e p_B \vee$
$p_A \pitchfork_e p_B$

Based on the completeness and minimality of the aforementioned set of relations, we can therefore conclude that they are sufficient to characterize the relationship of any two patterns. ■

Theorem 2 *Every two data compatible patterns p_A and p_B satisfy exactly one of the relations $\{p_A \cap p_B,\ p_A \pitchfork p_B,\ p_A \equiv p_B,\ p_A \succ p_B,\ p_B \succ p_A\}$ with respect to their approximate images.*

Proof: The proof follows the same steps with the proof of Theorem 1 using the approximate images of the patterns instead of the explicit ones. ■

5.1.6 Equality

So far, we have referred to pattern equivalence with respect to the images of the involved patterns. Naturally, except for the predicates that are defined with respect to the images of a pattern, we can define identity or equality in a broader sense. Specifically, we give two different definitions of equality:

- *Identity* ($=$). Two patterns p_1 and p_2 are identical if they have the same PID, i.e., $p_1.PID = p_2.PID$.
- *Shallow equality* ($=^s$). Two patterns p_1 and p_2 are shallow equal if their corresponding components (except for the PID component) are equal, i.e., $p_1.S = p_2.S$, $p_1.AD = p_2.AD$, $p_1.M = p_2.M$, and $p_1.MF = p_2.MF$. Note that, to check the equality for each component pair, the basic equality operator for the specific component type is used. It is easy to verify that $p_1 =^s p_2 \Rightarrow p_1 \equiv p_2$.

5.2 Pattern constructors

Having introduced all possible relationships among two patterns, we can now define pattern constructors resulting from such relations. Therefore, we introduce the intersection and union of two patterns. Then, we discuss issues of pattern similarity.

5.2.1 The intersection and union constructors

Constructing new patterns from new data sets is primarily a task for data mining methods; nevertheless, we can construct new patterns from the existing ones, in the PBMS, too.

Definition 27 *Let* $p_A = (pid_A, s_A, ad_A, m_A, mp_A)$ *and* $p_B = (pid_B, s_B, ad_B, m_B, mp_B)$ *be two data compatible patterns. The* intersection *of patterns* p_A *and* p_B*, denoted as* $p_A \bigcap p_B$*, is defined as*

$$p_A \bigcap p_B = \Big(newpid(), \{s_A, s_B\}, ad, f(m_A, m_B), mp_A \wedge mp_B\Big),$$

where $ad = \{x \mid (x \in ad_A \vee x \in ad_B)\}$ *and* f *some measure composition function. Furthermore,* $\mathcal{I}_e(p_A \bigcap p_B) = \{x \mid (x \in \mathcal{I}_e(p_A) \vee x \in \mathcal{I}_e(p_B)) \wedge (x \leadsto p_A \wedge x \leadsto p_B)\}$.

Practically, the pattern that is created by the intersection of two other patterns p_A and p_B consists of:

- A new *pid* created by some system function.
- A new structure $\{s_A, s_B\}$, which is a set having as members the values s_A and s_B that appear in the structure field of p_A and p_B respectively.
- A new relation $ad_{A \bigcup B}$ in the data field that contains all the data that were specified in the *data* fields of the input patterns. Note that since p_A and p_B are data compatible, the relations appearing in their data fields have compatible type, so the union of their values can appear under one of the initial relation schemas.
- A new measure $f(m_A, m_B)$, computed by employing a measure composition function f. The measure computation mechanism is currently a direction for future work.
- A new formula that is the conjunction of the two formula predicates of the intersecting patterns. Some attention should be paid in mapping the variables from the initial relations to the resulting one. We might be able to further simplify this formula using techniques from constraint theory. For example, if the two formula predicates of p_A and p_B are $mp_A = 5 < x < 7$ and $mp_B = 4 < x < 6$, then we can reduce $mp_A \wedge mp_B$ to $5 < x < 6$.

Note that the data mappings are created *based on the intensional description of the pattern.* That is, from the explicit images of the intersecting patterns, only those data satisfying the formula predicate of the new pattern, i.e., those that are approximately represented by the new pattern, are actually mapped to the intersection pattern. Figure 4 graphically illustrates this remark. The new pattern describes the shadowed area, thus, we would like to map onto it all the data inside the shadowed area. Note that data explicitly represented by p_A or p_B are in the intersection, but they are not *common* (i.e., none of them belongs to both datasets). Thus, if we mapped the data that are in the

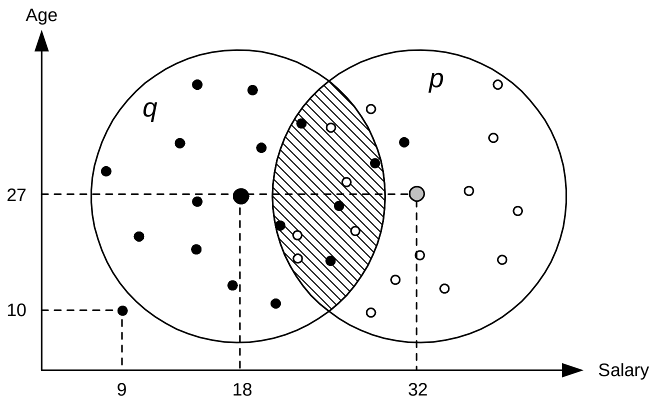


Fig. 4. The intersection operator. Solid points represent data items in the image of p_A and hollow points, represent data items in the image of p_B. The explicit images do not have any common point.

intersection of $\mathcal{I}_e^{p_A}$ and $\mathcal{I}_e^{p_B}$, instead of those that fall in the intersection of $\mathcal{I}^{p_A}$ and $\mathcal{I}^{p_B}$, we would end up with no data mapped to the new pattern. Therefore, we assume that pattern formulae are broader generalizations of explicit images (i.e., no point belonging to the explicit image of a pattern is lost from the pattern formula) and we compute the new explicit image based on the original approximate images. From our point of view, following the explicit intermediate mappings offers a rather narrow approach that we chose to completely avoid in this paper.

Definition 28 *Let* $p_A = (pid_A, s_A, ad_A, m_A, mp_A)$ *and* $p_B = (pid_B, s_B, ad_B, m_B, mp_B)$ *be two data compatible patterns. The* union *of patterns* p_A *and* p_B*, denoted as* $p_A \bigcup p_B$*, is defined as*

$$p_A \bigcup p_B = \Big(newpid(), \{s_A, s_B\}, ad, f(m_A, m_B), mp_A \vee mp_B\Big),$$

where $ad = \{x \mid x \in ad_A \vee x \in ad_B\}$ *and* f *some measure composition function. Furthermore* $\mathcal{I}_e(p_A \bigcup p_B) = \{x \mid (x \in \mathcal{I}_e(p_A) \vee x \in \mathcal{I}_e(p_B))\}$.

Practically, the new pattern resulting from the union operator is created as follows:

- First, the pattern gets a new *pid* from some system function.
- The *structure* of the pattern is a set composed of the values that appeared in the *structure* component of the initial patterns, exactly as in the case of *intersection*.
- The pattern *data* is the union of the *data* of the initial patterns.
- The new *measure* is computed, again, by employing a measure composition function, f.
- The new *formula* is the disjunction of the formulae of the initial patterns. Again reduction techniques can be applied as for the case of the *intersection*.

Lemma 1 *Let p_A and p_B be two patterns. Then* $(x \hookrightarrow p_A \vee x \hookrightarrow p_B) \Rightarrow (x \hookrightarrow p_{A \bigcup B})$.

Proof: Obvious. ■

Note that based on the above result, it is not required that $\mathcal{I}_e(p_A \bigcup p_B) = \mathcal{I}_e(p_A) \bigcup \mathcal{I}_e(p_B)$ as one might expect. This happens only if there has been imposed a constraint which requires that $\mathcal{I}_e \subseteq \mathcal{I}$.

5.2.2 The similarity operator ξ

Similarity between patterns is not an obvious notion. To the best of our knowledge there has been little work on the similarity of patterns [12,13], and none of it dealing with similarity of patterns of different type. As in [12] our notion of similarity is defined with respect to the data. Actually, as we did in the previous subsection, we define here two different versions of the similarity operator: (a) the *explicit similarity operator* ξ_e defined on $\mathcal{I}_e$, and (b) the *approximate similarity operator* ξ defined on $\mathcal{I}$.

The explicit similarity operator. The explicit similarity operator ξ_e is defined over the similarity of the represented datasets $\mathcal{I}_e$. A straightforward approach to defining this similarity would be to take into account how many common values exist in the explicit images of the compared patterns. One of the possible measures for such a similarity definition is the Jaccard coefficient:

Definition 29 *Let p_A and p_B be two patterns. The* explicit similarity *of p_A and p_B, denoted as $\xi_e(p_A, p_B)$, is defined by the expression* $\xi_e(p_A, p_B) = \frac{COUNT(\mathcal{I}_e(p_A) \bigcap \mathcal{I}_e(p_B))}{COUNT(\mathcal{I}_e(p_A) \bigcup \mathcal{I}_e(p_B))}$

Naturally, other measures apply, mainly from the field of Information Retrieval, involving the simple matching, cosine, or Dice's coefficients [14].

The approximate similarity operator. Definition 29 does not always reflect the similarity of the knowledge carried by the patterns. For example, in the case depicted in Figure 5 the two clusters carry approximately the same information: the fact that there is a concentration of employees around the age 30-31 with the same salary of 30k. Whereas this is visually obvious, it is not directly reflected in the relations between the two different data sets that are represented by the clusters, since they do not share even one common value. What we are actually comparing in this case are the whole areas in the data space that are defined by the pattern formulae and not the areas of the actual data. A measure for similarity that captures this notion is defined similarly to the explicit similarity.

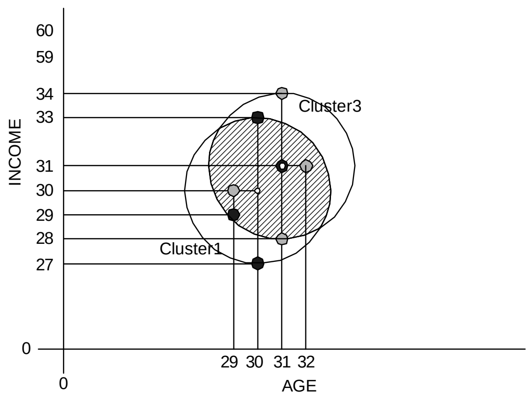


Fig. 5. Pattern similarity

Definition 30 *Let p_A and p_B be two data compatible patterns. The* similarity *of p_A and p_B, denoted as ξ, is defined by the expression $\xi(p_A, p_B) = \frac{V(\mathcal{I}(p_A) \bigcap \mathcal{I}(p_B))}{V(\mathcal{I}(p_A) \bigcup \mathcal{I}(p_A))}$, where V is a function that gives the volume of the region in the n-dimensional space of the image.*

Intuitively, the similarity of two patterns is defined as the ratio of the size of their intersection divided by the size of their union. Thus, two disjoint patterns have $\xi = 0$ and two equal patterns have $\xi = 1$. Whereas in the case of the disk clusters depicted in Figure 5, we could define the distance between their centers as a similarity measure, defining similarity over the size of their intersection and their union has an important property: *similarity is independent of the structure type of a pattern.* Thus, we can define the similarity between patterns that have different structures, i.e., they define regions with different shapes in the data space. Observe that since the data space in principle involves n attributes, the regions are defined over a n-dimensional space.

If the union or the intersection of the images of the patterns are not constrained in one or more dimensions, then we cannot define the approximate similarity operator. In this case, if spatial reasoning algorithms cannot be employed, we can still compute the similarity between the patterns by using the explicit similarity operator.

5.3 Operators defined over pattern classes

In the following, we introduce several operators defined over pattern classes. Some of them, like set-based operators, renaming and selection are quite close to their relational counterparts; nevertheless, some others, like join and projection have significant differences. Examples of the proposed operators will be provided in Section 5.5.

Set-based operators. Since classes are sets of patterns, usual set operators such as union, difference and intersection are defined for pairs of classes with compatible pattern types. The criterion for equality/identity of the patterns can be any of the ones mentioned in the previous subsection.

Renaming. Similarly to the relational context, we consider a renaming operator μ that takes a class and a renaming function and changes the names of the pattern component attributes according to the specified function. Each attribute is denoted by a path expression.

Measure projection. The measure projection operator reduces the measures of the input patterns by projecting out some components. Note that this operation does not modify the pattern mapping formula, since measures do not appear in it.

Let c be a class of pattern type pt. Let lm a list of attributes appearing in $pt.Measure$. Then, the measure projection operator is defined as follows:

$$\pi^m_{lm}(c) = \{(newid(), s, d, \pi_{lm}(m), f)|\ (\exists p \in c)(p = (pid, s, d, m, f))\}$$

In the previous definition, $\pi_{lm}(m)$ is the usual relational projection of the measure component.

Reconstruction. The reconstruction operator allows us to manipulate the pattern structure. This operation is useful for example for projecting out some components or for nesting or unnesting data. Since the type of the pattern structure is any type admitted by the type system, we define reconstruction as a query over the pattern structure, defined by using traditional complex value algebraic operators [9].

More formally, let c be a class of pattern type pt. Let $I, I_1, I_2, ...$ be sets of values of type $\tau, \tau_1, \tau_2, ...$ Let q be a query constructed by using the following subset of the nested relational algebra operators plus usual set-based operators [9].

Projection Let I be a set of type $\tau = [B_1 : \tau_1, ..., B_k : \tau_k]$. Then, $\pi_{B_1,...,B_l}(I)$ is a set of type $[B_1 : \tau_1, ..., B_l : \tau_l]$ such that $\pi_{B_1,...,B_l}(I) = \{[B_1 : v_1, ..., B_l : v_l]|\exists v_{l+1}, ..., v_k$ such that $[B_1 : v_1, ..., B_l : v_l, B_{l+1} : v_{l+1}, ..., B_k : v_k] \in I\}$

Constructors We consider two operations, *tup_create* and *set_create*.

- If $A_1, ..., A_m$ are distinct attributes, $tup\_create_{A_1,...,A_m}(I_1, ..., I_m)$ is of type $[A_1 : \tau_1, ..., A_m : \tau_m]$ and $tup\_create_{A_1,...,A_m}(I_1, ..., I_m) = \{[A_1 : v_1, ..., A_m : v_m]|\forall v_i \in I_i, i = 1, ..., m\}$.
- $set\_create(I)$ is a set of type $\{\tau\}$ and $set\_create(I) = \{I\}$.

Destructors We consider two operations, *tup_destroy* and *set_destroy*.

- If I is of type $\{\tau\}$, and $\tau = \{\tau_1\}$, then $set\_destroy(I)$ is a set of type τ_1 and $set\_destroy(I) = \{w|\exists v \in I, w \in v\}$.
- If I is of type $[A : \tau_1]$, $tup\_destroy(I)$ is a set of type τ_1 and $tup\_destroy(I) =$

$\{v|[A : v] \in I\}$.

Nesting and unnesting Let I_1 be a set of type t_1 such that $t_1 = [A_1 : \tau_1, ..., A_k : \tau_k, B : \{[A_{k+1} : \tau_{k+1}, ..., A_n : \tau_n]\}]$. Let I_2 be a set of type t_2 such that $t_2 = [A_1 : \tau_1, ..., A_k : \tau_k, A_{k+1} : \tau_{k+1}, ..., A_n : \tau_n]$. Then,

- $unnest_B(I_1)$ is a set of type t_2 and $unnest_B(I_1) = \{[A_1 : v_1, ..., A_n : v_n] | \exists y[A_1 : v_1, ..., A_k : v_k, B : y] \in I_1 \wedge [A_{k+1} : v_{k+1}, ..., A_n : v_n] \in y\}$;
- $nest_{B=(A_{k+1},...,A_n)}(I_2)$ is a set of type t_1 and $nest_{B=(A_{k+1},...,A_n)}(I_2) = \{[A_1 : v_1, ..., A_k : v_k, B : \{[A_{k+1} : v_{k+1}, ..., A_n : v_n]\}] | [A_1 : v_1, ..., A_n : v_n] \in I_2\}$.

Then, the reconstruction operator is defined as follows:

$\rho_q(c) = \{(pid, q(s), d, m, f) | p = (pid, s, d, m, f) \in c\}$.

Note that the formula of the resulting patterns has not to be updated since the only variables appearing in it come from the data source, which is not changed by the operation.

Selection. The selection operator allows one to select from a given class the patterns satisfying a certain condition, involving any pattern component. Conditions are constructed by combining through logical connectives ($\wedge$, $\vee$, $\neg$) atomic formulae corresponding to: (i) predicates over patterns presented in Section 5.1; (ii) similarity operators introduced in Section 5.2.2; (iii) predicates over pattern components, depending on their data type. Predicates of type (i) and (ii) must be used with respect to a constant pattern. Predicates of type (iii) are applied to pattern components s, ad, and m which, according to the proposed model, may have a complex value type. Therefore, predicates for complex values - e.g., equality ($=$), membership ($\in$), and set-containment ($\subseteq$) - can be used according to the proper type restrictions.

Let c be a class of pattern type pt. Let pr be a predicate. Then, the selection operator is defined as follows:

$\sigma_{pr}(c) = \{p | p \in c \wedge pr(p) = true\}$

Note that, since predicates over pattern components can only be applied to tuples, in order to select patterns with respect to inner components of the structure, a reconstruction operator may have to be applied before applying the selection operator.

Join. The join operation provides a way to combine patterns belonging to two different classes according to a join predicate and a composition function specified by the user.

Let c_1 and c_2 be two classes over two pattern types pt_1 and pt_2. A join predicate F is any predicate defined over a component of patterns in c_1 and a component of patterns in c_2. A composition function c for pattern types pt_1 and pt_2 is a

4-tuple of functions $c = (c_{StructureSchema}, c_{Domain}, c_{MeasureSchema}, c_{Formula})$, one for each pattern component. For example, function $c_{StructureSchema}$ takes as input two structure values of the correct type and returns a new structure value, for a possible new pattern type, generated by the join. Functions for the other pattern components are similarly defined. Given two patterns $p_1 = (pid_1, s_1, d_1, m_1, f_1) \in c_1$ and $p_2 = (pid_2, s_2, d_2, m_2, f_2) \in c_2$, $c(p_1, p_2)$ is defined as the pattern p with the following components:
$Structure : c_{StructureSchema}(s_1, s_2)$
$Domain : c_{Domain}(d_1, d_2)$
$Measure : c_{MeasureSchema}(m_1, m_2)$
$Formula : c_{formula}(f_1, f_2)$.

The join of c_1 and c_2 with respect to the join predicate F and the composition function c, denoted by $c_1 \bowtie_{F,c} c_2$, is now defined as follows:

$c_1 \bowtie_{F,c} c_2 = \{c(p_1, p_2) | p_1 \in c_1 \wedge p_2 \in c_2 \wedge F(p_1, p_2) = true\}$.

The general definition of join can be customized by considering specific composition operators. For example, by considering the union and intersection operators presented in Section 5.2.1, we can define the union join ($\bowtie^{\cup}$) and the intersection join ($\bowtie^{\cap}$) as follows:

$c_1 \bowtie_F^{\cup} c_2 = \{p_1 \cup p_2 | p_1 \in c_1 \wedge p_2 \in c_2 \wedge F(p_1, p_2) = true\}$.
$c_1 \bowtie_F^{\cap} c_2 = \{p_1 \cap p_2 | p_1 \in c_1 \wedge p_2 \in c_2 \wedge F(p_1, p_2) = true\}$.

5.4 Cross-over operators

Finally, we provide three operators for the navigation from the pattern to the data space and back. The operators are applicable to classes of patterns or data. Nevertheless, having defined the selection operator, it is straightforward to apply them to individual patterns, too.

Drill-Through. The drill-through operator allows one to navigate from the pattern layer to the raw data layer. Such operator, thus, takes as input a pattern class and returns a raw data set. More formally, let c be a class of pattern type *pt* which contains patterns $p_1, \ldots, p_n$. Then, the drill-through operator is denoted by $\gamma(c)$ and it is formally defined as follows: $\gamma(c) = \{d | \exists p, p \in c \ \wedge \ d \hookrightarrow p\}$ which is equivalent to: $\gamma(c) = \mathcal{I}_e(p_1) \bigcup \cdots \bigcup \mathcal{I}_e(p_n)$.

Data covering. Given a pattern p and a dataset D, sometimes it is important to determine whether p represents D or not. In other words, we wish to determine the subset S of D represented by p. Extended to a class c, this corresponds to determining the subsets S of D containing tuples represented by at least one pattern in c. To determine S, we use the mapping predicate as a query on the dataset. Let c be a class and D a dataset with schema $(a_1, ..., a_n)$, compatible with the source schema of p. The data covering operator, denoted by $\theta_d(c, D)$, returns a new dataset corresponding to all tuples in D represented by a pattern in c. More formally $\theta_d(c, D) = \{t \mid t \in D \wedge p \in c \wedge t \rightsquigarrow p\}$.

Note that, since the drill-through operator uses the intermediate mappings and the data covering operator uses the mapping formula, the covering $\theta_d(c, D)$ of the data set $D = \gamma(c)$ returned by the drill-through operator might not be the same. This is due to the approximating nature of the pattern formula.

Pattern covering. Sometimes it can be useful to have an operator that, given a class of patterns and a dataset, returns all patterns in the class representing that dataset (a sort of inverse data covering operation). Let c be a pattern class and D a dataset with schema $(a_1, ..., a_n)$, compatible with the source schema of the c pattern type. The pattern covering operator, denoted by $\theta_p(c, D)$, returns a set of patterns corresponding to all patterns in c representing D. More formally:

$$\theta_p(c, D) = \{p \mid p \in c \wedge \forall t \in D \; t \rightsquigarrow p\}.$$

Note that:

$\theta_p(c, D) = \{p \mid p \in c \wedge \theta_d(\{p\}, D) = D\}$

For some examples concerning the application of the proposed cross-over operators, we refer the reader to Section 5.5.

5.5 Motivating example revisited

Coming back to our motivating example (Section 1), we can translate the description of user operations into simple queries or operations over the pattern base. A data mining algorithm is executed over the underlying data and the results are stored in the pattern base. Assume that an algorithm generating frequent itemsets is employed and the results are stored in two classes named $\texttt{FIBranch}_1$ and $\texttt{FIBranch}_2$.

Which are the data represented by a pattern? The user selects an interesting pattern from the class $\texttt{FIBranch}_1$ and decides to find out which are the underlying data that relate to it. To this end, the drill through operator can be used. Thus, assuming that the user picks the pattern with $PID = 193$, then the operation performed is:

$$\gamma(\sigma_{PID=193}(\texttt{FIBranch}_1))$$

The result is stored in a transient data set $\texttt{Result}_1^d$.

Which are the patterns representing a data item? Out of a vast number of data that the user receives as an answer, he picks a random record and decides to see which other patterns relate to it (so that he can relate the originating pattern with these ones, if this is possible). To this end, the pattern covering operator can be used. Thus, assuming that the user picks a tuple t_1 with $RID = 733$, then the operation is:

$$\theta_p(\texttt{FIBranch}_1, t_1)$$

The result is stored in a transient pattern class $\texttt{Result}_1^p$.

Is a pattern suitable for representing a data set? The user artificially generates a new dataset and checks what kinds of patterns it supports. Then, on the basis of the result, he picks a certain pattern and tries to determine which set of data correspond to this pattern. The user creates the data set $\texttt{Cust}_3 = \sigma_{AGE>40}(\texttt{CustBranch}_1 \cup \texttt{CustBranch}_2)$ and uses again the pattern covering operator:

$$\theta_d(\texttt{FIBranch}_1 \cup \texttt{FIBranch}_2, \texttt{Cust}_3)$$

Again, the user can navigate back and forth between data and patterns through the aforementioned operators.

Could we derive new patterns from knowledge in the pattern base? The user combines patterns in classes $\texttt{FIBranch}_1$ and $\texttt{FIBranch}_2$ in order to identify pairs of intersecting frequent itemsets (i.e., pairs of frequent itemsets with a common ancestor in the frequent itemset lattice). To this end, the intersection join operator can be used as follows:

$$\texttt{FIBranch}_1 \bowtie^{\cap}_{\texttt{FIBranch}_1.p.s \cap \texttt{FIBranch}_2.p.s \neq \emptyset} \texttt{FIBranch}_2$$

Suppose the result is stored in a transient class $\texttt{Result}_1^{FJ}$. According to the intersection join definition, the structure of the patterns in $\texttt{Result}_1^{FJ}$ is a set containing two elements, corresponding to two intersecting frequent itemsets, one for each input class. The structure can be restructured in order to get just one set, corresponding for example to the union of the intersecting patterns, by applying the restructuring operator as follows:

$$\rho_{set\_destroy(p.s)}(\texttt{Result}_1^{FJ})$$

The result obtained by evaluating the previous expression can also be achieved by directly applying a join operation over classes $\texttt{FIBranch}_1$ and $\texttt{FIBranch}_2$, using the same join predicate and a composition function c defined as follows:

$c_{StructureSchema} = \texttt{FIBranch}_1.p.s \cup \texttt{FIBranch}_2.p.s$
$c_{Domain} = \texttt{FIBranch}_1.p.ad \cup \texttt{FIBranch}_2.p.ad$
$c_{MeasureSchema} = f(\texttt{FIBranch}_1.p.m, \texttt{FIBranch}_2.p.m)$
$c_{MappingFormula} = \texttt{FIBranch}_1.p.f \wedge \texttt{FIBranch}_2.p.f$

where f is the measure composition function also used in the intersection join (see Section 5.2.1).

In this case, the resulting expression would be the following:

$$\texttt{FIBranch}_1 \bowtie_{\texttt{FIBranch}_1.p.s \cap \texttt{FIBranch}_2.p.s \neq \emptyset, c} \texttt{FIBranch}_2.$$

How different are two pattern classes? The analyst is interested in determining the changes that have taken place between the patterns generated in two consecutive days. First, the user wants to find out which are the new patterns and which patterns are missing from the new pattern set. This involves a simple qualitative criterion: presence or absence of a pattern. Based on shallow equality, we take the difference between the frequent itemsets generated in the previous and the current day. If we tag the current set of frequent itemsets with *new* and the previous one with *old*, the operation for the first branch is: $\texttt{FIBranch}_1^{new} - \texttt{FIBranch}_1^{old}$. The result is stored in a transient pattern class $\texttt{Result}_2^p$.

Is the difference of two pattern classes significant? Then, the user wants some quantitative picture of the pattern set: Which patterns have significant differences in their measures as compared to their previous version? So, we need to take pairs of patterns whose similarity is above a threshold t and the difference in precision is larger than a threshold e. A simple query is formulated by the user:

$$\{(x, y) \mid x \in \texttt{FIBranch}_1^{new} \wedge y \in \texttt{FIBranch}_1^{old} \wedge \xi_e(x, y) > t$$

$$\wedge x.precision - y.precision > \epsilon\}$$

How similar are two patterns? Then the user wants to see trends: how could we predict that something was about to happen? So, he picks a new (deleted) pattern and tries to find its most similar pattern in the previous (new) pattern set, through a similarity function. Assuming the user picks pattern p from the results presented to him and the threshold is t:

$$\{\xi_e(x, p) \mid x \in \texttt{FIBranch}_1^{new} \wedge \xi_e(x, p) > t\}$$

6 Related work

Although significant effort has been invested in extending database models to deal with patterns, no coherent approach has been proposed and convincingly implemented for a generic model. However, since the problem of pattern management is very interesting and widespread, many researchers (both from the academic world and the industrial one) working in several different research areas are devoting efforts on this. The main involved areas are standards, inductive databases, and knowledge and pattern management systems. Of course, target issues in pattern management in each area can be different. For example, standardization efforts mainly deal with pattern representation in order to achieve interoperability and pattern exchange among different systems. On the other hand, efforts in the area of knowledge management systems try to address theoretical and practical aspects concerning modeling and manipulation of patterns.

In the area of standards, there exist several standardization efforts for modeling patterns. As already pointed out, the aim of those approaches is to provide support for the interchange of data mining results and metadata between heterogeneous systems. One of the most popular effort for modeling patterns is the Predictive Model Markup Language (PMML) [4], which is an XML-based modeling approach for describing data mining models (i.e., data mining results algorithms, procedures, and parameters). Its primary issue is to support the exchange of data mining models between different applications and visualization tools. The ISO SQL/MM standard [5] has been proposed with the same purpose. Differently from PMML, it is based on SQL technology and provides SQL user-defined types to model patterns and several methods for their extraction and manipulation. In the programming language context, the Java Data Mining API (JDMAPI) specification [6] has been proposed in order to provide data mining modeling support within the Java environment. It addresses the need for a language-based management of patterns. By adopting a JDM API a vendor may integrate its application with other existing mining applications (even if they are non proprietary) running in a Java environment. Finally, another important effort in this area is represented by the OMG Common Warehouse Model (CWM) framework [15], which is a more generic modeling approach, mostly geared towards data warehousing metadata.

All these approaches do not provide a generic model capable of handling arbitrary types of patterns. Rather, only some predefined pattern types are supported. For example, all the approaches support common pattern types like association rules and clusters. Moreover, such approaches mainly address representation issues; only SQL/MM and JDM consider manipulation problems, tailored to the supported pattern types. No similarity issues are considered. Further, in some cases (for example in PMML), they do not provide support for the representation and the management of the relationship between patterns and raw data from which they have been generated.

From a more theoretical point of view, often research has not dealt with the issue of pattern management per se, but, at best, with loosely related problems. For example, the paper by Ganti et. al. [12] deals with the measurement of similarity (or deviation, in the authors' vocabulary) between decision trees, frequent itemsets and clusters. Although this is already a powerful approach, it is not general enough for our purposes. The most relevant research effort in the literature concerning pattern management is found in the field of inductive databases, meant as databases that, in addition to data, also contain patterns

[16,17]. The field of inductive databases is very active and many research efforts have been devoted to it in the CINQ project [18]. The overall goal of this project is to define the basic theoretical and practical issues related to inductive query processing, which aims at extracting interesting knowledge from data. In this context, knowledge discovery is considered as an extended querying process involving both data and patterns. The CINQ project is under development and, until now, only few types of patterns have been considered (e.g. association rules, clusters, and – recently – frequent itemset [19]). No arbitrary, user-defined pattern types can be represented and manipulated. However, in the future, CINQ researchers plan to consider other pattern types (e.g., graph, episode, datalog queries, etc) and to provide a general inductive framework for all of them.

Our approach differs from approaches based on inductive database systems in two main aspects. First, 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. Second, unlike [16], we claim that the peculiarities of patterns in terms of structure and behavior, 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.

Finally, we remark that even if some languages have been proposed for pattern generation and retrieval [20,21,22,23], they mainly deal with specific types of patterns (in general, association rules) and do not consider the more general problem of defining sufficiently expressive languages for querying heterogeneous patterns. Such rule languages are usually completely integrated in a database environment and extend the expressive power of SQL by introducing primitives for rule generation and rule querying. The result is a query language providing support for extracting patterns with a certain type. For instance, the approaches in [20,21] deal only with association rules and clusters of association rules, whereas the approaches in [22,23] deal with association rules and patterns resulting from a classification mining process.

Recently a unified framework for data mining and analysis has been proposed: the 3W Model [24]. The model comprises three different worlds: the I-world, the E-world, and the D-world. The type of patterns that can be represented in such a model is the ‘region’, i.e., a collection of points in the space spanned by the data. In the I-world each region is intensionally described, whereas in the E-world it is extensionally represented (i.e., for each region the members are listed). Finally, raw data from which regions are populated (by means of a mining algorithm) are stored in the D-world, in the form of relations. Such kind of patterns is very suitable to represent patterns resulting from the application of mining algorithms which split a given data set into collections of subset, i.e., classification algorithms. For example, decision trees, clusters, and frequent itemsets can be conveniently represented. Besides the model for representing patterns by using the three worlds, an algebra for manipulating patterns and data has been proposed. Such an algebra, called *dimension algebra*, provides query operators (extending the relational ones) and operations to move from a world to another. In this way, it allows one to express and manipulate the relationships that exist among a pattern, its intensional description, and the raw data set from which it has been generated.

To our knowledge, this is the first effort towards a unified mining and analysis framework taking into account formal aspects. Unfortunately, the framework is not general enough to support the representation and manipulation of arbitrary pattern types. For example, it can be shown that I-world regions slightly correspond to the approximate image we associate with arbitrary patterns whereas the E-world is a sort of intermediate mapping. However, differently from the proposed framework, no clear separation between pattern structure and images is provided, which may however simplify some types of pattern management.

7 Implementation details

In order to assess the usability of the proposed pattern model and languages, we have developed a PBMS prototype, called PSYCHO (*Pattern base management SYstem arCHitecture prOtotype*) [7]. The following simplifications have been taken into account in designing PSYCHO:

- source data is assumed to be stored in an (object-)relational DBMS;
- the intermediate mapping has not been implemented (and therefore, predicates relying on explicit images have not been implemented too);
- the linear constraint theory has been used for the mapping formula representation.

Even with the above limitations, PSYCHO implements the heterogeneous pattern model proposed in Section 3 and a pattern query language (called PQL) offering query capabilities for selecting and combining heterogeneous patterns and for combining patterns and data (cross-over queries) according to what presented in Section 5. Moreover, PSYCHO provides a pattern definition language (called PDL), supporting the user in the definition of basic model entities - i.e., pattern types, patterns, and classes - and a pattern manipulation language (called PML), supporting the management of both patterns resulting from a mining process and user-defined patterns along with other advanced pattern manipulation operations, such as pattern-data synchronization.

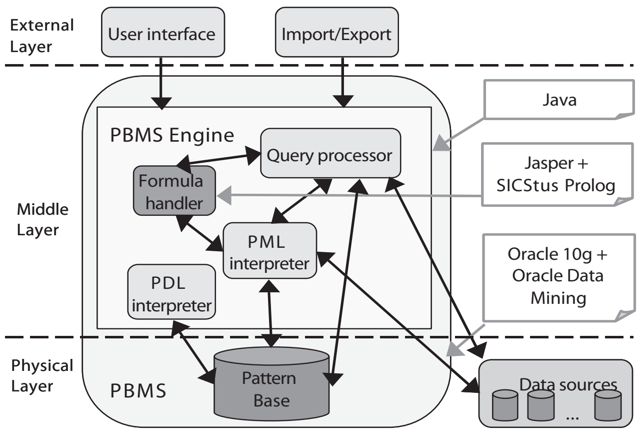


Fig. 6. The 3 layers architecture of PSYCHO

PSYCHO architecture relies on Oracle [25] and Java technologies and can exploit Oracle Data Mining (ODM) server functionalities when dealing with standard data mining patterns. The architecture is composed of three distinct but communicating layers (Fig. 6), resulting in a completely distributed architecture, where the pattern base, source data, and all the other PSYCHO modules can reside on different hosts. In the following, each layer will be described in more details.

Physical Layer The physical layer contains both the *Pattern Base* and the *Data Source*. The *Data Source* is a distributed database containing raw data from which patterns have been extracted. In the current PSYCHO version, we assume raw data are stored in a relational or object-relational DBMS. The *Pattern Base* corresponds to an object-relational database in Oracle 10g DBMS [25] storing pattern types, patterns, and classes. In particular, pattern types are implemented as Abstract Data Types (ADTs) and patterns of such pattern types are instances of those ADTs. Classes are collections of instances of a certain pattern type, thus they are implemented as tables typed over the corresponding ADT. The Pattern Base also contains PML and PQL interfaces, realized using PL/SQL functions and procedures which are invoked by the PBMS Engine (see below).

Middle layer The middle layer consists of the *PBMS Engine* component, whose aim is to execute requests sent to the PBMS. Together with the Pattern Base, it represents the core of the PSYCHO prototype. The PBMS Engine has been implemented in Java and is logically divided into three main submodules, dedicated to parse PQL, PML, and PDL requests, respectively, and to execute them through calls to the right functions and procedures defined in the Pattern Base.

Concerning PML requests, pattern extraction can use either mining functions provided by ODM or other mining functions provided by the PBMS. To this purpose, PSYCHO contains a library of predefined mining functions but new ones can be defined by the user. The Query Processor is used in case patterns have to be filtered (for example, only patterns with specific measures have to be generated, synchronized, deleted, or inserted in a given class).

PQL requests are managed by the *Query Processor* which first determines whether the query simply involves patterns or it requires to access both data and patterns (i.e., if it is a so called cross-over query). In the former case, the Query Processor interacts with the Pattern Base; in the second case, it interacts with both the Pattern Base and the Data Source. The *Formula Handler* is implemented as a Java module, using the Jasper package, interacting with SICStus Prolog environment [26]. It is used by the Query Processor for executing predicates relying on approximated pattern images.

External Layer The external layer corresponds to a set of user interfaces from which the user can send requests to the engine and import/export data in other formats. In the current PSYCHO version, user requests can be specified through a simple textual shell, where the user can specify his/her request using an SQL-like syntax. The request is translated into the corresponding PDL, PML or PQL request in SQL-like syntax and passed to the PBMS Engine for execution. Import/Export of association rules represented in PMML [4] is also supported, by exploiting, when possible, the import/export functionalities of the ODM server.

PSYCHO can be used as a unique, global environment for knowledge workers who wish to interact with both the PBMS - handling patterns - and the DBMS - managing source data represented by patterns. This is primarily due to the fact that all pattern languages provided by PSYCHO (i.e., the definition, the manipulation, and the querying languages) have an SQL-like syntax and the PSYCHO textual shell is able to process not only PDL, PML, or PQL commands but also standard SQL commands.

Moreover, the extensibility of the PBMS under a mining perspective - i.e., the user capability to exploit predefined and ODM mining algorithms, along with the possibility to define his own mining functions - is another innovative feature provided by PSYCHO.

In order to evaluate the potentialities of the system, we have experimented PSYCHO in several data mining and non data mining contexts [7]. The performed experiments showed that PSYCHO is a flexible enough tool to cope with heterogeneous pattern management issues.

In its current version, PSYCHO relies on proprietary technologies. We are currently working on a new version of PSYCHO relying on open-source technologies.

8 Conclusions and future work

In this paper we have dealt with the issue of modeling and managing patterns in a database-like setting. Our approach is based on the use of a Pattern-Base Management System, enabling the storage, querying, and management of interesting abstractions of data which we call patterns. In this paper, we have formally defined the logical foundations for pattern management through a model that covers data, patterns, and their intermediate mappings. Moreover, we have presented a pattern specification language for pattern management along with safety restrictions. Finally, we have introduced queries and query operators and identified interesting query classes.

Several research issues remain open. First, the proposed query operators are a starting basis for further theoretical work. It is straightforward to prove that the operators are *consistent* with the model, in the sense that they all produce pattern classes or raw data relations as their result, even if , in case of patterns, the result pattern type could be new with respect to the existing ones. We also conjecture (but have not proved yet) that the proposed operators are *minimal*, in the sense that no operator can be derived from the others, even if some relationships exist between them. An *axiomatization* of the proposed operators with respect to an appropriate calculus is a clear topic of future work, well beyond the scope of this paper. Second, it is an interesting topic to incorporate the notion of type and class hierarchies in the model [3]. Third, we have intensionally avoided an extensive discussion about statistical measures in this paper: it is not a trivial task to define a generic ontology of statistical measures for any kind of patterns out of the various proposed methodologies (general probabilities, Dempster-Schafer, Bayesian Networks, etc. [27]). Moreover, the computation of measures for newly derived patterns (e.g., through a join operation) is also another research topic. Finally, interesting topics include the identification of appropriate structures for the linkage between pattern- and data- bases, the identification of algebraic equivalences among operators, as well as query processing and optimization techniques that take access methods and algebraic properties into consideration.

References

[1] P. Lyman, H. R. Varian, How much information, http://www.sims.berkeley.edu/how-much-info (2000).

[2] J. Gray, The information avalanche: Reducing information overload, http://research.microsoft.com/~Gray/Talks/ (2002).

[3] S. Rizzi, E. Bertino, B. Catania, M. Golfarelli, M. Halkidi, M. Terrovitis, P. Vassiliadis, M. Vazirgiannis, E. Vrachnos, Towards a logical model for

patterns, in: Proceedings of 22nd International Conference on Conceptual Modeling (ER'03), Vol. 2813 of Lecture Notes in Computer Science, Springer, 2003, pp. 77–90.

[4] Predictive Model Markup Language (PMML), http://www.dmg.org/pmmlspecs_v2/pmml_v2_0.html (2003).

[5] ISO SQL/MM Part 6, http://www.sql-99.org/SC32/WG4/Progression_Documents/FCD/fcd-datamining-2001-05.pdf (2001).

[6] Java Data Mining API, http://www.jcp.org/jsr/detail/73.prt (2003).

[7] B. Catania, A. Maddalena, M. Mazza, Psycho: A prototype system for pattern management, in: Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05), 2005, pp. 1346–1349.

[8] M. Terrovitis, P. Vassiliadis, S. Skiadopoulos, E. Bertino, B. Catania, A. Maddalena, Modeling and language support for the management of pattern-bases, in: Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM'04), IEEE Computer Society, 2004, pp. 265–274.

[9] S. Abiteboul, C. Beeri, The power of languages for the manipulation of complex values, VLDB Journal 4 (4) (1995) 727–794.

[10] M. Escobar-Molano, R. Hull, D. Jacobs, Safety and translation of calculus queries with scalar functions, in: Proceedings of the 12th Symposium on Principles of Database Systems (PODS'93), ACM Press, 1993, pp. 253–264.

[11] D. Suciu, Domain-independent queries on databases with external functions, in: G. Gottlob, M. Y. Vardi (Eds.), Proceedings of Database Theory (ICDT'95), Vol. 893 of Lecture Notes in Computer Science, Springer, 1995, pp. 177–190.

[12] V. Ganti, R. Ramakrishnan, J. Gehrke, W.-Y. Loh, A framework for measuring distances in data characteristics, in: Proceedings of the 18th Symposium on Principles of Database Systems (PODS'99), ACM Press, 1999, pp. 126–137.

[13] I. Bartolini, P. Ciaccia, I. Ntoutsi, M. Patella, Y. Theodoridis, A unified and flexible framework for comparing simple and complex patterns, in: Proceedings of 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'04), Vol. 3202 of Lecture Notes in Computer Science, Springer, 2004, pp. 496–499.

[14] M. H. Dunham, Data Mining: Introductory and Advanced Topics, Prentice-Hall, 2002.

[15] Common Warehouse Metamodel (CWM), http://www.omg.org/cwm (2001).

[16] T. Imielinski, H. Mannila, A database perspective on knowledge discovery, Communications of the ACM 39(11) (1996) 58–64.

[17] L. De Raedt, A perspective on inductive databases, SIGKDD Explorations 4(2) (2002) 69–77.

[18] The CINQ project, http://www.cinq-project.org (2003).

[19] J. F. Boulicaut, Inductive databases and multiple uses of frequent itemsets: the CINQ approach, in: Database Support for Data Mining Applications: Discovering Knowledge with Inductive Queries, Vol. 2682 of Lecture Notes in Computer Science, Springer, 2004, pp. 3–26.

[20] R. Meo, G. Psaila, S. Ceri, An extension to SQL for mining association rules, Data Mining and Knowledge Discovery 2 (2) (1999) 195–224.

[21] T. Imielinski, A. Virmani, MSQL: A query language for database mining, Data Mining and Knowledge Discovery 2 (4) (1999) 373–408.

[22] J. Han, Y. Fu, W. Wang, K. Koperski, O.Zaiane, Dmql: A data mining query language for relational databases, in: Proceeding of the SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'96), Montreal, Canada, 1996.

[23] M. G. Elfeky, A. A. Saad, S. A. Fouad, ODMQL: Object data mining query language, in: Proceedings of International Symposium on Objects and Databases, Vol. 1944 of Lecture Notes in Computer Science, Springer, 2000, pp. 128–140.

[24] T. Johnson, L. V. S. Lakshmanan, R. T. Ng, The 3W model and algebra for unified data mining, in: Proceedings of 26th International Conference on Very Large Data Bases (VLDB'00), Morgan Kaufmann, 2000, pp. 21–32.

[25] Oracle 10g, http://www.oracle.com/database/.

[26] Sicstus prolog (v.3), http://www.sics.se/isl/sicstuswww/site/index.html.

[27] A. Siblerschatz, A. Tuzhillin, What makes patterns interesting in knowledge discovery systems, IEEE Transactions on Knowledge and Data Engineering 8 (6) (1996) 970–974.

[28] The PANDA project, http://dke.cti.gr/panda (2002).