Flexible Pattern Management within PSYCHO

Barbara Catania and Anna Maddalena

Dipartimento di Informatica e Scienze dell'Informazione
Università degli Studi di Genova (Italy)
{catania,maddalena}@disi.unige.it

Abstract. Patterns are concise, but rich in semantic, representation of data. The approaches proposed in the literature and by commercial systems for pattern management usually deal with few types of knowledge artifacts and mainly concern pattern extraction issues. Little effort has been posed in designing an overall framework dedicated to the management of different types of patterns, possibly user-defined, in an homogeneous way. PSYCHO (Pattern based SYstem arCHitecture prOtotype) is a recently developed tool, built on top of Oracle technologies, for generating, representing, and manipulating heterogeneous patterns, possibly user-defined. The aim of this paper is to present the PSYCHO system, by discussing the underlying theory, the reference architecture, and providing concrete examples of its usage.

1 Introduction

A pattern can be defined as a *compact* and *rich in semantics* representation of raw data. Clusters, association rules, frequent itemsets, symptom-diagnosis correlation, and moving object trajectories are common examples of patterns. Pattern management is an important issue in many different contexts, such as data mining, information retrieval, image processing, and clickstream analysis.

The specific characteristics of patterns make traditional DBMSs unsuitable for pattern representation and management. In particular, patterns can be generated from different application contexts resulting in very heterogeneous structures. Moreover, patterns can be mined (*a-posteriori patterns*) but also known by the users and used for example to check how well some data source is represented by them (*a-priori patterns*). To maintain the semantic alignment between patterns and raw data it is also important to determine whether existing patterns, after a certain time, still represent the data source from which they have been generated, possibly being able to update pattern information. Finally, all kinds of patterns should be manipulated (e.g. extracted, synchronized, deleted) and queried through dedicated languages. All the previous reasons motivate the need for ad hoc *Pattern Management Systems (PBMSs)*, i.e., *systems for handling (storing/processing/retrieving) patterns defined over raw data* [10].

Recently, several approaches have been provided for pattern management. Scientific community efforts mainly deal with the definition of a pattern management framework providing a full support for heterogeneous patterns. In the

3W Model [5] and in the PANDA framework [10], raw data are stored and managed in a traditional way by using a DBMS whereas patterns are stored and managed by a dedicated PBMS. In the inductive databases approach, mainly investigated in the context of the CINQ project [6], raw data and patterns are stored together, by using the same data model, and managed in the same way. On the other hand, industrial proposals mainly deal with standard representation purposes for patterns resulting from data mining, in order to support their exchange between different architectures. Examples of such approaches are the Predictive Model Markup Language (PMML) [11] and the Java Data Mining API (JDM) [8]. PMML simply deals with pattern representation issues, providing an XML-based format to represent data mining results and the used mining algorithm. JDM represents patterns within the Java environment and provides manipulation support through JAVA primitives. In all these cases, no user-defined patterns can be specified. Concerning commercial systems, the most important DBMSs (e.g., Oracle and MS-SQL Server) provide an applicational layer offering features for representing and managing typical data mining patterns.

Even if several approaches have been proposed, an integrated environment satisfying the requirements introduced above is still missing. Starting from these limitations and relying on the results achieved in the context of the PANDA Project [10], we have designed and implemented PSYCHO (*Pattern based System arCHitecture prOtotype*) [4, 12], a system built on top of Oracle technologies, for generating, representing, and manipulating heterogeneous patterns, possibly user-defined. According to our knowledge, PSYCHO is the first proposal of a PBMS system coping with most of the features cited above. Differently from existing proposals, that are mainly focused on common data mining patterns (e.g. association rules, frequent itemsets, and clusters), PSYCHO allows the design of user-defined pattern types and the management of both a-posteriori and a-priori patterns. Moreover, by exploiting the logical model proposed in [3], it allows the representation of pattern validity information and pattern hierarchies. Besides basic manipulation operations, PSYCHO supports synchronization between patterns and raw data. Concerning query capabilities, PSYCHO, by exploiting the power of the logical model, supports interesting queries combining both data and patterns in order to get a deeper knowledge of their correlations.

This paper is organized as follows. In Section 2, the model underlying the PSYCHO development is briefly discussed. Details about the PSYCHO architecture are presented in Section 3, while Section 4 is focused on PSYCHO usage. Finally, Section 5 presents some concluding remarks and outlines future work.

2 PSYCHO: the model

PSYCHO is a prototype of a Pattern Based Management System (PBMS) exploiting the logical framework for pattern management proposed in the context of the PANDA Project [10]. In particular, it relies on the logical model and the languages for pattern generation, manipulation, and querying introduced in [2–

4]. The PSYCHO logical model used to represent patterns is based on three basic concepts: *pattern type*, *pattern*, and *class* (see [3, 10] for further details).

A *pattern type* gives a formal description of the pattern structure of each of its instances. It is characterized by six components: (i) the *pattern name* n; (ii) the *structure schema* ss, which defines the structure of the patterns instances of the pattern type; (iii) the *source schema* ds, which describes the dataset from which patterns, instances of the pattern type being defined, are constructed; (iv) the *measure schema* ms, which is a tuple describing the measures which quantify the quality of the source data representation achieved by the pattern; (v) the *formula* f, carrying the semantics of the pattern. f is a constraint-based formula describing, possibly in an approximate way, the relation between data represented by the pattern and the pattern structure; (vi) the *validity period schema* vs, defining the schema of the temporal validity interval associated with each instance of the pattern type.

Patterns are instances of a specific pattern type, containing the proper instantiation of the corresponding schema components in the pattern type. In particular, the formula component of a pattern is obtained from the one in the corresponding pattern type pt by instantiating each attribute appearing in ss with the corresponding value, and letting the attributes appearing in ds range over the source space.

We remark that the data source represents the overall dataset over which the pattern has been extracted (in case of a-posteriori patterns). On the other hand, the formula represents, in an intensional and possibly approximated way, the specific subset of data represented by the pattern.

A *class* is a set of semantically related patterns and constitutes the key concept in defining a pattern query language. A class is defined for a given pattern type and contains only patterns of that type. A pattern may belong to any number of classes. If it does not belong to any class, it cannot be queried.

In the context of the PANDA Project [10], some interesting relationships supporting hierarchical pattern definition have also been proposed. Among them, we recall: the *composition* relationship - between a pattern and those used to define its structure - and the *refinement* relationship - between a pattern and those belonging to its data source. PSYCHO supports the definition of complex patterns based on refinement and composition hierarchy notions.

Based on the considered pattern model, PSYCHO provides three languages for the management of both a-priori and a-posteriori patterns: (i) the *Pattern Definition Language (PSY-PDL)*, used for defining new pattern types, classes, and *mining functions*, used for pattern generation; (ii) the *Pattern Manipulation Language (PSY-PML)*, used to perform operations such as insertion, extraction, deletion, update, synchronization of patterns, as well as insertion or removal of patterns into or from a class defined for the proper pattern type; (iii) the *Pattern Query Language (PSY-PQL)*, used to retrieve patterns and correlate them with data they represent (*cross-over queries*). For all these languages, an SQL-like syntax has been provided.

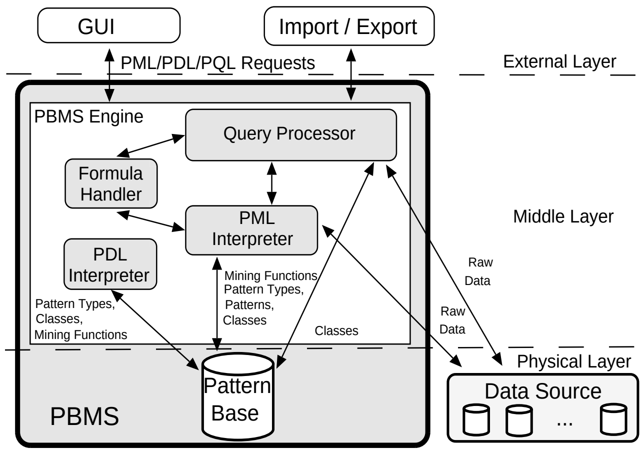

Fig. 1. The 3 layers architecture of PSYCHO

3 PSYCHO: the architecture

The PSYCHO [4] architecture relies on Oracle and Java technologies and it is composed of three distinct layers as depicted in Fig.1: (i) the physical layer, containing both patterns and data (possibly residing at different sites); (ii) the middle layer, coinciding with the kernel of the system and supporting all functionalities for pattern manipulation and retrieval; (iii) the external layer, corresponding to a set of user interfaces from which the user can send requests to the engine and import/export data in other formats. Due to design and implementation choices, the current version of PSYCHO is tightly coupled with Oracle technology and, when possible, it allows the user to exploit the Oracle Data Mining (ODM) server functionalities [9].

In the following, we describe each one of the PSYCHO layer (Sections 3.1,3.2, and 3.3); then, in Section 3.4, we discuss the technology adopted for the communication between layers.

3.1 Physical Layer

The Physical Layer contains both the *Pattern Base* and the *Data Source.*

The *Pattern Base* component contains pattern types, a-priori and a-posteriori patterns, and classes. PSYCHO relies on the object-relational model of Oracle 10g [9] for pattern storage. Concerning the pattern formula, we consider two distinct representations: an operational one, by which the formula is indeed a predicate over data source elements implemented as an Oracle PL/SQL stored function; a declarative one, by which the formula is just a representation of a linear constraint formula (see the Formula Handler). Since the provided implementation of the Pattern Base exploits the Oracle logical model, the PSY-PML and PSY-PQL interfaces are realized using PL/SQL functions and procedures which are invoked by the Java application implementing the PBMS Engine.

The *Data Source* is a distributed repository containing raw data from which patterns have been extracted (in case of a-posteriori patterns). Various technology can be used to store the source datasets: relational or object-relational DBMSs, XML dataset, streams, etc. In the current PSYCHO version, raw data are stored in Oracle 10g DBMS.

3.2 Middle Layer

The Middle Layer consists of the *PBMS Engine* component, which supports all functionalities for pattern manipulation and retrieval, by adequately using the Pattern Base and Data Sources when required. The PBMS Engine and the Pattern Base represent the core of the PSYCHO prototype. The PBMS Engine has been implemented in Java and is logically divided into three main submodules, each of which is dedicated to parse, interpret, and execute PSY-PQL, PSY-PML, and PSY-PDL requests, respectively, that are sent to the physical layer components. There is also a dedicated component to handle intensional pattern-data mapping, i.e. the pattern formula component.

The *PDL Interpreter* takes as input, from the higher layer, a PSY-PDL request for a pattern type or class definition and translates it into calls to the right functions and procedures defined in the Pattern Base.

The *PML Interpreter* takes as input a PSY-PML request and translates it into calls to the right functions/procedures of the Pattern Base. Some manipulation operations, such as, for instance, pattern extraction and synchronization, require an interaction with the Data Source to get the data from which patterns have to be generated. In the current release, pattern extraction is performed by using either mining functions provided by ODM server (i.e., a variant of the A-priori algorithm for association rules or the K-means or the proprietary O-cluster algorithm for clusters) or other mining functions (possibly defined by the user) stored in the PBMS. We outline that the Query Processor has to be invoked by the PML Interpreter in case patterns have to be filtered (e.g. only patterns with specific measures have to be generated, synchronized, deleted, or inserted in a given class).

The *Query Processor* translates an input PSY-PQL query into calls to the right functions and procedures defined in the Pattern Base. For non cross-over queries, only the Pattern Base and, eventually, the Formula Handler are involved in the query process. On the other side, to execute cross-over queries, the Data Source may also be required. We point out that queries can also use formulas for pattern comparison and selection. When formulas are used under the operational semantics, the queries are executed directly by the Query Processor. On the other hand, when they are used under the declarative semantics, the Formula Handler module is required to execute the query. As already said, some query requests can be generated by the PML Interpreter; in this case the Query Processor computes the answer and sends it directly to the PML Interpreter.

The *Formula Handler* deals with the declarative management of formulas, i.e., with constraints. It is used by the PSY-PML and PSY-PQL interpreters when computations over formula constraints are required. Within PSYCHO, the Formula Handler is implemented as a Java module, using the Jasper package [7] for interacting with SICStus Prolog environment [13]. Computations over formulas concern comparisons (equivalence, containment) between the specific sets of data from which patterns have been extracted and are implemented by the Formula Handler through typical logical operations such as equivalence or subsumption.

3.3 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. User requests can be specified through a *GUI*, providing a visual environment (in the current release, it is a simple shell), where the user can specify his/her request using an SQL-like syntax. The *Import/Export module* supports the import and the export in the PBMS of patterns already represented in standard formats (e.g. PMML models [11]).

3.4 Communication between Layers

As already stated, the Pattern Base is integrated within the Oracle DBMS managing the Data Source. The PBMS Engine is placed immediately above the Pattern Base. It creates and manages the connection with the Oracle DBMS and the calls to the stored functions and procedures defined in the Pattern Base. The communication between the Pattern Base and the PBMS Engine is, therefore, the classical communication between a Java application and a DBMS, through a JDBC driver. On the other hand, the communication between the PSYCHO Engine and the external layer is established using the mechanism of sockets. In this way the whole system is more flexible and a completely distributed architecture can be realized, where the different PSYCHO components - i.e., the pattern base, the data sources, the PBMS engine, and the external modules - can be placed on different hosts. In details, the PBMS Engine opens a socket on a fixed port and waits for connections from the outside. Whenever an external module needs to communicate with the engine, it makes a connection, creates a serializable object that encapsulate the request, and sends it to the PBMS Engine.

4 PSYCHO: usage

In the following, PSYCHO usage and its peculiarities in pattern management are highlighted by considering a typical data mining scenario dealing with market-basket association rules. Additional examples can be found at [12]. Association rules are a well-known data mining pattern type, therefore they are managed

n: AssociationRule
ss: TUPLE(head: SET(String),
body: SET(String))
ds: BAG(transaction: SET(String))
ms: TUPLE(confidence: REAL,
support: REAL)
f: $\forall x(x \in$ head $\vee$ x $\in$ body $\Rightarrow$
$x \in$ transaction)
vs: [start : DAY, end : DAY)

(a)

pid: 512
s: < head={'Boots'},
body={'Socks','Hat'} >
d: 'SELECT SETOF(article) AS transaction
FROM sales
GROUP BY transactionId'
m: < confidence=0.75, support=0.55 >
f: {transaction : $\forall x$(
$x \in$ {'Boots', 'Socks', 'Hat'}
$\Rightarrow x \in$ transaction)}
v: [1-DEC-2005,31-MAR-2006)

(b)

Fig. 2. Association Rules modeling: (a) the pattern type *AssociationRule* and (b) one of its pattern instance.

by any commercial system dealing with data mining [9, 8]. However, PSYCHO allows one to perform several operations that are not directly supported by other existing tools.

We assume source data is stored in a table with schema $(DSid, Item_1, ..., Item_n)$, where each tuple represents a sale transaction identified by $DSid$; $Item_i$ is either 1 or 0, and $Item_i = 1$ means that the corresponding transaction contains $Item_i$. According to the PANDA model, the pattern type for modeling association rules and one of its pattern instances are shown in Fig.2(a) and 2(b).

Besides the structure - i.e., head and body in the case of association rules - and the measures - i.e., support and confidence in the case of association rules - in PSYCHO each pattern type is associated with three additional components: the extensional formula, the intensional formula, and the validity period. The extensional formula is just a PL/SQL function that takes a source dataset and returns the subset of such dataset (possibly approximatively) represented by the pattern. The intensional formula has the same meaning, but it is intensionally represented through a Prolog predicate, defined by a set of linear constraints. Note that the two formulas used by PSYCHO (i.e., the extensional one and the intensional one) implement the formula component of the underlying logical model under an operational and a declarative perspective, respectively. The validity period is just a temporal interval inside which we assume the information represented by the pattern is reliable. The PSY-PDL command to create the pattern type *AssociationRule* is sketched in Fig.3(a).

In the following, we show a concrete example of PSYCHO usage by describing typical steps of a data mining process concerning pattern generation and system population, pattern analysis and querying, and pattern maintenance.

PBMS Population and Class Management In this step, we show how PSYCHO can be used to: (i) use various mining functions to extract a-posteriori

```
CREATE PATTERN TYPE AssociationRule
STRUCTURE head CharArray, body CharArray /* CharArray is an already
defined type*/
...
MEASURE support REAL, confidence REAL
...
FORMULA EXTENSIONAL ON varDS ...
/* retrieve items in the source dataset effectively represented by a
pattern of type AssociationRule*/
FORMULA INTENSIONAL ARFormula_INT; /* ARFormula_INT is an existing
Prolog predicate */
```

(a)

```
CREATE PATTERN TYPE ClusterOfAR
STRUCTURE ruleset ARSET
/*ARSET is an already defined type modeling arrays of AssociationRule
references*/
...
MEASURE Svalidity REAL
...
FORMULA EXTENSIONAL ON varDS ...
/* retrieve items in the source dataset effectively represented by a
pattern of this type*/
```

(b)

Fig. 3. (a) PSY-PDL definition of the pattern type *AssociationRules* and (b) PSY-PDL definition of the pattern type *ClusterOfAR*.

patterns; (ii) directly insert a-priori patterns; (iii) create new patterns by recomputing existing ones; (iv) handle pattern classes.

Pattern extraction Given a pattern type, several mining functions can be used to extract patterns of that type from a given dataset. For example, to mine association rules, we can use a PSYCHO proprietary Java mining function *apriori* implementing the well-known Apriori algorithm. [1]. We may specify that the support of the extracted rules must be higher than 0.4 and the confidence higher than 0.7; the validity period of the extracted rules is set from 01-jul-2005 to 10-aug-2005. Before extracting patterns, it may be useful to create a class, called *AR_PSY*, where extracted patterns are stored (if no class is used, patterns are stored in the system but they cannot be used in queries) (see Fig. 4(a)). Moreover, it is also possible to mine association rules by using different mining functions. For instance, other association rules can be extracted by using the mining function *Apriori_ODM* calling the one available in ODM [9] within a PSY-PML command similar to the previous one.

Direct insertion Single association rules can also be directly inserted in PSYCHO by using the 'DIRECT INSERT' PSY-PML command (see Fig.4(b)).

```
CREATE CLASS AR_PSY OF
AssociationRule;

EXTRACT PATTERNS OF
AssociationRule ar
FROM itemsDS_30
USING Apriori(0.4,0.7)
VALID FROM '01-jul-2005'
      TO '10-aug-2005'
INTO CLASS AR_PSY;
```
(a)

```
DIRECT INSERT PATTERN
OF AssociationRule ar
FROM itemsDS_30
STRUCTURE (
chararray('BREAD','MILK'),
chararray('JAM','BUTTER','WINE')
)
MEASURE (0.5,0.7)
VALID FROM '01-aug-2005'
      TO '15-aug-2005'
INTO CLASS AR_PSY;
```
(b)

```
RECOMPUTE PATTERNS
OF AssociationRule ar
ON itemsDS_5
USING AR_mf;
```
(c)

```
CREATE CLASS AR_high_support
OF AssociationRule;

INSERT INTO CLASS
AR_high_support ar
WHERE ar.m.support >= 0.5;
```
(d)

Fig. 4. Manipulation operations over *AssociationRules*: (a) Extraction of association rules using the *apriori* mining function; (b) Direct insertion of association rules into class *AR_PSY*; (c) Recomputation of association rules over dataset *itemsDS_5* by using function *AR_mf*; (d) Definition and population of class *AR_high_support*.

Recomputation New patterns can also be generated by recomputing measures of existing ones over a new data source. Various functions recomputing measure values for instances of a given pattern type, upon a given dataset can be defined. For association rules, PSYCHO provides a measure function for computing confidence and support over a given data source, named *AR_mf*. Fig. 4(c) reports an example of pattern recomputation using this measure function.

Class Management Suppose the user wants to define a class containing all association rules having support greater than 0.5%. Such class, named *AR_high_support*, can be created and association rules satisfying the previous condition can be inserted in it (see Fig.4(d)). The class can then be used for query purposes.

Querying PSY-PQL supports the following querying features: (i) simple queries involving predicates dealing with pattern components; (ii) pattern composition; (iii) nested queries; (iv) pattern-data reasoning (cross-over queries). In the following, we discuss each class of queries.

Simple queries Simple queries allow one to select patterns from a given class, according to a variety of predicates. In particular, PSYCHO supports selection based on two validity notions: temporal validity and semantic validity. A pattern is temporally valid with respect to a certain date if its validity period contains the specified date. A pattern is semantically valid with respect to a certain dataset and a set of thresholds if the pattern measures computed over the input datasets are better than those provided as input. Several examples of queries checking temporal and semantic validity are shown in Table 1 (e.g. Q2,Q4,Q7).

QID	**Query**	**PSY-PQL statement**

Q1	Retrieve all association rules from *AR_PSY* with confidence greater than or equal to 0.75	SELECT *
		 FROM AR_PSY ar
		 WHERE ar.m.confidence >= 0.75;

Q2	Retrieve all association rules that are valid on July 20, 2005	SELECT *
		 FROM AR_PSY ar
		 WHERE isTvalid(ar,'20-jul-2005')=1;

Q3	Retrieve all association rules valid during the period August,1 - August,10 2005	SELECT *
		 FROM AR_PSY ar
		 WHERE during(ar.v,
		 valPeriod('01-aug-2005','10-aug-2005'))=1;

Q4	Retrieve all semantically valid rules (with respect to their data source), with support and confidence greater than or equal to 0.4	SELECT *
		 FROM AR_PSY ar
		 WHERE isSvalid(ar,ar.d,'AR_mf',
		 AssociationRuleMeasure(null,0.4,0.4))=1;

Q5	Determine all association rules with confidence greater or equal to 0.75 or which are temporally valid in August,15 2005	SELECT *
		 FROM AR_PSY ar
		 WHERE ar.m.confidence >= 0.75
		 OR isTvalid(ar,'15-aug-2005')=1;

Q6	Determine all association rules, obtained as the transitive closure of two existing association rules, having at least two items in the body	SELECT *
		 FROM AR_PSY ar1 CJOIN AR_PSY ar2
		 WITH Trans_closure_ar
		 WHERE ar2.s.body.count >= 2;

Q7	Select among rules with at least confidence value equal to 0.7 the ones which are temporally valid in 15-aug-2005	SELECT *
		 FROM (SELECT *
		 FROM AR_PSY ar
		 WHERE ar.m.confidence >= 0.7) rule
		 WHERE isTvalid(rule,'15-aug-2005') = 1;

Q8	Which data are represented by association rules with confidence greater than 0.75?	DRILL THROUGH (
		 SELECT *
		 FROM AR_PSY ar
		 WHERE ar.m.confidence > 0.75) rule;

Q9	Determine whether the association rule with PID=1001348 is suitable for representing a certain dataset *itemsDS_30*	DATA COVERING (
		 SELECT *
		 FROM AR_PSY pr
		 WHERE pr.PID = 1001348) ar
		 FOR itemsDS_30;

Q10	Determine which patterns, belonging to class *AR_PSY* and having a confidence higher than 0.8, represent dataset *itemsDS_30*	PATTERN COVERING itemsDS_30 FOR AR_PSY ar WHERE ar.m.confidence >= 0.8;

Table 1. Several PSY-PQL queries over class *AR_PSY*

Pattern composition Within PSYCHO, different types of joins are available. In the actual release, two types of join are provided: a general one (*CJOIN*) and a specific one (*INTERSECTIONJOIN*). The *CJOIN* takes two classes and, for each pair of patterns, the first belonging to the first class, the second belonging to the second one, it applies a specified composition function, specifying the structure of the resulting patterns. On the other hand, the $INTERSECTIONJOIN$ takes two classes and returns new patterns, whose structure is a combination of the input structures and whose intensional formula is the conjunction of input intensional formulas. For instance, the user may be interested in calculating an association rule obtained as the transitive closure of two existing association rules $A \leftarrow B$ and $B \leftarrow C$ extracted from data sources D_1 and D_2, respectively. The new association rule $A \leftarrow C$ can be obtained by applying function $Trans\_closure\_ar$, which generates such a new rule by computing also a new data source, which is the union of D_1 and D_2, and a new validity period, which is the intersection of the validity periods of the two input rules. PSY-PDL supports the specification of such composition function (see [12] for more details). Query Q6 in Table 1 is an example of PSY-PQL CJOIN query.

Nested queries PSY-PQL queries can also be nested. In the current version, nesting is provided in the $FROM$ clause (see Q7 in Table 1).

Cross-over queries PSY-PQL supports pattern-data reasoning, i.e., it allows the user to specify queries involving both data and patterns. Such kind of queries are quite important in pattern management, since they allow the user to discover interesting (possibly new) correlations between patterns and data. Some examples of such queries are reported in Table 1 (Q8,Q9,Q10).

Advanced manipulation operations Differently from most existing systems and standards, PSYCHO supports various types of update operations (see Table 2): (i) synchronization, (ii) set validity, and (iii) validate.

Synchronization It allows the user to synchronize pattern measures with the current data source, that may be changed with respect to its status at extraction time, using a specific measure function. In order to perform this operation a measure function defined for the pattern type of the patterns you want to synchronize has to be used.

Validate Validating a pattern means synchronizing it, using a certain measure function, if the new measures are better then the original ones, or recomputing it, if this condition is not satisfied.

Set Validity Since PSYCHO supports a time validity associated with patterns, a manipulation operation to update the validity period of a pattern is provided.

Pattern Hierarchies PSYCHO supports hierarchies of patterns, by implementing refinement and composition relationships (see Section 2). Suppose we are interested in clusters of association rules describing correlations among sold products based on some grouping criteria (for instance, a simple clustering criteria could just divide a set of association rules into two clusters based on their semantic validity with respect to a certain dataset). By using PSY-PDL, a new pattern type *ClusterOfAR* can be defined exploiting the refinement relationship, i.e., classes of association rules previously created can be considered the data source (see Fig.3(b)). As in the case of non-hierarchical patterns, PSYCHO supports the manipulation and querying of hierarchical patterns.

Update Operation	**PSY-PML statement**

Synchronize all association rules using measure function *AR_mf*	UPDATE PATTERNS OF AssociationRule ar
	SYNCHRONIZE
	USING AR_mf
	WHERE INCLASS(ar,'AR_PSY')=1;

Validate rules in class *AR_PSY*	UPDATE PATTERNS OF AssociationRule ar
	VALIDATE USING AR_mf
	WHERE inclass(ar, 'AR_PSY')=1
	INTO CLASS AR_PSY;

Set the validity period of all association rules starting at 10-*jun*-2005 and ending at 31-*aug*-2005	UPDATE PATTERNS OF AssociationRule ar
	SET VALIDITY FROM '10-jun-2005' TO '31-aug-2005'
	WHERE INCLASS(ar,'AR_PSY')=1;

Table 2. PSY-PML update operations over class *AR_PSY*

5 Concluding remarks

In this paper, we have presented PSYCHO, a prototype system for pattern management developed on top of Oracle. After briefly presenting the underlying data model and architecture, we have presented an example of PSYCHO usage, based on a common market-basket scenario. The current PSYCHO version can be extended in several ways. For example, in the current PSYCHO release, the user interacts with the system through a simple textual shell. As a future work, we plan to extend PSYCHO with a user-friendly GUI. Another important issue under investigation consists in defining an open-source version of PSYCHO, relying on open-source technologies. To this purpose, we are currently investigating the opportunity of using the WEKA library [14], a collection of machine learning algorithms for data mining tasks written in Java, as part of the PSYCHO backend. Finally, we plan to investigate the relationships between our manipulation operations and adaptive/incremental mining solutions, with the aim of designing an incremental mining environment based on database solutions.

References

1. R. Agrawal, R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proc. of VLDB'94*, 487-499, 1994.
2. E. Bertino, B. Catania, and A. Maddalena. Towards a Language for Pattern Manipulation and Querying. In *Proc. of PaRMa'04*, 2004.
3. B. Catania, A. Maddalena, M. Mazza, E. Bertino, and S. Rizzi. A Framework for Data Mining Pattern Management. In *Proc. of PKDD'04*, pp 87-98, 2004.

4. B. Catania, A. Maddalena, and M. Mazza. PSYCHO: A Prototype System for Pattern Management. In *Proc. of VLDB'05*, pp 1346–1349, 2005.
5. L.V.S. Lakshmanan S. Johnson and R.T. Ng. The 3W Model and Algebra for Unified Data Mining. In *Proc. of VLDB'01*, pp 21–32, 2001.
6. The CINQ project. `http://www.cinq-project.org`.
7. Jasper Java Interface. `http://www.sics.se/sicstus/docs/latest/html/sicstus/Jasper.html`.
8. Java Data Mining API. `http://www.jcp.org/jsr/detail/73.prt`.
9. Oracle10g Database. `http://www.oracle.com/technology/ products/database/oracle10g/index.html`.
10. The PANDA Project. `http://dke.cti.gr/panda/`, 2002.
11. Predictive Model Markup Language (PMML). `http://www.dmg.org/pmmlspecs_v2/pmml_v2_0.html`.
12. PSYCHO Site. `http://www.disi.unge.it/person/CataniaB/psycho/`.
13. SICStus Prolog (v.3). `http://www.sics.se/isl/sicstuswww/site/index.html`.
14. WEKA site. `http://www.cs.waikato.ac.nz/~ml/weka/index.html`