Pattern Management

Irene Ntoutsi, Yannis Theodoridis

Information Systems Lab
Department of Informatics
University of Piraeus
Hellas

Technical Report Series
UNIPI-ISL-TR-2007-01

March 2007

Pattern Management*

Irene Ntoutsi and Yannis Theodoridis
Department of Informatics,
University of Piraeus, Greece
{ntoutsi,ythed}@unipi.gr

Abstract

The need for pattern management has become compulsory nowadays due to the spreading of the data mining technology as a result of the information flood. This need has been recognized by both academic and industrial parts resulting in a number of approaches towards efficient pattern management. In this chapter we review the work performed so far in the area of pattern management and outline open research directions regarding complete pattern management. [1]

1 Introduction

Nowadays a huge quantity of raw data is collected from different application domains (business, science, telecommunication, health care systems etc.). Also, the World Wide Web overwhelms us with information. According to a recent survey [19] the world produces between 1 and 2 exabytes of unique information per year, which is roughly 250 megabytes for every man, woman, and child on earth. Due to their quantity and complexity, it is impossible for humans to thoroughly investigate these data collections directly. Knowledge Discovery in Databases (KDD) and Data Mining provides a solution to this problem by generating compact and rich in semantics representations of raw data, called *patterns* [22]. Patterns are *compact* representations of raw data since they provide a high level description of raw data characteristics. Patterns are also *rich in semantics* since they reveal new knowledge hidden in the huge amount of raw data. Clusters, decision trees, association rules and frequent itemsets are some of the most well known patterns produced by Data Mining applications.

Due to the spreading of the Data Mining technology nowadays, as a result of the data flood, even the amount of patterns extracted from heterogeneous data sources is huge and, quite often, non–manageable by humans. Thus, there is a need for efficient pattern management including issues like modeling, storage,

*This research is partially supported by the Greek Ministry of Education and the European Union under a grant of the "Heracletos" EPEAEK II Programme (2003–07).

[1] Part of this work has been presented at the tutorial "Mining the Volatile Web" [23]

retrieval and manipulation of patterns. However, the majority of work in the data mining area mainly focus on efficient mining, putting aside the pattern management problem which only lately has began to gain more attention by both scientific and industrial parts. In the next sections we will review the work performed so far in the area of pattern management pointing out their advantages and disadvantages.

2 Current pattern management approaches

The need for pattern management has been recognized by both scientific and industrial parts and as a result several pattern management approaches have been proposed [10] in the literature. Among them, scientific community efforts try to deal with all the aspects of the pattern management problem including both representation and manipulation issues. Examples of this category are the inductive databases approach [15, 1], the 3–Worlds model [17] and the PANDA project approach [22, 8, 24].

On the other side, industrial approaches mainly focus on pattern representation and storage aiming at easy interchange of patterns between different vendors applications. Examples of this category include both specifications and standards like the Predictive Markup Language Model (PMML) [13], the Common Warehouse Metamodel [2], the SQL/MM Part 6 for Data Mining [4] and the Java Data Mining API (JDM API) [5] as well as extensions of existing commercial DBMS systems like Oracle 10g Data Mining [7], Microsoft SQL Server 2005 Analysis Manager [6] and IBM DB2 Intelligent Miner [3].

In the next sections we will review the approaches proposed so far taking into account the following aspects [10]:

- the **chosen architecture** with respect to the raw data
  - *integrated*, if patterns and data are stored together
  - *independent*, if patterns are stored independently of raw data
- the **adopted pattern model**, i.e. what pattern characteristics are supported by the model
- the capabilities of the proposed **pattern languages**, i.e. what operations are supported by the language
  - *pattern* operations that refer exclusively to patterns
  - *cross over* operations that relate patterns to raw data
- the support for **temporal management** of patterns. Since raw data from which patterns are extracted are usually dynamic, patterns are also dynamic, i.e. they are associated with some temporal notion. Considering the temporal notion of patterns results in a variety of applications, like monitoring, synchronization with respect to the underlying raw data space etc.

3 Scientific approaches

Scientific approaches try to provide an overall solution to the pattern management problem, providing both representation/ storage and retrieval/ manipulation capabilities. In the next sub–sections we will present in more detail three of these approaches, namely inductive databases, 3–Worlds model and PANDA model.

3.1 Inductive databases

The inductive databases framework, introduced by Imielinski and Manilla in 1996 [15], is inspired by the idea that the data mining process should be supported by the database technology. Thus why this approach relies on an integrated architecture where both data and patterns are stored together in the same repository.

Within the inductive databases framework the KDD process is considered as a kind of extended query processing in which users can query both data and patterns to gain insight about the data. To this aim a so–called *inductive query language* is used. An inductive query language is an extension of a database language that allows one to:

- select, manipulate and query *data* as in standard queries
- select, manipulate and query *patterns*
- execute *cross–over* queries over patterns, i.e. queries that associate patterns to raw data

Due to the importance of query languages within the inductive databases framework, a lot of such languages have been proposed which comprise SQL extensions. Among them, the most well known, are DMQL, MINE–RULE and MINE–SQL.

DMQL Han et al [14] proposed DMQL for the generation of patterns from relational data. Several kinds of rules, like generalized relations (a relation obtained by generalizing from a large set of low level data), characteristic rules (an assertion which characterizes a concept), discriminant rules (an assertion which discriminates concepts described by different classes), classification rules (a set of rules associated with a specific class problem) and association rules (which describes associations–relationships between the data) are supported by DMQL.

The general syntax of a query in DMQL is depicted in Figure 1.

An example of generating association rules through DMQL is depicted in Figure 2. In this example, "student" is the raw data relation from which patterns will be mined, "gpa", "birth place", "address" are the relevant attributes to be used, "association rules" is the type of patterns to be mined, "major = cs and birth place = Canada" are the conditions or constraints over the structure

```
⟨DMQL⟩ ::=
    use database ⟨database_name⟩
    {use hierarchy ⟨hierarchy_name⟩ for ⟨attribute⟩}
    ⟨rule_spec⟩
    related to ⟨attr_or_agg_list⟩
    from ⟨relation(s)⟩
    [where ⟨condition⟩]
    [order by ⟨order_list⟩]
    {with [⟨kinds_of⟩] threshold = ⟨threshold_value⟩
        [for ⟨attribute(s)⟩]}
```

Figure 1: The general syntax of DMQL (depicted from [14])

components of the generated rules and "support threshold=0.05, confidence threshold=0.7" are the conditions or constraints over the measures components of the generated rules.

```
find association rules
related to gpa, birth_place, address
from student
where major = "cs" and birth_place = "Canada"
with support threshold = 0.05
with confidence threshold = 0.7
```

Figure 2: Example of a DMQL query (depicted from [14])

Within DMQL the resulting rules can be stored into relational tables. The limitation of the DMQL, however, is that only pattern–extraction operations are supported, whereas no support is provided towards further retrieval and manipulation of the extracted patterns. Furthermore, only rule patterns are considered.

MINE-RULE Meo et al [20] proposed MINE–RULE, an extension of SQL, for discovering association rules from relational data.

A new operator, named MINE RULE has been introduced here. An example of its usage is depicted in Figure 3. The result of the above query is a new table called "SimpleAssociations" with schema "BODY, HEAD, SUPPORT, CONFIDENCE" (defined by the SELECT clause) where each tuple corresponds to a discovered rule. These rules come from the raw data relation "Purchase" (as specified by the FROM clause), where data are grouped by the attribute "transaction" (as specified in the GROUP BY clause). Regarding the mining parameters, namely "support" and "confidence", they are specified in the EXTRACTING RULES WITH clause.

As already demonstrated by the example, within MINE RULE the resulting rules are stored into relational tables. The limitation of the MINE RULE, however, is that only association rules discovery is supported;in particular, several

```
MINE RULE SimpleAssociations AS
SELECT DISTINCT 1..n item AS BODY,
                1..1 item AS HEAD,
                SUPPORT, CONFIDENCE
FROM Purchase
GROUP BY transaction
EXTRACTING RULES WITH SUPPORT: 0.1,
                      CONFIDENCE: 0.2
```

Figure 3: Example of a MINE RULE query (depicted from [20])

variants of simple association rules like association rules from filtered groups, association rules with clustering and association rules with mining conditions. Furthermore, further manipulation of the retrieved rules is no supported.

MINE-SQL Imielinski and Virmani [16] proposed MINE–SQL for the generation and querying of association rules. MSQL provides operators for generation, post–processing and crossover queries over patterns.

A rules–generation example is depicted in Figure 4. Here a set of rules is generated from the relation: "Transactions" with specific "confidence" and "support" criteria and the results are stored into the relation: "MarketAss-Rules".

```
GetRules (Transactions)
into MarketAssRules
where confidence >0.9 and support > 0.3
```

Figure 4: Example of a MSQL queries–rules generation (depicted from [16])

A pattern–post–processing example is depicted in Figure 5; from all generated rules of the previous query (relation: "MarketAssRules") only those with specific structure, i.e. "body=yes", are requested.

```
SelectRules (MarketAssRules)
where body has {(bread=yes)}
```

Figure 5: Example of a MSQL queries–rules post–processing (depicted from [16])

Finally, a cross–over query example is depicted in Figure 6, implemented through the operator VIOLATES ALL which returns all raw data tuples of the relation: "Transactions" that do not participate in any of the rules specified by the GetRules operator.

```
Select *
from Transactions
where VIOLATED ALL (
        GetRules (Transactions)
        where body has {(bread=yes)}
        and confidence > 0.75
        )
```

Figure 6: Example of a MSQL queries–cross–over query (depicted from [16])

Except for the VIOLATES ALL operator, further pattern–data consistency checking operators are provided like VIOLATES ANY, SATISFIES ALL and SATISFIES ANY.

As already noted, within MSQL the resulting rules are stored into relational tables and expect for their generation, post–processing and cross–over queries over rules are also supported. The limitation of the MSQL, however, is that only association rules manipulation is supported.

Overview of inductive databases Inductive databases are based on an integrated architecture where both data and patterns are stored in the same repository and treated in a similar manner; in fact data mining process is considered as an extended querying process. Within inductive databases framework, only specific types of patterns are supported, usually association rules and text mining patterns. Inductive databases treat patterns as permanent objects that can be stored (together with data into relational tables) and further manipulated through several languages, like DMQL, MINE RULE and MSQL. Among them MSQL seems more complete by means that it supports generate, manipulate and cross–over queries over patterns. There is no temporal information regarding patterns like for example their creation time and/or their validity period. The only attempt towards this aim is Mine–SQL that provides some cross over operations like SATISFIES ALL/ANY, VIOLATES ALL/ANY; actually these operators perform some kind of pattern synchronization or validity checking against a raw dataset rather than capturing pattern changes. However, they provide some kind of temporal notion of patterns.

3.2 3–Worlds model

The 3–Worlds model, introduced in 2000 by Johnson et al [17], provides a unified framework for pattern management, that supports combination of different mining, analysis and aggregation KDD tasks.

It relies to three key notions: regions, dimensions and hierarchies. Different pattern types serve to split a given data set into a collection of subsets of tuples,

which are called *regions*. For example, the cube operator merely splits up a data set into regions corresponding to all possible group by expressions. In general, the regions created vary on their spatial shapes and on whether they overlap. A set of related regions is called a *dimension*. As an example, consider the set of itemsets with support exceeding a given threshold or a set of itemsets containing a specific item p. Dimensions might come with interesting structure that relates their regions in the form of a *hierarchy*. In case of frequent itemsets, for example, the lattice of all possible subsets of the set of itemsets is the associated hierarchy, whereas in case of a decision tree, the associated hierarchy is the tree.

The 3–Worlds model is based on a separated architecture consisting of three distinct worlds:

- *Intensional world (I–World)*

  In the I–World, each region is represented in its intensional form, i.e. as a description of its members through constraints. For example, a cluster of products based on their price can be described as: $10 \leq price \leq 20$. Regions produced by most mining/analysis operations are convex and can be modeled as a set of linear inequalities. Non–convex regions can be modeled as a union of multiple convex regions.

- *Extensional world (E–World)*

  In the E–World, each region is represented in its extensional form, i.e. by an explicit enumeration of the tuples belonging to each region. Recalling, the previous mentioned example, the extensional representation contains all source data items with price between 10 and 20.

- *Data world (D–World)*

  The D–World consists of raw data, e.g. in the form of relations, from which regions and dimensions can be created as a result of mining. It can be viewed as consisting of a relational database.

In a few words, we can state that the I–World is the world of patterns, the D–World is the world of raw data from which patterns have been extracted through some mining process, whereas the E–World is the intermediate world that brings the gap between the two other words by determining which data produce which patterns.

Manipulating the worlds The manipulation of structures in each world can be performed through an algebra of choice. For the manipulation of regions and dimensions in the I–World authors propose the *dimension algebra* which is comprised of operators similar to the relational algebra as well as very different ones that add significant value to data mining and analysis. Examples of such operators are:

- *Selection operator*

  The selection operator invokes various spatial predicated such as overlap, containment, disjointness and non–containment. As an example, consider

the query "find all regions of a given decision tree that overlap with a given constant region: $age \leq 25$".

- *Purge operator*

  The purge operator removes inconsistent regions that might result after applying some operator on consistent regions. For example, although regions "$A \leq 2$" and "$A \geq 3$" are consistent, their conjunction "$(A \leq 2) \wedge (A \geq 3)$" is inconsistent.

- *Cartesian product*

  Cartesian product creates new regions which are obtained by taking the pairwise intersection of regions in the two dimension instances.

- *Union*

  In contrast to the traditional approaches, here union does not require union–compatibility between its operands and thus a fully heterogeneous union is permitted.

- *Minus*

  Just like the union operator, minus does not require union–compatibility between its operands. The result of minus contains exactly those regions in D_1 for which no equivalent region exists in D_2.

For the manipulation of extensional dimensions in the E–World, authors state that the relational algebra could be utilized extended with aggregation and with some modifications. Relational algebra (extended with aggregation) is also the natural algebra of choice for the D–World.

Connecting the worlds An important part of this work is the bridge operators between the three worlds which are depicted in Figure 7.

The bridging operators that facilitate moving in and out of the worlds are the following:

- *Populate*

  The populate operator amounts to "populating" the regions in an intensional dimension D with tuples from a given data set E, producing an extensional dimension E. Intuitively, for each region in D, there is a corresponding "extensional" region in E which contains just those tuples in E that satisfy the region description. For example, populating a collection of frequent sets with a transaction database would give an extensional dimension with each region containing the set of supporting transactions. So, the populate operator relates the I–world with the D–world letting us to move to the E–world.

- *Mine*

  Given a parameter p, the mine operation maps a relation E to an intensional dimension D. As an example of mine operator, consider the decision

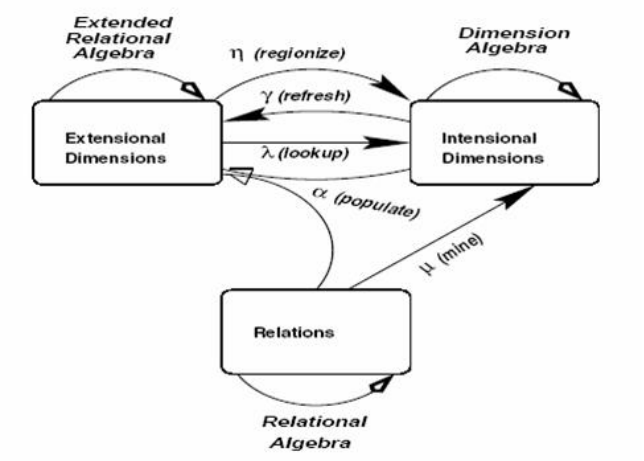

Figure 7: The different worlds in the 3–Worlds model and the bridges (depicted from [17])

tree extraction. The mine operator specifies the type of the model, but its exact definition and computation depends entirely on the specific mining task invoked. In practice, invocation of the mine operator would result in the running of a relevant (fast) mining algorithm. So, the mine operator bridges the D and I worlds.

- *Lookup*

  The lookup operator is evolved whenever one might want to recall or lookup which intentional dimension give raise to an extensional dimension. This operator is utilized if someone wants to find out the intensional descriptions of regions in E, as opposed to enumerations of their member tuples. So, the mine operator bridges the I and E worlds.

- *Refresh*

  Refresh is an incremental version of the populate operator. The refresh and lookup operators helps in maintaining extensional (resp., intensional) dimensions in sync with manipulations being done on the intensional (resp., extensional) dimensions.

Overview of 3–Worlds model The 3–Worlds model provides a generic framework towards unified mining supporting both patterns, underlying raw data and their interconnections management. It relies on an independent architecture, distinguishing thus the pattern management from raw data management, preserving, however, mappings between patterns and the underlying raw data. The 3–Worlds model is not restricted to a specific type of patterns,

it rather considers patterns as constraints and thus is it is more generic than inductive databases. Algebras for the manipulation of each world, as well as their relationships have been proposed. The temporal aspects of patterns are not considered within this model, although synchronization and consistency issues are considered through relevant query operators. The 3–Worlds model is very appealing in theory, in practice however, there is no work towards its continuation or some prototype implementation. This of course, does not inverse the fact that the 3–Worlds model is an important attempt towards unified data mining. Also, to the best of our knowledge, 3–Worlds model comprise the first attempt towards discrete pattern management which also considers linkages to the underlying raw data set from which patterns have been extracted.

3.3 PANDA framework

PANDA provides a unified framework for the representation of heterogeneous patterns, relying on a separated architecture. Its goal is the design of a Pattern Base Management System (PBMS) for patterns in correspondence to the Data Base Management Systems (DBMS) for raw data. The PBMS is a system for handling (storing/processing/retrieving) patterns defined over raw data in order to efficiently support pattern matching and to exploit pattern–related operations generating intensional information. The set of patterns managed by a PBMS is called *pattern–base*.

The PANDA logical model is composed of three basic building blocks:

- *Pattern type*

  A pattern type pt is a quintuple $pt = (n, ss, ds, ms, et)$, where n is the name of the pattern type; ss is the structure schema, ds is the source schema, ms is the measure schema and et is an expression template, written in a given language, which refers to type names appearing in the source and structure schemes. The *name* component declares the name of the pattern type, e.g. "association rules", "decision tree", etc. The *structure schema* component defines the pattern space by describing the structure of the patterns instances of the pattern type. The *source schema* component defines the related source space by describing the dataset from which patterns are constructed. The *measure schema* component describes the measures which quantify the quality of the source data representation achieved by the pattern. The *expression template* component describes the relationship between the source space and the pattern space, thus carrying the semantics of the pattern. An example of defining the pattern type "Association Rule" is depicted below:

$$
\begin{aligned}
n &: \mathsf{AssociationRule} \\
ss &: \mathsf{RECORD}(\mathsf{head} : \mathsf{SET}(\mathsf{STRING}), \mathsf{body} : \mathsf{SET}(\mathsf{STRING})) \\
ds &: \mathsf{BAG}(\mathsf{transaction} : \mathsf{SET}(\mathsf{STRING})) \\
ms &: \mathsf{RECORD}(\mathsf{confidence} : \mathsf{REAL}, \mathsf{support} : \mathsf{REAL}) \\
et &: \mathsf{head} \cup \mathsf{body} \subseteq \mathsf{transaction}
\end{aligned}
$$

- *Pattern*

  A pattern is an instantiation of the pattern type. More formally, a pattern p of type pt is a quintuple $p = (pid, s, d, m, e)$, where pid is the unique pattern identifier, s is the structure component, d is the source component, m is the measure component and e is the expression component. The *structure component* positions the pattern within the pattern space defined by its pattern type, the *source component* identifies the specific dataset the pattern relates to, the *measure component* estimates the quality of the raw data representation achieved by the pattern, whereas finally the *expression component* relates the pattern to the source dataset. For example, in case of an association rule, its structure is comprised of the head and the body components; its measure consists of the support and the confidence components; its source is the dataset from which the rule has been extracted; and finally, the expression component of the rule is an enumeration of the dataset tuples that support it. An example of defining a pattern belonging to the pattern type "Association Rule" is depicted below:

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

- *Class*

  A class is a set of semantically related patterns. As an example, consider the class "SalesRules" containing the set of patterns (including "512") generated from "sales" using e.g. the Apriori algorithm.

To increase the modeling expressiveness of the model authors have also proposed interesting relationships between patterns:

- *Specialization* Specialization allows new entities to be cheaply derived from existing ones, providing thus the model with extensibility and reusability. For example, the pattern type "Cluster" could be specialized to the type "Cluster of Integer".

- *Composition*

  Composition allows the creation of a part–of hierarchy of patterns, where the structure schema of a given pattern it is defined in terms of other patterns. As an example, consider the pattern type "Clustering" which refers to a set of patterns of type "Cluster". In this example, there is a composition relationship between the type "Clustering" and the type "Cluster".

- *Refinement*

  Refinement allows supporting the modeling of patterns obtained by mining other existing patterns. As an example, consider a pattern type "Cluster of Rules" that defines a cluster over association rules instead of raw data. In this example, the "Cluster of Rules" type refines the "Association Rule" pattern type.

Within the PANDA framework a Pattern Manipulation Language (PML) and a Pattern Query Language (PQL) have been proposed. Except for pattern queries, cross–over queries that relate patterns with the raw data from which they have been extracted have also been proposed.

The PBMS architecture of the PANDA approach is depicted in Figure 8.

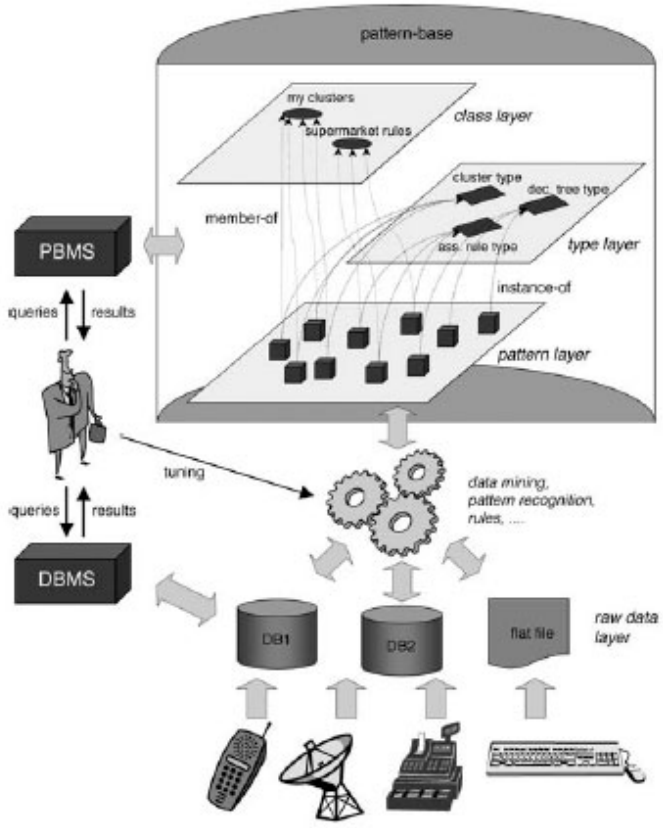

Fig. 2. The PBMS architecture.

Figure 8: The PBMS architecture of PANDA(depicted from [22])

The PANDA model [22] has been extended in [12] in order to address the need for temporal information management associated with patterns. More specifically, the notions of temporal validity, semantic validity and safety of patterns have been added to the definition of a pattern. The *temporal validity* is defined by the user and reflects the intuition of the user regarding the period that the patterns will be valid. The *semantic validity* is more strict by means that it depends on whether the underlying raw data space has been changed. If a pattern is both temporarily and semantically valid then it is considered as *safe*. The previously proposed PML and PQL [22] languages have been extended in [12] in order to be able to cope with temporal features during pattern manipulation and querying. As a proof of concept the PSYCHO prototype [11] has been implemented.

Recognizing the importance of dissimilarity operators for querying, indexing, mining etc. a generic and flexible framework [9] has been proposed by PANDA people capable of assessing dissimilarity between both simple patterns and complex ones (i.e. patterns defined over other patterns instead of raw data). The dissimilarity scores between complex patterns (e.g. clusterings) is conceptually evaluated in a bottom up fashion by aggregating the dissimilarities between the matched component patterns (e.g. clusters). Multiply coupling types and aggregation logics are supported. Furthermore, efficiency issues are considered by avoiding accesses to the raw data from which patterns have been extracted.

Overview of PANDA model The PANDA model provides a both generic and extensible framework for full pattern management, generic by means that it treats different pattern types in a uniform way and extensible by means that new pattern types are easily accommodated into the model. It relies on an independent architecture where patterns are treated as first class citizens, but also considers mappings to the underlying raw data space from which patterns have been extracted. Languages for querying and manipulating patterns as well as for their association to raw data have been proposed. The temporal aspects of patterns are taken into account in the core of the model and through the query languages.

Overview of theoretical approaches Summarizing the theoretical approaches, we can state that they try to fully facilitate the management of data mining results, considering issues like storage, representation, querying etc. Regarding the adopted architecture, the inductive databases rely on an integrated architecture, whereas 3–Worlds and PANDA model rely on a separate architecture distinguishing management of patterns from the management of raw data, although the intermediate mappings are preserved and exploited. Although, the second approach seems more appealing since it is specialized to patterns and their characteristics, the first approach seems more applicable by means that it exploits the infrastructure of already existing DBMS.

Regarding the modeling of patterns, the inductive databases concentrate mainly on association rules, whereas 3–Worlds and PANDA approaches con-

sider a generic model capable of handling different types of patterns. The first approach seems too restricted and un–efficient considering the large diversity of patterns, whereas the second approach seems more efficient since it can be easily extended. Ideally, patterns should be treated in a unified way so as new types can easily be managed with the existing infrastructure. Furthermore, this generality should not ignore the special characteristics of each pattern type, in the contrary, these characteristics should be exploited for efficiency reasons.

Regarding the querying of patterns, all approaches recognize the need for pattern querying and retrieval as well as the need for cross–over queries involving both patterns and raw data. Several issues arise here that need to be further investigated, like efficient querying through index structures, for example, or sophisticated pattern synchronization with respect to raw data.

Finally, the notion of temporariness of patterns has been indirectly recognized by all approaches by means that synchronization and refresh operators have been provided. Only within PANDA, however, temporal aspects of patterns have been considered in an integrated way in both modeling and querying of patterns.

4 Industrial approaches

Scientific approaches try to provide an overall solution to the pattern management problem, providing both representation/ storage and retrieval/ manipulation capabilities. In the next sub–sections we will present in more detail four of these approaches, namely PMML, SQL/MM, CWM and JDM API.

The most popular efforts for modeling patterns is the Predictive Model Markup Language [13], that uses XML to represent data mining models. Another approach, based on SQL, is the SQL/MM standard [4]; here, the supported mining models are represented by structured types that constitute first-class SQL types made accessible through the SQL:1999 base syntax. Another framework for metadata representation is proposed by the Common Warehouse Model [2]: the main purpose of CWM is to enable easy interchange of warehouse and business intelligence metadata between various heterogeneous repositories, and not the effective manipulation of these metadata. The Java Data Mining API [5] addresses the need for procedural support of all the existing and evolving data mining standards. The JDMAPI specification supports the building of data mining models, as well as the creation, storage, access and maintenance of data and metadata that represent data mining results.

From an architectural point of view, all the above solutions can be divided into two categories. In the first one, an additional layer is built on top of a DBMS that manipulates patterns, following the traditional object-relational approach. In the second approach, extension is achieved by allowing the integration of new system components (such as data types, access methods, storage structures, and low-level query processing techniques) into the DBMS kernel.

Overall, the listed approaches seem inadequate to represent and handle different classes of patterns in a flexible, effective, and coherent way: in fact, a

given list of predefined pattern types is considered only, and no general and extensible approach to pattern modeling is proposed.

4.1 Specification and standards

4.1.1 Predictive Model Markup Language (PMML)

PMML [13] is an XML-based language that provides a quick and easy way for companies to define data mining and statistical models using a vendor-independent method share models between PMML compliant applications allows users to develop models within one vendor's application, and use other vendors' applications to visualize, analyze, evaluate or otherwise use these models. PMML has been developed by the Data Mining Group (DMG), an independent, vendor led group for DM standards. Among the DMG members are IBM, Microsoft, Oracle, SPSS and SAP. PMML format is supported by major commercial products like Oracle, MSSQL and DB2.

PMML supports only predefined types. The types supported so far are: Association Rules, Decision Trees, Center/Distribution Based Clustering, (General) Regression, Neural Networks, Naive Bayes, Rulesets, Sequences, Text and Vector Machines.

Through PMML, quality measures associated with patterns can also be represented. Furthermore, is stored the relation between patterns and the subset of the source dataset represented by the pattern. Regarding the temporal notion of patterns,only the model date/time creation are stored.

An example of PMML representation for association rules is depicted in Figure 9. The raw data set used for the generation of the model is: $\{t_1$: Cracker, Coke, Water$\}$, t_2: {Cracker, Water}, t_3: {Cracker, Water}, t_4: {Cracker, Coke, Water}.

The data flow in PMML is depicted in Figure 10. Below we present each block of the flow in more detail:

—-

- *DataDictionary*

  The data dictionary block describes the raw input data hosted in external sources. More specifically, it defines how the model interprets these data according to their type (categorical) and value range.

- *Transformation*

  The transformation block converts the original values to internal values using e.g. normalization of numbers to $[0 \ldots 1]$, discretization of continuous fields (derived fields) etc.

- *MiningSchema*

  The mining schema block defines which fields a user has to provide in order to apply a specific model. It contains model specific data like field usage type (e.g. predicted, ignored, input).

```
<PMML> ...
<DataDictionary numberOfFields="2" >
            <DataField name="transaction" optype="categorical" /> <DataField name="item"
optype="categorical" />
  </DataDictionary>
<AssociationModel functionName="associationRules" numberOfTransactions="4" numberOfItems="3"
minimumSupport="0.6"    minimumConfidence="0.5"  numberOfItemsets="3"    numberOfRules="2">
<MiningSchema>
            <MiningField name="transaction" usageType="group" /><MiningField name="item"
usageType="predicted"/>
</MiningSchema>
<Item id="1" value="Cracker" /><Item id="2" value="Coke" /><Item id="3"value="Water"/>
<Itemset id="1" support="1.0" numberOfItems="1"> <ItemRef itemRef="1" /> </Itemset>
<Itemset id="2" support="1.0" numberOfItems="1">  <ItemRef itemRef="3" /> </Itemset>
<Itemset id="3" support="1.0" numberOfItems="2"> <ItemRef itemRef="1" />
<ItemRef itemRef="3" /></Itemset>
<AssocRule support="1.0" confidence="1.0"  antecedent="1" consequent="2" />
<AssocRule support="1.0" confidence="1.0"  antecedent="2" consequent="1" />
</AssociationModel>
</PMML>
```

Figure 9: Representing association rules through PMML (depicted from [13])

- *Model*

  The model block defines specific model parameters (e.g. tree, neural networks, regression).

- *Output*

  The output block depends on the specific kind of model. It is described by leaf nodes in case of a decision tree or by output neurons in case of a neural network.

- *Result*

  The result block is computed from the output of the model (e.g. in case of a neural network, the numeric output value should be de-normalized to original domain values).

PMML supports model composition through sequences of models and model selection. Two or more models combined into a sequence so that the results of one model are used as input into another model form a *sequence of models*. As an example, consider using regression elements within the nodes of a decision tree. Through *model selection* the result of a model can be used to select which model should be applied next.

PMML also supports *built in and user defined functions* like fine-grained transformations (e.g. sum, product, trim).

An important feature of PMML is *model verification* which gives providers and consumers of PMML models a mechanism to ensure that a model deployed

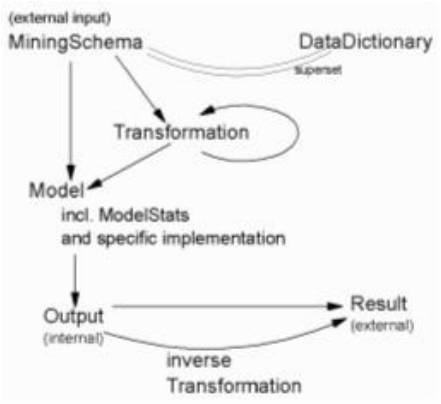

Figure 10: The data flow in PMML (depicted from [13])

to a new environment generates results consistent with environment where the model was developed. Recall, that due to differences in operating systems, data precision and algorithm implementation, the model's performance can be affected. To deal with this problem, a dataset of model inputs and known results that can be used to verify accurate results is generated, regardless of the environment.

From the above mentioned, is clear that PMML emphases on pattern representation issues, rather than management, providing a framework for pattern exchange between PMML compliant applications.

4.1.2 Common Warehouse Metamodel (CWM)

CWM [2] is a specification that enables easy interchange of metadata between data warehousing tools and metadata repositories in distributed heterogeneous environments.

CWM is based on three key industry standards:

- Unified Modeling Language (UML), which is a modeling standard. All classes in CWM are expressed in UML.
- Meta Object Facility (MOF), which a metamodeling and metadata repository standard
- XMI XML Metadata Interchange, which is a metadata interchange standard

CWM consists of a number of sub-metamodels representing common warehouse metadata in the areas of Data Resources, Data Analysis (OLAP, Data Mining,) and Warehouse Management.

The CWM Data Mining sub-metamodel represents three conceptual areas:

- *Model description*

  The model description is a generic representation of a data mining model. It consists of the *MiningModel*, which is a representation of the mining model itself, the *MiningSettings* that drives the construction of the model, the *ApplicationInputSpecification* that specifies the set of input attributes for the model and the *MiningModelResult* that represents the result set produced by the testing or application of a generated model.

- *Settings* The settings area elaborates further on the MiningSettings and their usage relationships to the attributes of the input specification (e.g. ClusteringSettings, AssociationRulesSettings etc).

- *Attributes* Two subclasses of Mining Attribute are defined: NumericAttribute and CategoricalAttribute.

The CWM specification has been designed to analyze large amounts of data and the data mining process is just a small part of it. Only predefined pattern types are supported like Clustering, Association Rules, Supervised, Approximation, Attribute Importance and Classification. CWM provides information concerning quality measures associated with patterns and supports links between a pattern and the subset of the source dataset represented by the pattern.

To conclude with CWM, we can state that it is not dedicated to patterns, rather it is a more generic specification for the interchange of metadata. Furthermore, within CWM representation issues are mainly considered rather than general management issues.

4.1.3 SQL/MM Part 6, Data Mining

SQL/MM [4] has been developed by the International Standards Organization (ISO) and it comprises a specification for supporting data management of common data types (text, spatial info, images, data mining results) relevant in multimedia and other knowledge intensive applications.

It consists of the following parts:

- Part 1: *Framework*
- Part 2: *Full-text*
- Part 3: *Spatial*
- Part 5: *Still Image*
- Part 6: *Data Mining*

SQL/MM defines first-class SQL types that can be accessed through SQL:1999 base syntax. In case of data mining models, every model has a corresponding SQL structured user-defined type. A set of predefined types completes the full definition of each model. These types are:

- *DM-*Settings*

  It stores various parameters of the data mining model (e.g. maximum number of clusters, depth of a decision tree etc.) The * symbol stands for *Class* in case of a classification model, *Rule* in case of a rule model, *Clustering* in case of a clustering model, *Regression* in case of a regression model.

- *DM-*TestResult*

  It holds the results of the testing during the training phase of the data mining models.

- *DM-*Result*

  It is created by running a data mining model against real data.

- *DM-*Task*

  It stores metadata that describe the process and control of testings/ runnings.

SQL/MM supports only the predefined types "Clustering", "Association Rules", "Supervised", "Classification" and "Regression". Furthermore, it stores the link between a pattern and the subset of the source dataset represented by the pattern. For each pattern type, a set of quality measures is provided. No temporal information is taken into account (except for the model time creation). Pattern manipulation and querying are possible through SQL. An example of SQL/MM - DM is depicted in Figure 11.

Concluding with SQL/MM, we can state that although only a part of it is dedicated to patterns, it provides both representation and management capabilities through SQL statements. However, since its goals concern management of general multimedia data, rather than patterns, it does not deepen on efficient pattern management.

4.1.4 Java Data Mining API

Java Data Mining API [5] addresses the need for an independent of the underlying data mining system Java API that will support the creation, storage, access and maintenance of data and metadata. It has been developed by Sun and is supported by a majority of companies like IBM, Oracle, SPSS, Blue Martini Software etc.

JDMAPI provides a standardized access to data mining patterns represented in various formats, including CWM, SQL/MM DM and PMML. It supports four conceptual areas: settings, models, transformations and results.

It consists of three logical components that are implemented either as one executable or in a distributed environment:

- *Application Programming Interface (API)*

  API is the end-user-visible component of a JDMAPI implementation that allows access to services provided by the DME

```
DM_RuleModel type represents models which are the result of the search for assoc.
rules
  <!--definition -->
  CREATE TYPE DM_RuleModel AS
  (
     DM_content CHARACTER LARGE OBJECT(DM_MaxContentLength)
  )
  <!—public members -->
  STATIC METHOD DM_impRuleModel
           (input CHARACTER LARGE OBJECT(DM_MaxContentLength))
     RETURNS DM_RuleModel
  METHOD DM_expRuleModel ()
     RETURNS CHARACTER LARGE OBJECT(DM_MaxContentLength
  METHOD DM_getNORules ()
     RETURNS INTEGER
  METHOD DM_getRuleTask ()
     RETURNS DM_RuleTask
```

Figure 11: SQL/MM - DM example regarding association rules (depicted from [4])

- *Data Mining Engine (DME)*

  DME provides the infrastructure that offers a set of data mining services to its API clients.

- *Metadata Repository (MR)*

  MR sstores the data mining objects. It can be based on the CWM framework, specifically using the CWM Data Mining metamodel, or implemented using a vendor representation. MR might exist in a file-based environment or in a relational database.

JDMAPI supports only the predefined types: "Clustering", "Association Rules", "Classification", "Approximation" and "Attribute Importance". It also supports associating patterns with quality measures. Both pattern representation and management issues are considered within JDMAPI.

An example of using JDMAPI for building a clustering model is depicted in Figure 12.

Overview of Specifications and Standards Summarizing the specifications and standards, we can state that several efforts have been carried out in this direction, focusing mainly on pattern representation issues rather than full pattern management issues. This is due to the fact that specifications and standards emphasize on the efficient and effective interchange of patterns between different vendors applications. Actually, they act as an agreement between data mining producers and consumers. Further pattern management issues like

```
/* create a PhysicaDataSet object specifying the location of a table via a URL and
                    save the object into the default schema */
(1)PhysicalDataSetFactory pdsFactory = (PhysicalDataSetFactory)dme-
Conn.getFactory("javax.
datamining.data.PhysicalDataSet");
(2)PhysicalDataSet buildData = pdsFactory.create( url, true );
(3) dmeConn.saveObject( "myPhysicalData", buildData );

/* create a LogicalData instance based on the physical data */
(4) LogicalDataFactory ldFactory = (LogicalDataFactory)
dmeConn.getFactory("javax.datamining.data.
LogicalData");
(5) LogicalData ld = ldFactory.create( buildData );

/* create a ClusteringSettings instance providing a name, logical data, the
maximum number of clusters and the minimum cluster size desired */
(6)ClusteringSettingsFactory csFactory = (ClusteringSettingsFactory)dme-
Conn.getFactory("javax.
                                    datamining
.clustering.ClusteringSettings");
(7) ClusteringSettings clusteringSettings = csFactory.create();
(8) clusteringSettings.setLogicalData( ld );
(9) clusteringSettings.setMaxNumberOfClusters( 20 );
(10) clusteringSettings.setMinClusterCaseCount( 5 );
(17) if( ExecutionState.success.isEqual( status.getState() ) )
/* task completed successfully... */
```

Figure 12: JDMAPI example - building a clustering model (depicted from [5])

querying, indexing etc. are not considered at all or are payed small attention. The most popular effort in this direction is PMML which is dedicated to data mining patterns and statistical models and is supported by major vendors in the field of data mining.

4.2 DBMS extensions

4.2.1 Oracle Data Mining (ODM)

ODM [7] provides data mining functionality embedded in Oracle Database 10g to mine data, extract hidden patterns and insights and build advanced business intelligence applications. All the data mining functionality is embedded in the Oracle DB (Figure 13).

ODM supports a variety of data mining tasks including "Attribute importance", "Classification and Regression", "Clustering", "Associations", "Feature extraction", "Text mining" and "Sequence matching and alignment". ODM functionality is available to end users through ODM Java API, ODM DBMS PL/SQL API and the Data Mining Client (GUI).

To build a model, the user defines the input data and the mining function settings. Models are built and stored in the DMS. ODM supports the persistence of mining results as independent, named entities in the DMS. A mining results object contains the operation start time and end time, the name of the model

Figure 13: Oracle Data Mining embedded into Oracle DB (depicted from [7])

used, input data location, and output data location (if any) for the data mining operation. The user can query model settings and details as well as test the model.

ODM works towards supporting PMML import /export. Currently, only Naive Bayes and Association Rules import /export into PMML format are supported.

4.2.2 Microsoft SQL Server 2005

MSSQL [6] provides an integrated environment for creating and working with data mining models. Currently, the supported models are "Decision Trees", "Clustering", "Association Rules", "Itemsets", "Naive Bayes", "Sequence Clustering", "Time Series", and "Neural Networks". It also supports PMML 2.1 format.

The major contributions of MSSQL regarding data mining are the following:

- *OLE DB for Data Mining interface*, which provides the functionality to mine knowledge from relational data sources or multi-relational repositories.
- *the Data Mining Extensions (DMX) to SQL language*, which provides a familiar query interface for data mining purposes. More specifically, model creation and training become familiar "CREATE" and "INSERT INTO" statements, whereas prediction and content discovery become familiar "SELECT" statements. Also, the storage and manipulation of patterns is performed in an SQL like style and the results are stored into relational tables.

The data mining process in MSSQL is depicted in Figure 14

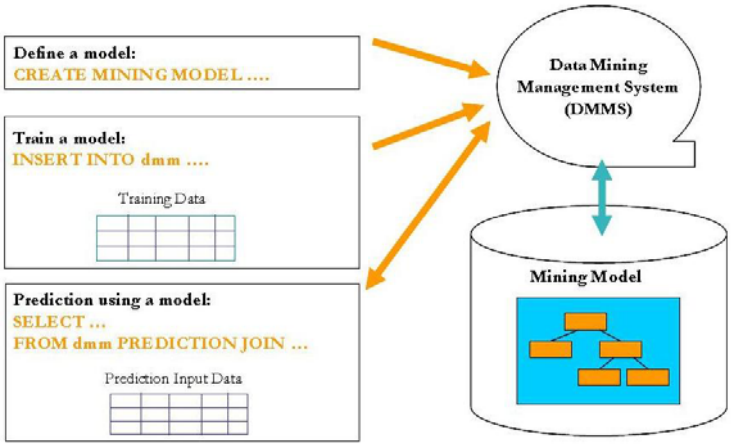

Figure 14: Data Mining process in MSSQL (depicted from [6])

4.2.3 IBM DB2 Intelligent Miner

IBMs DB2 Intelligent Miner [3] is a suite of tools for supporting Knowledge Management. The supported data mining algorithms are "Association Rules", "Clustering" and "Classification". Intelligent Miner produces PMML data mining models which are stored as Binary Large Objects (BLOBs).

The end user might interact with the system through an SQL API, thus data mining results can be integrated within business applications using external programming languages.

Furthermore, Intelligent Miner supports patterns-data synchronization through a scoring mechanism that can be started up by some triggers monitoring raw data changes.

Overview of DBMS extensions Summarizing the DBMS extensions to pattern management, we can state that industrial approaches have recognized the emerging need for pattern management and try to incorporate it into existing DBMSs. So far, issues like pattern storage, simple querying and naive synchronization with respect to raw data are considered. These issues are usually addressed by exploiting the infrastructure of DBMSs, e.g. in case of patterns' retrieval SQL like languages are utilized. Actually, it seems that these approaches treat patterns just as another type of (complex) data and rely on some integrated architecture by storing patterns and raw data in the same repository. We should also note that all approaches are PMML compatible, so as to facilitate the interchange of data mining models between different vendor applications.

5 Conclusions - Open issues - Research directions

In this chapter we have overview the work performed so far in the area of pattern management by both academic and industrial parts. Academic approaches try to provide an overall and generic solution to the pattern management problem, starting sometimes from scratch, whereas industrial approaches try to incorporate pattern management into existing DBMSs. Pattern management includes several issues like representation/ storage, querying/ retrieval, indexing, visualization etc.

So far all approaches deal with the pattern representation issue either by adopting discrete representation for each pattern type, like PMML, or by adopting a common model for all pattern types, like PANDA or 3-Worlds models. Extensibility for novel mining models is a desired feature for a pattern base management system. Ideally, a general model should be adopted that would also preserve and utilize pattern type specific characteristics.

Regarding the pattern storage issue, there are approaches that adopt the integrated architecture where patterns and raw data are stored in the same repository, like inductive databases and industrial approaches, and others that adopt a discrete architecture like PANDA and 3-Worlds model. For the storage of patterns existing data storage approaches are utilized like relational databases, object-relational databases, XML and data files. Although this is a straightforward solution of vendors of commercial DMBSs, it could be revisited since the data mining results are far from the traditional data [18].

A lot of work remains to be done in the area of pattern querying, especially in that of efficient querying through sophisticated index structures. So far, only naive querying capabilities are supported, like retrieval of patterns or simple cross over queries involving both patterns and raw data. Interesting extensions are sophisticated cross-over queries as well as similarity queries and queries involving patterns of different pattern types [21].

References

[1] The CINQ (Consortium on Discovering Knowledge with Inductive Queries) project. http://www.cinq-project.org, (valid as of July 2006).

[2] Common Warehouse Metamodel (CWM). http://www.omg.org/cwm, (valid as of July 2006).

[3] IBM DB2 Intelligent Miner. http://www-306.ibm.com/software/data/iminer, (valid as of July 2006).

[4] ISO SQL/MM Part 6. http://www.sql-99.org/SC32/WG4/Progression-Documents/FCD/fcd-datamining-2001-05.pdf, (valid as of July 2006).

[5] Java Data Mining API. http://www.jcp.org/jsr/detail/73.prt, (valid as of July 2006).

[6] Microsoft SQL Server 2005. http://www.microsoft.com/sql/2005/, (valid as of July 2006).

[7] Oracle10g Data Mining Concepts. http://download-uk.oracle.com/docs/cd/B19306-01/datamine.102/b14339/toc.htm, (valid as of July 2006).

[8] The PANDA Project. http://dke.cti.gr/PANDA/, (valid as of July 2006).

[9] I. Bartolini, P. Ciaccia, I. Ntoutsi, M. Patella, and Y. Theodoridis. A unified and flexible framework for comparing simple and complex patterns. In *PKDD*, pages 496–499. Springer, 2004.

[10] B. Catania and A. Maddalena. *Pattern Management: Practice and Challenges*, pages 280–317. Processing and Managing Complex Data for Decision Support. Idea Group Publishing, 2006.

[11] B. Catania, A. Maddalena, and M. Mazza. Psycho: A prototype system for pattern management. In *VLDB*, pages 1346–1349, 2005.

[12] B. Catania, A. Maffalena, A. Mazza, E. Bertino, and S. Rizzi. A framework for data mining pattern management. In *ECML/PKDD*, 2004.

[13] DMG. Predictive Model Markup Language (PMML), (valid as of July 2006).

[14] J. Han, Y. Fu, K. Koperski, W. Wang, and O. Zaiane. DMQL: A data mining query language for relational databases. In *SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, 1996.

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

[16] T. Imielinski and A. Virmani. MSQL: A Query Language for Database Mining. *Data Mining and Knowledge Discovery*, 3:373–408, 1999.

[17] T. Johnson, L. Lashmanan, and T. Ravmond. The 3W Model and Algebra for Unified Data Mining. In *VLDB*, 2000.

[18] E. Kotsifakos, I. Ntoutsi, and Y. Theodoridis. Database support for data mining patterns. In *Panhellenic Conference on Informatics*, 2005.

[19] P. Lyman and H. R. Varian. How Much Information?, 2000 (valid as of December 2006).

[20] R. Meo, G. Psaila, and S. Ceri. A New SQL–like Operator for Mining Association Rules. In *VLDB*, pages 122–133, 1996.

[21] I. Ntoutsi and Y. Theodoridis. Measuring and evaluating dissimilarity in data and pattern spaces. In *Proceedings of VLDB'05 Phd Workshop*, 2005.

[22] S. Rizzi, E. Bertino, B. Catania, M. Golfarelli, M. Halkidi, M. Terrovitis, P. Vassiliadis, M. Vazirgiannis, and E. Vrachnos. Towards a Logical Model for Patterns. In *ER*, pages 77–90, 2003.

[23] M. Spiliopoulou, Y. Theodoridis, and I. Ntoutsi. Mining the Volatile Web - Tutorial. In *ECML/PKDD*, 2005.

[24] Y. Theodoridis, M. Vazirgiannis, P. Vassiliadis, B. Catania, and S. Rizzi. A Manifesto for Pattern Bases. PANDA TR-2003-03. Technical report, Research Academic Computer Technology Institute, 2003.