Pattern Management:
Practice and Challenges

Barbara Catania and Anna Maddalena

Barbara Catania
Department of Informatics and System Science,
University of Genova (Italy)
Telephone: +390103536728
Fax: +3900103536699
E-mail: catania@disi.unige.it

Anna Maddalena
Department of Informatics and System Science,
University of Genova (Italy)
Telephone: +390103536617
Fax: +3900103536699
E-mail: maddalena@disi.unige.it
Pattern Management: Practice and Challenges

ABSTRACT

Knowledge intensive applications rely on the usage of knowledge artifacts, called patterns, to represent in a compact and semantically rich way huge quantity of heterogeneous raw data. Due to the specific characteristics of patterns, specific systems are required for pattern management, in order to model, store, retrieve, and manipulate patterns in an efficient and effective way. Several theoretical and industrial approaches (relying on standard proposals, metadata management, and business intelligence solutions) have already been proposed for pattern management. However, no critical comparison of the existing approaches has been proposed so far. The aim of this chapter is to provide such a comparison. In particular, specific issues concerning pattern management systems, pattern models, and pattern languages are discussed. Several parameters are also identified that will be used in evaluating the effectiveness of theoretical and industrial proposals. The chapter is concluded with a discussion concerning additional issues in the context of pattern management.
INTRODUCTION

The huge quantity of heterogeneous raw data that we collect from modern, data-intensive applicational environments do not constitute knowledge by itself. A knowledge extraction process and data management techniques are often required to extract from data concise and relevant information that can be interpreted, evaluated and manipulated by human users in order to drive and specialize business decision processing. Of course, since raw data may be very heterogeneous, several kinds of knowledge artifacts exist that can represent hidden knowledge. Clusters, association rules, frequent itemsets, symptom-diagnosis correlations are common examples of such knowledge artifacts, generated by data mining applications. Equations or keyword frequencies are other examples of patterns, relevant for example in a multimedia context. All those knowledge artifacts are often called patterns. In a more concise and general way, patterns can be defined as compact and rich in semantics representation of raw data. The semantic richness of a pattern is due to the fact that it reveals new knowledge hidden in the huge quantity of data it represents. Patterns are also compact since they represent interesting correlations among data providing, in many cases, a synthetic, high level description of some data characteristics. Patterns are therefore the knowledge units at the basis of any knowledge intensive application.

Due to their specific characteristics, ad hoc systems are required for pattern management, in order to model, store, retrieve, analyze, and manipulate patterns in an efficient and effective way.

Many academic groups and industrial consortiums have devoted significant efforts towards solving this problem. Moreover, since patterns can be seen as a special type of metadata, pattern management has also some aspects in common with metadata management. In general, scientific community efforts mainly deal with the definition of a pattern management framework providing a full support for heterogeneous pattern generation and management, thus providing back-end technologies for pattern management applications. Examples of these approaches are the 3W Model (Johnson et al., 2000), the inductive databases approach – investigated in particular in the CINQ project (CINQ, 2001), and the PANDA framework (PANDA, 2001; Catania et al., 2004). In the context of inductive databases, several languages have also been proposed supporting the mining process over relational (or object-relational) data by extending the expressive power of existing data query language with primitives supporting the mining process. Examples of such approaches are MSQL proposal (Imielinski & Virmani, 1999), Mine-Rule (Meo et al., 1998), DMQL (Han et
al., 1996), and ODMQL (Elfeky et al., 2001). On the other hand, industrial proposals mainly
deal with standard representation purposes for patterns resulting from data mining and data
warehousing processes, in order to support their exchange between different architectures.
Thus, they mainly provide the right front-end for pattern management applications. Examples
of such approaches are the Predictive Model Markup Language (PMML, 2003), the Common
Warehouse Model (CWM, 2001), and the Java Data Mining API (JDM, 2003).

In general, existing proposals can be classified according to the following aspects: (i) the
chosen architecture to manage patterns together with data; (ii) the pattern characteristics
supported by the data model; (iii) the type of operations and queries supported by the
proposed languages. As far as we know, even if several proposals exist, no critical
comparison of the existing approaches has been proposed so far. We believe that such a
comparison would be very useful in order to determine whether the existing approaches are
sufficient to cover all pattern requirements and to guide application developers in the choice
of the more adequate solution in developing knowledge discovery applications.

The aim of this chapter is to provide such a comparison. We first present a definition of
patterns and pattern management. Then, specific issues concerning pattern management
systems, pattern models, and pattern languages are discussed, pointing out possible alternative
solutions, in the context of a given scenario. Several parameters relevant to the pattern
management context will also be identified and then used to evaluate the effectiveness of
various theoretical and industrial proposals. Moreover, relationships between pattern and
general metadata management will also be identified, discussing some existing approaches for
metadata representation. Finally, we will briefly discuss solutions for pattern management
supported by some popular commercial DBMSs. The chapter is concluded with a discussion
of additional issues and possible future trends in the context of pattern management.

**PATTERN MANAGEMENT: BACKGROUND**

In many different modern contexts, a huge quantity of raw data is collected. An usual
approach to analyze such data is to generate some compact knowledge artifacts (i.e., clusters,
association rules, frequent itemsets, etc.) through data processing methods, that reduce the
number and size of data, to make them manageable from humans while preserving as much as
possible their hidden information or discovering new interesting correlations. Those
knowledge artifacts, that constitute our knowledge unit, are called *patterns.*
**Definition 1**: A pattern is a compact and rich in semantics representation of raw data.

Patterns, thus, can be regarded as knowledge units that effectively describe entire subsets of data (in this sense, they are compact). The quality of the representation achieved by a pattern can be quantified by using some statistical measures. Depending on their measures, patterns can describe relevant data properties (in this sense, they are rich in semantics).

Pattern management is an important issue in many different contexts and domains. The most important context in which pattern management is required is business intelligence and data mining. Business intelligence concerns a broad category of applications and technologies for gathering, storing, analyzing, and providing access to data to help enterprises in business decisions. Data mining is one of the fundamental activities involved in business intelligence applications besides querying and reporting, OLAP processing, statistical analysis, and forecasting. The knowledge units resulting from the data mining tasks can be quite different. As an example, in the context of the market-basket analysis association rules, involving sold items derived from a set of recorded transactions, are often generated. In addition, in order to perform a market segmentation, the user may also be interested in identifying clusters of customers, based on their buying preferences, or clusters of products, based on customer buying habits. In financial brokerage, users cope with stocks trends derived from trading records. In epidemiology, users are interested in symptom-diagnosis correlations mined from clinical observations.

Pattern management is a key issue also in many other domains not involving directly a data mining process. For instance, in information retrieval, users are interested in extracting keyword frequencies and frequent sets of words appearing in the analyzed documents in order to specialize searching strategies and to perform similarity analysis. Content-based music retrieval is another domain in which patterns have to be managed in order to represent and query rhythm, melody, harmony (Conklin, 2002). In image processing, recurrent figures in shapes can be interpreted as specific types of patterns (Nakajima, C. et al., 2000). In machine learning (Mitchell, T.M., 1997), predictions and forecasting activities are based on classifiers, which can be interpreted as specific types of patterns.

Recently, pattern management is becoming much more important not only in centralized architectures but also in distributed ones. Indeed, the diffusion of the Web and the improvement of networking technologies speed up the requirement for distributed knowledge discovery and management systems. For instance, in the Web context, sequences of clicks
collected by Web Servers are important patterns for clickstream analysis. Moreover, knowledge representation and management, in terms of patterns, is a fundamental issue in the context of the Semantic Web and in agent-based intelligent systems where metadata have to be shared among different parties.

Depending on the specific domain, different processes can be used for pattern extraction, e.g., knowledge discovery processes for data mining patterns, feature extraction processes in multimedia applications, or manual processes when patterns are not extracted but directly provided by the user or the application (for example a classifier not automatically generated from a training set).

Patterns share some characteristics that make traditional DBMSs unable to represent and manage them. As discussed above, patterns can be generated from different application contexts resulting in very heterogeneous structures. Moreover, often, heterogeneous patterns have to be managed together. For instance, in a Web context, in order to well understand e-commerce buying habits of the users of a certain Web site, different patterns can be combined, for example: (i) navigational patterns (identified by clickstream analysis) describing their surfing and browsing behaviour; (ii) demographic and geographical clusters, obtained with market segmentation analysis based on personal data and geographical features; (iii) frequencies of the searching keywords specified by the user when using a searching engine (typical information treated in information retrieval); and (iv) metadata used by an intelligent agent-based crawling system (typical of the artificial intelligence domain) the user may adopt. Additionally, patterns can be generated from raw data by using some data mining tools (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). Since source data change with high frequency, another important issue consists in determining whether existing patterns, after a certain time, still represent the data source from which they have been generated, possibly being able to change pattern information when the quality of the representation changes. 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 the design of ad hoc Pattern Management Systems (PBMSs), i.e., according to (Rizzi et al., 2003), systems for handling (storing/processing/retrieving) patterns defined over raw data.
**Definition 2**: A *Pattern Base Management System* (PBMS) is a system for handling (storing/processing/retrieving) patterns defined over raw data in order to efficiently support pattern matching and to exploit pattern-related operations generating intensional information. The set of patterns managed by a PBMS is called *pattern base*.

The Pattern Base Management System (PBMS) is therefore not a simple repository for the extracted knowledge (patterns); rather it is an engine supporting pattern storage (according to a chosen logical model) and processing (involving also complex activities requiring computational efforts).

The design of a PBMS relies on solutions developed in several disciplines, such as: data mining and knowledge discovery, for a-posteriori pattern extraction; database management systems, for pattern storage and retrieval; data warehousing, for providing raw datasets; artificial intelligence and machine learning, for pattern extraction and reasoning; metadata management. Pattern management can therefore be seen as a relatively new disciplines lying in the intersection of several well-known application contexts.

**KEY FEATURES IN PBMS EVALUATION**

In the following we first present a typical data mining scenario and then, based on it, we present useful parameters in comparing existing pattern management solutions. In particular, we consider three different aspects: (i) architecture for a pattern management system, (ii) pattern models, and (iii) pattern languages.

**The Scenario**

The market-basket analysis is a typical data mining application concerning, in our example, the task of finding and handling *association rules* and *clusters* concerning customer's transactions. Given a domain $D$ of values and a set of transactions, each corresponding to a subset of $D$, an association rule takes the form $B \Rightarrow H$, where $B \subseteq D$, $H \in D$, $H \cap B = \emptyset$. $H$ is often called the head of the rule, while $B$ is its body. The informal meaning of the rule is that, given a transaction $T$, it often happens that when $T$ contains $B$ then it also contains $H$. This qualitative information can be quantified by using two measures: the *support* (i.e. the ratio between the number of transactions satisfying the body of the rule and the total number of
transactions) and the confidence (i.e. the ratio between the number of transactions satisfying both rule body and head and the number of transactions satisfying just the body).

Suppose a commercial vendor traces shop transactions concerning milk, coffee, bread, butter, and rice, and applies data mining techniques to determine how he can further increase his sales. The vendor deals with different kinds of patterns: association rules, representing correlations between sold items; clusters of association rules, grouping rules with respect to their similarity; clusters of products, grouping products with respect to their type and price. Now, we suppose the vendor wants to execute the following operations:

**Step 1: modeling heterogeneous patterns.** Since the vendor deals with (at least) three different types of patterns he would like to generate and manage those patterns all together, in the same system, in order to be able to manipulate all these knowledge in an integrated way.

**Step 2: periodic pattern generation.** At the end of every month, the vendor mines from his transaction data association rules over sold products by filtering interesting results with respect to certain thresholds. He assumes to consider reliable the rules extracted from the instant in which they have been generated till the last day of the month. The vendor then groups the rules into clusters.

**Step 3: pattern querying.** The vendor may be interested in analyzing patterns stored in the system, i.e., to retrieve patterns satisfying certain conditions, to combine them in order to construct new patterns, to establish whether a pattern is similar to another one, and to correlate patterns and raw data. For instance, the vendor may be interested in retrieving all association rules mined during March 2005 involving ‘Bread’ or similar items, or in identifying all association rules extracted from a certain set of transactions, with a reasonable level of detail (i.e., with quality measures higher than specified thresholds). In order to solve the last query, both the data management and the pattern management system have to be used.

**Step 4: promotion of a new product.** From April 2005, the vendor will start to sell a certain product P. To promote P in advance, he may promote some other products he already sells, for which there exists a correlation with P, in order to stimulate the demand for P. In this way, it is possible that customers will start to buy P without the need for a dedicated advertising campaign. In order to know, for example, whether ‘bread’ may stimulate the sale of P, he may insert in the system an association rule such as ‘bread → P’ (not automatically generated) and verify whether it holds or not with respect to the recorded transactions.

**Step 5: pattern update, synchronization, and deletion.** Patterns may have to be updated. For instance, the user may know that the quality of the representation that a pattern achieves with respect to its data source has been changed because source data have been changed, thus
the pattern quality measures (evaluated at insertion time) have to be updated, since the pattern may no longer be semantically valid, i.e. it may not correctly represent the updated source data. As an example, when on April 1 2005, the vendor starts to sell a new product $P$, new raw data concerning sales are collected, new patterns are generated and, at the same time, patterns previously extracted may not correctly represent source data. Thus, a synchronization is required between data and patterns to reflect to patterns changes occurred in raw data. In this case, the measures may change. Finally, there is the need for pattern deletion operations. For example, the vendor may be interested in deleting all patterns that are no more semantically valid or in removing from the system all rules having ‘rice’ as value in its head or its body.

Architecture for a Pattern Base Management System

The architecture of a PBMS can be integrated or separated. In an integrated architecture, raw data and patterns are stored together, by using the same data model, and managed in the same way. On the other side, in a separated architecture, raw data are stored and managed in a traditional way by a DBMS whereas patterns are stored and managed by a dedicated PBMS. Since in the integrated architecture a unique data model is used for both data and patterns, design of the pattern base is simplified. For example, an association rule can be represented in the relational model by using a set of relational tuples, each containing the head of the rule and one element in the body. However, traditional data models may not adequately represent all patterns characteristics, thus making manipulation operations more complex. Further, by storing patterns with data, we rely on traditional DBMS capabilities for what concerns query expressive power and query optimization. In particular, under an integrated architecture, the mining process is usually seen as a particular type of query. However, patterns may require sophisticated processing that, in traditional systems can only be implemented through user-defined procedures.

Separated architectures manage data and patterns by using two distinct systems. Thus, two models and languages have to be used to deal with pattern-based applications. The usage of a specific pattern data model guarantees a higher and more tailored expressive power in pattern representation. Moreover, operations over data are activated by the PBMS only by demand, through the so-called cross-over queries. The PBMS can therefore support specific techniques for pattern management and retrieval. Mining operations are not part of the query language;
rather, they are specific manipulation operators. Finally, specific query languages can be
designed providing advanced capabilities, based on the provided pattern representation.

Pattern Models

We can define a pattern model as a formalism by which patterns are described and
manipulated inside the PBMS. In defining a pattern model, we believe that the following
aspects should be taken into account.

**User-defined pattern types support.** The ability to model heterogeneous patterns is very
important to make the PBMS flexible and usable in different contexts. Most of the systems
allow the user to manipulate different types of patterns (see Step 1 of the scenario) that
usually correspond to different data mining results, such as association rules, clusters, etc.
However, in many cases, they cannot be used “together” in a unified framework. Moreover,
often, it is not possible for the user to define new pattern types, which are therefore
predefined.

**Relation between raw data and patterns.** Often patterns are generated from raw data through
the application of some mining technique; it may be useful to store the relation between
patterns and raw data, in order to make the pattern richer in semantics and provide additional
significant information for pattern retrieval. Most of the systems recognize the importance of
this aspect and provide a mechanism to trace the source data set from which a pattern has
been generated. In the proposed scenario, this corresponds to maintain information
concerning the dataset from which association rules have been extracted. Such information
can then be used to solve some of the queries pointed out in Step 3 of the scenario. Besides
the source dataset, it may be useful to exactly know the subset of the source dataset
represented by the pattern. For examples, to generate rule ‘bread $\Rightarrow$ milk’ only transactions
containing ‘bread’ and ‘milk’ are considered from the overall set of transactions in the source
dataset. This subset can be represented in a precise way, by listing its components, or
approximate way, by providing a formula satisfied by the elements of the source dataset from
which the pattern has been probably generated. Most of the systems do not support the
representation of this relationship or support it only in an approximate way.

**Quality Measures.** It is important to be able to quantify how well a pattern represents a raw
data set, by associating each pattern with some quantitative measures. For example, in the
identified scenario, each association rule mined from data is associated with confidence and
support values. Most of the systems allow the user to express this quality information, which is in general computed during pattern generation and never modified.

**Temporal features.** Since source data change with high frequency, it is important to determine whether existing patterns, after a certain time, still represent the data source from which they have been generated. This happens when, given a pattern $p$ extracted at time $t$, the same pattern $p$ can be extracted at time $t' > t$ from the same raw dataset, with the same or better measure values. In this case, we say the pattern is semantically valid at time $t'$. When this happens and measure change, the system should be able to change pattern measures. In practice, it may be useful to assign to each pattern a validity period, representing the interval of time in which it can be considered reliable with respect to its data source.

**Hierarchies over types.** Another important feature a pattern management system should provide is the capability to define some kind of hierarchy over the existing pattern types, in order to introduce relationships such as specialization or composition that increase expressivity, reusability and modularity. For instance, in the proposed scenario, the vendor deals with association rules and with more complex patterns that are clusters of association rules (see Step 1). Thus, a composition relationship is exploited.

### Pattern Languages

As in any DBMS, in a PBMS at least two different types of languages must be provided: the **Pattern Manipulation Language** (PML), providing the basic operations by which patterns can be manipulated (e.g. extracted, synchronized, deleted), and the **Pattern Query Language** (PQL), supporting pattern retrieval. PQL queries take as input patterns and data sets and returns patterns. On the other hand, PML operations take as input a pattern dataset and return a new pattern set, which replaces in the pattern base the input one. Aspects concerning both manipulation and query languages for patterns will be evaluated by means of several parameters introduced in the following.

#### Pattern Manipulation Language Parameters.

**Automatic extraction.** This is the capability of a system to generate patterns starting from raw data using a mining function. It corresponds to the data mining step of a knowledge data discovery process and generates a-posteriori patterns. In the proposed scenario, association rules generated in Step 2 of the scenario represent a-posteriori patterns.
Direct insertion of patterns. There are patterns that the user knows a-priori and wish to verify over a certain data source. They are not extracted from raw data, but inserted directly from scratch in the system. Ad hoc primitives are therefore needed to perform this operation. In the proposed scenario, patterns described in Step 4 are examples of a-priori patterns.

Modifications and deletions. Patterns can be modified or deleted. For example, users may be interested in updating information associated with patterns (such as their validity in time or the quality of raw data representation they achieve, represented in terms of measures) or in removing from the system patterns satisfying (or do not satisfying) certain characteristics. For instance, in the proposed scenario (Step 5), the user is interested in removing an association rule when it does not correctly represent anymore the source data set. Not all the systems guarantee both deletion and update operations over patterns; in many cases, only pattern generation and querying are provided.

Synchronization over source data. Since modifications in raw data are very frequent, it may happen that a pattern, extracted at a certain instant of time from a certain data source, does not correctly represent the data source after several modifications occur (Step 5). Thus, the need for a synchronization operation arises in order to align patterns with data source they represent. This operation is a particular type of update operations for patterns. For instance, in the proposed scenario (Step 5), the user is interested in updating the measure values associated with a certain rule (such as ‘bread→P’) when the source data change.

Synchronization can also be executed against a different dataset in order to check whether a pattern extracted from a certain data source holds also for another data source. In this case, we call it ‘recomputation’. For example, suppose the vendor receives a data set DS concerning sales in the month of January 2005 in another supermarket. He may be interested in checking whether the association rules mined from his data set represent reasonable patterns for the new data set DS. Unfortunately, synchronization between raw data and patterns (Step 5) is rarely supported.

Mining function. Patterns are obtained from raw data by applying some kind of mining function, e.g. the APriori (Agrawal & Srikant, 1994) algorithm can be used to generate association rules (Step 2). The presence of a library of mining functions and the possibility to define new functions if required makes pattern manipulation much more flexible.

Pattern Query Language Parameters.

Queries against Patterns. The PBMS has to provide a query language to retrieve patterns according to some specified conditions. For example, all association rules having ‘bread’ in
their body may need to be retrieved (Step 3). In general, pattern collections have to be supported by the system in order to be used as input for queries. Similarly to the relational context where a relation contains tuples with the same schema, patterns in a collection must have the same type. Moreover, it is highly desirable the language to be closed, i.e. each query over patterns must return a set of patterns of the same type over which other queries can be executed.

Pattern combination. Operations for combining patterns together should be provided as an advanced form of reasoning. Combination can be seen as a sort of ‘join’ between patterns. For example, transitivity between association rules can be seen as a kind of pattern join.

Similarity. An important characteristic of a pattern language is the ability to check pattern similarity, based on pattern structure and measures. Only few general approaches for pattern similarity have been provided that can be homogeneously applied to different types of patterns. Moreover, only few existing pqls support such kind of operation.

Queries involving source data. According to the chosen architecture and the logical model, a system managing patterns has to provide operations not only for querying patterns but also data. Such queries are usually called cross-over queries. When the system adopts a separated architecture, cross-over operations require the combination of two different query processors in order to be executed. In our scenario, the second query of Step 3 is a cross-over query.

THEORETICAL PROPOSALS

As we have already stressed, the need for a unified framework supporting pattern management is widespread and it covers many different contexts and domains. Thus, many efforts have been put in the formalization of the overall principles under which a PBMS can be developed, providing the background for the development of back-end technologies to be used by pattern-based applications. In the following we briefly present and compare the following proposals by considering all the parameters previously introduced:

- Inductive Databases Approach (Imielinsky & Mannila, 1996; De Raedt, 2002; CINQ, 2001): an inductive framework where both data and patterns are stored at the same layer and treated in the same manner.
- 3-Worlds Model (Johnson et al., 2000): a unified framework for pattern management based on the definition of three distinct worlds: an intensional world (containing intensional descriptions of patterns), an extensional world (containing an explicit representation of patterns), and a world representing raw data.
• Panda Project (PANDA, 2001): a unified framework for the representation of heterogeneous patterns, relying on a separated architecture.

**Inductive Databases Approach**

Inductive databases (Imielinsky & Mannila, 1996; De Raedt, 2002) rely on an integrated architecture. Thus, patterns are represented according to the underlying model for raw data. More precisely, the repository is assumed to contain both datasets and pattern sets. Within the framework of inductive databases, knowledge discovery is considered as an extended querying process (Meo, R. et al., 2003; De Raedt, L., et al., 2002). Thus, a language for an inductive database is an extension of a database language that allows one to: (i) select, manipulate and query data as in standard queries; (ii) select, manipulate and query patterns; and (iii) execute cross-over queries over patterns. Queries can then be stored in the repository as views, in this way datasets and pattern sets are intensionally described.

Inductive databases have been mainly investigated in the context of the CINQ project of the European Community (CINQ, 2001), which tries to face with both theoretical and practical issues of inductive querying for the discovery of knowledge from transactional data. CINQ covers several different areas, spreading from data mining task to machine learning. The considered data mining patterns are itemsets, association rules, episodes, data dependencies, clusters, etc. In the machine learning context, interesting patterns considered by the project are equations describing quantitative laws, statistical trends, and variations over data.

From a theoretical point of view, a formal theory is provided for each type of patterns, providing: (i) a language for pattern description; (ii) evaluation functions, for computing measures and other significant data related to patterns; (iii) primitive constraints for expressive basic pattern properties (e.g. minimal/maximal frequency, minimal accuracy). By using primitives constraints, extraction and further queries (seen as a postprocessing step in the overall architecture) can be interpreted as constraints and executed by using techniques from constraint programming, using concepts from constraint-based mining. Other manipulation operations, such as the insertion of a-priori patterns, are delegated to the underlying DBMS, since an integrated architecture is exploited. Note that since a theory is provided for each type of patterns, integration is not an issue of the project. Moreover, no support for temporal management and pattern hierarchies is provided.

From a more practical point of view, extension of existing querying standards, such as SQL, have been provided in order to query specific types of patterns, mainly association rules. The
combination of a data mining algorithm (usually some variant of the Apriori algorithm (Agrawal, R. & Srikant, R., 1994)) with a language such as SQL (or OQL) offers some interesting querying capabilities. Among the existing proposals, we recall the following:

- **DMQL (Data Mining Query Language)** (Han et Al., 1996) is an SQL-based data mining language for generating patterns from relational data. An object-oriented extension of DMQL, based on Object Query Language (OQL) (Cattell & Barry, 2000), has been presented in (Elfekey et al., 2001). Discovered association rules can be stored in the system, but no post-processing (i.e. queries over the generated patterns) is provided. Indeed, they are simply presented to the user and a further iterative refining of mining results is possible only through graphical tools. The obtained rules can be specialized (generalized) by using concept hierarchies over source data. Besides association rules, other patterns can be generated such as: data generalizations (a sort of aggregates), characteristic rules (assertions describing a property shared by most data in certain data set, for example the symptoms of a certain disease), discriminant rules (assertions describing characteristics that discriminate a dataset from another one), data classification rules (patterns for data classification). For each type of pattern, a set of measures is provided (confidence and support for association rules) and conditions over them can be used in order to generate only patterns with a certain quality level (Figure 1(a)).

- **MINE RULE** (Meo et Al., 1998) extends SQL with a new operator, named MINE RULE, for discovering association rules from data stored in relations. By using the MINE RULE operator, a new relation with schema (BODY, HEAD, SUPPORT, CONFIDENCE) is created, containing a tuple for each generated association rule. The body and the head itemsets of the generated rules are stored in dedicated tables and referred within the rule-base table by using foreign keys. The cardinality of the rule body as well as minimum support and confidence values can be specified in the MINE RULE statement. MINE RULE is very flexible in specifying the subset of raw data from which patterns have to be extracted as well as conditions that extracted patterns must satisfy. However, no specific support for post-processing (queries) is provided, even if standard SQL can be used since rules are stored in tables. Similarly to DMQL, hierarchies over raw data can be used to generalize the extracted association rules, or said better, to extract only association rules at a certain level of generalization. A similar operator, called XMine, for extracting association rules from XML documents has been presented in (Braga et al., 2002).
- **Mine-SQL (MSQL)** (Imielinsky & Virmani, 1999) is another SQL-like language for generating and querying association rules. Similarly to MINE RULE, only association rules are considered. Also in this case input transactions and resulting rules are stored in relations. With respect to MINE RULE, it supports different types of statements, one for rule extraction (GetRules), one for rule post-processing (SelectRules), and some predicates for cross-over queries (Satisfy, Violate). Concerning extraction, MSQL is less flexible in specifying the source data set; indeed, it must be an existing table or view. However, similarly to MINE RULE, constraints over the rules to be generated can be specified. Extracted queries can be queried using the SelectRules operator. Various conditions can be specified, depending on the body and the head of the rules. By using the SelectRules statement, it is also possible to recompute measures of already extracted rules over different datasets. In order to explicitly support cross-over queries, MSQL proposes the operators satisfy and violate. They determine whether a tuple satisfies or violates at least one or all the association rules in a given set, specified by either using GetRules or SelectRules commands.

Figure 1 presents an example of usage of the just presented languages for extracting association rules from transactions stored in relation Transactions, storing them, when possible, in relation MarketAssRules.

We finally recall that results achieved in the context of the CINQ project have been experimented in the context of machine learning in the implementation of a molecular fragment discovery demo system (MOLFEA, 2004). In the context of the association rule mining, they have been experimented in the demo version of Minerule Mining System (Minerule System, 2004).

```sql
(a) Find association rules from Transactions with support threshold=0.3 with confidence threshold=0.9

(b) Select distinct 1..n item as Body,
    1..n item as Head,
    Support,
    Confidence
    from Transactions
    group by tr#
    extracting rules with
    Support:0.3,
    Confidence:0.9

(c) MINE RULE MarketAssRules AS
    select distinct
    1..n item as Body,
    1..n item as Head,
    Support,
    Confidence
    from Transactions
    group by tr#
    extracting rules with
    Support:0.3,
    Confidence:0.9

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

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

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

**Figure 1: Inductive languages: examples**
3-Worlds Model

The 3-Worlds (3W) Model (Johnson et al., 2000) is a unified framework for pattern management, based on a separated architecture. Under this approach, the pattern model allows one to represent three different worlds: the intensional world (I-World), containing the intensional description of patterns; the extensional world (E-World), containing an extensional representation of patterns; the data world (D-World), containing raw data. In the I-World, patterns correspond to (possibly overlapping) regions in a data space, described by means of linear constraints over the attributes of the analyzed data set. For example, a cluster of products based on their price in dollars can be described by the following constraint “10<=price <=20” (call it region cheap_product). More complex regions can be defined, composed of a set of constraints. In the E-World, each region is represented in its extensional form, i.e., by an explicit enumeration of the members of the source space satisfying the constraint characterizing the region. Thus, the extension corresponding to region cheap_product (contained in the I-world) contains all source data items with price between 10 and 20. Finally, the D-World corresponds to the source data set, in the form of relations, from which regions and dimensions can be created as result of a mining process. Note that regions in the I-World are not predefined, thus user-defined patterns are allowed. Each region can be associated with a number of attributes, including measures, which do not have a special treatment. Additionally, the framework does not support a-priori patterns. Indeed, operations to directly insert patterns in the system are not supported. Moreover, no pattern temporal management is provided.

Query languages for all the worlds have been proposed. In particular, for the D-World and the E-World, traditional relational languages can be used (with some minor extensions for the E-World). On the other hand, a dimension algebra has been defined over regions in the I-World, obtained by extending relational languages. The main operations of this language are described in the following:

- The selection operation allows pattern retrieval by invoking various spatial predicates such as overlap (||), containment (⊂), etc. between regions.
- The projection operation corresponds to the elimination of some property attribute i.e. it amounts to setting this value to ‘true’ in every region.
- A purge operator, returning inconsistent regions, i.e., regions whose constraint cannot be satisfied by any data point (thus, with an empty extensional representation). For
instance, a region with constraint price > 20 AND price < 10 is clearly inconsistent, since the constraint is intrinsically unsatisfiable.

- Traditional relational operators (cartesian product, union, minus, and renaming) have then been extended to cope with sets of regions.

The following cross-over operators are also provided that allow the user to navigate among the three worlds: (a) automatic extraction of patterns (mine); (b) the assignment of an extension to a region (populate), given a certain data source; (c) the detection of the regions corresponding to a certain extension (lookup); (d) a sort of synchronization, providing the computation of new extensions starting from combinations of regions and a given dataset (refresh). Note that all the previous operators but mine can be interpreted as cross-over query operators. We remark that, even if no PML is explicitly provided, some of the proposed operators can be interpreted as PML operations when assuming to change the three worlds according to the query result. For example, the mine operator can be seen as a PML operator when the result of the mining is made persistent in the I-World.

PANDA Project

The purposes of the PANDA (PAtterns for Next-generation DAtabase systems) project of the European Community (PANDA, 2001) are: (1) to lay the foundations for pattern modeling; (2) to investigate the main issues involved in managing and querying a pattern-base; and (3) to outline the requirements for building a PBMS.

The PANDA approach relies on a separated architecture. The proposed model provides the representation of arbitrary and heterogeneous patterns, by allowing the user to specify its own pattern types. It provides support for both a-priori and a-posteriori patterns and it allows the user to define ad-hoc mining functions to generate a-posteriori patterns.

Under this modeling approach, pattern quality measures are explicitly represented, as well as relationships between patterns and raw data that can be stored in an explicit or approximated way. For example, a cluster of products based on their price in dollars can be described in an approximate way by follow constraint “10<=price<=20”. However, not necessarily all products with price between 10 and 20 belongs to this cluster. Thus, an explicit representation of the relationship between patterns and raw data will list the exact set of products belonging to the cluster. Moreover, the definition of hierarchies involving pattern types has been taken into account, in order to address extensibility and reusability issues. Three types of hierarchies
between pattern types have been considered: specialization, composition, and refinement. Specialization is a sort of inheritance between pattern types. On the other hand, composition is a sort of aggregation. Finally, refinement allows patterns to be used as source data. As an example, in the proposed scenario, clusters of association rules rely on a refinement relationship with association rules. If the representative of such cluster is an association rule, then there exists also a composition relation between them.

In this context, languages for pattern manipulation and querying have also been defined. In particular, the pattern manipulation language supports the main manipulation operations involving patterns, such as pattern insertion and deletion. Both a-priori and a-posteriori patterns can be manipulated by using the language proposed. On the other hand, by using the proposed pattern query language patterns inserted in the system (directly or mined by applying a mining function) can be retrieved and queried by specifying filtering conditions involving all pattern characteristics supported by the model. Additionally, it allows the user to combine different patterns and to correlate them with raw data, i.e. it supports cross-over operations. An approach for pattern similarity has also been provided (Bartolini et al., 2004).

Starting by the PANDA approach (PANDA, 2001; Catania et al., 2004), an extended model for patterns has been proposed. Such a model addresses the need for temporal information management associated with patterns. In this way, it becomes possible to exploit and manage information concerning pattern semantics and temporal validity, including synchronization and recomputation. Furthermore, the previously proposed PML and PQL have been extended in order to cope with temporal features during pattern manipulation and querying.

**Concluding Discussion**

Table 1 summarizes the features of the frameworks presented above, according to the previously introduced parameters. In the table, the PANDA approach refers to the extended temporal model (Catania et al., 2004).

Concerning the architecture, only inductive databases adopt an integrated approach. On the other hand, 3W and PANDA rely on a separated PBMS. For what concerns the model, the more general approach seems to be PANDA, where there is no limitation on the pattern types that can be represented. PANDA is also the only approach taking into account temporal aspects, hierarchies, and providing both a precise and an approximated relationship of patterns.
with respect to source data. In particular, it can be shown that the approximated representation in PANDA is quite similar to the region representation in 3W.

Concerning the manipulation language, 3W and CINQ do not support direct insertion of patterns as well as deletion and update operations. On the other hand, all the proposals take into account synchronization (recomputation) issues. Concerning the query language, all the approaches propose either one (or more) calculus or algebra languages, providing relational operators.

<table>
<thead>
<tr>
<th>Type of Architecture</th>
<th>Inductive Databases</th>
<th>3W Model</th>
<th>PANDA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Predefined Types</strong></td>
<td>- Itemsets</td>
<td>- Association Rules</td>
<td>Users can define their own types, that must be represented as sets of constraints</td>
</tr>
<tr>
<td></td>
<td>- Association Rules</td>
<td>- Sequences</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Sequences</td>
<td>- Clusters</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Clusters</td>
<td>- Equations</td>
<td></td>
</tr>
<tr>
<td><strong>Link to source data</strong></td>
<td>Yes. Datasource is part of the architecture.</td>
<td>Yes. Datasource is one of the layers of the architecture. Relationship is precise</td>
<td>Yes. Datasource is one of the layers of the architecture. Relationship can be either precise or approximated.</td>
</tr>
<tr>
<td><strong>Quality measures</strong></td>
<td>Yes</td>
<td>Yes, but not explicit</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Mining function</strong></td>
<td>Yes. The mining process is a querying process.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Temporal features</strong></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Hierarchical types</strong></td>
<td>No</td>
<td>Yes</td>
<td>Specialization, composition, and refinement.</td>
</tr>
<tr>
<td><strong>Manipulation Language</strong></td>
<td>Manipulation through constraint-based querying and SQL.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Automatic extraction</strong></td>
<td>Yes. Constraint-based queries.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Direct insertion</strong></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Modifications and Deletions</strong></td>
<td>Yes (SQL)</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Synchronization over source data</strong></td>
<td>Yes, recomputation</td>
<td>Yes, recomputation</td>
<td>Yes, recomputation &amp; synchronization</td>
</tr>
<tr>
<td><strong>Mining function</strong></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Queries against patterns</strong></td>
<td>Constraint-based calculus</td>
<td>Algebra</td>
<td>Algebra &amp; Calculus</td>
</tr>
<tr>
<td><strong>Pattern combination</strong></td>
<td>No</td>
<td>Yes, Cartesian Product</td>
<td>Yes, Join</td>
</tr>
<tr>
<td><strong>Similarity</strong></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Queries involving source data</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 1: Features Comparison: Theoretical Proposals

Specific characteristics of languages provided in the context of inductive databases are summarized in Table 2. Concerning extracted patterns, MINE RULE and MSQL deals only with association rules, whereas DMQL and ODMQL deal with many different types of
patterns. When patterns are stored, SQL can be used for manipulation and querying (including cross-over queries). Among the proposed languages, however, only MSQL proposes ad-hoc operators for pattern retrieval and post-processing.

As a final consideration, we observe that when dealing with applications managing different types of patterns (this is the case of advanced knowledge discovery applications), the 3W Model and PANDA theoretical frameworks are the best solutions since they provide support for heterogeneous patterns in a unified way. On the other side, the inductive databases approach provides better solutions for specific data mining context, such as association rules management, with a low impact on existing SQL-based applications.

<table>
<thead>
<tr>
<th>Model</th>
<th>DMQL &amp; ODMQL</th>
<th>MINE RULE</th>
<th>MSQl</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Manipulation Language</th>
<th>DMQL &amp; ODMQL</th>
<th>MINE RULE</th>
<th>MSQl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only extraction, but no storage</td>
<td></td>
<td>Only extraction</td>
<td>Yes</td>
</tr>
<tr>
<td>Automatic extraction</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Direct insertion</td>
<td>No</td>
<td>Using standard SQL</td>
<td>Using standard SQL</td>
</tr>
<tr>
<td>Modifications and Deletions</td>
<td>No</td>
<td>Using standard SQL</td>
<td>Using standard SQL</td>
</tr>
<tr>
<td>Synchronization over source data</td>
<td>No</td>
<td>No</td>
<td>Yes, recomputation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query Language</th>
<th>DMQL &amp; ODMQL</th>
<th>MINE RULE</th>
<th>MSQl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only visualization and browsing</td>
<td></td>
<td>Using standard SQL</td>
<td>SQL-like</td>
</tr>
<tr>
<td>Pattern combination</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Similarity</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Queries involving source data</td>
<td>No</td>
<td>Using standard SQL</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 2: Features Comparison: Theoretical Proposals (query language)

STANDARDS

The industrial community proposed some standards to support pattern representation and management in the context of existing programming languages and database (or data warehousing) environments in order to achieve interoperability and (data) knowledge sharing. Thus, they provide the right front-end for pattern management applications.

In general, they do not support generic patterns and, similarly to the inductive database approach, specific representations are provided only for specific types of patterns. Moreover, they do not provide support for inter-pattern manipulation.
Some proposals, such as Predictive Model Markup Language (PMML, 2003) and Common Warehouse Data Model (CWM, 2001), mainly deal with data mining and data warehousing pattern representation, respectively, in order to support their exchange between different architectures. Some others, such as Java Data Mining (JDM, 2003) and SQL/MM Data Mining (ISO SQL/MM part 6, 2001), provide standard representation and manipulation primitives in the context of Java and SQL, respectively. In the following, all these proposals will be briefly presented and compared with respect to the parameters previously introduced.

**Predictive Model Markup Language**

PMML (PMML, 2003) is a standardization effort of DMG (Data Mining Group) and it consists in an XML–based language to describe data mining models (i.e., the mining algorithm, the mining parameters, and mined data) and to share them between PMML compliant applications and visualization tools. Figure 2 shows an extract of a PMML association rule mining model. Since PMML is primary aimed to the exchange of data between different architectures, no assumptions about the underlying architecture are done.

PMML traces information concerning the data set from which a pattern has been extracted by allowing the user to specify the data dictionary, i.e. the collection of raw data used as input for the mining algorithm. Concerning the mining function, it is possible to express the fact that a certain pattern has been mined from a certain raw data set by using a specified mining algorithm. However, no assumption is made on the existence of a mining library. For instance, in the example shown in Figure 2, the represented association rule is the result of the application of the A-Priori algorithm (algorithmName = “Apriori”).

Moreover, PMML does not allow the user to define its own types. Indeed, one can only define models of one of the predefined types, that, however, cover a very large area of the data mining context (see Table 3). It is also important to note that PMML allows the user to represent also information concerning quality measures associated with patterns.

Due to its nature, PMML does not provide temporal features. Even if no general support for pattern hierarchies is provided, in PMML 3.0 refinement is supported for decision trees and simple regression models. More general variants may be defined in future versions of PMML.

All major commercial products supporting data knowledge management and data mining attempt to be compliant with PMML standard. Among them we recall Oracle Data Mining
tool in Oracle 10g (Oracle DM), DB2 Intelligent Miner tools (DB2), and MS SQL Server 2005 Analysis Services (MS SQL).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PMML>
  ...
  <!-- items in input data for the mining of association rule #1 -->
  <Item id="1" value="milk"/>
  <Item id="2" value="coffee"/>
  <Item id="3" value="bread"/>
  ...
  <!-- definition of the mining model used -->
  <AssociationModel modelName="mba">
    <!-- mining algorithm used -->
    <algorithmName>Apriori</algorithmName>
    <!-- tuples in Transactions -->
    <numberOfTransactions>10</numberOfTransactions>
    <!-- thresholds for support and confidence -->
    <minimumSupport>0.3</minimumSupport>
    <minimumConfidence>0.9</minimumConfidence>
    ...
    <!-- item sets involved in association rule #1 -->
    <ItemSet id="1" numberOfItems="1">
      <ItemRef itemRef="2"/>
    </ItemSet>
    <ItemSet id="2" numberOfItems="1">
      <ItemRef itemRef="3"/>
    </ItemSet>
    ...
    <!-- association rules -->
    <AssociationRule support="0.3" confidence="1.0" antecedent="1" consequent="2"/>
    ...
  </AssociationModel>
</PMML>
```

Figure 2: PMML example

Common Warehouse Metamodell

CWM (CWM, 2001) is a standardization effort of the ODM (Object Management Group) and it enables easy interchange of warehouse and business intelligence metadata between warehouse tools, platforms and metadata repositories in distributed heterogeneous environments. CWM is based on three standards: (i) UML (Unified Modeling Language) (UML, 2003), an object oriented modeling language, used for representing object models; (ii) MOF (Meta Object Facility) (MOF, 2003), which defines an extensible framework for defining models for metadata and provides tools with programmatic interfaces to store and access metadata in a repository; (iii) XMI (XML Metadata Interchange) (XMI, 2003), which allows metadata compliant with the MOF meta-model to be interchanged as streams or files with a standard XML-based format.

CWM has been defined as a specific metamodel for generic warehouse architectures. Thus, it is compliant with the MOF metamodel and relies on UML for the object representation and notation. Since MOF is a metamodel for metadata, also UML metamodels can be represented in MOF. This means that both CWM metadata and UML models can be translated into XML documents by using XMI through the mapping with MOF.

CWM consists of various meta-models, including a meta-model for data mining (CWM-DM), by which mining models and parameters for pattern extraction can be specified.
Unfortunately, CWM has been designed to analyze large amounts of data, where the data mining process is just a small part. Only few pattern types can be represented: clustering, association rules, supervised classification, approximation, and attribute importance. The user has not the capability to define its own pattern types and no temporal and hierarchical information associated with patterns can be modeled.

Finally, no dedicated languages for query and manipulation are proposed, since it is assumed manipulation is provided by the environment importing CWM-DM metadata. Due to the complexity of the model, CWM is supported to some extent by most commercial systems providing solutions for data warehousing, such as Oracle, IBM (within DB2), Genesis and Iona Technologies (providing e-datawarehouse solutions), Unisys (providing backbone solutions for UML, XMI and MOF, core tools for CWM development). However, CWM-DM is rarely integrated in specific solutions for data mining where often only import/export in PMML, providing a much more simple pattern representation, is provided.

**SQL/MM – DM**

The International Standard ISO/IEC 13249 (ISO SQL/MM part 6, 2001) “Information technology - Database languages - SQL Multimedia and Application Packages (ISO SQL/MM)” is a specification for supporting data management of common data types (text, spatial information, images, data mining results) relevant in multimedia and other knowledge intensive applications in SQL-99. It consists of several different parts. Part number 6 is devoted to data mining aspects. In particular, it attempts to provide a standardized interface to data mining algorithms that can be layered at the top of any object-relational database system and even deployed as middleware when required, by providing several SQL user-defined types (including methods on those types) to support pattern extraction and storage. Differently from PMML and CWM, it does not only address the issue of representation but also of manipulation. Thus, it can be used to develop specific data mining applications on top of an object-relational DBMS (ORDBMS).

Four types of patterns are supported (thus, the set of pattern types is not extensible and no support for user-defined pattern types is provided): association rules, clusters, regression (predicting the ranking of new data based on an analysis of existing data), and classification (predicting which grouping or class new data will best fit based on its relationship to existing
data). For each pattern type, a set of measures is provided. For each of those models, various activities are supported:

- **Training**: the mining task (also called the model) is specified by choosing a pattern type, setting some parameters concerning the chosen mining function, and then applying the just configured mining function over a given dataset.
- **Testing**: when classification or regression are used, a resulting pattern can be tested by applying it to known data and comparing the pattern predictions with that known data classification or ranking value.
- **Application**: when clustering, classification or regression are used, the model can then be applied to all the existing data for new classifications or cluster assignment.

All the previous activities are supported through a set of SQL user-defined types. For each pattern type, a type DM_*Model (where the '*' is replaced by a string identifying the chosen pattern type) is used to define the model to be used for data mining. The models are parameterized by using instances of the DM_*Settings type, which allows various parameters of a data mining model, such as the minimum support for an association rule, to be set.

Models can be trained using instances of the DM_ClassificationData type and tested by building instances of the DM_MiningData type that holds test data, and instances of the DM_MiningMapping type that specify the different columns in a relational table that are to be used as a data source. The result of testing a model is one or more instances of the DM_*TestResult type (only for classification and regression). When the model is run against real data, the obtained results are instances of the DM_*Result type. In most cases, instances of DM_*Task types are also used to control the actual testing and running of your models.

Since SQL/MM is primarily aimed at enhancing SQL with functionalities supporting data mining, no specific support is provided for a-priori patterns. Advanced modeling features, such as the definition of pattern hierarchies and temporal information management, are not taken into account. However, queries over both data and patterns can be expressed, through SQL. In the same way, the specified mining model and patterns can be modified or deleted.

**Java Data Mining API**

The Java Data Mining (JDM) API (JDM, 2003) specification addresses the need for a pure Java API to facilitate the development of data mining applications. While SQL/MM deals with representation and manipulation purposes inside an ORDBMS, Java Data Mining is a
pure Java API addressing the same issues. As any Java API, it provides a standardized access
to data mining patterns that can be represented according to various formats, including
PMML and CWM-DM. Thus, it provides interoperability between various data mining
vendors by applying the most appropriate algorithm implementation to a given problem
without having to invest resources in learning each vendor’s API.
JDM supports common data mining operations, as well as the creation, storage, access, and
maintenance of metadata supporting mining activities, under an integrated architecture,
relying on three logical components: (i) Application Programming Interface (API), which
allows end-users to access to services provided by the data mining engine (DME); (ii) Data
Mining Engine (DME), supporting all the services required to the mining process, including
data analysis services; and (iii) Mining Object Repository (MOR), where data mining objects
are made persistent together with source data. Various technologies can be used to implement
the MOR, such as a file-based environment, or a relational/object database, possibly based on
SQL/MM specifications. The MOR component constitutes the repository against which
queries and manipulation operations are executed.
Through the supported services, a-posteriori patterns of predefined types (see Table 3) can be
generated by using several different mining functions. Similarly to SQL/MM, pattern
extraction is executed through tasks, obtained by specifying information concerning the type
of patterns to be extracted, the source dataset, the mining function and additional parameters.
Each generated pattern is associated with some measures, representing the accuracy with
respect to raw data. Patterns are then stored in the MOR and then used for mining activities.
JDM supports various import and export formats, including PMML.

Concluding Discussion
Table 3 summarizes the features of the presented standards, according to the previously
introduced parameters. From the previous discussion, it follows that all the commercial
proposals described above rely on an integrated architecture. Among them, PMML and
CWM-DM simply address the problem of pattern representation. On the other hand,
SQL/MM and JDM cope with both pattern representation and management.
All standards provide a support for the representation of common data mining patterns.
Among them, PMML provides the largest set of built-in pattern types. No user-defined
patterns can be modeled, i.e., the set of pattern types is not extensible.
All standards allow users to specify the mining function/algorithm they want to apply. However, in PMML it is just a string used only for user information purposes. Furthermore, all considered approaches support measure computation and description of the source dataset, which is used in SQL/MM and JDM for pattern extraction.

<table>
<thead>
<tr>
<th>Model &amp; Architecture</th>
<th>PMML</th>
<th>JDM API</th>
<th>SQL/MM</th>
<th>CWM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type of Architecture</strong></td>
<td>Only representation of patterns. No Architecture</td>
<td>Integrated</td>
<td>Integrated</td>
<td>Only representation of patterns. No architecture</td>
</tr>
<tr>
<td><strong>Predefined Types</strong></td>
<td>-Association Rules -Decision Trees -Center/Distribution Based Clustering -(General) Regression -Neural Networks -Sequences</td>
<td>-Clustering -Association Rules -Classification -Approximation -Attribute Importance</td>
<td>-Clustering -Association Rules -Classification -Regression</td>
<td>-Clustering -Association Rules -Supervised -Classification -Approximation -Attribute Importance</td>
</tr>
<tr>
<td><strong>Link to source data</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Quality measures</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Temporal features</strong></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Hierarchical types</strong></td>
<td>Partial</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Manipulation Language</strong></td>
<td>No</td>
<td>Java API</td>
<td>SQL</td>
<td>No</td>
</tr>
<tr>
<td><strong>Automatic extraction</strong></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Direct insertions</strong></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Modifications and deletions</strong></td>
<td>No</td>
<td>Possible through direct access to objects via Java</td>
<td>Possible through direct access to objects via SQL</td>
<td>No</td>
</tr>
<tr>
<td><strong>Synchronization over source data</strong></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Mining function</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Query Language</strong></td>
<td>No</td>
<td>Java-API</td>
<td>SQL</td>
<td>No</td>
</tr>
</tbody>
</table>

**Table 3: Feature comparison: Industrial proposals**

None of the standards supports advanced modeling features concerning patterns such as temporal information management associated with patterns and definition of hierarchies involving patterns. Moreover, no specific support for a-priori patterns is provided by such approaches even if imported patterns in JDM can be seen as a sort of a-priori patterns. Concerning pattern management, no dedicated languages for pattern manipulation are supported. In ISO SQL/MM and JDM, since raw data and patterns are stored together, manipulation and querying are possible by using typical languages used for accessing data. Finally, we outline that since PMML and CWM simply address the issue of pattern representation, they can be used in any PBMS architecture. As we will see later, most commercial systems support PMML, which guarantees a clear XML representation that can
be easily integrated with other XML data; on the other hand, due to its complexity, CWM-DM is rarely supported. Differently, SQL/MM and JDM can be used to develop specific data mining applications on top of existing technologies. In particular, SQL/MM can be put on top of an ORDBMS environment, whereas JDM works in a JAVA-based environment, providing an implementation for the proposed API.

METADATA MANAGEMENT

Pattern can be interpreted as a kind of metadata. Indeed, metadata in general represent data over data and, since patterns represent knowledge over data, there is a strong relationship between metadata management and pattern management. However, as we have already stressed, pattern management is a more complex problem since patterns have some peculiar characteristics that general metadata do not have. Indeed, metadata are usually provided for maintaining process information, as in data warehousing, or for representing knowledge in order to guarantee interoperability, as in the Semantic Web and intelligent agent systems. Specific pattern characteristics, such as quantification of importance through quality measures, are not taken into account. Usually, metadata are not used to improve and drive decision processes. Since however metadata management has somehow influenced pattern management, in the following we briefly describe some approaches defined in this context. In the artificial intelligence area many research efforts have been invested in the Knowledge Sharing Effort (KSE, 1997), a consortium working on solutions for sharing and reuse of knowledge bases and knowledge bases systems. Standards proposals of such a consortium are computer-oriented, i.e. they are not dedicated to human users even if in some cases they can take advantages in using the proposed standard languages. The most important contributions developed by the consortium are Knowledge Interchange Format (KIF) and Knowledge Query and Manipulation Language (KQML) specification. The first one is a declarative language to express knowledge about knowledge and it used to exchange knowledge units among computers. It does not provide support for internal knowledge representation, thus each computer receiving KIF data translate them into its internal logical model in order to be able to apply some computation process. On the other side, the second contribution proposes a language and a protocol supporting interoperability and cooperation among collections of intelligent agents involved in distribute applications. KQML can be used as a language by an application to interact with an intelligent agent system or by two or more intelligent systems to interact cooperatively in problem solving.
Concerning the emerging Semantic Web research area, Web metadata management problem has been taken into account by the W3C and a framework for representing information in the Web, Resource Description Framework (RDF, 2004), has been proposed. One of the essential goals of RDF (and RDF-Schema) is to allow - in a simple way - the description of Web metadata, i.e. information about Web resources and how such resources can be used by third-party, in order to make them available not only for human users but also for machines and automatic processes. RDF uses an XML-based syntax and it exploits the URI identification mechanism. Recently, an emerging research field coping with the integration of ontology management and Web data management has been started up. In this context, W3C proposes a recommendation for a dedicated language: the Web Ontology Language (OWL, 2004). OWL is primary dedicated to applications that need to process the content of information instead of just presenting information to human users. OWL supports better machine interpretability of Web content than XML, RDF, or RDF Schema (RDF-S) solutions since it provides an extended vocabulary along with a more precise semantics.

Issues concerning metadata management have also been extensively considered in the context of the Dublin Core Metadata Initiative (DCMI, 2005), “an organization dedicated to promoting the adoption of interoperable metadata standards and developing specialized metadata vocabularies for describing resources that enable more intelligent information discovery systems”. The main aim of DCMI is to support Internet resources identification through the proposal of metadata standards for discovery across domains and frameworks (tools, services, and infrastructure) and for the metadata sharing.

### PATTERN SUPPORT IN COMMERCIAL DBMSs

Since the ability to support business intelligence solutions enhances the market competitiveness of a DBMS product, all the most important DBMS producers supply their products with solutions for business intelligence supporting data mining and knowledge management processes. Pattern management in commercial DBMSs is provided in the context of such environments. In the remainder of this section we will briefly discuss data mining solutions proposed by three leading companies in database technology: Oracle, Microsoft, and IBM.

#### Oracle Data Mining Tool

Starting from release 9i, Oracle technology supports data mining processing (Oracle DM, 2005). In Oracle Data Mining 10g, basic data mining features have been specialized and
enhanced. Oracle Data Mining (ODM) is a tool tightly integrated with Oracle DBMS supporting basic data mining tasks, such as classification, prediction, and association, but also clustering and ranking attribute importance. By using ODM, the user can extract knowledge (in the form of different kinds of patterns) from corporate data lying in the underlying Oracle databases or data warehouses. Supported patterns include categorical classifiers (computed by applying Naïve Bayes Network or Support Vector Machines), continuous/numerical classifier relying on linear or non-linear regression models (obtained by Support Vector Machines), association rules, clusters (produced by the K-Means algorithm or a proprietary clustering algorithm). Mining algorithms and machine learning methods are built-in in ODM, but the user may change some settings and/or define new parameters for the mining model through the ODM Java API. Statistical measures can be associated with classifiers and association rules.

In the latest release (i.e., ODM 10g-Release1), ODM’s functionalities can be accessed in two ways: through a Java-based API or through the PL/SQL interface. Up to now, Java API and PL/SQL API are not interoperable, i.e. a model created in Java cannot be used in PL/SQL and vice-versa. To overcome this limitation, the next ODM release (10g-Release2) will adhere to the JDM standard specification and a JDM API will be implemented as a layer on top of ODM PL/SQL API and the current Java API will be abandoned.

Concerning other data mining standards, ODM supports PMML import and export, but only for Naïve Bayes and association rule models. Exchanges through PMML documents are fully supported between Oracle database instances, but the compatibility with PMML models produced by some other vendors can be achieved only if they use core PMML standard.

**Microsoft SQL Server 2005**

The business solution proposed by Microsoft SQL Server exploit OLAP, data mining and data warehousing tools (MS SQL, 2005). The pioneer data mining functionalities appeared in SQL Server 2000 (only two types of patterns were supported: decision trees and clusters), but they have been consolidated and extended in the recent SQL Server 2005 beta release. Within the SQL Server environment, there are tools supporting data transformation and loading, pattern extraction and analysis based on OLAP services.

SQL server 2005 allows the user to build different types of mining models dealing with traditional mining patterns (such as decision tree, clusters, Naïve Bayes classifier, time series, association rules and neural networks) and to test, compare, and manage them in order to
drive the business decision processes. Seven mining algorithms are provided by SQL Server 2005. The entire knowledge management process is performed through a Mining Model Editor to define, view, compare, and apply models. Besides this editor, additional tools are provided to exploit other mining phase (for example, data preparation). Within the SQL Server 2005, through OLE DB for Data Mining (OLEDB, 2005), it is possible to mine knowledge from relational data sources or multi-relational repository. OLE DB for Data Mining extends SQL to integrate data-mining capabilities in other database applications. Thus, it provides storage and manipulation features for mined patterns in an SQL style. Using OLE DB for Data Mining, extracted patterns are stored in a relational database. To create a mining model, a CREATE statement quite similar to the SQL CREATE TABLE statement can be used; to insert new patterns in your mining model, the INSERT INTO statement can be used; finally, patterns can be retrieved and predictions made by using the usual SQL SELECT statement. For the sake of interoperability and compatibility with standards, OLE DB for Data Mining specification incorporates PMML.

IBM DB2 Intelligent Miner

DB2 databases management environment provides support for knowledge management by mean of a suite of tools, called DB2 Intelligent Miner (DB2, 2005), dedicated to the basic activities involved in the whole data mining process. Thus, users may use data mining functionalities as any other traditional relational function provided by the DBMS. The interaction between DB2 Intelligent Miner’s tools takes place through PMML standard. In particular, an ad-hoc DB2 Extender for data mining allows the automatic construction of mining models within DB2/SQL applications and their update with respect to changes occurring in the underling raw data. The generated mining models are PMML models and are stored as Binary Large Objects (BLOBs). The other DB2 tools supporting training, scoring (or prediction), and visualization of a model works on PMML models, thus they can manage without additional overhead third-party PMML models. It is quite important to note that the scoring tool has the ability to score a mining model over data recorded not only on DB2 databases but also on Oracle ones. This capability has a great impact in applications development since it may reduce design and building costs. Since DB2 Intelligent Miner’s tools are tightly integrated with the database environment and the mining results are stored as BLOB, the user may interact with the system through an SQL
API. In particular, by using SQL it is possible to perform association rules discovery, clustering and classifications techniques provided by the DB2 environment.

Then, through ODBC/JDBC or OLE DB data mining results can be integrated within business applications developed using an external powerful programming language.

Concluding discussion

As we have seen, commercial DBMSs do not provide a comprehensive framework for pattern management; rather they support business intelligence by providing an applicational layer offering data mining features, in order to extract knowledge from data, and by integrating mining results with OLAP instruments in order to support advanced pattern analysis. For this reason, in general, they do not provide a dedicated logical model for pattern representation and querying, since these aspects are demanded to the applications using the mined results.

An exception is represented by SQL Server 2005, where through OLE DB, pattern storage, manipulation, and querying is made through OLE DB for Data Mining, which can be considered an SQL-based language for pattern management.

<table>
<thead>
<tr>
<th>Predefined Types</th>
<th>Oracle Data Mining (10g)</th>
<th>Microsoft SQL Server 2005</th>
<th>IBM DB2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Association rules</td>
<td>Association rules and itemsets</td>
<td>Association rules and itemsets</td>
<td>Association rules</td>
</tr>
<tr>
<td>Discrete and continuous</td>
<td>Clusters</td>
<td>Clusters</td>
<td>Clusters</td>
</tr>
<tr>
<td>classifier</td>
<td>Decision trees</td>
<td>Decision trees</td>
<td></td>
</tr>
<tr>
<td>Clusters</td>
<td>Naïve Bayes classifier</td>
<td>Time series</td>
<td></td>
</tr>
<tr>
<td>Attribute importance</td>
<td>Neural Networks</td>
<td></td>
<td>Classifiers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quality measures</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mining function</td>
<td>Built-in (user-defined settings)</td>
<td>Built-in (user-defined settings)</td>
<td>Built-in (user-defined</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>settings)</td>
</tr>
<tr>
<td>Temporal features</td>
<td>No</td>
<td>No</td>
<td>Yes (scoring phase)</td>
</tr>
<tr>
<td>Hierarchical types</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Supported standards</td>
<td>PMML</td>
<td>PMML</td>
<td>PMML</td>
</tr>
<tr>
<td></td>
<td>JDM (ODM 10g-Release2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Features Comparison: commercial DBMSs

All systems do not allow the user to define its own pattern types. Moreover, mining functions are built-in in the system; however, the user can modify some settings specializing the algorithm to the case he is interested in. Finally, none of the DBMSs takes into account advanced modelling aspects involving patterns, such as temporal information management and the existence of hierarchical relationships between patterns. Actually, only DB2 considers patterns-data synchronization issues, through a scoring mechanism that can be started up by some triggers monitoring raw data changes.

Table 4 summarizes the features of the described commercial DBMSs, by considering a subset of the previously introduced parameters.
ADDITIONAL ISSUES

In order to make PBMSs a practical technology, besides issues concerning architectures, models, and languages, additional issues have to be taken into account when developing a PBMS. Among them, pattern reasoning, physical design, query optimizations, and access control are fundamental issues that have only been partially taken into account by existing proposals. In the following, such topics will be discussed in more details.

**Pattern reasoning.** Pattern reasoning is supported only in few theoretical proposals, in the form of similarity check (PANDA) or pattern combination (3W-model and PANDA). However, an overall approach for reasoning about possibly heterogeneous patterns needs more sophisticated techniques, describing the semantics of pattern characteristics. As an example, consider measures. In general, various approaches exist for measure computation (general probabilities, Dempster-Schafer, Bayesian Networks - see for example (Silberschatz and Tuzhillin, 1996)). It is not clear how pattern, possibly having the same type but characterized by different measures, can be compared and managed together. Probably, measure ontologies can be used to support such kind of quantitative pattern reasoning.

**Physical design.** Since patterns are assumed to be stored in a repository, specific physical design techniques must be developed. Unfortunately, it is not clear up to now what should be a reasonable physical layer for patterns. Most commercial DBMSs store patterns as BLOB that are then manipulated using specific methods. However in order to provide a more efficient access, specific physical representations, clustering, partitioning, caching, and indexing techniques should be developed. Concerning theoretical proposals, as we have already seen, in the context of the 3W model, patterns are represented as regions, thus techniques developed for spatial databases can be used for their physical management.

**Query optimization.** Query optimization for pattern queries has been only marginally investigated. Some preliminary work, concerning query rewriting, has been proposed in the context of the 3W framework. An overall query optimization approach, taking into account choices concerning the physical design, has not been defined yet. Assuming to deal with a separated architecture, the main issue is how to perform data and patterns computations in an efficient way. An important issue here is how it is possible to use patterns to reduce data access in data and cross-over queries and how data and pattern query processors can be combined. On the other side, under an integrated architecture, where extraction is a kind of query, the main issue is the optimization of pattern generation. Some work has been done in
the context of inductive databases, where approaches to optimize pattern extraction, based on constraints on pattern properties, (Ng et al., 1998), or to refine the set of generated patterns (Baralis & Psaila, 1999), have been proposed. Techniques for reducing the size of the generated pattern sets, by representing them using condensed representations, have also been proposed for itemsets and association rules (CINQ).

**Access control.** Patterns represent high-sensitive information. Their access has therefore to be adequately controlled. The problem is quite similar to access control in presence of inference (Farkas & Jajodia, 2002). In general, assuming a user has access to some non-sensitive data, the inference problem arises when, through inference, sensitive data can be discovered from non-sensitive one. In terms of patterns, this means that users may have the right to access some patterns, for example some association rules, and starting from them they may infer additional knowledge over data, over which they may not have the access right. Techniques already proposed in the inference context should be adapted and extended to cope with the more general pattern management framework. Some of these approaches rely on pre-processing techniques, checking through mining techniques whether it is possible to infer sensitive data; some others can be applied at run-time, i.e. during the knowledge discovery phase, releasing patterns only when they do not represent sensitive information; finally modifications over original data, such as perturbation and sample size restrictions, do not disturbing data mining results can be also applied in order to encrypt the original data and to prevent unauthorized user data access.

## CONCLUSIONS

Patterns refer to knowledge artifacts used to represent in a concise and rich in semantics way huge quantity of heterogeneous raw data or some of their characteristics. Patterns are relevant in any knowledge intensive application, such as data mining, information retrieval, or image processing. In this chapter, after presenting a sample scenario of pattern usage, specific issues concerning pattern management have been pointed out, in terms of the used architecture, models, and languages. Several parameters have also been identified and used in comparing various pattern management proposals.

From the analysis proposed in this chapter, it follows that there is a gap between theoretical proposals and standard/commercial ones, that spans from a lack of modelling capabilities (such as no support for user-defined patterns, pattern hierarchies, or temporal features management in standard/commercial proposals) to a lack of manipulation and processing
operations and tools (no manipulation of heterogeneous patterns, no support for similarity, pattern combination and synchronization in standard/commercial proposals). More generally, the analysis has shown that, even if several proposals exist, an overall framework, in terms of the current standards, to represent and manipulate patterns is still missing. In particular, aspects related to the physical management of patterns have not been considered at all.

On the other hand, the diffusion of knowledge intensive applications that may benefit from pattern technology is increasing. A combined effort of the academic community with industries is therefore required for establishing the real need of such features and the extension of existing standards in these directions. We however remark that the support of pattern combination in the last PMML version seems to answer positively this question.

ACKNOWLEDGMENTS

The authors would like to thank Maurizio Mazza for his valuable contribution to this chapter.

REFERENCES


Han, J., et al. (1996). DMQL: A data mining query language for relational databases. In *Proc.of SIGMOD’96 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD’96)*.


JDM (2003). Java Data Mining API. http://www.jcp.org/jsr/detail/73.prt


SIGKDD (2002). SIGKDD Explorations - Special Issue on Constraint-Based Mining


