An XML-Based Database for Knowledge Discovery

Rosa Meo[1] and Giuseppe Psaila[2]

[1] Università di Torino, Dipartimento di Informatica,
corso Svizzera 185, I-10149, Torino, Italy,
e-mail: meo@di.unito.it
[2] Università di Bergamo, Facoltà di Ingegneria,
Viale Marconi 5, I-24044 Dalmine (BG), Italy,
e-mail: psaila@unibg.it

Abstract. Pattern Management Systems and Inductive Databases, are proposed as a new generation of general purpose databases with the aim to manage data mining patterns and work as knowledge bases in support to the deployment of the KDD process. One of the main problems to be solved is the integration between data and patterns and pattern maintenance when data update. Unfortunately, the heterogeneity of the patterns that represent the extracted knowledge and of the different conceptual tools used to find the patterns make difficult this integration in a unique framework.
In this paper, we explore the feasibility of using XML as the unifying framework for inductive databases, and present a model, named XDM (XML for Data Mining). We will show the basic features of the model, such as the storage in the same database of both data and patterns. To store patterns, we consider determinant for their interpretation the storage of the pattern derivation process which is described by the concept of statement, based on data mining operators. Some of the statements are automatically generated by the system while maintaining consistence between source and derived data. Furthermore, we show how the use of XML namespaces allows the effective coexistence of different data mining operators and provides extensibility to new operators. Finally, we show that with the use of XML-Schema we are able to define the schema, the state and the integrity constraints of an inductive database.

1 Introduction

Data mining applications are called to extract descriptive and predictive patterns, typically used for decision making, from the data contained in traditional databases and from other unconventional information systems such as the web.

Inductive Databases (IDB) have been launched in [8] as general-purpose databases in which both the data and the patterns can be represented, retrieved and manipulated with the goal to assist the deployment of the *Knowledge Discovery Process* (KDD). Thus, KDD becomes a querying sequence in a query

language designed for a specific data mining problem. Pattern Management Systems (PMS) [12] have been proposed instead with the main concern of representing and managing in a unique system a collection of heterogeneous patterns. Consequently, both of them should integrate several heterogeneous data mining patterns that deal with very different, and complex data models. For example, classification tools usually adopt a data model that is a classification tree or rules, while basket analysis usually represent patterns by means of set enumeration models. In [12] a logic-based model of a pattern management system is studied which satisfies the basic requirements of generality, extensibility and reusability. In [5] object oriented modeling techniques have been applied to obtain a uniform model of a pattern management database. In [11] UML has been applied, instead.

In this paper, we present a semi-structured data model specifically designed for inductive databases and, more generally, for *Knowledge Discovery Systems*. This model is called XDM (XML for Data Mining). It is based on XML and is devised to cope with several distinctive features at the same time. At first, it is semi-structured, in order to be able to represent an apriori infinite set of data models. Second, it is based on two simple and clear concepts, named *Data Item* (Section 2) and *Statement* (Section 3): a data item is a container of data and/or patterns; a statement is a description of an operator application. Third, with XDM the inductive database state is defined (Section 5) as the collection of data items and statements, and the knowledge discovery process is represented as a set of relationships between data items and statements. Fourth, with the aid of XML-Schema[3] (see the official documents [13, 2] for detailed descriptions) we define (Section 5) the concept of database schema which provides the set of integrity constraints over the operators' inputs and outputs and constitutes part of the meta-data of the KDD process. The above detailed features of the model set the foundations for the interoperability of the operators inside a unique framework. Finally, the adoption of XML as syntactic format provides several benefits; in particular, the concept of *namespace* opens the way to the integration of several data formats and operators inside the same framework. Throughout the paper we demonstrate the feasibility of the model to support KDD processes presenting a sample process (Section 4).

A similar set of functionalities, i.e. the application of XML to the deployment of the KDD process, is available also in [1] while [3] provides a framework for extracting knowledge from XML documents. XDM, however, provides a set of features that is peculiar to the integration of different patterns and models in inductive databases. At first, source data and patterns are represented at the same time in the model. Patterns may be stored either extensionally or intensionally, i.e. by storing only the statements that generate the patterns. Second, the pattern derivation process is stored in the database: this is determinant in many situations, such as for the maintenance of the patterns (that are a kind of

[3] XML-Schema specifications constrain the structure of XML documents and overcome the limitations of classical DTDs by adding the concept of data type for attributes.

derived data) when the original data is updated, in the phase of pattern interpretation and allows pattern reuse. Furthermore, the framework can be easily extended with new data mining operators (thanks to the embedding provided by namespaces). Thus IDB based on XDM really become open systems and a unifying framework in which various KDD data transformation phases take place. Finally, the use of XML allows the inclusion in IDB of semi-structured data.

To conclude, we want to highlight an important difference of XDM w.r.t. other XML formats for data mining and knowledge discovery, such as PMML [6]. PMML is a format to exchange patterns among different systems, thus it is focused on the description of patterns. XDM is not focused on the description of specific patterns; it is a framework for knowledge discovery activities, and the XML structure is the suitable format to host in a flexible way different representations for data and patterns, included PMML documents.

2 XDM Data Items

Tree Model. An XML document or fragment is represented as a tree of nodes, called *ElementNode*s. An *ElementNode* n has a possibly empty set of *attributes*, denoted as $n.Attributes$, i.e. pairs $(Name : String, Value : String)$ (they are the attributes defined by the XML syntax within tags). Furthermore, an *ElementNode* has the following properties: the element name $n.Name$ and the prefix associated to the name space $n.Prefix$ (that, with the notation $Prefix$:$Name$, we will use to refer to *ElementNodes*); the name space $r.NameSpace$ specifying the name space URI. Finally, an element node has a content $n.Content$, which is a possibly empty sequence of *ElementNode*.

Definition 1: An *XDM Data Item* is a tree fragment defined as follows.

• The root r is an *ElementNode* `XDM:DATA-ITEM`, belonging to the standard XDM name space whose prefix is `XDM`. In the content $r.Content$, only `XDM:DERIVATION` and `XDM:CONTENT` nodes are allowed (defined hereafter).
• The root node r has a set of attributes, $r.Attributes$, denoting the data item features: `Name`, `Date` and `Version`. `Virtual` denotes if the data item is materalized. □

Definition 2: The `XDM:CONTENT` node is an *ElementNode* (defined in the XDM name space), denoted as c. The `XDM:CONTENT` node has no attributes, and only one child *ElementNode* n in $c.Content$. □

Example 1: Consider the following XDM data item.

```
<XDM:DATA-ITEM Name="Purchases" Version="1"
     Date="..." Virtual="NO" xmlns:XDM="http://.../NS/XDM">
  <XDM:CONTENT>
    <TRANSACTIONS>
      <PRODUCT TID="1" CUST="c1" ITEM="A"  PRICE="25"/>
      <PRODUCT TID="1" CUST="c1" ITEM="B"  PRICE="12"/>
      <PRODUCT TID="2" CUST="c3" ITEM="C"  PRICE="30"/>
```

```
      .  .  .
    </TRANSACTIONS>
  </XDM:CONTENT>
</XDM:DATA-ITEM>
```

The start tag `XDM:DATA-ITEM` defines the attributes for the XDM data item named `Purchases`. Notice the name space definition `xmlns:XDM="http://.../NS/XDM"`, which says that all element nodes prefixed as `XDM` belong to the name space identified by the specified URI.
This data item is materialized (`Virtual="NO"`); therefore the content of the `Purchases` data item consists of the actual set of purchase transactions, with the details of each purchase. □

Definition 3: A `XDM:DERIVATION` node is an *ElementNode* (defined on the standard XDM name space), here denoted as *d*. In this case, *d.Attributes* contains only one mandatory attribute, `statement`, which contains the identifier of the XDM *statement* that generated the data item (see next sections for a detailed discussion on derivation). This is the statement that defines how the derived data item is generated and can be used later to recover the definition of that data item (for instance for documentation purposes or when original data is updated and the system needs to maintain consistency between the derived data item and the original data).

In addition, in derived data items *d.Content* could be empty. This is a case of a virtual data item (with `Virtual`="YES") that is very useful in the management and derivation of large data volumes. In fact, a virtual data item is not materialized, but only the statement that will be used to generate the data item is stored. This statement could be executed later, at the most convenient time for load-balancing of the system. □

Note that the `XDM:DERIVATION` node is required for derived data items (it stores the derivation process of the data item). On the contrary, a non-derived data item contains the `XDM:CONTENT` node only.

Example 2: The following XDM code shows the first version of a derived XDM data item, named `Rules`, containing the association rules extracted from the source data given in input, and materialized in the database. These data items are shown in the left hand side of Figure 1 that shows a sample KDD process.

```
<XDM:DATA-ITEM Name="Rules" Version="1" Date="..." Virtual="NO"
 xmlns:XDM="http://.../NS/XDM">
 <XDM:DERIVATION Statement="00128"/>
 <XDM:CONTENT>
 <AR:ASSOCIATION-RULE-SET xmlns:AR="http://...NS/DATA/AssRules">
      <AR:RULE>
          <AR:BODY>
              <AR:ELEMENT Name="ITEM">A</AR:ELEMENT>
              <AR:ELEMENT Name="ITEM">B</AR:ELEMENT>
          </AR:BODY>
          <AR:HEAD>
```

```
            <AR:ELEMENT Name="ITEM">C</AR:ELEMENT>
        </AR:HEAD>
        <AR:MEASURES>
            <AR:SUPPORT VALUE="0.5">
            <AR:CONFIDENCE VALUE="0.8">
            <AR:AVG-PRICE VALUE="22.33">
        </AR:MEASURES>
    </AR:ASSOCIATION-RULE>
    Other association rules
</AR:ASSOCIATION-RULE-SET>
</XDM:CONTENT>
</XDM:DATA-ITEM>
```

The `XDM:DERIVATION` node specifies (attribute `Statement`) the statement whose execution generated the item (statements will be described in Section 3). In particular, this item is generated by a statement based on the `MR:MINE-RULE` operator (see Section 3) which extracts association rules from a data item. The generated set of association rules is included in the `XDM:CONTENT` element. The description of association rules included here can be in PMML or XML-compliant. The version presented is an extension of PMML for the presence of the `ELEMENT` and `MEASURES` nodes. `ELEMENT` describes (by means of `Name`) the schema of the antecedent and consequent itemsets (named `BODY` and `HEAD`) in terms of their constituting elements, as defined in the data mining statement that generated them (generally speaking, `BODY` and `HEAD` schemas could be different). `MEASURES` helps the introduction by the users of new evaluation measures that define the validity of association rules and that can satisfy the different requirements of the various applications.

This XML fragment is based on a specific name space (with prefix `AR:` and its own URI) defined for association rule sets descriptions. Note that this difference w.r.t. the standard XDM name space is due to the fact that XDM is independent of the operators, that can be added to the XDM-based system when necessary. In the example, the association rule {`A,B`} ⇒{`C`} with the values of three evaluation measures (support, confidence and an aggregate value, average of the items price involved in the rule). □

Schema for XDM Data Items. The XML syntactic structure of XDM data items is very rich. However, Behind this structure, we can identify the concept of *Schema*. Given an XDM data item di, the schema of di, denoted as $Schema(di)$, is a four-tuple $Schema(di) = \langle NameSpace, Prefix, Root, xsd \rangle$ where $NameSpace$ is the namespace URI on which the content of the XDM data item is defined, $Prefix$ is the namespace prefix associated to the namespace URI, *root* is the root node element of the XML fragment within the `XDM:CONTENT` element in the data item; finally, xsd is the name of the file containing the XML-Schema definition that defines the XML structure for documents belonging to the specified namespace URI. For example, the schema of item in Example 2 is

⟨`"http://...NS/DATA/AssRules","AR","ASSOCIATION-RULE-SET","ar.xsd"`⟩

3 XDM Statements

The XDM model is devised to capture the KDD process and therefore it provides also the concept of *statement*. This one specifies the application of an operator (for data manipulation and analysis tasks) whose execution causes the generation of a new, derived data item.

Definition 4: An XDM statement s is specified by an XML fragment, whose structure is the following.

• The root of the fragment is an element node `XDM:STATEMENT` denoted as s. s has the attribute `ID`, which is the statement identifier.

• The *Content* of `XDM:STATEMENT` is a non empty list of `XDM:SOURCE-ITEM` nodes (where each of them specifies an operator input), followed by an `XDM:OPERATOR` node (describing the application of the operator), followed by a non empty list of `XDM:OUTPUT-ITEM` nodes (that specify the operator output). □

Example 3: The following example shows the main features of a statement of `MR:MINE-RULE` (see [9] for a complete description).

```
<XDM:STATEMENT ID="00128" xmlns:MR="http://.../NS/XDM">
 <XDM:SOURCE-ITEM Role="RawData" Name="Purchases" Version="1"/>
 <XDM:OPERATOR>
  <MR:MINE-RULE xmlns:MR="http://.../NS/MINE-RULE">
   <MR:GROUPING select="TRANSACTIONS/PRODUCT" common-value="@TID"/>
    <MR:RULE-ELEMENT name="ITEM" select="@ITEM"/>
     <MR:MEASURES>
      <MR:SUPPORT threshold="0.4"/>
      <MR:CONFIDENCE threshold="0.75"/>
     </MR:MEASURES>
  </MR:MINE-RULE>
 </XDM:OPERATOR>
<XDM:OUTPUT-ITEM Name="Rules" Role="AssociationRules"
  Root="MR:RULE-SET" NS="http://.../NS/DATA/Rules"/>
</XDM:STATEMENT>
```

The operator and its specific element nodes are defined in the name space prefix `MR`. `SOURCE-ITEM` specifies the input of the operator providing `Name` and `Version` number of the XDM data item while `OUTPUT-ITEM` identifies the output node by means of the `Name` and the `Root` node of its tree fragment, as well as the name space in which the output format is defined (attribute `NS`). The `Role` attribute is exploited by the operator to distinguish the role of each data item w.r.t. the operator application.

The `MINE-RULE` operator analyzes the source item named `Purchases` (shown in Example 2) looking for association rules which associate values of attribute `ITEM` (see `MR:RULE-ELEMENT`); items are selected from elements `TRANSACTIONS/PRODUCT` logically grouped by the value of attribute `TID`, (see `MR:GROUPING`) since association rules must denote regularities w.r.t. single transactions. Finally, rules are extracted if their evaluation measures (support and confidence) are greater than the respective thresholds (see `MR:MEASURES`). □

Schema for XDM Statements. Statements are based on the XML Syntactic structure and, as well as XDM data items, it is possible to identify the concept of *schema*. Given an XDM statement s, the schema of s, denoted as $Schema(s)$, is a four-tuple $Schema(s) = \langle NameSpace, Prefix, Root, xsd \rangle$ where $NameSpace$ is the namespace URI associated to the operator application described in `XDM:STATEMENT`, $Prefix$ is the namespace prefix associated to the namespace URI, *root* is the root element of the XML fragment describing the operator application, *xsd* is the XML-Schema definition that defines the XML structure for the operator application belonging to the specified namespace URI. For example, the schema of the `MINE-RULE` statement of Example 3 is

⟨`"http://.../NS/MINE-RULE","MR","MINE-RULE","mr.xsd"` ⟩

It is important to note that data items and statements have the same concept of schema. This is important, because it shows that they are dual: data items and statements are really two faces of the same coin, i.e. the KDD process.

4 A Sample KDD Process

In the following we present a sample instance of a knowledge discovery process that involves the presented XDM data items and is instantiated by the excution of data management and data mining operators. This example will show how XML, and XDM in particular, is suitable for the inductive database framework and to provide effective support to the deployment of the KDD process. Figure 1 shows the sample KDD process: data items are rectangles; statements are circles labeled with operator name and `ID`; edges are labeled with input/output roles (written in italic).

In the figure, we find the application of the `MINE-RULE` operator on market basket data that produces the data item `Rules`. Then, another operator, namely `EVALUATE-RULE`, computes cross-references between rules and original data. It is discussed in detail in the Example 4, where it retrieves the list of customers that satisfy each association rule. Customers are then selected by a data manipulation operator, `SELECT-DATA-ITEM`, that performs selection on customers, according to the association rules they satisfy. Later selected customers are joined by operator `JOIN-DATA-ITEM` with the data item containing customers' details (named `Customers`). Finally, a clustering operator analyzes better the selected customers by clustering them on their detailed data. In the following examples we will discuss the operators and the data items generated.

Example 4: The operator `EVALUATE-RULE` retrieves the original data (stored in XDM data items) for which already extracted association rules (in other XDM data items) are satisfied. A complete description of this operator, in an SQL version for relational databases, can be found in [10].

The application of an `EVALUATE-RULE` statement is shown in Figure 1. This instance of the operator (identified by `ID="00133"` and reported in the following) takes in input two `XDM` data items: with the role of `RawData` it takes the first version of the data item named `Purchases`, and with the role of `AssociationRules`

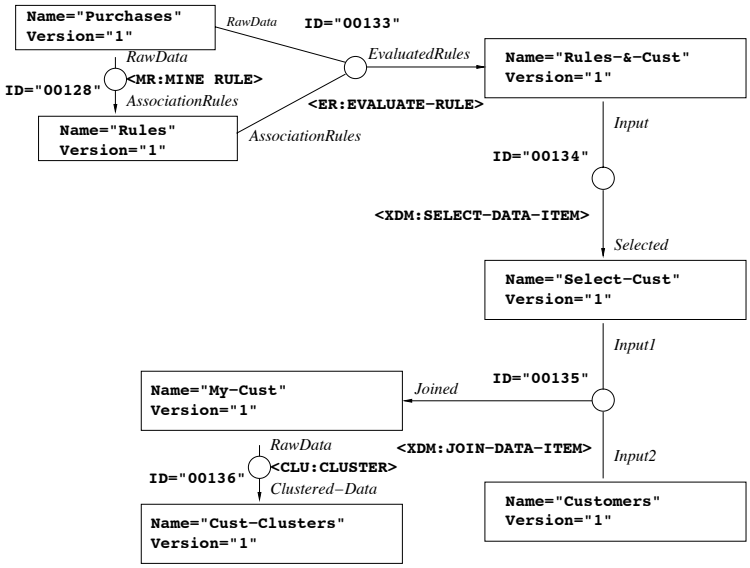

Fig. 1. A sample knowledge discovery process based on XDM.

the first version of the data item named `Rules`. It gives in output the first version of a new data item, named `Rules-&-Cust` containing for each association rule the list of customers for which it holds; this data item has the role `Evaluated Rules`. For lack of space we omit the details of the clauses in the operators.

```
<XDM:STATEMENT ID="00133" xmlns:ER="http://.../NS/XDM">
  <XDM:SOURCE-ITEM Role="RawData" Name="Purchases" Version="1"/>
  <XDM:SOURCE-ITEM Role="AssociationRules" Name="Rules" Version="1"/>
  <XDM:OPERATOR>
    <ER:EVALUATE-RULE xmlns:ER="http:.../NS/EVALUATE-RULE">
      . . . tags with operator clauses . . .
    </ER:EVALUATE-RULE>
  </XDM:OPERATOR>
  <XDM:OUTPUT-ITEM Name="Rules-&-Cust" Version='1'
   Virtual="NO" Role="EvaluatedRules" Root="DATA-AND-RULE-SET"
   NS="http://xdm.unito.it/NS/DATA/Data-With-Rules"/>
</XDM:STATEMENT>
```

Notice that the specific element nodes of the operator are defined in the namespace `ER:`, corresponding to the URI `"http:.../NS/EVALUATE-RULE"`, outside the standard XDM namespace.

The first `SOURCE-ITEM` node specifies the data item with the role of `RawData`; it is the data set over which rules are evaluated. The second `SOURCE-ITEM` node specifies the data item with the role of `AssociationRules`, i.e. the set of association rules to evaluate.

The operator produces a new data item, which contains the same rules extended with the list of customers for which each rule holds. The output data item is specified by the XDM:OUTPUT-ITEM. Evaluated association rules have a specific role (EvaluatedRules) and in the case of this statement will be materialized (attribute Virtual=NO). The name of the output node is defined by the Root attribute and its namespace (ERD) is defined by a NS specification. □

The data item produced by the previous statement is the following.

```
<XDM:DATA-ITEM Name="Rules-&-Cust" Version="1" Virtual="NO"
     Date="..." xmlns:XDM="http://.../NS/XDM">
 <XDM:DERIVATION Statement="00133"/>
 <XDM:CONTENT>
   <EAR:EVALUATED-ASSOCIATION-RULE-SET
     xmlns:AR="http://.../NS/DATA/EvAssRules">
      <EAR:RULE>
        <EAR:BODY>
          <EAR:ELEMENT Name="ITEM"> A </EAR:ELEMENT>
          <EAR:ELEMENT Name="ITEM"> B </EAR:ELEMENT>
        </EAR:BODY>
        <EAR:HEAD>
          <EAR:ELEMENT Name="ITEM"> C </EAR:ELEMENT>
        </EAR:HEAD>
        <EAR:SUPPORT value="0.5"/>
        <EAR:CONFIDENCE value="0.8"/>
        <EAR:EVALUATED-FOR>
          <EAR:ELEMENT Name="CUST"> c1 </EAR:ELEMENT>
          <EAR:ELEMENT Name="CUST"> c3 </EAR:ELEMENT>
        </EAR:EVALUATED-FOR>
      </EAR:RULE>
   </EAR:EVALUATED-ASSOCIATION-RULE-SET>
 </XDM:CONTENT>
</XDM:DATA-ITEM>
```

Notice that this data item is structurally similar to the one shown in Example 2, named Rules. In particular, it is defined on a different namespace associated with the prefix EAR; furthermore, for each rule an element named EAR:EVALUATED-FOR is added, which contains the set of customers for which the rule holds (notice that each customer is described by an occurrence of element EAR:ELEMENT, whose attribute Name specifies the feature whose value is reported in the content).

Example 5: We want to select now the customer identifiers that satisfy a particular association rule of interest, such as {A, B}⇒ {C}, shown in Example 2. We perform a selection operation on Rules-&-Cust data item. The operator SELECT-DATA-ITEM, in the XDM namespace, makes use of the SOURCE-ITEM and OUTPUT-ITEM nodes for the specification of the input and output, while the OPERATOR node introduces the kind of operator. Notice the roles for the input

and output data items. The SELECT-DATA-ITEM statement follows, for which we omitted the details for lack of space.

```
<XDM:STATEMENT ID="00134" xmlns:XDM="http://.../NS/XDM">
  <XDM:SOURCE-ITEM Role="Input" Name="Rules-&-Cust" Version="1"/>
  <XDM:OPERATOR>
     <XDM:SELECT-DATA-ITEM>
      . . . tags with operator clauses . . .
     </XDM:SELECT-DATA-ITEM>
  </XDM:OPERATOR>
  <XDM:OUTPUT-ITEM Name="Select-Cust" Role="Selected" Version='1'
   Virtual="NO" Root="CONTENT"
   NS="http://xdm.unito.it/NS/XDM"/>
</XDM:STATEMENT>
```

The output of this SELECT-DATA-ITEM statement is the first version of the data item named Select-Cust. It will be materialized and will contain the list of customer identifiers in EAR:ELEMENT nodes under the root node CONTENT. The role of the data item puts in evidence that this object is a selected version of the input. □

Example 6: Consider the XDM data item storing the details of customers.

```
<XDM:DATA-ITEM Name="Customers" Version="1"
     Date="..." Virtual="NO" xmlns:XDM="http://.../NS/XDM">
 <XDM:CONTENT>
 <CUST ID="c1" SALARY="53.000" AGE="56" SEX="M" COUNTRY="FL"/>
 <CUST ID="c2" SALARY="46.500" AGE="32" SEX="M" COUNTRY="VA"/>
 <CUST ID="c3" SALARY="60.000" AGE="44" SEX="F" COUNTRY="CA"/>
      .  .  . Other customers  . . .
 </XDM:CONTENT>
</XDM:DATA-ITEM>
```

Now we would like to better analyze the customers that satisfy the association rule of interest and whose identifiers are listed in Select-Cust. In particular we would like to perform a clusterization step that groups similar customers according to their details. Therefore, we retrieve the details of selected customers by joining Select-Cust with the data-item Customers. We use a data manipulation operator named JOIN-DATA-ITEM, shown in the following, which combines the element nodes in the first source data item with the element nodes in the second data item specified by the SOURCE-ITEM nodes. The element nodes from the first and second source items are retrieved according to a select condition that consists in an XPath expression. Its application results in a set of valid node pairs included under the first version of the output data item named My-Cust under the root node CONTENT. In the following the details of the condition are omitted for lack of space.

```
<XDM:STATEMENT ID="00135" xmlns:XDM="http://.../NS/XDM">
  <XDM:SOURCE-ITEM Role="Input1" Name="Select-Cust" Version="1"/>
  <XDM:SOURCE-ITEM Role="Input2" Name="Customers" Version="1"/>
```

```
  <XDM:OPERATOR>
     <XDM:JOIN-DATA-ITEM>
      . . . tags with operator clauses . . .
     </XDM:JOIN-DATA-ITEM>
  </XDM:OPERATOR>
  <XDM:OUTPUT-ITEM Name="My-Cust" Role="Joined"
   Version='1' Virtual="NO" Root="CONTENT"
   NS="http://xdm.unito.it/NS/XDM"/>
</XDM:STATEMENT>
```

The ouput of this `JOIN-DATA-ITEM` statement, under the `XDM` namespace, has role `Joined` that puts in evidence the nature of the output whose content is determined by the operator inputs. Follows the new data item, named `My-Cust`:

```
<XDM:DATA-ITEM Name="My-Cust" Version="1"
     Date="..." Virtual="NO" xmlns:XDM="http://.../NS/XDM">
  <XDM:DERIVATION Statement="00135"/>
  <XDM:CONTENT>
    <EAR:ELEMENT Name="CUST"
     xmlns:AR="http://.../NS/DATA/EvAssRules> c1 </ELEMENT>
    <CUST ID="c1" SALARY="53.000" AGE="56" SEX="M" COUNTRY="FL"/>
    <EAR:ELEMENT Name="CUST"
     xmlns:AR="http://.../NS/DATA/EvAssRules> c3 </ELEMENT>
    <CUST ID="c3" SALARY="60.000" AGE="44" SEX="F" COUNTRY="CA"/>
       . . . Other customers . . .
  </XDM:CONTENT>
</XDM:DATA-ITEM>
```

Notice the `DERIVATION` node referring to the statement that generated this derived data item. Notice also the customer identifier in `EAR:ELEMENT` that comes from evaluated rules generated by `EVALUATE-RULE`. It is still in the namespace `EAR` of that operator, since it could still be checked and modified by that operator. □

Example 7: Finally, to conclude our sample knowledge discovery process by XDM, we would like to perform a clusterization step that groups similar customers according to their details. A clustering operator is called:

```
<XDM:STATEMENT ID="00136" xmlns:CLU="http://.../NS/CLU">
  <XDM:SOURCE-ITEM Role="RawData" Name="My-Cust" Version="1"]"/>
  <XDM:OPERATOR>
     <CLU:CLUSTER>
      . . . tags with operator clauses . . .
     </CLU:CLUSTER>
  </XDM:OPERATOR>
  <XDM:OUTPUT-ITEM Name="Cust-Clusters" Role="Clustered-Data"
   Version='1' Virtual="NO" Root="CONTENT" NS="http://.../XDM"/>
</XDM:STATEMENT>
```

This operator has the namespace `CLU` for clustering. It specifies the source data item with the role `RawData` and the ouput one with a specific role. □

5 XDM Database Schema and State

Defined the two basic XDM concepts, we can formally define the concepts of XDM *database schema* and XDM *database state*. They help us to define some integrity constraints over the possible inputs and outputs of a given statement, or, dually, over the possible statements that can be applied to a given data item. These are meta data on the KDD process that can be exploited by querying the schema of the XDM database to check (automatically by the system or explicitly by the user) the consistence among the operations performed over the data.

Definition 5: The *schema of an XDM database* is a 4-tuple $\langle \overline{S}, \overline{I}, \overline{In}, \overline{Out} \rangle$, where $\overline{S}$ is a set of statement schemas, and $\overline{I}$ is a set of data item schemas.

$\overline{In}$ is a set of tuples $\langle Operator, InputRole, InputFormat \rangle$, where $Operator$ is an operator whose schema is described by a tuple in S (in the form $prefix : root$); $InputRole$ is the role expected by the operator for the input data; $InputFormat$ is a data item content root (whose schema is described by a tuple in I) allowed for the role (if the operator does not require any particular data format for the specified role, $InputFormat$ is $*$).

$\overline{Out}$ is a set of tuples $\langle Operator, OutputRole, OutputFormat \rangle$ where $Operator$, $OutputRole$ and $OutputFormat$ are analogously defined. □

Example 8: With reference to the KDD scenario described in previous examples, this is the schema of our database.

$\overline{In}$=$\{$⟨`MR:MINE-RULE, RawData,*`⟩, ⟨`ER:EVALUATE-RULE, RawData,*`⟩,
⟨`ER:EVALUATE-RULE, AssociationRules, AR:ASSOCIATION-RULE-SET`⟩,
⟨`XDM:SELECT-DATA-ITEM, Input,*`⟩,⟨`XDM:JOIN-DATA-ITEM, Input1,*`⟩,
⟨`XDM:JOIN-DATA-ITEM, Input2,*`⟩,⟨`CLU:CLUSTER, RawData,*`⟩$\}$

$\overline{Out}$=$\{$⟨`MR:MINE-RULE, AssociationRules,AR:ASSOCIATION-RULE-SET`⟩,
⟨`ER:EVALUATE-RULE, EvaluatedRules,ERD:DATA-AND-RULE-SET`⟩,
⟨`XDM:SELECT-DATA-ITEM,Selected,*`⟩,⟨`XDM:JOIN-DATA-ITEM,Joined,*`⟩,
⟨`CLU:CLUSTER, Clustered-Data, CLU:CLUSTERS`⟩$\}$ □

Definition 6: The state of an XDM database is represented as a pair

$$\langle DI : Set\ Of(DataItem), ST : Set\ Of(Statement) \rangle$$

where DI is a set of XDM data items (see Definition 1), and ST is a set of XDM statements (see Definition 4). The following constraints hold.

• *Data Item Identity.* Given a data item d and its mandatory attributes `Name`, and `Version`, the pair ⟨`Name`, `Version`⟩ uniquely identifies the data item d in the database state.

• *Statement Identity.* Given a statement s and its mandatory attribute `ID`, its value uniquely identifies the statement s in the database state.

• *Relationship between statements and source data items.* Consider an XDM statement s. The attributes `Name` and `Version` of each `XDM:SOURCE-ITEM` appearing in s must denote one and only one XDM data item.

State	*DI*	*ST*
S_0	{⟨Purchases,1⟩},⟨Customers,1⟩}	∅
S_1	{⟨Purchases,1⟩,⟨Customers,1⟩,⟨Rules,1⟩}	{00128}
S_2	{⟨Purchases,1⟩,⟨Customers,1⟩,⟨Rules,1⟩,⟨Rules-&-Cust,1⟩}	{00128,00133}
S_3	{⟨Purchases,1⟩,⟨Customers,1⟩,⟨Rules,1⟩,⟨Rules-&-Cust,1⟩, ⟨Select-Cust,1⟩}	{00128,00133, 00134}
S_4	{⟨Purchases,1⟩,⟨Customers,1⟩,⟨Rules,1⟩,⟨Rules-&-Cust,1⟩, ⟨Select-Cust,1⟩,⟨My-Cust,1⟩}	{00128,00133, 00134,00135}
S_5	{⟨Purchases,1⟩,⟨Customers,1⟩,⟨Rules,1⟩,⟨Rules-&-Cust,1⟩, ⟨Select-Cust,1⟩,⟨My-Cust,1⟩, ⟨Cust-Clusters,1⟩}	{00128,00133, 00134,00135, 00136}

Table 1. Database states for the example of Figure 1

• *Relationship between derived data items and statements.* Consider a derived XDM data item d. The value specified by the `Statement` attribute of the `XDM:DERIVATION` element must identify one and only one XDM data item. □

Example 9: With reference to the KDD process described in Figure 1 the database has moved between five states that are reached after the application of each statement, to some of the data items in `DI`. After each statement execution, the statement identifier is added to `ST` and the output data items are added to `DI`. □

Observe that an XDM database is both a data item base and a statement base. When a new statement is executed, the new database state is obtained by adding both the executed statement and the new data item. This structure represents the two-fold nature of the knowledge discovery process: data and patterns are not meaningful if considered in isolation; in contrast, patterns are significant if the overall process is described, because their meaning is clarified by the data mining operators that generated them. Considering this, the patterns representation provided by PMS [12] does not seem to be so specifically defined.

6 Implementation of a Prototype of the XDM System

We implemented a prototype based on the XDM framework. This prototype demonstrated the feasibility of the approach and gave us useful indications to study practical problems related with extensibility issues and performance issues.

The XDM System is fully realized in Java, and is based on open source components only. The architecture of the XDM System (Figure 2) is organized in four overlapped layers, each of them hiding the lower layers to the upper ones.

The top-most components are The *User Interface* and the *XDM API* allow to interact with the XDM System: the *XDM API* is used by applications; the *User Interface* is used in interactive sessions with the system.

The second layer is constituted by the *XDM Manager*, and by *Operators*, i.e. components which implement data management or data mining operators.

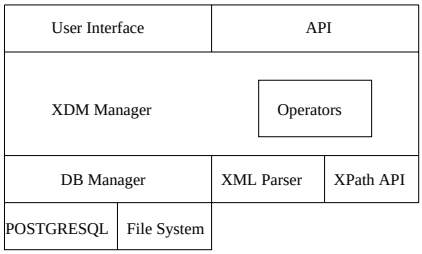

Fig. 2. XDM System Architecture.

XDM Manager interprets statements coming from interfaces, activates execution of tools, exploits *DB Manager* to access and store both meta data and data items. *Operators*, are responsible to implement the actual semantics given to the operators; they can interact with the system through an API provided by *XDM Manager*. This embedding is beneficial because it provides an inner and immediate compatibility and security check on which operations are allowed by operators. This is a fundamental feature of an open system where new operators are allowed to be added freely by users at any time.

XDM Manager exploits components in the third layer: these are *DB Manager*, *XML Parser* and *XPath API* components; in particular, since both *XML Parser* and *XPath API* might be used by *Operators* for reading data items, *XDM Manager* provides a controlled access to these components (in the sense that these latter components can be exploited by various tools in *Operators*). For both *XML Parser* and *XPath API* we adopted `xerces` XML Parser and the XPath API library available in the `xalan` XSLT processor (open source implementations developed by the *Apache Software Foundation*).

DB Manager encapsulates all data management operations. In particular, it currently exploits `POSTGRESQL` DBMS to manage the meta-schema of the XDM framework, and the file system to store unstructured and semi-structured data items. This latter choice is motivated by efficiency reasons. However, we plan to study the integration of an XML DBMS in *DB Manager*.

In the current version, operators read XDM items by a SAX parser, which has the advantage of avoiding the construction and storage of the DOM tree corresponding to their XML structure. For the future, we also plan the development of a fast XPath interpreter on top of the parser so that only the relevant XML fragments are identified and sent through the channel (avoiding to send the entire XML document). Another possible solution is the usage of XML compressors. Furthermore, we also plan to investigate the problem of getting data items from different data sources, such as relational databases or native XML database. Finally, we plan to evolve XDM system into a distributed, grid like, system, where both distributed data sources and distributed computational sources are connected through the Internet.

7 Conclusions

In this paper we presented an XML-based data model, named XDM. It is designed to be adopted inside the framework of inductive databases. XDM allows the management of semi-structured and complex patterns thanks to the semi-structured nature of the data that can be represented by XML.

In XDM the pattern definition is represented together with data. This allows the reuse of patterns by the inductive database management system. In particular, XDM explicitly represents the statements that were executed in the derivation process of the pattern. The flexibility of the XDM representation allows extensibility to new pattern models and new mining operators: this makes the framework suitable to build an open system, easily customized by the analyst. We experimented the XDM idea by means of a system prototype that resulted to be easily and quickly extendible to new operators.

References

1. P. Alcamo, F. Domenichini, and F. Turini. An xml based environment in support of the overall kdd process. In *FQAS*, 2000.
2. P. V. Biron and A. Malhotra. Xml schema part 2: Data types. Technical Report REC-xmlschema-2-20010502, World Wide Web Consortium, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/, May 2001.
3. D. Braga, A. Campi, S. Ceri, M. Klemettinen, and P. L. Lanzi. Discovering interesting information in xml data with association rules. In *ACM SAC*, 2003.
4. A.G. Buchner and M.Baumgarten. Data mining and xml: Current and future issues. *Int. Conf. on Web Information Systems Engineering*, 2000.
5. B. Catania, M. Maddalena, Mazza, E. Bertino, and S. Rizzi. A framework for data mining pattern management. In *Proc. of ECML-PKDD 2004*, Italy, Sept. 2004.
6. DMG. The pmml language. *http://www.dmg.org/pmml-v2-0.htm*.
7. A. N. Edmonds. Xmlminer, xmlrule and metarule white paper. Technical report, Scientio, Inc., http://www.kdnuggets.com/news/2001/n14/7i.html, 2002.
8. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Coomunications of the ACM*, 39(11):58–64, November 1996.
9. R. Meo, G. Psaila, and S. Ceri. An extension to SQL for mining association rules. *Journal of Data Mining and Knowledge Discovery*, 2(2), 1998.
10. G. Psaila. Enhancing the kdd process in the relational database mining framework by quantitative evaluation of association rules. In *Knowledge Discovery for Business Information Systems*. Kluwer Academic Publisher, January 2001.
11. S. Rizzi. Uml-based conceptual modeling of pattern-bases. In *Proc. of PaRMA'04*.
12. 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.
13. H: S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. Xml schema part 1: Structures. Technical Report REC-xmlschema-1-20010502, World Wide Web Consortium, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/, May 2001.