A Tool for Extracting XML Association Rules from XML Documents

Daniele Braga[1], Alessandro Campi[1], Stefano Ceri[1], Mika Klemettinen[2], Pier Luca Lanzi[1*]

[1]Dipartimento di Elettronica e Informazione
Politecnico di Milano
P.zza Leonardo da Vinci
I-20133 Milano, Italy
{braga,campi,ceri,lanzi}@elet.polimi.it

[2]Nokia Research Center
Nokia Group, P.O.Box 407
FIN-00045 Nokia Group, Finland
mika.klemettinen@nokia.com

Abstract

The recent success of XML as a standard to represent semi-structured data, and the increasing amount of available XML data, pose new challenges to the data mining community. In this paper we present the XMINE *operator a tool we developed to extract XML association rules for XML documents. The operator, that is based on XPath and inspired by the syntax of XQuery, allows us to express complex mining tasks, compactly and intuitively.* XMINE *can be used to specify indifferently (and simultaneously) mining tasks both on the content and on the structure of the data, since the distinction in XML is slight.*

1. Introduction

Knowledge discovery in databases (KDD) deals with the process of extracting *interesting* knowledge from large amounts of data, usually stored in large (relational) databases. Knowledge can be represented in many different ways (clusters, decision trees, decision rules, etc.). Among the others, association rules [4] proved effective in discovering interesting relations in massive amounts of *relational* data. The recent years have seen the dramatic development of the eXtensible Markup Language (XML) [12] as a major standard for storing and exchanging information. XML enables the self-description of hierarchies of semi-structured information, by intermixing data content with semantic tags which describe such content. At the moment, the use of XML within the data mining community is still quite limited; there are some proposals to exploit the XML syntax to express interfaces to knowledge representation artifacts, including association rules, so that they can be easily exchanged among data mining tools (e.g., PMML [8]); but there are no significant extensions of data mining research taking full advantage of the intrinsic properties of XML. However, it is easy to foresee that the spreading of XML will cause an increasing interest on this subject, going beyond a mere syntactic adaptation to XML of data mining artifacts and techniques.

In this paper, we present the XMINE operator, a tool that can be used to extract association rules from *native* XML documents, shortly "XML association rules", which we first introduced in [6, 5]. The paper is organized as follows. In Section 2 we overview association rules in the context of relational databases. In Section 3 we shortly discuss the notion of association rules for XML while we refer the readers to [6, 5] for additional details about the their theoretical foundations. In Section 4 we present the XMINE operator through a serie of intuitive examples. In Section 5 we introduce some basic concepts needed to discuss implementation details. In Section 6 we discuss how XML association rules are extracted from an XML document through XMINE by composing an execution environment for XPath expressions and an algorithm for discovering frequent itemsets. In Section 7 we give some implementation details discussing the current state of the prototype we developed and an outline of the planned future development. We conclude the paper with a discussion of future research directions.

2. Relational Association Rules

Association rules [4] are implications of the form $X \Rightarrow Y$ where the rule *body* X and the rule *head* Y are subsets of the set $\mathcal{I}$ of *items* ($\mathcal{I} = \{I_1, \ldots, I_n\}$) within a set of *transactions* $\mathcal{D}$. The rule $X \Rightarrow Y$ states that the transactions $T \in \mathcal{D}$ which contain the items in X ($X \subset T$) are *likely* to contain also the items in Y ($Y \subset T$). Association rules are characterized by two measures: the *support*,

*Artificial Intelligence and Robotics Laboratory (AirLab) http://airlab.elet.polimi.it

A	B	C
1	0	0
1	1	1
1	0	1
0	1	1

(a)

rules	*supp.*	*conf.*
$A \Rightarrow B$	0.25	0.33
$C \Rightarrow A$	0.5	0.66
$AB \Rightarrow C$	0.25	1.00
$BC \Rightarrow A$	0.25	0.50
...	...	...

(b)

Figure 1. An example of source data (a) and association rules generated from the data (b).

which measures the percentage of transactions that contain both items X and Y; the *confidence*, which measures the percentage of transactions that contain the items Y within the transactions that also contain the items in X. More formally, given the function $freq(X, \mathcal{D})$, denoting the percentage of transactions in $\mathcal{D}$ which contains X, we define:

$$\begin{aligned} support(X \Rightarrow Y) &= freq(X \cup Y, \mathcal{D}) \\ confidence(X \Rightarrow Y) &= freq(X \cup Y, \mathcal{D}) / freq(X, \mathcal{D}) \end{aligned}$$

Suppose there is an association rule "*bread,butter* $\Rightarrow$ *milk*" with confidence 0.9 and support 0.05. The rule states that customers who buy *bread* and *butter*, also buy *milk* in 90% of the cases and that this rule holds in 5% of the transactions.

The problem of mining association rules consists of finding all the association rules, from a set of transactions $\mathcal{D}$, that have support and confidence greater than the two defined thresholds: the minimum support (*minsupp*) and the minimum confidence (*minconf*). Association rules assume a relational representation of data (e.g., [4]), in which the transactions $T \in \mathcal{D}$ correspond to tuples, while the items $I \in \mathcal{I}$ correspond to binary attributes. For instance, a rather typical although simple example of source data is depicted in Figure 1(a). The attributes A, B, C represent the set of items $\mathcal{I}$; the tuples represent the transactions, in particular the first tuple $\langle 1, 0, 0 \rangle$ represents a transaction with only the item A; the last tuple $\langle 0, 1, 1 \rangle$ represents a transaction with both the items B and C. Some association rules extracted from the data in Figure 1(a) are shown in Figure 1(b). For instance, the first rule $A \Rightarrow B$ states that in 25% of the tuples, item A and item B appear together and that the 33% of tuples which contain A contain *also* B. To support the specification of complex association rule mining tasks, a number of query languages have been proposed. In particular, [10] introduced the `MINE RULE` operator, an extension of SQL specifically designed for modeling the problem of mining association rules from relational data.

```
<DEPARTMENT>
 <PhDCourses>
  <Course teacher="fp1" title="Advanced...">
   <TimeTable>...</TimeTable>
   <Student ref="ps1" />
   <Student ref="ps2" />
  </Course>
  <Course teacher="fp3" title="XML...">
   <TimeTable>...</TimeTable>
   <Student ref="ps2" />
   <Student ref="ps3" />
  </Course>
 </PhDCourses>
 <People>
  <PhDStudent id="ps2" advisor="fp3">
   <PersonalInfo email="fp3@atlantis.edu">
    <Name>...</Name>
   </PersonalInfo>
   <Subscription year="2001" />
   <Publications> ... </Publications>
  </PhDStudent>
  <FullProfessor id="fp3">
   <PersonalInfo email="fp3@atlantis.edu">
    <Name> ... </Name>
   </PersonalInfo>
   <Publications>
    <Article title="Golden Data Mines...">
     <Author>...</Author>
     <Conference name="VLDB" year="2001" />
    </Article>
    <Article title="P=NP - The Final Chapter">
     <Author>...</Author>
     <Journal year="2000" month="4" volume="4"
              name="DMKD" publisher="Kluwer" />
    </Article>
    <Book year="2001" title="XML ...">
     <Author>...</Author>
     <Publisher>...</Publisher>
     <Keyword>XML</Keyword>
       ...
     <Keyword>XQuery</Keyword>
    </Book>
   </Publications>
   <Award year="2001" society="IEEE">...</Award>
  </FullProfessor>
 </People>
</DEPARTMENT>
```

Figure 2. A sample document containing various information about the research activities of a university department (`http://www.atlantis.edu/research.xml`).

3. Association Rules for XML Documents

In this section we overview the basic concepts of association rules for XML documents, shortly "XML association rules" [6, 5]; we refer the interest reader to [6, 5] for further details. Let us consider the XML document depicted in Figure 2 where various information about a department are represented, e.g., the available Ph.D. courses (identified by the tag `<PhDCourse>`) and the people in the department (`<People>`). These can be either students (`<PhDStudent>`) or professors (`<FullProfessor>`). For each of them, some personal information are stored (`<PersonalInfo>`) as well as the list of works published

(<Publications>), e.g., books (<Book>), journal papers, or conference papers (<Article>). Consider the problem of mining frequent associations among people that appear as coauthors in the publications represented in the document research.xml in Figure 2. In practice, we are interested in finding associations of the form:

"*Wilson* ⇒ *Holmes*"

which states that, in the department, the papers which are authored by *Wilson* are also likely to have *Holmes* as author. Since any part of an XML document is a tree, usually called XML *fragment*, both $\mathcal{D}$ and $\mathcal{I}$ are collections of trees. In particular, the transactions $T \in \mathcal{D}$ are the XML fragments that define the *context* in which the items $I \in \mathcal{I}$ must be counted; the *items* $I \in \mathcal{I}$ are the fragments that must be counted. If we consider the problem of mining association rules among authors appearing in the same papers, we have that $\mathcal{D}$ are the set of trees (i.e., *fragments*) of all the publications authored by people within the department. The items $I \in \mathcal{I}$ are the set of *fragments* of all the authors who appear in the works published by people within the department. This situation is depicted in Figure 3 where an example of the research.xml document is represented. The overall document is represented by the *grey triangle*. The *white triangles* represent the *fragments* corresponding to the various publications authored by people of the department. The *black triangles* represent the *fragments* corresponding to the authors who appear in various publications. Note that, the *value* of the <Author> tag is depicted at the bottom of the black triangle, thus a black triangle labeled Wilson is equivalent to the XML fragment <Author>Wilson</Author>. While the *value* of a white triangle, corresponding to a <Book> tag or to a <Article> tags, represents an entire XML fragment which corresponds to a book or to an article, e.g.:

```
<Article title="Classifier Fitness ...">
 <Author>Wilson</Author>
 <Journal year="1995" month="4" volume="4"
          name="Evolutionary Computation"
          publisher="MIT Press" />
</Article>
```

In the example in Figure 3, $\mathcal{D}$ contains the four white triangles and $\mathcal{I}$ contains the the three black triangles corresponding to the authors *Holmes*, *Stolzmann*, and *Wilson*. While an itemset $X \subset \mathcal{I}$ is a set containing black triangles. It is now possible to compute $freq(X, \mathcal{D})$ as the percentage of XML fragments in $\mathcal{D}$ which contains the fragments in X. In particular, $freq(X, \mathcal{D})$ is computed as the percentage of white triangles which contain all the *black triangles* in X. Thus, in the example depicted in Figure 3, $freq(\{\texttt{Wilson},\texttt{Holmes}\},\mathcal{D})$ is 0.5 since the fragments <Author>Wilson</Author>

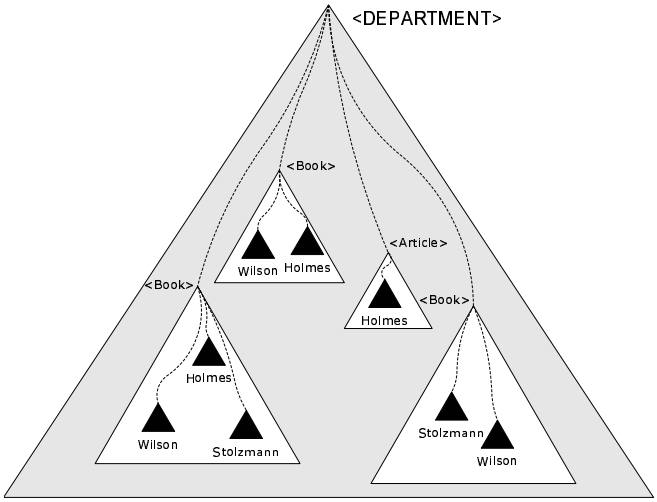

Figure 3. A graphical representation of the XML document described in Figure 2.

and <Author>Holmes<Author> appear together in two white triangles out of four, i.e., in 50% of the cases. Likewise, $freq(\{\texttt{Wilson}\},\mathcal{D})$ is 0.75 since the fragment <Author>Wilson</Author> appears in three white triangles out of four, i.e., in 75% of the cases. Therefore we can now compute the *confidence* of the (XML) association rule:

Wilson ⇒ Holmes

as:

$$\frac{freq(\{\texttt{Wilson}, \texttt{Holmes}\}, \mathcal{D})}{freq(\{\texttt{Wilson}\}, \mathcal{D})}$$

which returns 0.66, i.e., in 66% of papers published by *Wilson* appears also *Holmes* or, in the XML context, in 66% of the fragments in $\mathcal{D}$ in which appears the fragment <Author>Wilson</Author> appears also the fragment <Author>Holmes</Author>. While the *support* of the association rule is computed as:

$$freq(\{\texttt{Wilson},\texttt{Holmes}\},\mathcal{D})$$

which returns 0.5 since <Author>Wilson</Author> appears together with <Author>Holmes</Author>, in two white triangles out of four, i.e., in 50% of the cases.

4. The XMINE Operator

We devised XMINE operator as a tool to specify complex association rule mining tasks in XML documents. XMINE allows the specification of complex mining task

compactly and intuitively, and puts in evidence the most relevant semantic features of the problem. XMINE is based on XPath [13], inspired to the syntax of XQuery, and to the work on MINE RULE [10] in the context of relational databases. Following the approach of [10], we present XMINE through a set of examples; additional examples, as well as the XMINE formal syntax and the theoretical foundations of XML association rules, can be found in [6, 5].

Example 1. In this first example, we consider the problem of mining frequent associations among people who appear as coauthors (see Section 2). With XMINE we formulate this task with the following statement.

```
XMINE RULE
IN document("www.atlantis.edu/research.xml")
FOR ROOT IN //People/*/Publications/*
LET BODY := ROOT/Author,
    HEAD := ROOT/Author
EXTRACTING RULES WITH
     SUPPORT = 0.1 AND CONFIDENCE = 0.2
RETURN
<RULE support={ SUPPORT }
      confidence={ CONFIDENCE }>
 <BODY> { FOR $item IN BODY
          RETURN <Item> { $Item } <Item> }
 </BODY>

 <HEAD> { FOR $item IN HEAD
          RETURN <Item> { $Item } <Item> }
 </HEAD>
</RULE>
```

The statement begins with the IN clause which specifies the data source, i.e., the document found at the URL http://www.atlantis.com/research.xml. In the FOR section, the special variable ROOT specifies the fragments which represent the transactions $T \in \mathcal{D}$ by means of one (or more) XPath expressions. In the example in Figure 3, ROOT identifies the white triangles. The next two special variables, BODY and HEAD, identify the fragments in the source data which should appear respectively in the rule body and in the rule head, i.e., the set of items $I \in \mathcal{I}$. Note that both BODY and HEAD *are* defined with respect to the *context* (i.e., to the ROOT variable) since confidence and support values have meaning only with respect to the *context* defined by the ROOT (i.e., the set of transactions $\mathcal{D}$). More specifically, BODY and HEAD are defined as Author elements appearing in some Publications. Since the Author element appears both in articles and in books, Author is specified as a subelement of Publications/*. The EXTRACTING RULES WITH clause specifies *constraints* on the resulting rules. In the above statements, these constraints are very simple since they only specifies the minimum support and the minimum confidence values for the output association rules with the clauses, "SUPPORT=0.1" and "CONFIDENCE=0.2". The final RETURN clause specifies the structure of the association rules produced. An association rule is defined by a RULE tag, with two attributes, support and confidence which specify the support and confidence values; and two subelements, <BODY> and <HEAD>, which specify the items in the rule body and in the rule head. Some of the rules produced by the above statement on the data depicted in Figure 2 are the following:

```
<RULE support="0.25" confidence="0.33">
  <BODY>
    <Item> <Author>Holmes</Author> </Item>
  </BODY>
  <HEAD>
    <Item> <Author>Stolzmann</Author> </Item>
  </HEAD>
</RULE>
 ...

<RULE support="0.25" confidence="0.50">
  <BODY>
    <Item> <Author>Stolzmann</Author> </Item>
    <Item> <Author>Wilson</Author> </Item>
  </BODY>
  <HEAD>
    <Item> <Author>Holmes</Author> </Item>
  </HEAD>
</RULE>
```

Where, the latter RULE fragment represents the rule with <Author>Stolzmann</Author> and <Author>Wilson</Author> as body, and <Author>Holmes</Author> as head; the rule has support 0.25 and confidence 0.5. Note that in the remaining, we will omit the RETURN clause from the examples, since the one discussed above can be applied to all of them. In addition, XMINE assumes that whenever RETURN is missing, the output association rules will be produced according to the PMML 2.0 specification [8] .

Example 2. In the second example, we require that at least one of the authors in the body of the generated rules is a member of the department. This means that an additional *constraint* must be added to the EXTRACTING RULES WITH clause used in the previous example. In particular, we add an XQuery predicate requiring that at least one of the authors' names (denoted by the $a variable) be contained in the PersonalInfo element, denoting the members of the department. In addition, we add a *filtering* condition, with an appropriate WHERE clause, so to restrict the source data to the publications printed in 2001. The resulting statement is the following.

```
XMINE RULE
IN document("www.atlantis.edu/research.xml")
FOR ROOT IN //People/*/Publications/*
LET BODY := ROOT/Author,
```

```
    HEAD := ROOT/Author
WHERE ROOT//@year = 2001
EXTRACTING RULES WITH
SUPPORT = 0.1 AND CONFIDENCE = 0.2 AND
SOME $a IN BODY SATISFIES
not(empty(//People/*/PersonalInfo/Name[=$a]))
```

The WHERE clause has an obvious side effect on the generated rules, since all the publications that do not satisfy the condition are pruned before rule generation. Therefore, the resulting context is a subset of the previous one, and both support and confidence take different values. Note that the WHERE clause is just a *filtering* condition, *not* a constraints. This means that the WHERE clause must be considered before the association rules are generated.

Example 3. So far we presented rules in which the head and body fragment sets are defined by the same path expressions, and thus pertain to the same domain. The next example shows instead rules mining collaborations extracting authors when they publish papers on the same topic. This means that the rule HEAD will be extracted from the Author elements, and the rule BODY from the Keyword elements. A proper association between authors and keywords imposes that the ROOT be set as /People/*/Publications/* (all publications).

```
XMINE RULE
IN document("www.atlantis.edu/research.xml")
FOR ROOT IN /People/*/Publications/*
LET BODY := ROOT/Author,
    HEAD := ROOT/Keyword
EXTRACTING RULES WITH
    SUPPORT = 0.1 AND CONFIDENCE = 0.2 AND
    count(BODY) >= 2
```

Since the intrinsic meaning of the task is to find collaborations, an obvious restriction to the generated rules is that they should contain at least two authors in the body, and this is expressed by the generic XQuery predicate count(BODY)>=2, in addition to predicates defining the minimum support and confidence.

Example 4. The next example shows the use of several path expressions. We want to discover the associations between publications and awards, looking for which type of publication is mostly related to which kind of award. Since the publishers are represented as attributes in Journal elements and as PCDATA content in Books, while the conference types are stored as name attributes, the BODY fragment set is the *set union* of the nodes which three different path expressions evaluate to.

```
XMINE RULE
IN document("www.atlantis.edu/research.xml")
FOR ROOT IN /People/*
LET BODY := ROOT/Publications/Book/Publisher,
ROOT/Publications/Article/Journal/@publisher,
ROOT/Publications/Article/Conference/@name,
    HEAD := ROOT/Award/@society,
EXTRACTING RULES WITH
    SUPPORT = 0.1 AND CONFIDENCE = 0.2
```

Example 5. In XML, there is a slight distinction between the document data and the document structure. Accordingly, with XMINE we can intermix data and structural information in the same mining task. As an example, consider the problem of discovering which kinds of awards may influence the career, i.e., best correlates with the specific position of the members of the department. This mining task can be expressed through the following statement.

```
XMINE RULE
IN document("www.atlantis.edu/research.xml")
FOR ROOT IN /People/*
LET BODY := ROOT/Award/@society,
    HEAD := ROOT/name()
EXTRACTING RULES WITH
    SUPPORT = 0.1 AND CONFIDENCE = 0.2
```

Note that the the positions "covered" by the members are expressed as different *tagnames* (that are related to the document structure), and not as regular data. In order to compactly denote all career of a Department's member, the HEAD definition exploits the XPath name() step, that extract the tagname of the items to which it is applied. This feature is particularly relevant when the mining activity is addressed arbitrary tags in arbitrary positions, e.g., when documents describe the same application domain with different DTDs.

Additional Features. There are other features of XMINE that are not discussed here. Probably, the most relevant is the GROUP BY clause that can be use to restructure the source data when these do not allow the specification of adequate rule context [5]. We refer the reader to [6, 5] for further details.

5. Support and Confidence for XML Association Rules

Before we can discuss how the first prototype of XMINE has been implemented we must define how the support and the confidence values of an XML association rule are computed. Let X_R be the set of XPath expressions specified in the ROOT section; X_B be the set of XPath expressions specified in the BODY section; X_H be the set of XPath expressions specified in the HEAD section; $X_\mathcal{I}$ be $X_B \cup X_H$. In particular, let us focus on the simple case in which the BODY and the HEAD sections have identical sets of XPath expressions, $X_\mathcal{I} = X_B = X_H$, as in the initial examples in Section 4.

Let *value* be a function that, given an XPath expression p and an XML document d returns the set of XML fragments in d identified by p. Let *satisfy* be a function that, given an XML fragment f and an XQuery where clause w, returns one if f satisfies w, zero otherwise. Let *contains* be a function that, given an XML fragment x and an XML fragment y, returns one if x contains y, zero otherwise. First, we compute the set $F_{\mathcal{D}}$ as the collection of all the XML fragments defined by the XPath expressions in X_R, more formally:

$$F_{\mathcal{D}} = \bigcup_{p \in X_R} value(p, d)$$

Likewise, we compute the set $F_{\mathcal{I}}$ as the collection of XML fragments that are defined by the XPath expressions in $X_{\mathcal{I}}$.

$$F_{\mathcal{I}} = \bigcup_{p \in X_{\mathcal{I}}} value(p, d)$$

We now define $F'_{\mathcal{D}}$ and $F'_{\mathcal{I}}$ by filtering the sets $F_{\mathcal{D}}$ and $F_{\mathcal{I}}$ through the XQuery where clause w specified in the `WHERE` section. $F'_{\mathcal{D}}$ and $F'_{\mathcal{I}}$ are defined as follows:

$$\begin{aligned} F'_{\mathcal{D}} &= \{f \mid f \in F_{\mathcal{D}} \wedge satisfy(f, w)\} \\ F'_{\mathcal{I}} &= \{f \mid f \in F_{\mathcal{I}} \wedge satisfy(f, w)\} \end{aligned}$$

From the definitions of $F'_{\mathcal{D}}$ and $F'_{\mathcal{I}}$ it is now possible to compute the *frequency* of an itemset X, $X \in F'_{\mathcal{I}}$, as follows:

$$freq(X, F'_{\mathcal{D}}) = \frac{\sum_{c \in F'_{\mathcal{D}}} \prod_{f \in X} contains(c, f)}{|F'_{\mathcal{D}}|}$$

Given the function *freq* it is then possible to extract frequent (XML) itemsets from an XML document. Therefore it is possible to extract the association rules, as specified in the `XMINE` statement. Note that all the three functions used to define the *freq* are available or easy to implement. In particular, the function *value* can be directly implemented by any of the available XPath interpreter (e.g., Xalan [11]). The function *satisfy* can be implemented by an XPath interpreter, if the conditions in the `WHERE` clause are simple enough; by an XQuery interpreter when this will become available. The function *contains* can be again easily implemented on top of an XPath interpreter (see Section 6).

6. The Architecture of XMINE

`XMINE` has been devised as an extension of the XML Query language [14] and therefore it could be easily implemented on the top of an XQuery interpreter. However, because of the lack of a robust XQuery execution environment, the current implementation is built on the coupling of the Xalan [11] XPath interpreter and an algorithm to extract association rules from relational data. The `XMINE` prototype has been implemented as a three-step process [7], namely, *preprocessing*, *mining*, and *postprocessing*. The overall process is illustrated in Figure 4. In the initial preprocessing step (illustrated in Figure 4 with arrows from 1 to 4), the `XMINE` statement is processed to generate a representation of the XML mining problem as a relational table R which can subsequently elaborated by a known association rule mining algorithm. Then in the following *mining* step, association rules are extracted from R; during this phase the constraints specified in the `EXTRACTING RULES` clause are exploited by the mining algorithm (Figure 4, arrow labeled 5). In the final *post processing* step, the association rules extracted from relation R are finally mapped into the XML representation (Figure 4, arrow labeled 6).

Preprocessing. First the sets $F_{\mathcal{D}}$ and $F_{\mathcal{I}}$ (see Section 5) are generated (Figure 6, arrow from 1 to 3). These sets contain the XML fragments addressed by the XPath expressions specified in the clauses `ROOT` ($F_{\mathcal{D}}$), `HEAD`, and `BODY` ($F_{\mathcal{I}}$). In general, this step could be implemented by any of the available XPath interpreters; in our case, we use the Java implementation of Xalan [11]. Then the sets $F'_{\mathcal{D}}$ and $F'_{\mathcal{I}}$ containing the XML fragments in $F_{\mathcal{D}}$ and $F_{\mathcal{I}}$ which satisfy the `WHERE` clause are generated. This step is implemented on the top of the Xalan interpreter of Xpath. Note that since there is no XQuery interpreter currently available, our initial implementation supports only basic `WHERE` conditions mainly based on XPath expressions such as: `ROOT//@year=2001`. With the sets $F'_{\mathcal{D}}$ and $F'_{\mathcal{I}}$ the relational table R is generated as follows.

- R has as many attributes (i.e., columns) as is the number of XML fragments in $F'_{\mathcal{I}}$, i.e., $R \subset \{0, 1\}^{|F_{\mathcal{I}}|}$.
- R has as many tuples (i.e., rows) as the number of fragments in $F'_{\mathcal{D}}$, i.e., $R = \{r_1, \ldots, r_{|F'_{\mathcal{D}}|}\}$;
- for every XML fragment $c_i \in F'_{\mathcal{D}}$ and every XML fragment $f_j \in F'_{\mathcal{I}}$, a tuple $r_i \in R$, $r_i = \langle r_{i,1} \ldots r_{i,|F_{\mathcal{I}}|} \rangle$, is defined as $r_{i,j} = contains(c_i, f_j)$.

From Section 5 we recall that the function $contains(c_i, f_j)$ returns one if the fragment c_i contains f_j. Note that the current implementations of XPath do not provide any method to check whether a fragment appear in another fragment. Accordingly, in our prototype the function *contains* is implemented on the top of the XPath interpreter as follows. Given two fragments c_i and f_j, their textual representation is generated exploiting the functionalities of the XPath interpreter. Then it is checked whether the textual representation of f_j actually appear in the textual representation of c_i; if it does, one is returned, otherwise zero. This solution is indeed computationally expensive, but on the other hand,

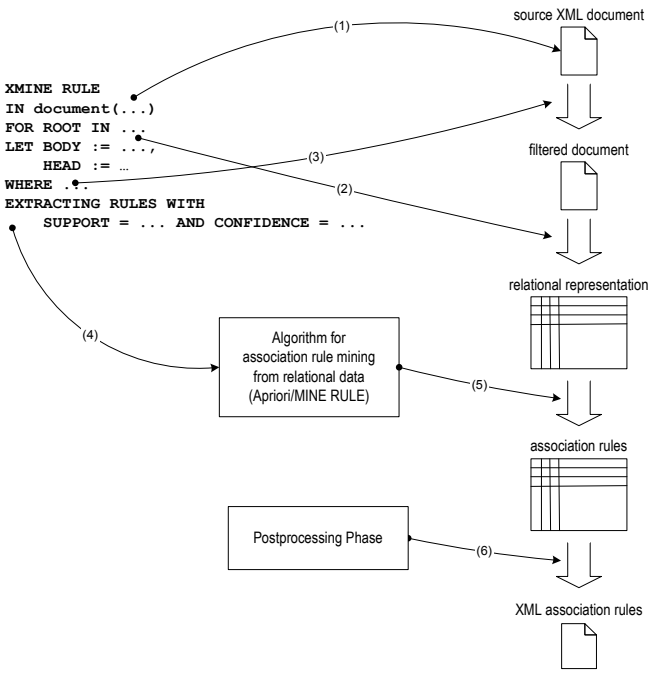

Figure 4. The architecture of the XMINE prototype.

the comparison of XML fragments is one of the major open issues in the XML community. Thus, it might be possible in the future to reduce the complexity of this phase through some advanced technique introduced in the next versions of XPath and XQuery. Finally, the constraints specified in the EXTRACTING clause are represented so to be exploited by the association rule mining algorithm, in the next phase (Figure 6, arrow from 4).

Mining. With the table R and the constraints extracted from the EXTRACTING RULE clause in the previous phase, association rules are extracted (Figure 6, arrow from 5). In the initial implementation presented here this phase is based on the plain Apriori algorithm. Note that among the many constraints which might be specified, our implementation currently considers only constraints concerning the composition of the rules (e.g., the number of items in the rule head or body), the minimum support, and the minimum confidence. In fact, some complex constraints (e.g., those described in [5]) are currently not possible with a tabular representation– but they would require mining the XML association rules directly from the DOM (Document Object Model) representation. DOM offers a platform–and language–neutral API (Application Programming Interface) allowing programs to dynamically access and update the content, structure and style of the documents. Recently we decided to implement this mining step by using MINE RULE [10] since it allows a better specification of complex constraints. The new version of XMINE is still under implementation.

Postprocessing. In the final step (depicted in Figure 6 with arrow 6), association rules extracted from relational data are mapped into the XML representation. Here the information which have been memorized during the preprocessing, such as the correspondence between the XML fragments in $F'_{\mathcal{D}}$ and $F'_{\mathcal{I}}$ and the attributes in R, is exploited in order to print an XML version of the association rules generated through the Apriori algorithm.

7. Notes on the Implementation

The XMINE prototype has been implemented in Java on the top of the Xalan XPath 1.0 [11] interpreter and DOM; the mining phase was initially implemented with the plain Java implementation available in Weka [3]. Efficiency has not been of our main concern in developing the initial implementation of XMINE. However, the first experiments of use are good thanks to the excellent performance of the Xalan XPath interpreter. In addition, our initial experiments suggests that the preprocessing and postprocessing steps are reasonably fast with respect to the central mining phase. Our first experiments conducted on some simple bibliographic XML documents obtained by using BibTeXML [2] showed that in general, association rule mining is quite a complex task if compared to the simple preprocessing needed to produce the relational representation

needed for mining. More specifically, in our initial experiments we noted that preprocessing and postprocessing together required less than the 20% of the overall computing time [7]. Note however that since the structure of source documents was quite simple with respect to that of most XML documents, these preliminary results surely lack of generality. These considerations suggest that an efficient execution of XMINE statements is feasible with current technology as long as the statement includes only XPath expressions or basic XQuery WHERE conditions. Our three-step process, which exploits an intermediate relational representation, allowed us to implement a basic prototype smoothly. On the other hand, the lack of a robust XQuery execution environment does not allow us to compute the complex predicates supported by the XMINE syntax on a native XML representation. The implementation we presented here (see also [7]) leaves many open issues, which must be addressed to have a fully functional and efficient implementation of the XMINE operator. We believe that the most important open issue regards the support evaluation, which should be used to extract association rules from XML documents. There are currently no XQuery interpreters to be used in filtering XML fragments and although XPath processors are quite advanced, they do not yet provide all the required functionalities either. Many of these functionalities should be included in the new XPath 2.0 version, while some operations needed for advanced options [5] are still not defined but are still requirements of the W3C consortium. Therefore, many required computation must be coded directly on the raw DOM representation of the documents (e.g., the filtering of WHERE clauses) which causes some computational overhead. Note that in our prototype we implemented some of these functionalities on the top of DOM and XPath 1.0 Xalan interpreter, however, we do not plan to implement all the functionalities needed by XMINE. In fact, we expect reliable XPath 2.0 and XQuery execution environments to be available in the very near future.

8. CONCLUSIONS

In this paper, we overviewed the XMINE operator, a tool we have devised to extract association rules from native XML data. Although this research provides a good starting point, there are still many open issues for future research–even within the work presented in this paper. Additionally, association rules from XML data could be extended to the case of episode rules [9]. A more general question is how to consider and connect all available metadata–not only the XML DTD or schema information, but also RDF metadata and DAML+OIL format ontologies–for a specific mining task in a specific domain. This additional metadata provides enhanced possibilities to constrain the mining queries both automatically and manually, but it also increases the complexity and thus poses lots of new requirements for the data mining query language.

Acknowledgments

In this work, the authors have been supported by the *consortium on discovering knowledge with Inductive Queries* (*cInQ*) [1] of the EU-IST Programme (Contract no. IST-2000-26469). The authors wish to thank Roberto Carnevale for his implementation of the first XMINE prototype.

References

[1] **c**onsortium on discovering knowledge with **In**ductive **Q**ueries (*cInQ*). http://www.cinq-project.org.

[2] BibTeXML an XML representation of BibTeX. http://www.bibtexml.org.

[3] WEKA. www.cs.waikato.ac.nz/ml/weka/.

[4] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'93)*, pages 207 – 216. ACM, May 1993.

[5] D. Braga, A. Campi, S. Ceri, M. Klemettinen, and P. L. Lanzi. Discovering interesting information in xml data with association rules. Technical Report 2002-15, Dipartimento di Elettronica e Informazione – Politecnico di Milano, 2002.

[6] D. Braga, A. Campi, M. Klemettinen, and P. L. Lanzi. Mining association rules from xml data. In *Proceedings of the 4^{th} International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2002). September 4-6, Aix-en-Provence, France*, 2002. accepted.

[7] R. Carnevale. Algoritmi per stemming ed estrazione di regole di associazione su documenti xml. Master's thesis, Apr. 2002. Master thesis supervised by Marco Colombetti and Pier Luca Lanzi. (available in Italian).

[8] D. M. Group. PMML 2.0 Predictive Model Markup Language. http://www.dmg.org, Aug. 2001.

[9] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 210 – 215. AAAI Press, 1995.

[10] R. Meo, G. Psaila, and S. Ceri. An extension to SQL for mining association rules. *Data Mining and Knowledge Discovery*, 2(2):195 – 224, 1998.

[11] The Apache Software Foundation. The Apache XML Project. http://xml.apache.org/xalan-j/.

[12] World Wide Web Consortium. Extensible Markup Language (XML) Version 1.0 (W3C Recommendation). http://www.w3c.org/xml/, Feb. 1998.

[13] World Wide Web Consortium. XML Path Language (XPath) Version 1.0 (W3C Recommendation). http://www.w3c.org/tr/xpath/, Nov. 1999.

[14] World Wide Web Consortium. XQuery 1.0: An XML Query Language (W3C Working Draft). http://www.w3.org/TR/2001/WD-xquery-20011220, Dec. 2001.