XQUAKE
An XQuery-like Language for Mining XML Data

Andrea Romei and Franco Turini
Department of Computer Science, University of Pisa, Largo B. Pontecorvo, 3, Pisa 56127, Italy

Keywords: Data mining, Knowledge discovery, Inductive databases, XML, XQuery, Query language.

Abstract: The rapid growth of semi-structured sources raises the need of designing and implementing environments for knowledge discovery out of XML data. This paper presents an Inductive Database System in which raw data, mining models and domain knowledge are represented as XML documents, stored inside XML native databases. In particular, we discuss our experiences in the design and development of XQuake, a mining query language that extends XQuery. Features of the language are an intuitive syntax, a good expressiveness and the capability of dealing uniformly with raw data, induced and background knowledge. The language is presented by means of examples and a sketch of its implementations and the evaluation of its performance is given.

1 INTRODUCTION

Inductive Databases (IDBs) are general purpose databases in which both the data and the knowledge are represented, retrieved, and manipulated in an uniform way (Imielinski and Mannila, 1996). A critical aspect in IDBs is the choice of what kind of formalism is more suited to represent models, data sources, as well as the queries one might want to apply on them. A considerable number of different papers propose to integrate a mining system with a relational DBMS. Relational databases are fine for storing data in a tabular form, but they are not well suited for representing large volumes of semi-structured data fields. However, data mining need not be necessarily supported by a relational DB.

The past few years have seen a growth in the adoption of the *eXtensible Markup Language* (XML). On the one hand, the flexible nature of XML makes it an ideal basis for defining arbitrary languages. One such example is the Predictive Modelling Markup Language (PMML) (The Data Mining Group, 2009), an industry standard for the representation of mined models as XML documents. On the other hand, the increasing adoption of XML has also raised the challenge of mining large collections of semi-structured data, for example web pages, graphs, geographical information and so on. From the XML querying point of view, a relevant on-going effort of the W3C is the design of a standard query language for XML, called *XQuery* (W3C World Wide Web Consortium, 2007), which is drawing much research and for which a large number of implementations already exists.

The goal of our research is the design and implementation of a mining language in support to an IDB in which an XML native database is used as a storage for Knowledge Discovery in Databases (KDD) entities, while Data Mining (DM) tasks are expressed in an XQuery-like language, in the same way mining languages on relational databases are expressed in a SQL-like format. Features of the language are the expressiveness and flexibility in specifying a variety of different mining tasks and a coherent formalism capable of dealing uniformly with raw data, induced knowledge and background knowledge.

Due to space restrictions, we do not present here the data model on which XQuake is based, but we focus on the basic ideas behind the language, discussing several examples of its concrete usage. All examples assume the availability of XML data in a native XML database. Specifically, Figure 1 shows a fragment of the online `dblp` database[1], containing bibliographic information on major computer science journals and proceedings. Figure 1 depicts a sample document including various information about researchers in a university department. Finally, the XML document `mondial`[2] of Figure 1 contains a collection of XML tags storing geographical, economic and political characteristic of a country.

[1] www.dblp.uni-trier.de/xml/

[2] www.dbis.informatik.uni-goettingen.de/Mondial/

Romei A. and Turini F. (2010).
XQUAKE - An XQuery-like Language for Mining XML Data.
In *Proceedings of the 2nd International Conference on Agents and Artificial Intelligence - Artificial Intelligence*, pages 20-27
DOI: 10.5220/0002703400200027

```
<dblp>
  <article key="tr/dec/SRC1997-018">
  <author dep="id392">
    <first-name>Tomasz</first-name>
    <last-name>Imielinski</last-name>
  </author>
  ....
  <title>A database perspective .... </title>
  <journal>Comm. of the ACM</journal>
  <volume>39(11)</volume>
  <pages>58-64</pages>
  <year>1996</year>
  <keywords>
    <keyword>data mining</keyword>
    <keyword>databases</keyword>
    ....
  </keywords>
  <abstract>The concept of ....</abstract>
  </article>
<inproceedings key="conf/ic/NayakWT02">
  <author dep="id949">....</author>
  ....
</inproceedings>
....
</dblp>
```

```
<Department id="id145">
  <Name> ... </Name>
  <University> ... </University>
  <People>
    <Employee>
      <PersonalInfo>
        <Name> ... </Name>
      </PersonalInfo>
      <Education>
        <Higher>
          <PhD year="1999"> yes </PhD>
        </Higher>
      </Education>
      <Awards>
        <Award year="2001" society="IEEE">
          <Description>This ...</Description>
        </Award>
      </Awards>
      <Projects>
        <Project active="yes" isDirector="yes"
                 name = "p1">
          <Description> The ... </Description>
        </Project>
        ....
      </Projects>
    </Employee>
    ....
  </People>
</Department>
```

```
<mondial>
 <country car_code="I" area="301230"
          capital="cty-Italy-Rome"
          memberships="org-EU ...">
   <name>Italy</name>
   <population discr-as="med">57460274</population>
   <gdp_total>1088600</gdp_total>
   ....
   <inflation discr-as="low" >5.4</inflation>
   <government>republic</government>
   <encompassed continent="europe"
                percentage="100"/>
   <religions perc="98">Catholic</religions>
   <province capital="cty-Italy-Rome">
     ....
   </province>
   ....
 </country>
 ....
 <organization id="org-WHO"
               headq="Switzerland-4">
   <name>World Health Organization</name>
   <abbrev>WHO</abbrev>
   <established>1946-07-22</established>
   <members type="member"
            country="FR AL ESP GR...."/>
 </organization>
 ....
</mondial>
```

Figure 1: XML fragments of the `dblp` dataset (a), the `departments` dataset (b) and the `mondial` dataset (c).

2 XQUAKE

In this section, the XQuake (acronym of XQUery-based Applications for Knowledge Extraction) mining language is presented.

2.1 The XQuake Philosophy

Essentially, XQuery expressions are used to identify XML data as well as mining fields and metadata, to express constraints on the domain knowledge, to specify user preferences and the format of the XML output.

A mining query begins with a collection of XQuery functions and variables declarations followed by an XQuake operator. The syntax of each operator includes three basic statements[3]: (i) *task and method specification*; (ii) *domain entities identification*; (iii) *exploitation of constraints and user preferences*. The outline of a generic operator is explained below.

Task and Method Specification. Each XQuake operator starts specifying the kind of KDD activity. Constructs for preprocessing, model extraction, knowledge filtering, model evaluation or model meta-reasoning are possible. As an instance, the following XQuake fragment denotes a data sampling task:

```
prepare sampling doc("my-out") using
  alg:my-sampling-alg(my-params ...).
```

[3]More precisely, the language also allows the construction of the generated results. However, this feature is not described in this paper.

The `doc("my-out")` expression directs the result of the mining task to a specific native XML database for further processing or analysis. The `using` statement introduces the kind of mining or preprocessing algorithm used, together with atomic parameters.

Domain Entities Identification. Any KDD task may need to specify the set of relevant entities as input of the analysis. The syntax is an adaptation of the standard XQuery FLOWR syntax, in which the result of the evaluation of an expression is linked to a variable in `for` and `let` clauses. Below you can see a simple statement that can be used to locate input data.

```
for data $tuple in <XQuery expr>
where <XQuery expr on $tuple>
let active field $field := <XQuery expr on $tuple>.
```

Input data is typically a sequence of XML nodes. The *<for>* expression above binds the variable `$tuple` to each item during the evaluation of the operator, while the *<let>* clause identifies an attribute of the data. XQuake also offers an optional *<where>*, used to express data filtering constraints. The user specifies them through an XQuery condition that is typically processed before the mining task. The *<let>* clause defines a data attribute with name `$field`, whose values are obtained by means of the XQuery expression in the body and whose type is omitted. The keyword after the *<let>* refers to the role of such an attribute in the mining activity of interest. More specifically:

- *<active>* specifies that the field is used as input of the analysis;
- *<predicted>* specifies that it is a prediction attribute;

- <*supplementary*> states that it holds additional descriptive information.

Mining fields in input to the mining task are required to be atomic, i.e. an instance of one of the built-in data types defined by XML schemas, such as string, integer, decimal and date. A richer set of types is included into the data model by extending the system type of XQuery to support discrete, ordering and cyclical data types. Moreover, XQuake maintains the typing philosophy of XQuery by offering a method to equip attributes with logical information. In the query fragment below, we are interested in the specification of a mining attribute that indicates whether a journal paper published by the 'ACM' focusses on the KDD field. An explicit boolean type is specified by the user for the field `$has-kdd-keyword`.

```
for data $paper in doc("dblp")//article
where fn:contains($paper/journal, "ACM")
let active field $has-kdd-keyword as xs:boolean :=
    some $keyword in $paper/keywords/keyword
    satisfies $keyword eq "KDD".
```

Either if an input field has an explicit or an implicit type, it is validated against a required type, that depends on the context in which it appears. For instance, the target attribute of a classification task is required to be discrete. An error is raised whenever the type of an expression does not match the expected type.

Exploitation of Constraints and User Preferences. XQuake admits a special syntax to specify domain knowledge[4], particularly useful for the definition of domain-based constraints. In contrast to active and predicted mining fields, a metadata field may include also non-atomic types, such as XML nodes or attributes. For example, below we assign an hypothetical XML hierarchy to a table column as metadata information.

```
for data $country in doc("mondial")//country
let metadata field $hier :=
 let $cap := $country//city[@is-capital='yes']
 return doc("hierarchy")/root/city[.=$cap].
```

For each distinct `<country>` element of the `mondial` dataset, the <*metadata*> keyword defines a special field used to bind domain knowledge (an XML taxonomy in this case) to the capital of that country. This paper reports, in the next section, several examples to show how the user can express personalized sophisticated constraints based on the domain knowledge.

[4]The domain, or background, knowledge is the information provided by a domain expert about the domain to be mined.

2.2 Constraint-based XML Frequent Itemset Mining

One of the most important open issues in frequent itemset mining is the too large number of generated patterns. The constraint-based pattern mining paradigm has been introduced with the aim of providing the analyst with a domain-dependent tool for driving the discovery process directly towards potentially interesting patterns. We present an operator to handle constrained itemset mining out of XML data.

Example 1. We wish to discover correlations among authors in the `dblp` dataset. In our analysis, we consider only the patterns with a minimum support of 10%, in which some "leader" author occurs, and in which at most two authors received a PhD after 2002. We consider as "leaders" those authors that received an award from the "IEEE" society or, alternatively, are prime investigators of an active project. The resulting XQuake operator is given in Figure 2.2. ◁

The query is not hard to understand for readers familiar with XQuery. The algorithm used is the Apriori algorithm (Agrawal and Srikant, 1994), with the relative minimum support expressed as a parameter (<*using*> clause).

The set of involved XML transactions, i.e. both proceeding and journal papers, is specified through the <*for data*> clause. The next statement uses a similar syntax to identify items of a transaction, i.e. the authors of a publication, binding each XML node `<Author>` to the variable `$author`. The keyword <*item*> specifies in this case that we are iterating over the items of a transaction. The <*let active*> clause uses a built-in function to format the required author name, i.e. the atomic values in the itemset. In addition, we bind to the variable `$employee` an XML element encoding the domain knowledge (i.e. the employee information of the department of interest to each distinct author - see also Figure 1). This is achieved through the <*let metadata*> statement containing, in the body, an XQuery expression that first looks for the department of interest in the collection of the various departments, and then looks for the author name among its employees. Notice the use of both the `$aut` and `$aut-name` variables in the body expression.

The set of itemset constraints occur in the <*having*> clause. Specifically, they constrain the number of the items of an itemset that satisfies a particular condition to have a certain threshold. They have the following format (where $n > 0$).

having at least *n* **item satisfies** `<XQuery predicate>`.

The operator <*at least*> (similar are <*at most*> and

```
mine itemsets doc("leader-co-authors") using alg:apriori(0.1)
for data $paper in doc("dblp")/(inproceedings|article)
for item $aut in $paper/Author
let active field $aut-name := fn:concat($aut/FirstName," ", $aut/LastName)
let metadata field $employee :=
            let $dept := collection("Dep")/Department[@id=$aut/@dep]
            return $dept//Employee[//Name=$aut-name]
having at least 1 item satisfies
           (some $award in $employee/Awards/* satisfies $award="IEEE" or
            some $p in $employee/Projects/* satisfies
            $p/@active="yes" and $p/@is-director="yes"),
       at most 2 item satisfies $employee/PhD/@year > 2002

mine itemsets doc("co-countries") using alg:apriori(0.35)
for data $d in doc("mondial")//organization/members
for item $i in for $j in fn:tokenize(string($d/@country)," ") return $j
let active field $f := fn:concat("country=", $i)
let metadata field $country := doc("mondial")//country[./car_code eq $i]
having at least 1 item satisfies
      some $j in $country/encompassed/@continent satisfies $j eq "europe",
        exactly $ALL item satisfies $country/government eq "republic".
```

Figure 2: Two examples of the MINE ITEMSETS operator at work.

<*exactly*>) is true for all itemsets which have at least a specified number of items that satisfy the XQuery predicate. The latter one can be expressed on the variables previously defined that, in the example 1, denote both the author's name and the employee metadata. The examples below highlight their usage.

```
exactly 1 item satisfies $aut-name eq "A. Einstein"
at least 4 item satisfies fn:true()
at most fn:round($ALL div 2) item satisfies
 count($employee//Project[@active]="yes") > 1
exactly $ALL item satisfies $aut/@dep eq "Cambridge".
```

The first condition above finds out who publishes frequently together with "A. Einstein". The next clause looks for itemsets of length at least 4. The third condition imposes that at most half of the authors in the itemset are involved in at least two active projects. The special variable `$ALL` stands for the length of the current itemset that should be validated against the constraint. Finally, the last predicate finds correlations among the publications of the authors at the "Cambridge University".

In the above queries we have supposed the availability of metadata. In several cases, XQuery can be also used to build metadata from scratch. The variable below stores an XML element containing the number of publications for each distinct author in the `dblp` database.

```
declare variable local:np as node()* :=
 for $a in distinct-values(doc("dblp")//author)
 return element publication {
  <name>{$a}</name>,
  <number>{count(/dblp/*[aut=$a])}</number>};
```

The fragment of the XQuake query below returns all itemsets which have "Enrico Fermi" as an author, and at least one co-author with a number of publications greater than 30.

```
let metadata field $n as xs:decimal :=
      local:np/number[./name=$aut-name]
having exactly 1 item satisfies $aut-name eq "Fermi",
      at least 1 item satisfies
       $aut-name ne "Fermi" and $n > 30.
```

The predicate below aims at extracting frequent itemsets in which it does not exist one author that is also editor.

```
let metadata field $is-editor :=
      not(empty(doc("dblp")//editor[.=$aut]))
having exactly $ALL item satisfies not($is-editor).
```

The versatility of the <*mine itemsets*> operator permits to comply easily with the nature of the domain and the interpretation of the items. In the next example, we exploit the `mondial` dataset as a source for constraint-based frequent itemset mining.

Example 2. In the `mondial` dataset, we are interested in finding correlations among the countries that appear frequently as members of the 168 mondial organizations (e.g. FAO, ONU, etc.). For instance, the itemset {`country=FR, country=USA`} `supp=0.2` means that France and USA occurred about 20% of the times together as members of some organization. Also, we require that in each itemset at least one item is an European country and that every country in the itemset is a republic. The query of Figure 2.2 accomplishes this task. ◁

Notice that in the example above, the transactions are all the countries located in the `<organization>` XML tags, items are the car codes of the countries

and the metadata information is the set of `<country>` elements (see Figure 1).

2.3 XML Classification

The next example focuses on the XML classification task, and in particular on the issue of specifying the input to a decision tree algorithm. In the example, we select the `mondial` database as an XML table suitable as an input to a classification algorithm intended to build a model of the geographical, economic and political information of the countries.

Example 3. The goal is to classify new countries in two different categories: the countries that are good candidates to become a new member of the UNESCO and those that are not. The classification scenario is as follows. Records of the training set are the countries located by means of the homonymous XML element. The set of the attributes includes country's properties like (i) the government type; (ii) the values of the level of inflation and the population of the capital city; (iii) a binary attribute indicating whether the capital of the country has an extension greater than a fixed value. The DM task for generating the classification model can be specified by means of the <*mine tree*> operator, as shown in Figure 3. ◁

The statement can be read as follows. The <*using*> clause specifies the mining algorithm, ID3 (Mitchell, 1997) in this case, and the confidence for pruning as parameter. The <*for*> clause identifies the XML nodes that denote the records of the training set. Each <*let*> element specifies the attributes in the source data set that are considered for mining, i.e. the mining schema. Since the ID3 algorithm is restricted to deal with discrete sets of values, the operator forces the type of each field to be `xs:string` or one of its subtypes (e.g. discrete or ordinal). An example of discrete attribute definition is the last clause above, that specifies the target attribute `$is-unesco`, whose only admitted values are "yes" or "no".

XQuake also offers operators to apply the generated itemsets and classification trees to further data or to do preprocessing.

3 THE PHYSICAL LEVEL

3.1 The XQuake System Prototype

XQuake is supported by a *tightly-coupled* architecture designed directly over BaseX (Holupirek et al., 2009), a Java-based open source native XML database developed by the Database and Information Systems Group at the University of Konstanz. Basically, the XQuake system prototype defines and implements the mining operations via extended XQuery programs. On the one hand, complex operations over data structures are implemented in Java and the I/O of such operators is integrated within the XQuery program by external functions.

On the other hand, such an extension regards a new iterative construct, named `wfor`, encapsulated into XQuery, whose aim is to provide a better implementation of the mining tasks. It tries to overcome two main deficiencies of the traditional FLOWR expressions in order to define queries over windows of data (useful in data preparation tasks) and to manage local temporary variables in an iterative computation. The general schema of the `wfor` is the following.

```
wfor $binding-var in <sequence>
   declare state variable $state-var
                as <type> := <expression>
   init <expression>
   iterate <expression> while <condition>
   return <expression>.
```

It iterates over an input sequence of length $N > 0$ and it binds a variable to each item of the sequence. The effect of the `declare` clause is to introduce a new variable - say *state variable* - and to initialize it with the value of the given expression body. The state variable is in the scope of all the rest of the `wfor` expression. Also, it is updated at each iteration with the result of the `iterate` clause, whose aim is to consume the sequence item by item. The `while` breaks the cycle and forces the execution of the `return` clause, that returns an output value. At this point, if additional items exist in the input sequence, the computation continues by re-initializing the state variable with the result of the `init` statement and by evaluating `iterate` again. Intuitively, the `while` specifies when the `return` clause should be evaluated and a new value returned as output. At one extreme, if it is `true()` or absent, then a single value is returned (e.g. for model extraction operators). At the other extreme, if it is `false()`, a sequence of length N is returned as the answer (e.g. for preprocessing operators). Moreover, `wfor` expressions can be nested to iterate over an input sequence several times. Due to space restrictions, the syntax and the semantic aspects of `wfor` are out of the core of this paper.

The high level XQuake architecture is shown in Figure 4. A mining task is expressed in XQuake via a specific *text editor*. A *compiler* automatically generates the appropriate (extended) XQuery code that is then interpreted. The compiler is designed to provide a good level of extensibility to accomodate the definition of new algorithms. The core of the mining pro-

```
mine tree doc("is-unesco-member") using alg:id3()
for data $c in doc("mondial-discr")/country
let active field $government := string($c/government)
let active field $population := $c/population/@disr-as
let active field $inflation := $c/inflation/@discr-as
let active field $is-fao := if (fn:contains($c/memberships, "org-FAO"))
                            then "yes" else "no"
let active field $ext-cap := let $c-d := $c//city[@is-capital = "yes"]/gdp
                             return if ($c-d > 10000) then "t" else "f"
let predictive field $class as xs:discrete<("yes","no")> :=
            if (fn:contains($c/memberships, "UNESCO")) then "yes" else "no"
```

Figure 3: Extraction of a classification tree from the `mondial` dataset.

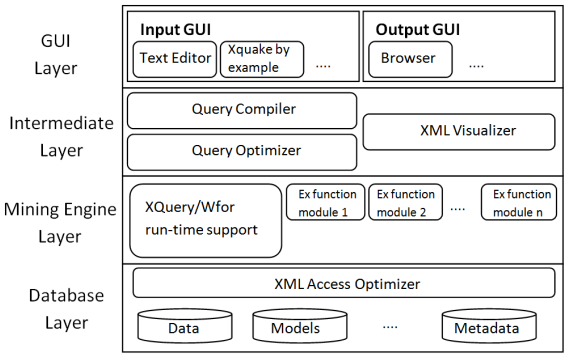

Figure 4: The XQuake system architecture.

cess is performed by the *mining engine* that contains the run-time support of the BaseX XQuery engine extended with the `wfor` iterator. This component uses the *external function modules* responsible for providing an XQuery interface to external user-defined Java functions over data structures. At the bottom, we have the BaseX native XML database containing the input and output of mining tasks, as well as XML metadata. The database is accessed by means of the *XML access optimizer* component that will contain (proprietary or native) indexing techniques to speed-up the access to input data. Finally, the *XML visualizer module* translates an XML document stored in the DB into a visualization form, accepted by data and model visualization tools, and presented by means of an *output GUI* to the user. Currently, the XML result is transformed into HTML browsable format via XSL style sheets.

3.2 Performance Evaluation

In the following, we analyze the impact of the architecture on the frequent itemsets problem. In order to compare the performance with an existent Apriori implementation, we tested the effects of a very simple XQuake query, without considering the encapsulation of any kind of constraints.

Table 1: Summary of datasets for experiments.

Name	Real	Avg trans	Num trans	Num items	Min supp	Num patterns
census	yes	15	48841	135	2	70826
mushroom	yes	23	8122	119	10	574513
connect-4	yes	43	67556	129	88	55115
T20I6D100K	no	20	100K	1K	0.2	25820
T30I10D100K	no	30	100K	1K	0.2	108634
T20I6D300K	no	20	300K	3K	0.2	7457

We used both real and synthetic datasets. The real datasets are from the UCI repository[5]. The synthetic databases are generated by means of the IBM generator[6]. These datasets are named "TxIyDz" according to the parameters x, y, z indicating the average number of items in the transactions, the average number of items in the large itemsets, and the number of transactions in the database, respectively. A summarization of the databases used in our experiments is outlined in Table 1, in which the last two columns identify the lower minimum threshold of the support used in the experiments and the number of extracted patterns.

We carried out our tests on a dual core Athlon 4000+ running Windows XP. We assigned 1.5Gbyte of memory to the Java Virtual Machine. Figures 5, 6, 7 and 8 report the performance obtained on the datasets by varying the value of the minimum support. In order to have an idea of the performance of XQuake, we compared its execution time on the three real datasets with those obtained by running the well-known Java-based Weka system[7] (Figures 5, 6 and 7).

The first group of experiments shows good and promising results. When the mining becomes hard, XQuake outperforms Weka and the differences between the two implementations tend to increase with respect to lower values of the minimum support threshold. The scalability is also acceptable on artificial datasets (Figure 8), on which the performance of the algorithm resulted to be quite stable. The perfor-

[5] http://archive.ics.uci.edu/ml/

[6] www.almaden.ibm.com/cs/disciplines/iis/

[7] http://www.cs.waikato.ac.nz/ml/weka

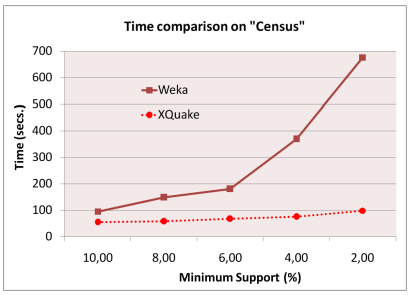

Figure 5: Runtime comparison among XQuake and Weka on the `census` dataset.

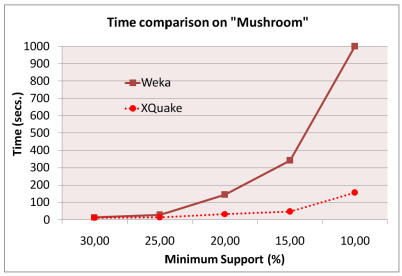

Figure 6: Runtime comparison among XQuake and Weka on the `mushroom` dataset.

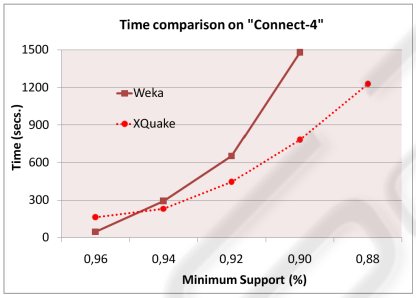

Figure 7: Runtime comparison among XQuake and Weka on the `connect` dataset.

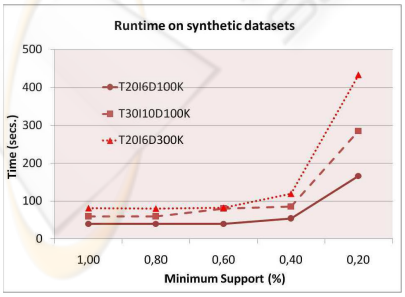

Figure 8: XQuake performance on the synthetic datasets.

mance overhead introduced by external Java functions integrated in XQuery is modest. This is confirmed by the graphs of Figures 3.2, 3.2 and 3.2, which report the overall execution time on the test datasets as a sum of: (i) the *initialization time*, i.e. the time to compile the query and to prepare the data structures; (ii) the *data iteration time*, i.e. the time to iterate over the data several times, according to the number of cycles of the Apriori; (iii) the *mining time*, i.e. the time to update data structures at each iteration; (iv) an estimation of the *overhead due to external functions* and finally (v) the *serialization time* to produce the output. The performance of the Apriori tends to worsen when the number of generated patterns is very large - more than 500,000 itemsets (Figure 3.2) . Such a behaviour is mainly due to the context-switching overhead due to their serialization.

4 FINAL REMARKS

4.1 Related Work

An important issue in DM is how to make all the heterogeneous patterns, sources of data and other KDD objects coexist in a single framework. A solution considered in the last few years is the exploitation of XML as a flexible instrument for IDBs (Meo and Psaila, 2006; Romei et al., 2006; Euler et al., 2006; Braga et al., 2003).

In (Meo and Psaila, 2006) XML has been used as the basis on which a semi-structured data model designed for KDD, called XDM, is defined. In this approach both data and mining models are stored in the same XML database. This allows the reuse of patterns by the inductive database management system. The perspective suggested by XDM is also taken in KDDML (Romei et al., 2006) and RapidMiner (Euler et al., 2006). Essentially, the KDD process is modeled as an XML document and the description of an operator application is encoded via an XML element. Both KDDML and XDM integrate XQuery expressions into the mining process. For instance, XDM encodes XPath expressions into XML attributes to select sources for the mining, while KDDML uses an XQuery expression to evaluate a condition on a mining model. In our opinion, XQuake offers a deeper amalgamation with the XQuery language and consequently a better integration among DM and XML native databases.

Finally, the XMineRule operator (Braga et al., 2003) defines the basic concept of association rules for XML documents. Two are the main differences with respect to XQuake. From the physical point of view, XMineRule requires that the data are mapped to the relational model and it uses SQL-

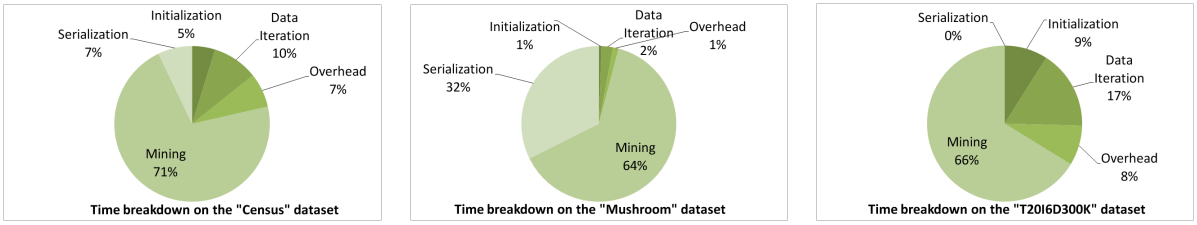

Figure 9: Time breakdown on the `census` dataset (a), `mushroom` dataset (b) and `T20I6D300K` dataset (c).

oriented algorithms to do the mining. Also the output rules are translated into an XML representation. As a consequence, the loosely-coupled architecture of XMineRule makes it difficult to use optimizations based on the pruning of the search space, since constraints can be evaluated only at pre- or post-processing time. From the semantics perspective, items have an XML-based hierarchical tree structure in which rules describe interesting relations among fragments of the XML source (Feng and Dillon, 2004). In contrast, in our approach, items are denoted by using simple structured data from the domains of basic data types, favouring both the implementation of efficient data structures and the design of powerful domain-specific optimizations evaluated as deeper as possible in the extraction process. Domain knowledge is linked to items through XML metadata elements.

4.2 Conclusions and Future Work

In this paper, we proposed a new QL as a solution to the XML data mining problem. In our view, an XML native database is used as a storage for KDD entities. DM tasks are expressed in an XQuery-like language. The syntax of the language is flexible enough to specify a variety of different mining tasks by means of user-defined functions in the statements. These ones provide to the user personalized sophisticated constraints, based, for example, on domain knowledge. The first empirical assessment reported in Section 3.2 exhibits promising results, even if only related to the XML frequent itemsets mining problem.

Summing up, our project aims at a completely general solution for DM. Clearly, the generality is even more substantial in XML-based languages, since no general-purpose XML mining language has been yet proposed (at the best of our knowledge). An interesting on-going work includes the exploitation of ontologies to represent the metadata. As an example, ontologies may represent enriched taxonomies, used to describe the application domain by means of data and object properties. As a consequence, they may provide enhanced possibilities to constrain the mining queries in a more expressive way. This opportunity is even more substantial in our project, since ontologies are typically represented via the Web Ontology Language (OWL) (W3C World Wide Web Consortium, 2004), de facto an XML-based language.

REFERENCES

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules. In *VLDB '94*, pages 487–499, Santiago de Chile, Chile.

Braga, D., Campi, A., Ceri, S., Klemettinen, M., and Lanzi, P. (2003). Discovering interesting information in XML data with association rules. In *SAC '03*, pages 450–454, Melbourne, Florida.

Euler, T., Klinkenberg, R., Mierswa, I., Scholz, M., and Wurst, M. (2006). YALE: rapid prototyping for complex data mining tasks. In *KDD '06*, pages 935–940, Philadelphia, PA, USA.

Feng, L. and Dillon, T. S. (2004). Mining Interesting XML-Enabled Association Rules with Templates. In *KDID '04*, pages 66–88, Pisa, Italy.

Holupirek, A., Grün, C., and Scholl, M. H. (2009). BaseX and DeepFS joint storage for filesystem and database. In *EDBT '09*, pages 1108–1111, Saint Petersburg, Russia.

Imielinski, T. and Mannila, H. (1996). A database perspective on knowledge discovery. *Comm. Of The Acm*, 39(11):58–64.

Meo, R. and Psaila, G. (2006). An XML-based database for knowledge discovery. In *EDBT '06*, pages 814–828, Munich, Germany.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Romei, A., Ruggieri, S., and Turini, F. (2006). KDDML: a middleware language and system for knowledge discovery in databases. *Data Knowl. Eng.*, 57(2):179–220.

The Data Mining Group (2009). The Predictive Model Markup Language (PMML). Version 4.0. `www.dmg.org/v4-0/GeneralStructure.html`.

W3C World Wide Web Consortium (2004). OWL Web Ontology Language. W3C Recommendation 10 February 2004. `http://www.w3.org/TR/owl-features`.

W3C World Wide Web Consortium (2007). XQuery 1.0: An XML Query Language. W3C Recommendation 23 January 2007. `http://www.w3.org/TR/Query`.