IOSR Journal of Computer Engineering (IOSRJCE)
ISSN: 2278-0661 Volume 4, Issue 6 (Sep.-Oct. 2012), PP 13-22
www.iosrjournals.org

Efficient Data Mining for XML Queries – Answering Support

G. Seshadri Sekhar[1], Dr.S. Murali Krishna[2]
[1]*M.Tech Student, Dept of CSE, Madanapalle Institute of Technology and Science, AP, India.*
[2]*Professor and Head, Dept of CSE, Madanapalle Institute of Technology and Science, AP, India.*

Abstract: *We describe an approach based on Tree-based Association Rules(TARs) mined rules, which provide approximate, intensional information on both the structure and the contents of XML documents and can be stored in XML format as well. There are two main approaches to XML document access: Keyword-based Search and Query-Answering. The idea of mining association rules to provide summarized representations of XML documents has been investigated in many proposals either by using languages XQuery,JQuery etc., and techniques developed in the XML context or by implementing graph-or-tree-based algorithms. In this paper, we introduce a proposal for mining and storing TARs (Tree-based Association Rules) as a means to represent intensional knowledge in native XML.*

Keywords: *XML, Keyword search, Intensional answers, Approximate answering, Succinct answers.*

I. INTRODUCTION

Data mining is usually used to extract interesting knowledge from large amounts of data stored in databases or data warehouses. This knowledge can be represented in many different ways such as clusters, decision trees, decision rules etc. Among them, association rules have been proved effective tool to discovering interesting relations in massive amounts of data. During the recent years, we have seen the dramatic development of the eXtensible Markup Language (XML) as a major standard for storing and exchanging information. The database research field has concentrated on XML as an expressive and flexible hierarchical model suitable to represent huge amounts of data with no absolute and fixed schema, and with a possibly irregular and incomplete structure. Despite its impressive growth in popularity, XML is still lacking appropriate techniques to retrieve datasets available to casual users.

There are two main approaches to XML document access: 1) Keyword-based search, 2) Query-answering. The first one comes from the tradition of information retrieval [9], where most searches are performed on the textual content of the document; this means that no advantage is derived from the semantics conveyed by the document structure. As for Query-answering, since query languages for semi structured data rely on the document structure to convey its semantics, in order for query formulation to be effective users need to know this structure in advance, which is often not the case, In fact, it is not mandatory for an XML document to have a defined schema: 50% of the documents on the web do not possess one [5]. When users specify queries without knowing the document structure, they may fail to retrieve information which was there, but under a retrieve information which was there, but under a different structure. This limitation is a crucial problem, which did not emerge in the context of relational database management systems. Frequent, dramatic outcomes of this situation are either the information overload problem, where too much data are included in the answer because the set of keywords specified for the search captures too many meanings, or the information deprivation problem, where either the use of in appropriate keywords or the wrong formulation of the query, prevent the user from receiving the correct answer. As a consequence, when accessing for the first time a large dataset, gaining some general information about its main structural and semantic characteristics helps investigation on more specific details. Discovering recurrent patterns inside XML (documents) provides high-quality knowledge about the document content: Frequent patterns are in fact intentional information about the data contained in the document itself, that is, they specify the document in terms of a set of properties rather than by means of data. As opposed to the detailed and precise information conveyed by the data, this information is partial and often approximate, but synthetic and concerns both the document structure and its content.

The idea of mining association rules[1],[2] to provide summarized representations of XML documents has been investigated in many proposals either by using languages such as XQuery,JQuery etc., and techniques developed in the XML context or implementing graph or tree-based algorithms. In this paper we introduce a proposal for mining and storing TARs (Tree-based Association Rules) as a means to represent intentional knowledge in native XML. Intuitively, a TAR represents intentional knowledge in the form of $S_B \Rightarrow S_{H,}$ where S_B is the body of tree and S_H is head tree of the rule and S_B is a sub tree of S_H. The rule $S_B \Rightarrow S_H$ states that, if the tree S_B appears in an XML document D, it is likely that the "wider", tree S_H also appears in D. Graphically, we render the nodes of the body of a rule by means of black circles and the nodes of the head by empty circles.

The intentional information embodied in TARs provides a valid support in several cases: 1) It allows to obtain and store implicit knowledge of the documents, useful in many respects: (i) When a user faces a dataset for the first time, s/he does not know its features and frequent patterns provide a way to understand quickly what is contained in the data set; (ii) Besides intrinsically unstructured documents, there is a significant portion of XML documents which have some structure, but only implicitly, that is, their structure has not been declared via a DTD or an XML-Schema[13]. Since most work on XML Query languages has focused on documents having a known structure, querying the above mentioned documents is quite difficult, because users have to guess the structure to specify the query conditions correctly. TARs represent a data guide that helps users to be more effective in query formulation; (iii) It supports query optimization design, first of all because recurrent structures can be used for physical query optimization, to support the construction of indexes and the design of efficient access methods for frequent queries, and also because frequent patterns allow to discover hidden integrity constraints, that can be used for semantic optimization; (iv) For privacy reasons, a document answer might expose a controlled set of TARs instead of the original document, as a summarized view that masks sensitive details[7]. 2) TARs can be queried to obtain fast, although approximate answers. This is particularly useful not only when quick answers are needed but also when the original documents are unavailable. In fact, once extracted, TARs can be stored in a (smaller) document and be accessed independently of the dataset they were extracted from.

Summarizing, TARs are extracted for two main purposes: 1) To get a concise idea-the gist-of both the structure and the content of an XML document, and 2) To use them for intentional query answering, that is, allowing the user to query the extracted TARs rather than the original document. In this paper, we concentrate mainly on the second task (Efficient Keyword search on XML documents). Our techniques are very useful in the Journalism area. We can post the data into XML storage file and users have facility to search the incidents such as news headlines, sports news, world news etc., without having any database knowledge. We have implemented sample application on journalism concept by applied with TARs technique. Our research is useful on not only particular area but also all areas such as educational, Financial, Scientific and Engineering etc.

Goal and Contribution

We provide a method for deriving intentional knowledge from XML documents in the form of TARs, and then storing these TARs as an alternative, synthetic dataset to be queried for providing quick and summarized answers. Our procedure is characterized by the following key aspects: a) It works directly on the XML documents, without transforming the data into any intermediate format, b) It looks for general association rules, without the need to impose what should be contained in the antecedent and consequent of the rule, c) It stores association rules in XML format, and d) It translates the queries on the original dataset into queries on the TARs set. The aim of our proposal is to provide a way to use intentional knowledge as a substitute of the original document during querying and not to improve the execution time of the queries over the original XML dataset.
Accordingly, the paper's contributions are:

1) An improved version of the TARs extraction algorithm introduced in[10], which was based on Path Join. The new version uses the better performing CMTreeMiner [6] to mine frequent sub trees from XML documents.

2) Approach validation by means of experimental results, considering both the previous and the current algorithm and showing the improvements.

3) Automatic user-query transformation into "equivalent" queries over the mined intentional knowledge.

4) As a formal corroboration of the accuracy of the process, the proof that our intentional-answering process is sound and complete up to a frequency threshold.

Structure of the paper

Our paper is organized as follows. Section II defines Tree-based Association Rules (TARs) and introduces their usage, while section III presents how these rules are extracted from XML documents. Section IV presents the main interesting application of TARs that is their use to provide intesional answers to queries. Section V describes a prototype that implements our proposal. Section VI draws the conclusion and future work.

II. TREE-BASED ASSOCIATION RULES

Association rule is an implication of the form $X \Rightarrow Y$, where the rule body X and head Y are subsets of the set I of items (I= {I1, I2…In}) within a set of transactions D and $X \cap Y = \varnothing$. A rule $X \Rightarrow Y$ states that the transaction T that contain the items in X are likely to contain also the terms in Y. Association rules are characterized by two measures: the support, which measures the percentage of transactions in D that contain both items X and Y ($X \cup Y$); the confidence, which measures the percentage of transactions in D containing the items X that also contain the items Y (support($X \cup Y$)/support(X)). In XML context, both D and I are collection of trees. In this work we extend the notion of association rule introduced in the context of relational databases to adapt it to the hierarchical nature of XML documents. Following the Infoset conventions, we represent an XML document as a tree $\langle N, E, r, l, c\rangle$ where N is the set of nodes, $r \in N$ is the root of the tree, E is the set of edges, $l : N \rightarrow L$ is the label function which returns the tag of nodes (with L is the domain of all tags) and $c : N \rightarrow C \cup \{\perp\}$ is the content function which returns the content of nodes(with C the domain of all contents). We consider the element-only Infoset content model [14], where XML non-terminal tags include only other elements and/or attributes, while the text is confined to terminal elements. We are interested in finding relationships among sub trees of XML documents. Thus, since both textual content of leaf elements and values of attributes convey "content", we do not distinguish between them. As a consequence, for the sake of readability, we do not report the edge label and the node type label in the label in the figures. Attributes and elements are characterized by empty circles, where as the textual content of elements, or the value of attributes, is reported under the outgoing edge of the element or attribute.

A. Fundamental concepts

Given two trees $T = (N_T, E_T, r_T, l_T, c_T)$ and $S = (N_S, E_S, r_S, l_S, c_S)$, S is an induced subtree of T if and only if there exists a mapping $\theta : N_S \rightarrow N_T$ such that for each node $n_i \in N_S$, $l_T(n_i) = l_S(n_j)$ and $c_T(n_i) = c_S(n_j)$, where $\theta(n_i) = n_j$, and for each edge $e = (n_1, n_2) \in E_S$, $(\theta(n_1), \theta(n_2)) \in E_T$. Moreover, S is a rooted subtree of T if and only if S is an induced subtree of t and $r_S = r_T$. Given a tree $T = \langle N_T, E_T, r_T, l_T, c_T\rangle$, a subtree of T, $t = \langle N_t, E_t, r_t, l_t, c_t\rangle$ and a user-fixed support threshold s_{min}: (i) t is frequent if its support is greater or at least equal to s_{min}; (ii) t is maximal if it is frequent and none of its proper supertrees is frequent; (iii) t is closed if none of its proper supertrees has support greater than that of t.

A Tree-based Association rule (TAR) is a tuple of the form $Tr = \langle S_B, S_H, s_{Tr}, c_{Tr}\rangle$, where $S_B = \langle N_B, E_B, r_B, l_B, c_B\rangle$ and $\langle N_H, E_H, r_H, l_H, c_H\rangle$ are trees and s_{Tr}, and c_{Tr} are real numbers in the interval [0, 1] representing the support and confidence of the rule respectively. A TAR describes the co-occurrence of the two trees S_B and S_H in an XML document. For the sake of readability we shall often use the short notation $S_B \Rightarrow S_H$; S_B is called the body or antecedent of T_r while S_H is the head or consequent of the rule. Furthermore, S_B is a subtree of S_H with an additional property on the node labels; the set of tags of S_B is contained in the set of tags of S_H with the addition of the empty label "ϵ": $\ell_{S_B}(N_{S_B}) \subseteq \ell_{S_B}(N_{S_B}) \cup \{\epsilon\}$. The empty label is introduced because the body of a rule may contain nodes with unspecified tags, that is, blank nodes. Moreover:

1) A rooted TAR(RTAR) is a TAR such that S_B is a rooted subtree of S_H
2) An extended TAR(ETAR) is a TAR such that S_B is an induced subtree of S_H

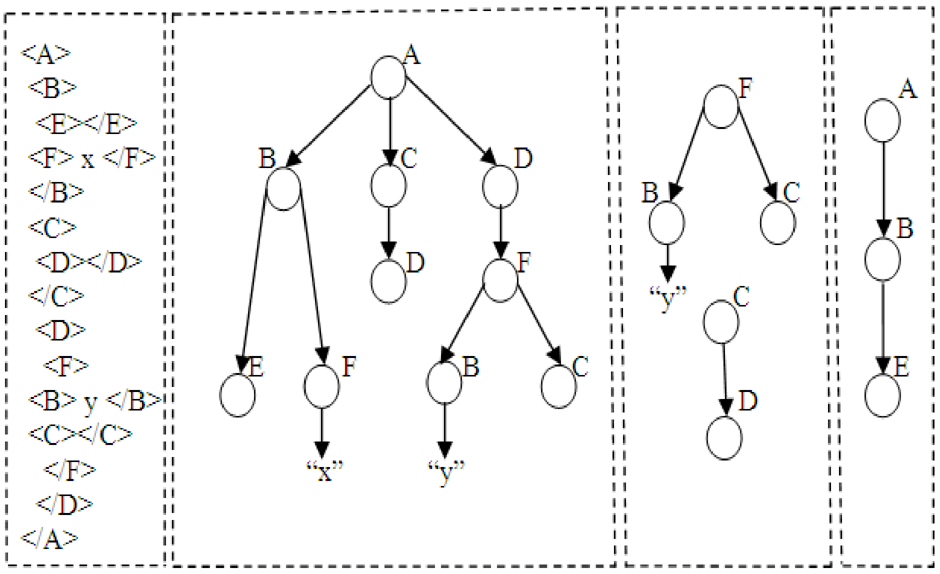

Fig.1 a) An example of XML document, b) Tree-based representation, c) Two induced subtrees, and d) A Rooted subtree.

Let count(S, D) denote the number of occurrences of a subtree S in the tree D and cardinality (D) denote the number of nodes of D. We formally define the support of a TAR $S_B \Rightarrow S_H$ as count (S_H, D)/cardinality (D) and its confidence as a count (S_H, D)/count (S_B, D).

Notice that TARs, in addition to associations between data values, also provide information about the structure of frequent portions of XML documents; thus they are more expressive than classical association rules which only provide frequent correlations of flat values.

Given an XML document, we extract two types of TARs:

1) A TAR is a structure TAR (sTAR) iff, for each node n contained in S_H, $c_H(n) = \perp$, that is, no data value is present in sTARs, i.e. they provide information only on the structure of the document.

2) A TAR, $S_B \Rightarrow S_H$, is an instance TAR(iTAR) iff S_H contains at least one node n such that $c_H(n) \neq \perp$, that is, iTARs provide information both on the structure and on the data values contained in a document.

According to the definitions above we have: structure-Rooted-TARs (sTARs), structure-Extended-TARs (sTARs), instance-Rooted-TARs (iRTARs) and instance-Extended-TARs (iETARs).

Since TARs provide an approximate view of both the content and the structure of an XML document, (1) sTARs can be used as an approximate Data Guide of the original document, to help users formulate queries; (2) iTARs can be used to provide intentional, approximate answers to user queries.

III. TAR EXTRACTION

TAR mining is a process composed of two steps: 1) Mining frequent sub trees, that is, sub trees with a support above a user defined threshold, from the XML document; 2) Computing interesting rules, that is, rules with a confidence above a user defined threshold, from the frequent sub trees. As we discuss the problem of finding frequent sub trees has been widely treated in the literature [1], [3], [16], [2]. Algorithm 1 presents our extension to a generic frequent subtree-mining algorithm in order to compute interesting TARs. The inputs of Algorithm 1 are the XML document D, the threshold for the support of the frequent sub trees minsupp, and the threshold for the confidence of the rules, minconf.

Algorithm 1 Get-Interesting-Rules (*D, minsupp, minconf***)**

```
1: // frequent subtrees
2: $F_S$ = FindFrequentSubtrees ($D$, $minsupp$)
3: ruleSet = $\emptyset$
4: for all $s \in F_S$ do
5: // rules computed from s
6: tempSet = Compute-Rules($s$, $minconf$)
7: // all rules
8: ruleSet = ruleSet $\cup$ tempSet
9: end for
10: return ruleSet
```

Function 1 Compute-Rules (*s, minconf***)**

```
1: ruleSet = $\emptyset$; blackList = $\emptyset$
2: for all $c_s$, subtrees of $s$ do
3: if $c_s$ is not a subtree of any element in blackList then
4: $conf = supp(s) / supp(cs)$
5: if $conf \geq$ minconf then
6: newRule = $\{c_s, s, conf, supp(s)\}$
7: ruleSet = ruleSet $\cup$ {newRule}
8: else
9: blackList = blackList $\cup$ $c_s$
10: end if
11: end if
12: end for
13: return ruleSet
```

Algorithm 1 finds frequent sub trees and then hands each of them over to a function that computes all the possible rules. Depending on the number of frequent sub trees and their cardinality, the amount of rules generated by a naïve Compute Rules function may be very high. Given a subtree with n nodes, we could generate 2^n-2 rules, making the algorithm exponential. This explosion occurs in the relational context too, thus, based on similar considerations, it is possible to state the following property that allows us to propose the optimized version of compute rules shown in function 2.

Remark 1. If the confidence of a rule $S_B \Rightarrow S_H$ is below the established threshold minconf then the confidence of every other rule $S_{Bi} \Rightarrow S_{Hi}$, such that its body S_{Bi} is an induced subtree of the body S_B, is no greater than minconf. Consider Fig 2, which shows a frequent subtree (Figure 2a) and three possible TARs mined from the tree; all the three rules have the same support k and confidence to be determined. Let the support of the body tree of rule (1) be *s*. Since the body trees of rules (2) and (3) are sub trees of the body tree of rule (1), their support is at least *s*, and possibly higher. This means that the confidences of rules (2) and (3) are equal, or lower, than the confidence of rule (1).

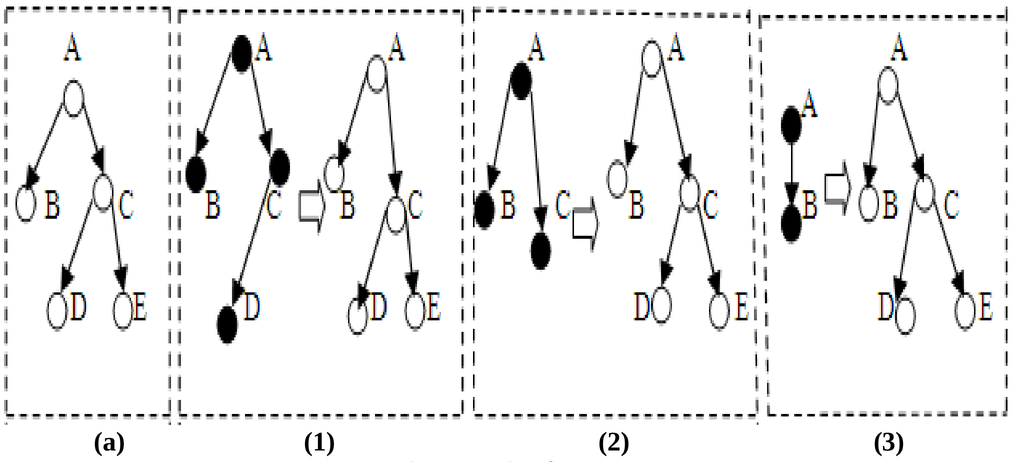

Fig.2. Rule examples for Property 1.

In Function 2 TARs are mined exploiting Remark 1 by generating first the rules with the highest number of nodes in the body tree. Consider two rules *Tr*1 and *Tr*2 whose body trees contain one and three nodes respectively; suppose both rules have confidence below the fixed threshold. If the algorithm considers rule *Tr*2 first, all rules whose bodies are induced subtrees of *Tr*2 will be discarded when *Tr*2 is eliminated. Therefore, it is more convenient to first generate rule *Tr*2 and in general, to start the mining process from the rules with a larger body. Using this solution, we can lower the complexity of the algorithm, though not enough to make it per form better than exponentially. However, notice that the process of deriving TARs from XML documents is only done periodically. Since intensional knowledge represents frequent information, to update it, it is desirable to perform such process after a big amount of updates have been made on the original document. Therefore, in the case of stable documents (that is, those that are rarely updated) the algorithm has to be applied few times or only once (for documents that do not change). Once the mining process has finished and frequent TARs have been extracted, they are stored in XML format. This decision has been taken to allow the use of the same language (XQuery in our case) for querying both the original dataset and the mined rules. Each rule is saved inside a <rule> element, which contains three, attributes for the ID, support and confidence of the rule. Follows the list of elements, one for each node in the rule head. We exploit the fact that the body of the rule is a subtree of the head, and use a Boolean attribute in each node to indicate if it also belongs to the body. Each blank node is described by an element <blank>.

Finally, the rules in the XML file are sorted on the number of nodes of their antecedent; this is an important feature that is used to optimize the answering of queries containing a count operator. One of the (obvious) reasons for using TARs instead of the original document is that processing the document. To take full advantage of this, we introduce indexes on TARs to further speed up the access to mined trees - and in general of intentional query answering. In general, path indexes are proposed to quickly answer queries that follow some frequent path template, and are built by indexing only those paths having highly frequent queries. We start from a different perspective: we want to provide quick, and often approximate, answers also to casual queries.
Given a set R of rules, the index associates, with every path p present in at least one rule of R, the references to rules that contain p in S_H. An index is an XML document containing a set of trees T1,. . , Tn such that each node n of each tree Ti contains a set of references to the rules containing in S_H the path from the root of Ti to n. A TAR-index contains references both to iTARs, sTARs, and is constructed by Algorithm 2.

Algorithm 2 Create-Index (D)

```
1: for all $Di \in D$ do
2:  for all $dj \in Di$ with $j \in \{2, 3 . . . n\}$ do
3:   references (root ($d1$)) = references (root ($d1$)) $\cup$ references (root ($dj$))
4:   sumChildren ($d1$, $dj$)
```

```
5: end for
6: end for
7: return D
```

Function 2 sumChildren (*T*1, *T2*)

```
1: for all x ∈ children (root (T2)) do
2: if ∃ c ∈ children(root(T1)) | c = x then
3: references (root(c)) = references (root(c)) ∪ references (root(x))
4: c = sumChildren(c, x)
5: else
6: addChild (root (T1), x)
7: end if
8: end for
9: return
```

Before applying the algorithm, two sets A and C are constructed containing respectively the antecedent and consequent trees of all the TARs to be indexed. Each tree Ti in the index is annotated in a way that each node contains the reference to the ID of the rule it comes from; then trees are scanned looking for those that have the same root. After this step two sets P = {P1… Pn} and D = {D1... Dm} are obtained that are partitions of A and C respectively, where each Pi and Di contains trees having the same root. Algorithm 3 is applied to merge the trees in each set using the same rationale behind the Data Guide construction procedure [13]. In particular, for each set, the first tree is merged together with the others, that means that the references of its root are added to the references of the roots of the other trees (line 3) and the same procedure is applied recursively to the children of the two roots (line 4). For each node n of the resulting tree a set of references is stored, pointing to the TARs that contain the path from the root of the tree to node n. Once the algorithm is applied to all TARs, the result is a set of trees whose nodes contain references to one or more rules and which are stored in an XML file to be queried later on. Notice that both antecedents and consequents of rules are indexed because we work on the assumption that an answer to a user query is a set of rules whose antecedents or consequents match user requests. However, they are indexed separately because for some categories of user queries we need both of them to provide the answer, while for others we need only antecedents. Detailed explanations will be given in followed section i.e. Section IV.

IV. INTENSIONAL ANSWERS

iTARs(intentional Tree-based Association Rules) provide an approximate intensional view of the content of an XML document, which is in general more concise than the extensional one because it describes the data in terms of its properties, and because only the properties that are verified by a high number of items are extracted. A user query over the original dataset can be automatically transformed into a query over the extracted iTARs. The answer will be intensional, because, rather than providing the set of data satisfying the query, the system will answer with a set of properties that these data “frequently satisfy”, along with support and confidence. There are two major advantages: i) querying iTARs requires less time than querying the original XML document; ii) approximate, intensional answers are in some cases more useful than the extensional ones (see the Introduction). Not all queries lend themselves to being transformed into queries on iTARs; we list three classes of queries that can be transformed by preserving the soundness; moreover, we explain how such transformation can be automatically done.

The classes of queries that can be managed with our approach have been informally introduced in [8] and further analyzed in the relational database context in [4].They includes the main *retrieval* functionalities of XQuery, i.e. path expressions, FLOWR expressions, and the COUNT aggregate operator. We have not considered operators for adding new elements or attributes to the result of a query, because our purpose is to retrieve slender and approximate descriptions of the data satisfying the query, as opposed to modifying, or adding, new elements to the result. Moreover, since aggregate operators require an exact or approximate numeric value as answer, they do not admit intensional answers in the form of implications, thus queries containing aggregators other than COUNT are excluded. Note however that mined TARs allow us to provide *exact answers* to counting queries. The emphasized objects are meta-expressions (queries or variables) which need to be replaced in the actual query.

Class 1: ***σ*/*π*-**queries. Used *to* impose a simple, or complex (containing AND and OR operators), restriction on the value of an attribute or the content of a leaf node, possibly ordering the result. The query imposes some conditions on a node’s content and on the content of its descendants, orders the results according to one of them and returns the node itself. For example, "Retrieve all incidents where country types of incident were used, ordered by the date the incident was reported".

This class of queries is rewritten using Algorithm 5. The result is query q_I that looks for the rules which satisfy the conditions imposed in the where clause and returns those obtained by combining the previous sets using the logical connectives in the same order as in the where clause of q_E, possibly ordered by the variable specified in the order by clause. The query q_I contains calls to the functions references, references and ruleset presented in [9], used to access the TARs and their index. Consider a Class-1 query q_E applied to a document D_E, and its answer E_A. Consider the document D_I containing the extracted TARs. We now show that, if we extract TARs from the answer to query q_E, we obtain a superset of the answers I_A obtained by applying q_I to the TARs D_I extracted from D_E, i.e. the intensional answer constitutes a *representation* of the frequent properties of the extensional one.

Algorithm 3 Class1-Query (v_F ,VW,$CONN$,vOB)

```
1: // the intensional query is empty
2: IQ =ϵ
3: if $VW \neq \emptyset$ then
4:      // get instance rules for paths with a constraint
5:      IQ = IQ • get iTARs($v_F$ ,$VW$,$CONN$,false)
6: else
7: // structure rules for the path without constraint
8:      IQ = IQ • get sTARs($v_F$ )
9: end if
10: // order the results
11: IQ = IQ • "for $r in $Rules/Rule
        order by $r/$v_F$ /$vOB$
        return $r"
12: return IQ
```

Function 3 get iTARs (for, variables, connectives, count)

```
1: Q = ϵ
2: for all $vj \in$ variables do
3: if count = true then
4: // for count queries match only in the antecedent
5: Q=Q•"let $RefI_j:=referencesA(for, $vj$ )"
6: else
7: // for queries without count match both in antecedent and consequent
8: Q=Q • "let $RefI_j:=references(for, $vj$ )"
9: end if
10: end for
11: Q=Q • "let $Rules := "
12: for all $vj \in$ variables, $j \in \{1 . . . n\}$ do
13: Q=Q • "ruleset($RefI_j) $connectivej$ "
14: end for
15: return Q
```

Function 4 get sTARs (variable)

```
1: Q="let $RefS:=references(variable," ")
let $Rules := ruleset($RefS)"
2: return Q
```

Syntax

```
        for variable in path
        [where condition [(and|or) condition]]
        [order by element [asc|desc]]
        return variable
```

Class 2: count-queries. Used to count the number of elements having a specific content. The query creates a set containing the elements which satisfy the conditions and then returns the number of elements in such set. For example, "Retrieve the number of incidents".

This class of queries is rewritten using Algorithm 4. The result is a query q_I that specifies the iTARs, which satisfy the original query conditions, and returns *the support of the first rule, which has been found, divided by its confidence*. Notice that, since rules are ordered according to the number of nodes in their antecedent, the first rule will be either the one, which satisfies all, *and only* the requested conditions or its best approximation (that is, a rule whose antecedent satisfies all the desired conditions and contains the least number of nodes).

Algorithm 4 Class2-Query (*v_F ,VW,CONN*)

```
1: // the intensional query is empty
2: IQ = ϵ
3: // get instance rules for paths with a constraint
4: IQ = IQ• get iTARs($v_F$ ,VW,CONN,true)
5: IQ = IQ • get count()
6: IQ = IQ • “return $supp div $conf”
7: return IQ
```

Function 5 get count **()**

```
1: Q=“let $supp:=$Rules/Rule[1]@support
let $conf:=$Rules/Rule[1]@confidence”
2: return Q
```

Syntax

```
        let $set := ( class 1 query )
        return count($set)
```

Class 3: top-k queries. Used to select the best k answers satisfying a counting and grouping condition. The query counts the occurrences of each distinct value of a variable in a desired set; then orders the variables with respect to their occurrences and returns the most frequent k. For example, “Retrieve the k most used types of country”. This class of queries is rewritten using Algorithm 5. The result will be a query q_I that for each distinct value of a variable finds the corresponding sTARs and uses them to compute the number of occurrences of each value; ranks the values according to the computed count and returns all the rules associated with the first k ranked values.

Algorithm 5 Class3-Query (*vDV ,vF ,VW,CONN*)

```
1: IQ =ϵ // the intensional query is empty
2: IQ = IQ • get sTARs(vDV ) // get instance rules for paths with a constraint
3: IQ = IQ • “for $v in distinct-values($Rules/vDV )”
4: IQ = IQ • get iTARs(vF ,VW,CONN,true)
5: IQ = IQ • get count()
6: IQ = IQ • “order by $supp div $conf descending
return $Rules) [position()<=k]”
7: return IQ
```

Syntax

```
        (for variable in distinct-values(path)
        let $set := ( class 1 query )
        order by count($set) desc
        return variable )[position() <= k]
```

Notice that, in all classes of queries, conditions can be imposed on the descendants of the element that is returned and not on its ancestors. That is, a query containing conditions on the contents of an element is supposed to be as depicted in (where x is the element returned by the query). As shown in Fig.3 given query q_E, a file containing iTARs and the index file, it is possible to obtain the intensional answer in two steps: 1) rewrite q_E into q_I ; 2) apply *qI* on the intensional knowledge. That is: (a) access the index retrieving the references to the rules satisfying the conditions in q_I ; (b) access the iTARs file returning the rules whose references were found in step a). In step 1, we start from the extensional query q_E and apply a rewriting algorithm to obtain the intensional query q_I. We first extract from q_E the following variables and lists:

• v_F , the path in the FOR clause of q_E
• v_{OB}, the variable in the ORDER BY clause of q_E
• v_{DV} , the variable in the distinct-values function of q_E
• *VW* = (*vwj* |*vwj* is a variable of the paths in the WHERE clause of q_E, in the same order*)*
• *CONN* = (*connk*|*connk* is a connective in the WHERE clause of q_E, in the same order)

These objects are the input of Algorithm 3 and its variants, whose output is the intensional query q_I. In the following we describe the algorithm for obtaining the rewritten query q_I for each class of queries. Each algorithm progressively builds the query q_I by concatenating pieces of the query. Notice that the operator • is used to denote concatenation of strings.

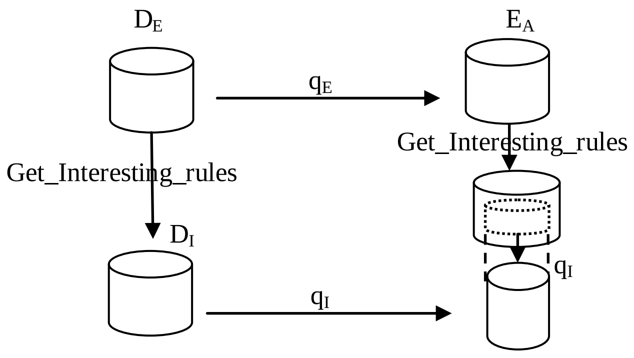

Fig. 3 Intentional query answering.

V. THE TREE RULER PROTOTYPE

Tree Ruler is a tool that integrates the functionalities proposed in our approach. Given an XML document, it enables users to extract intensional knowledge and compose traditional queries as well as queries over the intensional knowledge, receiving both extensional and intensional answers [9][15]. Users formulate XQueries over the original data, and queries are automatically translated and executed on the intensional knowledge.

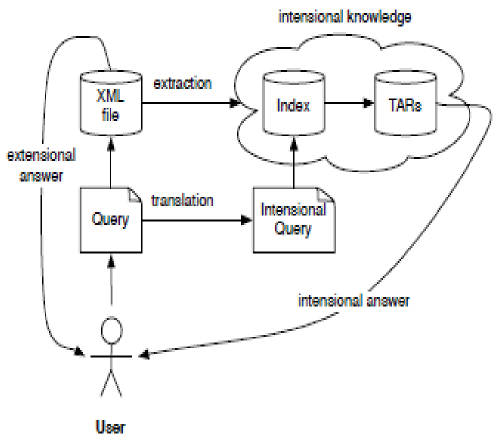

Fig. 4 : Tree Ruler Architecture

Tree Ruler is a tool that integrates the functionalities proposed in our approach. Given an XML document, it enables users to extract intensional knowledge and compose traditional queries as well as queries over the intensional knowledge, receiving both extensional and intensional answers. Users formulate XQueries over the original data, and queries are automatically translated and executed on the intensional knowledge.

- Get the Gist allows intensional information extraction from an XML document, given the support, confidence and the files where the extracted TARs and their index are to be stored.
- Get the Idea allows showing the intensional information as well as the original document, to give users the possibility to compare the two kinds of information.
- Get the Answers allows querying the intensional knowledge and the original XML document. Users have to write an extensional query.

When the query belongs to the classes we have analyzed it is translated and applied to the intensional knowledge. Finally, once it is executed, the TARs that reflect the search criteria are shown in Fig.4.

VI. CONCLUSION AND FUTURE WORK

The main goals we have achieved in this work are: 1) Mine all frequent association rules without imposing any a-priori restriction on the structure and the content of the rules; 2) Store mined information in XML format; 3) Use extracted knowledge to gain information about the original datasets. 4) Casual users can search the data by a keyword without any data base knowledge from XML storage media. We have not discussed the updatability of both the document storing TARs and their index. As an ongoing work, we are studying how to incrementally update mined TARs when the original XML datasets change and how to further optimize our mining efficient algorithm; moreover, for the moment we deal with a (substantial) fragment of XQuery; we would like to find the exact fragment of XQuery, which lends itself to translation into intensional queries.

REFERENCES

[1] Mirjana Mazuran, Elisa Quintarelli, and Letizia tanca. Data Mining for XML query-answering support. *IEEE Transactions on Knowledge and Data Engineering*, Volume:PP Issue:99, 2011

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. of the 20th Int. Conf. on Very Large Data Bases*, pages 487–499. Morgan Kaufmann Publishers Inc., 1994.

[3] T. Asai, H. Arimura, T. Uno, and S. Nakano. Discovering frequent substructures in large unordered trees. In *TechnicalReportDOI-TR216,DepartmentofInformatics,Kyushuuniversity. http://www.i.kyushuu.ac.jp/doitr/trcs216.pdf*,2003

[4] E. Baralis, P. Garza, E. Quintarelli, and L. Tanca. Answering xml queries by means of data summaries. *ACM Transactions on Information Systems*, 25(3):10, 2007.

[5] D. Barbosa, L. Mignet, and P. Veltri. Studying the xml web: Gathering statistics from an xml sample. *World Wide Web*, 8(4):413–438, 2005.

[6] Y. Chi, Y. Yang, Y. Xia, and R. R. Muntz. Cmtreeminer: Mining both closed and maximal frequent subtrees. In *Proc. of the 8th Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, pages 63–73, 2004.

[7] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *Proc. of the 8th ACM Int.Conf. on Knowledge Discovery and Data Mining*, pages 217–228, 2002.

[8] S. Gasparini and E. Quintarelli. Intensional query answering to xquery expressions. In *Proc. of the 16th Int. Conf. on Database and Expert Systems Applications*, pages 544–553, 2005.

[9] M. Mazuran, E. Quintarelli, and L. Tanca, Mining Tree-based association rules from xml documents. In *Technical Report, Politecnico di Milanohttp://home.dei.polimi.it/quintare/Papers/MQT09-RR.pdf*, 2009.

[10] M. Mazuran, E. Quintarelli, and L. Tanca. Mining tree-based frequent patterns from xml. In *Proc. of the 8th Int. Conf. on Flexible Query Answering Systems*, pages 287–299, 2009.

[11] J. Paik, H. Y. Youn, and U. M. Kim. A new method for mining association rules from a collection of xml documents. In *Proc. of Int. Conf. on Computational Science and Its Applications*, pages 936–945, 2005.

[12] A. Termier, M. Rousset, and M. Sebag. Dryade: A new approach for discovering closed frequent trees in heterogeneous tree databases. In *Proc. of the 4th IEEE Int. Conf. on Data Mining*, pages 543–546, 2004.

[13] World Wide Web Consortium. XML Schema, 2001. http://www.w3C.org/TR/xmlschema-1/.

[14] World Wide Web Consortium. XML information Set, 2001. http://www.w3C.org/xml-infoset/.

[15] WWW Consortium. XQuery 1.0: An XML query language, 2007. http://www.w3C.org/TR/xquery.

[16] K. Wang and H. Liu. Discovering typical structures of documents: a road map approach. In *Proc. of the 21st Int. Conf. on Research and Development in Information Retrieval*, pages 146–154, 1998.

AUTHOR'S PROFILES

G. Seshadri Sekhar
M.Tech (CSE),
Madanapalle Institute of Technology and Science (MITS),
Madanapalle - 517325, Andhrapradesh India.

Dr.S. Murali Krishna,
B.Tech, M.Tech, Ph.D Professor & Head CSE,
Madanapalle Institute of Technology and Science (MITS),
P.B.No. 14, Angallu,Madanapalle - 517325, India,
(Chittoor District, Andhrapradesh),Cell No – 9849356444.