Mining Changes from Versions of Dynamic XML Documents

Laura Irina Rusu[1], Wenny Rahayu[2], and David Taniar[3]

[1,2] LaTrobe University, Department of Computer Science & Computer Eng, Australia
lirusu@students.latrobe.edu.au
wenny@cs.latrobe.edu.au
[3] Monash University, School of Business Systems, Clayton, VIC 3800, Australia
David.Taniar@infotech.monash.edu.au

Abstract. The ability to store information contained in XML documents for future reference becomes a very important issue these days, as the number of applications which use and exchange data in XML format is growing continuously. Moreover, the contents of XML documents are dynamic and they change across time, so researchers are looking to efficient solutions to store the documents' versions and eventually extract interesting information out of them. This paper proposes a novel approach for mining association rules from changes between versions of dynamic XML documents, in a simple manner, by using the information contained in the consolidated delta. We argue that by applying our proposed algorithm, important information about the behaviour of the changed XML document in time could be extracted and then used to make predictions about its future performance.

1 Introduction

The increasing interest from various applications in storing and manipulating their data in XML format has determined, during the last few years, a growing amount of research work, in order to find the most effective and usable solutions in this respect. One main focus area was XML warehousing [9, 10], but a large volume of work have been also concentrating on the issue of mining XML documents [7, 8, 11]. The later one evolved in a quite sensitive issue, because the users became interested not only in storing the XML documents in a very efficient way and accessing them at any point in time, but also in getting the most of the interesting information behind the data.

In addressing first part of the problem, i.e. XML warehousing, we have identified at least two types of documents which could be included in a presumptive XML data warehouse: *static XML documents*, which do not change their contents and structures in time (e.g. an XML document containing the papers published in a proceedings book) and *dynamic XML documents*, which change their structures or contents based on certain business processes (e.g. the content of an on-line store might change hourly, daily or weekly, depending on the customer behavior). While the first category of XML documents was the subject of intense research during the recent years, with various methods for storing and mining them being developed, there is still work to be done in finding efficient ways to store and mine dynamic XML documents [1].

The work in this paper continues the proposal made in [1], visually grouped in the general framework presented in Figure 1. In this framework, we focused on both warehousing and mining dynamic XML documents, in three main steps, i.e. (*i*) storing multiple versions of dynamic XML documents (Fig. 1A), (*ii*) extracting historic changes for a certain period of time (Fig.1B) and (*iii*) mining the extracted changes (Fig.1C) to obtain interesting information (i.e. association rules) from them.

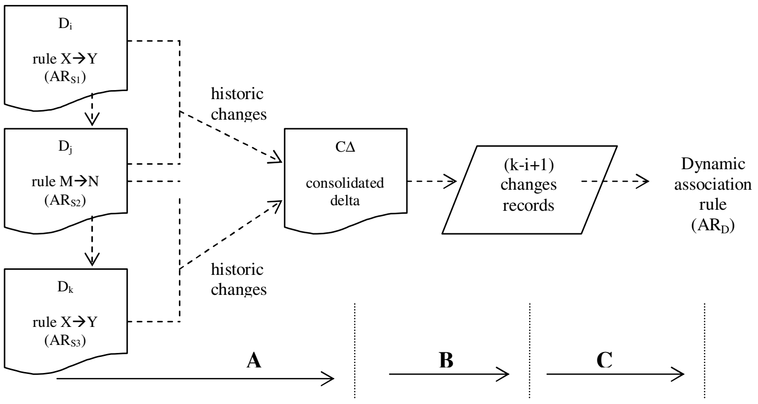


Fig. 1. A visual representation of the mining historic changes process, using consolidated delta

In this paper, we are focusing on the part C of the above mentioned framework, i.e. extracting association rules from changes affecting dynamic XML documents. We believe this knowledge would be very useful in determining if there are any relationships between changes affecting different parts of the documents and making predictions about the future behaviour of the document.

2 Related work

To our knowledge, there is no much work done in the area of mining changes between versions of dynamic XML documents. The existing work is more focused on determining interesting knowledge (e.g. frequently changing structures, discovering association rules or pattern-based dynamic structures) from the multiple versions of the document themselves, not from the actual changes happened in the specified interval of time. We detail below some of this work, noting in the same time that the list of examples is nor complete or exhaustive.

In [2], the authors focus on extracting the FCSs (Frequently Changing Structures). They propose an H-DOM model to represent and store the XML structural data, where the history of structural data is preserved and compressed. Based on the H-DOM model, they present two algorithms to discover the FCSs.

X-Diff algorithm is proposed in [3] and it deals with unordered trees, defined as trees where only the ancestor relationship is important, but not the order of the siblings. This approach is considered to be better and more efficient for the purpose of database applications of the XML. In [3], changes in a XML document over the time are determined by calculating the *minimum-cost edit script*, which is a specific sequence of operations which can transform the XML tree from the initial to the final phase, with the lowest possible cost. In introduces the notion of *node signature* and a new type of matching between two trees, corresponding to the versions of a document, utilized to find the minimum cost matching and cost edit script, able to transform one tree into another.

Another algorithm, proposed by [4], deals with the unordered tree as well, but it goes further and does not distinguish between elements and attributes, both of them being mapped to a set of labeled nodes.

In [5], the authors focus on discovering the pattern-based dynamic structures from versions of unordered XML documents. They present the definitions of *dynamic metrics* and *pattern-based dynamic structure mining* from versions of XML documents. They focus especially on two types of pattern-based dynamic structures, i.e. *increasing dynamic structure* and *decreasing dynamic structure*, which are defined with respect to dynamic metrics and used to build the pattern-based dynamic structures mining algorithm.

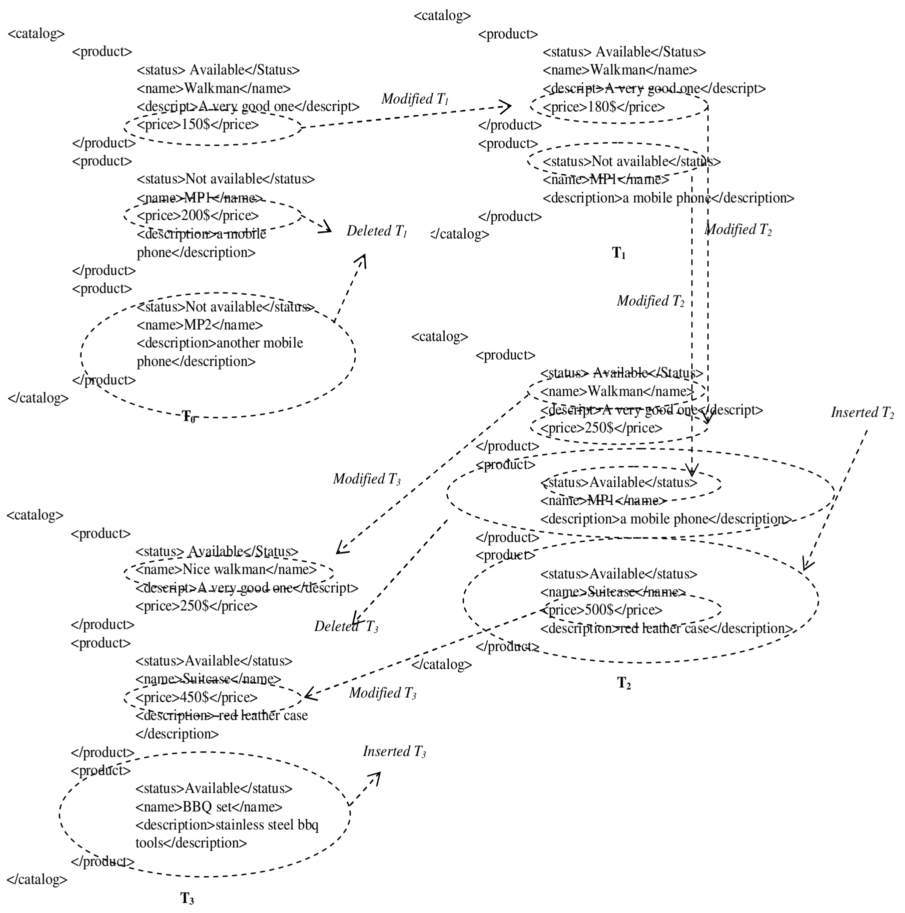
3 Problem Specification

To exemplify the problem, in Figure 2 we present one XML document, at the time T_0 (the initial document), followed by three versions, at three consecutive moments of time, i.e. T_1, T_2 and T_3. Each version brings some changes to the previous one, visually represented by the dotted lines.

One technique for storing the changes of a dynamic XML document (i.e. which changes its context in time) was proposed in [1]. Three main features of this technique are: (*i*) the resulting XML document is much smaller in size than the sum of all versions' sizes; (*ii*) it allows running a simple algorithm to extract any historic version of the document and (*iii*) the degree of redundancy of the stored data is very small, only the necessary information for quick versioning being included.

By running the consolidated delta algorithm [1], we obtain a single XML document containing the historic changes on top of the initial document. We resume here the main steps in building the consolidated delta and few important concepts, for a document changing from the version D_i (at time T_i) to version D_j (at time D_j):

- unique identifiers are assigned for the new inserted elements in the D_i version;
- version D_j is compared with the previous one D_i and for each changed element in the D_j version, a new child element is inserted in the consolidated delta, namely <stamp>, with two attributes: (*a*) "time" , which contain the T_i value (e.g. month, year etc) and (*b*) "delta" , which contain one of *modified*, *added*, *deleted* or *unchanged* values, depending on the change detected at the time T_i; there are some rules to be observed when adding the <stamp> elements [1];
- the D_i version is removed from the data warehouse, as it can be anytime recreated using the consolidated delta. The D_j version is kept until a new version arrives or until a decision to stop the versioning process is taken; *D*j will be removed after the last run of the consolidated delta algorithm;
- at the end of the process, the consolidated delta will contain enough historical information to allow for versioning.


```
<catalog>
    <product>
        <status> Available</Status>
        <name>Walkman</name>
        <descript>A very good one</descript>
        <price>150$</price>
    </product>
    <product>
        <status>Not available</status>
        <name>MP1</name>
        <price>200$</price>
        <description>a mobile
        phone</description>
    </product>
    <product>
        <status>Not available</status>
        <name>MP2</name>
        <description>another mobile
        phone</description>
    </product>
</catalog>
```

```
<catalog>
    <product>
        <status> Available</Status>
        <name>Walkman</name>
        <descript>A very good one</descript>
        <price>180$</price>
    </product>
    <product>
        <status>Not available</status>
        <name>MP1</name>
        <description>a mobile phone</description>
    </product>
</catalog>
```

```
<catalog>
    <product>
        <status> Available</Status>
        <name>Walkman</name>
        <descript>A very good one</descript>
        <price>250$</price>
    </product>
    <product>
        <status>Available</status>
        <name>MP1</name>
        <description>a mobile phone</description>
    </product>
    <product>
        <status>Available</status>
        <name>Suitcase</name>
        <price>500$</price>
        <description>red leather case</description>
    </product>
</catalog>
```

```
<catalog>
    <product>
        <status> Available</Status>
        <name>Nice walkman</name>
        <descript>A very good one</descript>
        <price>250$</price>
    </product>
    <product>
        <status>Available</status>
        <name>Suitcase</name>
        <price>450$</price>
        <description>red leather case
        </description>
    </product>
    <product>
        <status>Available</status>
        <name>BBQ set</name>
        <description>stainless steel bbq
        tools</description>
    </product>
</catalog>
```

Fig. 2. The "catalog.xml" document in four consecutive versions

We need to mention that the D_0 version of the XML document (i.e. the initial one) will be included in the initial consolidated delta; from that point, only the changes for any subsequent version will be recorded in the consolidated delta as described above.

After running the consolidated delta algorithm [1] to capture all the changes affecting the running example document in period T_0 – T_3, we will obtain an XML document where each initial element from D_0 has attached a history of its changes. Note that, if an element was either deleted at updated at a time Ti, $0<i<3$, its children do not have attached any stamp elements for that specific time and this helps in limiting as much as possible the degree of redundancy of the data stored in the consolidated delta.

In our working example, the changes affecting the initial XML document during the consecutive transformations from a version to another are presented in Table 1.

Table 1. The list of changes for the working example, for three consecutive versions of an XML document

$T_0 \rightarrow T_1$	Price – modified; Product – deleted; Price – deleted
$T_1 \rightarrow T_2$	Product – inserted; Price – modified; Status – modified
$T_2 \rightarrow T_3$	Name – modified; Product – deleted; Price – modified; Product – inserted

If, in our working example, we consider the sets of changes in periods $T_0 \rightarrow T_1$, $T_1 \rightarrow T_2$ and $T_2 \rightarrow T_3$ (see Table 1) as transactions, it can be noticed that the pairs "Price-modified" and "Product-inserted" appear in 2 of the 3 (66%) of the transactions. If the minimum support required is set at a level lower than 66%, the association rule extracted would be: "when a *price* element is modified, a *product* is inserted as well, and this happens in 66% of the sets of changes appearing from a version to another".

This paper proposes to build a generic algorithm for extracting association rules from the changes which affect dynamic XML documents, i.e. to discover if there are any relationships between modifications, deletions or insertions of some elements or another. As exemplified above, the resulting rules could be very informative about how parts of the document are changing together so the user can make predictions about the future behaviour of the dynamic XML document.

4 Mining Changes – The Proposed Algorithm

The algorithm for mining changes from historic versions of dynamic XML documents is an improved Apriori one, redesigned to be applicable to XML documents; it has a preparation step and four main working steps, as follows:

```
For each Ti, 0<i<n
  Get all nodes with timestamp Ti
    For each node with timestamp Ti and delta not "unchanged"
      If the delta is "modified" or "inserted"
        If the node has no other children elements except stamp
          Record timestamp, value and delta
        End if
      Else   ' i.e. delta is "deleted"
        Record timestamp and delta
      End if
    Next
Next
```

Fig. 3. Generic algorithm for extracting historic changes (ECD) from consolidated delta document

Preparation step:
Because we want to mine the actual changes which influenced the initial XML document, we use the consolidated delta to extract the set of changes, using the algorithm proposed in Figure 3. We will name the resulting document ECD (the extracted changes document) further in the paper.

For each moment of time *Ti* when the document was changed, $0<i<=n$, we will have a *transaction* containing the elements changed at the time *Ti*. Each combination {element – change} will actually become an *item* in our mining algorithm.

During this step, also count the number of transactions in the ECD. It will be relevant for calculating the support of the association rules discovered.

Step 1:
During this step, a new XML document is built, to store the number of modifications, deletions, insertions for each element E_i in ECD, together with the associated support for each pair "element-change'. The document will be named MCdoc (matrix of changes in the document), further in the paper.

The number of modifications, insertions and deletions will be stored as attributes of each element in MCdoc. Each time a new element is found in the extracted changes document, the *modified*, *inserted* or *deleted* attribute will take value 1. If the same element is found again to be changed during a different transaction, the corresponding attribute will be updated to reflect the current number of changes; the support of the pair {element – change} will be updated too, with regard to the total number of transactions (see preparation step).

If during the preparation step, we also include the paths of the elements in the extracted changes document, the algorithm will be able to identify only the distinct elements and their changes, so elements with same name but different positions in the hierarchy will be recorded separately. The algorithm for Step 1 is detailed in Figure 4.

In our working example, the total number of changes extracted is 10 and the result of applying this step is the document in Figure 6.

```
For each element Ei in ECD
    Search MCdoc for the element Ei
    If not found
        Include it in the MCdoc, with value 1 to the appropriate argument
    Else
        'check if the path is different
        If path attribute is different
            Include it in the MCdoc, with value 1 to the appropriate argument
        Else
            Update the appropriate argument to reflect the change attached
            Update the support of the pair "element – change'
        End if
    End if
Next element Ei
```

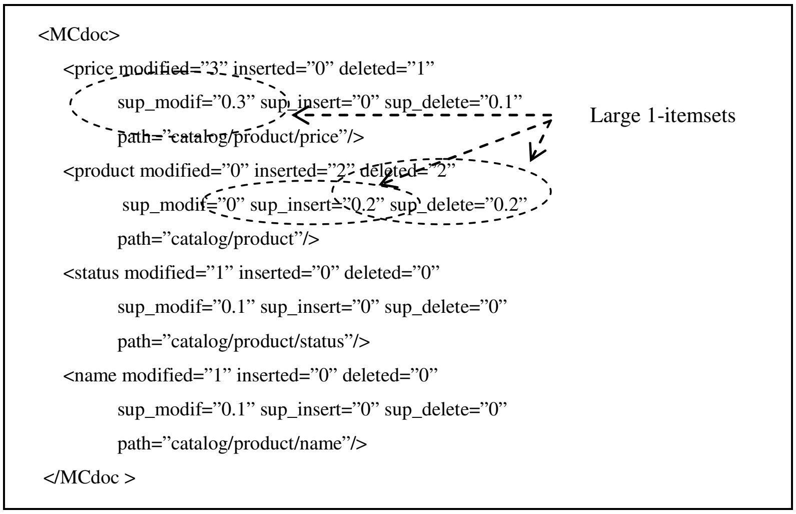
Fig. 4. Step 1 of the proposed algorithm, i.e. building the document with changing items and their support from the ECD (extracted changes document)

```
Set L = set of large 1-itemsets
For each element in MCDoc
  If sup_modif > min_sup
          Add {"element" – modif} pair to L
  If sup_insert > min_sup
          Add {"element" – inserted} pair to L
  If sup_delete > min_sup
          Add {"element" – deleted} pair to L
Next element
```

Fig. 5. Step 2 of the proposed algorithm, i.e. determining the large 1-itemsets

Step 2:
The 1-large itemsets will be extracted from the document build at step 1, i.e. those items (element - change pairs) which have the support in ECD higher than the min_sup set at the beginning of the process (see Figure 5 for the algorithm).

In our example, we set the minimum support to 20%, so the frequent 1-itemsets are three: {price - modified}, {product - inserted} and {product-deleted} (see dotted lines in Figure 6). We emphasise again the fact that, in our proposed algorithm, not the actual elements from the initial document are the items in transactions, but the combinations (pairs) element – change.


Large 1-itemsets

```
<MCdoc>
   <price modified="3" inserted="0" deleted="1"
          sup_modif="0.3" sup_insert="0" sup_delete="0.1"
          path="catalog/product/price"/>
   <product modified="0" inserted="2" deleted="2"
          sup_modif="0" sup_insert="0.2" sup_delete="0.2"
          path="catalog/product"/>
   <status modified="1" inserted="0" deleted="0"
          sup_modif="0.1" sup_insert="0" sup_delete="0"
          path="catalog/product/status"/>
   <name modified="1" inserted="0" deleted="0"
          sup_modif="0.1" sup_insert="0" sup_delete="0"
          path="catalog/product/name"/>
</MCdoc >
```

Fig. 6. The document resulted by applying Step 1 of the proposed mining changes algorithm, highlighting the large 1-itemsets in the working example

Step 3:
Similar with the Apriori-based algorithm, the *k*-itemsets (*k*>1) are built starting from the 1-itemsets; for each of them the support is calculated with respect to the total

number of changes from ECD. This step is repeated until all the large *n*-itemsets are found. This step will be influenced by the observation that any larke *k*-itemset (i.e. which has a support greater than minimum required) needs to have all its subsets large. In figure 7, we show a general *k* step.

In our example, the 2-itemset {price – modified, product - inserted} has the support=0.66, because both pairs appear in two out of three transactions extracted.

```
For each large (k-1) itemset in $L_{k-1}$
  Extend to k itemset by adding a 1-large itemset
    ‘calculate its support in ECD (extracted changes document)
    For each transaction in ECD
      Set bPairFound=false
      For each pair {element-change} in the k itemset
          If (transaction//element and transaction//change) not null
              Set bPairFound= true
        Else
              Set bPairFound= true
      Next pair
      If bPairFound=true  ‘found all pairs from k itemset
          Update $S_k$ - support of k itemset
    Next transaction
    If $S_k$ > min_sup then add k itemset to $L_k$ - the list of large k itemsets
Next (k-1) itemset
```

Fig. 7. Step 3 of the proposed algorithm, i.e. determining the large k-itemsets from the (k-1) large itemsets

Step 4:

Based on the large *n*-itemsets extracted at Step 3, we determine the association rule and calculate their confidence. In the above working example, the rule extracted is “if a price element is modified, a product is inserted as well, and this happens in 66% of the sets of changes appearing from a version to another” and it will have a confidence of 66%.

5 Evaluation of Experimental Results

To evaluate the proposed approach we used the consolidated delta obtained from successive versions of the same XML documents of various sizes, i.e. 25kb, 63kB, and 127kB, data being downloaded from the SIGMOD dataset [6]. Firstly, we built a number of versions for each type of document by using a changes simulator, created by us, which takes as input the D_i version of the document ($0<i<n$) and returns a modified D_{i+1} version, where the desired percentages of deletions, insertions and modifications can be controlled through a user-friendly interface and elements to be changed are randomly chosen. For each new version of each document, we calculate the corresponding consolidated delta, observing the rules in [1];

By applying the algorithm proposed in this paper and implemented in Visual Basic, we got a number of association rules and we made measurements of the running time, for different values of the minimum support required. The graph in Figure 8 shows the time of running for each consolidated delta obtained after few sets of changes (number showed in parenthesis on the graph legend) and for different values of minimum support required.

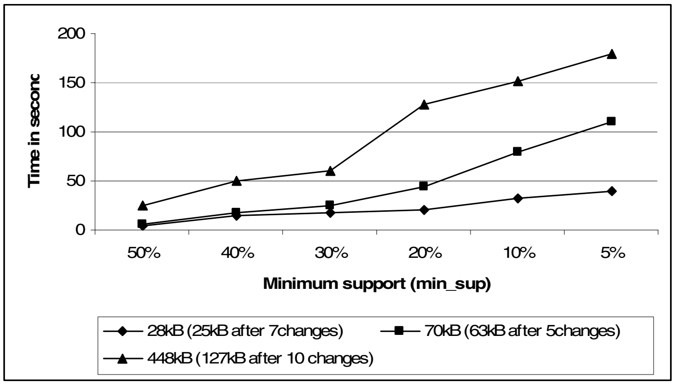


Fig. 8. Running time for three different sized consolidated delta and six different percentages of minimum support imposed

As it can be noticed from the graph, the smaller consolidated delta has the best results as time of running, as one would expect and even for large XML documents (as it is the 448kB consolidated delta) the time is kept under 3 minutes. We need to mention that the first two steps, i.e. preparation step (when the ECD - extracted changes document is built) and step 1 (when the large 1-itemsets are identified) are the most expensive ones, as time and processor resources. Our future work is to explore the possibilities of making this steps more efficient, so the overall performance of the algorithm to be improved.

6 Conclusions

In conclusion, this paper presents a novel approach for mining changes extracted from versions of dynamic XML documents, by looking into the actual changes and into the associations between them. The motivation for this research was that the user not only needs to know which are, for example, the most changing parts of the document, but also which are the relationships between the changes of the document's parts, e.g. modifications of some parts of the document might be related with insertions of some new parts or with deletions of other parts. The information extracted would be very useful to predict the future behaviour of a dynamic XML document. We hence propose an algorithm to mine these changes, in few clear steps, with examples easy to understand and replicate.

References

1. Rusu, L.I., Rahayu W., Taniar D., "Maintaining Versions of Dynamic XML Documents", *Proceed. of The 6th International Conference on Web Information Systems Engineering (WISE 2005)*, New York, LNCS 3806, pp. 536-543, 2005
2. Zhao, Q., Bhowmick, S.S., Mohania, M., Kambayashi, Y., "FCS Mining: Discovering Frequently Changing Structures from Historical Structural Deltas of Unordered XML", *In Proceedings of the 13th Conference on Information and Knowledge Management (CIKM 2004)*, pp. 188-197
3. Wang Y., DeWitt D.J., Cai J.Y., "X-Diff: An Effective Change Detection Algorithms for XML Documents", *In Proceedings of ICDE 2003*, pp.519-530, IEEE Computer Society, 2003
4. Zhao, Q., Bhowmick, S.S., Mohania, M., Kambayashi, Y., "Discovering Frequently Changing Structures from Historical Structural Deltas of Unordered XML*", Proceedings of ACM CIKM'04*, pp.188-197, November 8-13, Washington, US, 2004
5. Zhao, Q., Bhowmick, S.S., Mandria, S., "Discovering Pattern-based Dynamic Structure from Versions of Unordered XML Documents", In *Proceedings of the 6th International Conference on Data Warehousing and Knowledge Discovery (DaWak 2004)*, pp.77-86, Zaragoza, Spain, September 1-3, 2004
6. www.cs.washington.edu/datasets - SIGMOD XML dataset
7. Yin, M., Goh, D.H-L, Lim, E-P., and Sun, A., "Discovery of Content Entities from Web Sites Using Web Unit Mining", *International Journal of Web Information Systems*, 1(3):123-136, 2005.
8. Zhou, B., Hui, S.C., and Fong, A.C.M., "A Web Usage Lattice Based Mining Approach for Intelligent Web Perzonalization", *International Journal of Web Information Systems*, 1(3):137-146, 2005.
9. Quang, N.H., Rahayu, W., "XML Schema Design Approach", *International Journal of Web Information Systems*, 1(3):161-178, 2005.
10. Rusu L.I., Rahayu W., Taniar D., "A methodology for building XML data warehouses", *International Journal of Data Warehousing and Mining, vol.1, no.2*, pp.67-92, April - June 2005
11. Feng.L, Dillon T., "An XML-enabled data mining query language: XML-DMQL", *International Journal of Business Intelligence and Data Mining, vol 1, no 1*, 22-41, 2005-11-30