AUSMS: An environment for frequent sub-structures extraction in a semi-structured object collection

P.A Laur [1] – M. Teisseire [1] – P. Poncelet [2]

[1] LIRMM, 161 rue Ada, 34392 Montpellier cedex 5, France
{laur,teisseire}@lirmm.fr
[2] EMA/LGI2P, Ecole des Mines d'Alès
Site EERIE, Parc Scientifique Georges Besse, 30035 Nîmes cedex 1, France
Pascal.Poncelet@ema.fr

Abstract. Mining knowledge from structured data has been extensively addressed in the few past years. However, most proposed approaches are interested in flat structures. With the growing popularity of the Web, the number of semi-structured documents available is rapidly increasing. Structure of these objects is irregular and it is judicious to assume that a query on documents structure is almost as important as a query on data. Moreover, manipulated data is not static because it is constantly being updated. The problem of maintaining such sub-structures then becomes as much of a priority as researching them because, every time data is updated, found sub-structures could become invalid. In this paper we propose a system, called A.U.S.M.S. (Automatic Update Schema Mining System), which enables us to retrieve data, identify frequent sub-structures and keep up-to-date extracted knowledge after the sources have evolved.

1. Introduction

The search for knowledge in structured data has been extensively addressed in the few past years. Most of the proposed approaches concern flat or highly structured structures. With the growing popularity of the World Wide Web, the number of semi-structured documents produced has quickly soared. In contrast to traditional database applications, where we first describe the structure itself (e.g. the type or the schema) and where we then create instances of these types, within semi-structured data, data has no predefined schema, and each object holds its own structure. In most cases, "on line" documents, such as the HTML/XML, Latex, Bibtex, or SGML files are semi-structured. Consequently, the structure of the objects is irregular and it is judicious to think that a request on the structure of the documents is as significant as a request on the data [22]. However in spite of this structural irregularity, structural similarities among semi-structured objects can exist. It

is frequently noted that semi-structured objects which describe the same type of information have similar structures. The analysis of such implicit structures in semi-structured data can then provide significant information : to optimize the evaluations of requests, to obtain general information on the contents, to facilitate the integration of data resulting from various information sources, to improve storage, to facilitate the installation of index or views and to contribute to the classification of semi-structured documents. Applications fields are very numerous and gather, for example: bio-data processing, Web Content Mining and Web Usage Mining. In this last case, frequent substructures discovered in various users' navigations constitute very useful knowledge to dynamically optimize the hypertext organization of a server or they can be used by a proxy server in order to improve access to pages.
Recently, new approaches have been defined in this context. Very efficient approaches were proposed to seek such substructures [4, 12, 18, 21, 24]. Unfortunately, the handled data are not static because new updates are constantly carried out. The problem of keeping such substructures up to date becomes very significant then because, as updates are carried out, the previously found sub-structures can become invalid.
In this article we are interested in the extraction of such substructures with a detailed attention for their evolution. We propose a system, called AUSMS (Automatic Update Schema Mining System), which allows the collection of data, the search for frequent substructures, and the maintenance of extracted knowledge during evolution of sources.
The article is organized in the following way. In section 2 we present the problems of searching frequent substructures and data maintenance. Section 3 presents the functional architecture of the system by detailing the various stages. We also present some experiments undertaken with the prototype on real data files from the Web. A short related work on the approaches of extracting and maintaining knowledge is proposed in paragraph 4. Lastly, in paragraph 5, we conclude by evoking the continuations from this work.

2. Problem Statement

In this section, we give the formal definitions related to the problem of searching frequent substructures in semi-structured objects.

2.1 Definitions

The goal of our proposal is to discover structural similarities among a set of semi-structured objects. We will consider for the following example a tree as an acyclic connected graph and a forest as an acyclic graph. In our context a cyclic graph can be transformed into an acyclic graph which in itself can be described by a tree while replicating the divided sub nodes [22]. A forest is thus a collection of trees where each tree is a connected component of the forest. An ordered tree is a rooted tree in which the children of each node are ordered. The order is given according to the type of application and it follows either the lexicographical order (set-of), or the imposed order (list-of). To express the differences between orders, we will respectively use the notations "{}" to represent a "set of" and "< >" to represent a "list of". In the rest of the paper, we will assume that we are working with labelised and ordered trees with a common root. Dealing with only rooted trees and two types of orders enables us to address various types of traditional data set such as the data from Web pages.

Example: Let us consider figure 1, where we have two types of order: the first one is lexicographical (address, id) where address has the following child: city, street, zipcode, the second one is imposed by the application: name, firstname (denoted by dotted arrows on figure 1). The representation of the tree is as follows: [root: {address root: {city, street, zipcode}, id: < name, firstname >}].

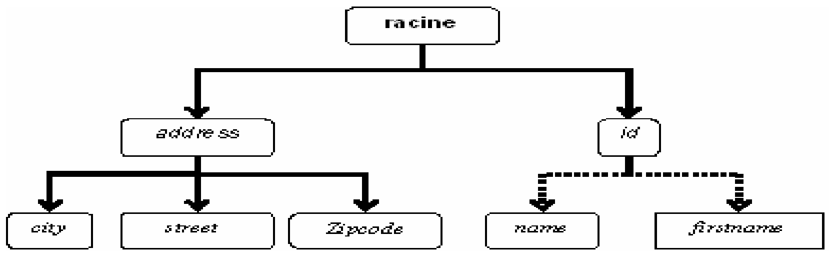


Fig. 1. Example of a tree

Let *r* be the root of tree *T*. Let *x* be a node of *T*. A node *y* from the single path *r* to *x* is named *ancestor* of *x* and is written as $y \leq_k x$, where *k* is the *y* to *x* path length. If *y* is an *x* ancestor, then *x* is a *y* successor (each node is at the same time his ancestor and his successor). If $y \leq_1 x$, i.e. *y*

is a direct ancestor, then y is named parent of x and x is named child of y.
Let T be a tree such that $T = (N, B)$, where N is the labelised node set and B is the edge set. Let a tree $S=(N_s, B_s)$ be an imbricated tree in T, written as $S \leq_{ST} T$ iff : $N_s \leq N$ and $b=(n_x, n_y) \in B_s$ iff $n_y \leq_1 n_x$, i.e. n_x is the n_y child in T. If $S \leq_{ST} T$ then T is included in S or S is a T substructure.

Example: for example, the tree {address: {city, street, zipcode}, category, name} is a subtree of {address: {city, street, zipcode}, category, name, nearby: {category, name, price}}. However the tree {address: {city, street, zipcode}, category, name, price} is not a subtree because the element price is not on the same level in the graph as shown on figure 2.

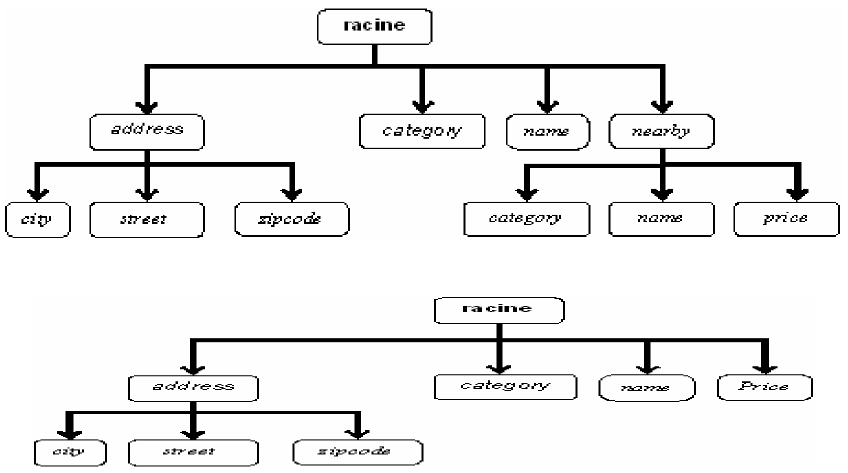


Fig. 2. Sub tree inclusion

2.2 Problem

Let us consider DB a tree database also named structures, i.e. a forest where each tree T is composed of an identifier and a structure included in the forest. All the trees are sorted either by lexicographical order, or by an imposed order. Figure 3 illustrates a database example. Let supp (p) be the support value for a structure corresponding to the number of occurrences of this structure in the database DB. In other words, the

support of a structure *p* is defined as the percentage of all the trees in the database which contain *p*. A tree of the database contains *p* iff *p* is a substructure of this tree. In order to decide whether a structure is frequent or not, a value of minimal support is specified by the user (minSupp) so a structure is frequent if the condition supp (p)≥minSupp holds.
Being given a tree database dB, the problem of searching for regularities in semi-structured data thus consists in finding all the maximum structures which are in dB and whose support is higher than minSupp.

Trans_id	Structure
t_1	*[person :{identity :{address, name}}]*
t_2	*[person :{identity : { address : <street , zipcode >, company, director : <name, firstname >, name}}]*
t_3	*[person : {identity : { address : <street, zipcode>, id }}]*
t_4	*[person :{identity : { address , company, name}}]*
t_5	*[person : { identity : {address, name}}]*
t_6	*[person : {identity : { address : < street , zipcode>, director : <name, firstname >, name }}]*

Fig. 3. A database illustration

Example: In order to illustrate the problem of mining regularities in semi-structured data, let us consider the DB of figure 3. Let us suppose that the support value specified by the user is 50%, i.e. to be frequent; a substructure must appear in at least three trees. The only frequent structures in dB are the following ones: [identity: {address, name}] and [identity: {address: < street, zipcode >}]. The first one appears in t_1 as well as in t_4 and t_5. On the other hand, the structure [identity: {address: < street, zipcode >, director: < name, firstname >, name}] are checked by t_2 and t_6 but is not frequent since the number of trees which hold this structure is lower than the minimal support.
Let us now consider the evolution of the data sources. That is to say db the database increment where new information is added or removed. Let U=DB $\cup$db, be the updated database holding all structures from DB and db. Let L^{DB} be the frequent substructures set in DB. The problem of keeping of up to date discovered knowledge is to seek the frequent substructures in U, noted L^{U}, by respecting the same support value. Moreover, maintenance must take previously extracted knowledge into account so as to avoid restarting retrieval algorithms from scratch when the data are updated.

3. The A.U.S.M.S. System

The aim of A.U.S.M.S. (Automatic Update Schema Mining System) is to propose an environment of discovery and knowledge extraction for semi-structured data from the recovery of information until the update of extracted knowledge. These general principles illustrated by figure 4 are rather similar to those of a process of knowledge extraction. The process can be broken into three principal phases. First of all starting from rough semi-structured data files, a preprocessing is necessary to eliminate the irrelevant data and to ensure their transformation. In the second phase, a knowledge extraction algorithm is used to find the frequent substructures. So as to allow the maintenance of extracted knowledge, the information obtained at the knowledge discovery phase is kept in a database. Lastly, the exploitation by the user of the results obtained is facilitated by a frequent substructures visualization tool.

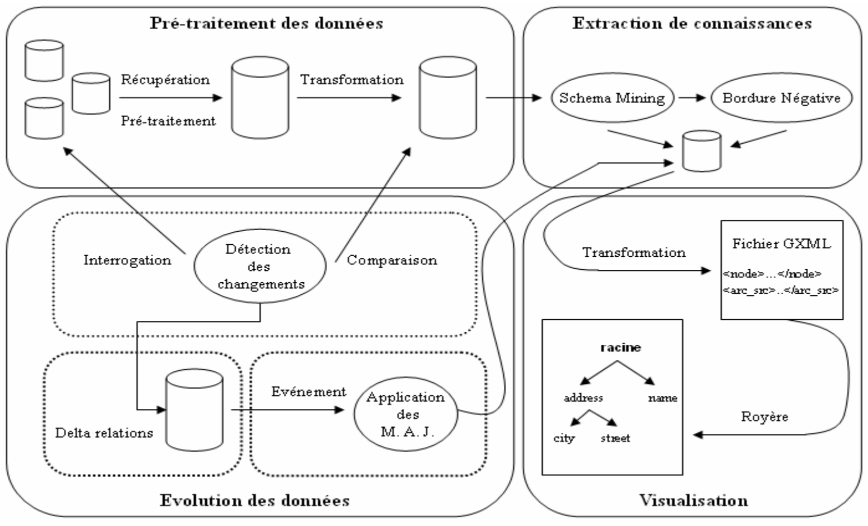


Fig. 4. General Architecture

The various introduced phases are detailed in the following paragraphs.

3.2 Data Pre-processing

From the sources, a process of extraction and transformation is carried out and the extracted data are stored in a database. Within the framework of Web data a process of filtering is carried out so as to eliminate the data which are not useful for the analysis: image, sounds, video.... Moreover, according to the user point of view, the substructures which are not of interest are also removed. For each extracted tree, we associate an identifier which will be used as a primary key. Each extracted structure is transformed before being stored. So as to preserve the levels of overlaps of the various trees, the transformation is carried out in the following way:

- Taking into account depth of overlap and complex type: each extracted element is considered separately, and an integer describing the depth of overlap of this element in the complex structure, is added to the element.
- Creation of simple elements sets lists: when two elements are on the same level and if the first is directly followed by the second, we gather them in the same set otherwise they are included in two separate sets. The notion of order in "list-of", on the other hand, is taken into account by creating new sets between elements. The composite transaction, which results from the union of these sets which have been created from the initial transactions, describes a sequence of modified simple elements and the order of this sequence can then be perceived like a navigation in a "depth-first manner" of the transactions.

At the end of this phase, the various structures are stored in a database.

Example: To illustrate the transformation phase, let us consider the two following trees: $t_1 = \{a, \{c, \{d, f\}\}\}$ and $t_2 = \{a, <e, \{b,f\},d,<,h\ g>>,c\}$. For the illustration, each element is preceded respectively by the letter S for "set-of" and the letter L for "list-of". In the first tree, since all the simple elements appear in a set of values, the S symbol is assigned to them. Concerning the transaction level of overlap, we assign to each simple element its level compared to the highest set of the hierarchy. The simple elements *a*, *b*, *c*, *d* and *f* are then transformed in the following way: Sa_1, Sc_2, Sd_3, Sf_3. By affecting each simple element in a new set and by traversing in-depth first manner the structure t_1, we obtain the following transformation: $t_1 = (Sa_1)\ (Sc_2)\ (Sd_3\ Sf3)$. By applying the same principle for t_2, we obtain: $t_2 = (Sa_1)\ (Le_2)\ (Sb_3\ Sf_3)\ (Ld_2)\ (Lh_3)\ (Lg_3)\ (Sc_1)$. We can note that the modification of t_2 respects the in-depth first order of course. Let us examine in detail the part "list-of" < *g, h*> of t_2. As the elements *g* and *h* are ordered because of the value "set-of", we consider that even if they interpose on the same level, and even if g follows *h* directly, they cannot be gathered in the same set.

3.2 Knowledge Extraction

We showed in [11] that there was a bijection between the problems of searching substructures such as we defined it and that of searching sequential patterns defined in [2]. To find the frequent structures in the database obtained from the previous phase, we use an algorithm largely inspired by those defined for the sequential patterns research and whose general principles are explained below:

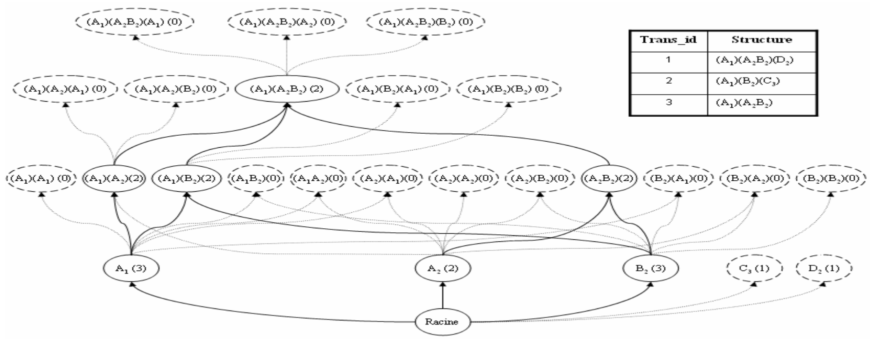
Extraction Algorithm

```
    Input: minimal support (minSupp), a data base DB
    Output: the set L of the maximum frequent structures which check the
    minimal support constraint and a graph G constituting the negative border
    k = 1 ;
    C1 = {{i}/ i ∈ set of atomic elements transformed by the preceding phase}
    while ($C_k \neq \emptyset$ ) do
        for each d ∈ D do VerifyCandidate (d,k) ;
        $L_k$ = { c ∈$C_k$/support (c) ≥ minSupp} ;
        k += 1 ;
        GeneratingCandidate (k) ;
        GenerateBN(G,k) ;
    return $L^{DB}$ where $L^{DB}$ is the union of j=0 at k of $L_j$
```

In a general way, the algorithm carries out a DB traverse to determine which elements play a part frequently enough to be retained. From these size 1 structures, which check the support, we generate size 2 structures which are named candidate structures. A new traverse on the database makes it possible to retain all the candidate structures of size 2 of which the number of occurrences is higher than the minimal support. Then, the algorithm continues in the following way: with each stage k, the database is traversed to count the support of the candidates (Veri-

fyCandidate procedure). From the candidates of which the number of occurrences is higher than the minimal support, the set of the frequent structures is built: L^k. From this set, new candidates can be built (GeneratingCandidate procedure). The algorithm stops when the generation of the candidates procedure provides an empty set or that the VerifyCandidate procedure turns over a set that does not contain frequent substructures.
With an aim of improving the candidate generating procedure as well as the management of candidate elements, we use a bitmap representation inspired by [3]. This structure offers the advantage of considerably reducing the storage space and the ability to generate candidates easily. Moreover it is particularly adapted in the search of long structures. At the time of the search for candidates we also generate the negative border [17]. This is made up of all the structures which are not frequent but whose substructures are frequent. This negative border will be used in the following phase to take into account the data sources evolutions.

Example: Let us consider figure 5 representing the lattice associated with the sample database. For a minimal support of 50 %, on level 1, only the A_1, A_2 and B_3 elements are frequent and can be used to create more complex structures. We thus store in the negative border, the B_2, C_3 and D_2 elements. On level 2, only *(A_1) (A_2)*, *(A_1) (B_2)*, *$(A_2$ $B_2)$* are frequent, we preserve in the negative border those of the preceding level elements which were frequent.



Trans_id	Structure
1	$(A_1)(A_2B_2)(D_2)$
2	$(A_1)(B_2)(C_3)$
3	$(A_1)(A_2B_2)$

Fig. 5. Example of negative border

3.3 Taking into account of data sources evolutions

The negative border obtained in the previous stage enables us to take into account the updates and to maintain extracted knowledge. Indeed, to avoid applying the preceding algorithm again at the time of each update, we store in the negative border the minimal information required to quickly compute the frequent substructures. The taking into account of data sources evolutions follows the general principles which are explained below:

Update algorithm

```
Input: S Set of data sources, BN the negative border, $BN^{Limit}$,
$L^{DB}$ the set
    of frequent structures, minSupp the minimal support specified
by the user
Output: the updated sources S, BN updated and $L^{DB}$ updated
while t $\in$ delay do
foreach s $\in$ S do
if $s_{new} \neq s_{old}$ then
    updateDeltaRelation ($\Delta_s$, opmaj, t)
enddo
$\Delta_S \leftarrow \bigcup_{i=0}^{S} \Delta_s$

if Validate($\Delta_S$, $BN^{Limit}$) then
    Update ($L^{DB}$)
```

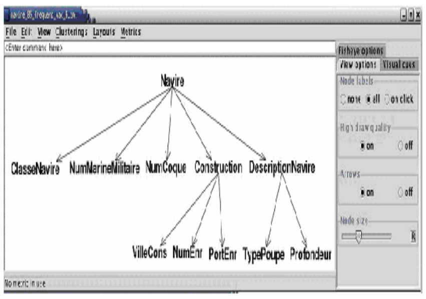
From a time specified by the user (delay), the data sources are compared (s_{old} represents the initial data sources, i.e. during the last analysis and s_{new} represents the data being analyzed). This operation is carried out in the AUSMS system by an agent which acts either in a temporal way (fixed time difference since last update), or in a direct way (user activation). The agent is in charge of comparing the data sources and propagating the modifications. Thus, if the data source was modified, the updates are stored as a Δs set which manages the history of the modifications (UpdateDeltaRelation procedure). This procedure, inspired from the delta relations of the active rules, makes it possible to reflect the side effects of the modifications of the structure, i.e. it contains only the side effect resulting from the modifications [7].

From the information contained as Δs set, a comparison is carried out, by the procedure Validate (Δ_S, BN^{Limit}), with the elements contained in the negative border which are likely to change quickly, i.e. those which can become frequent or not, up to one element. This procedure also takes into account the addition or the suppression of new sources which generate of course a modification of the support value. If one of the conditions is then verified the modifications are brought directly into the negative border to update the set of the frequent structures (procedure Update (L^{DB})). Due to a lack of space, we do not detail this algorithm here (the interested reader can refer to [13]) but we give the general principles of them below.

The first stage consists in deferring the modifications in the negative border as soon as structures are added or removed. Indeed, such an operation causes the calculation of the support value to be modified for the whole base. For each structure, we thus examine the value in the negative border and if this one is lower than the support, the branches of the tree resulting from this structure are pruned. Otherwise the other elements are re-examined and the negative border is updated according to their frequency. When the operations consist of the addition or the removal of elements in existing structures, we analyze the negative border while starting with level 1 so as to verify how frequently the elements appear. If elements become frequent the various levels of the lattice are built recursively with those which were already frequent. If frequent elements become infrequent, the various branches of the lattices resulting from the substructure are pruned. At the end of this phase, the frequent elements are extracted and L^{DB} is updated.

3.4 Visualization

Whereas previous modules are charged to provide and maintain frequent substructures, this module makes it possible to visualize these structures and offers a formalism to describe them. For that, we use, initially, GraphXML [10] which is a graph description language in XML especially designed for drawing and display systems. GraphXML, in addition to providing a language of description makes it possible for the user to add much information with the handled graphs: date, color of the arcs, semi-structured information (for an arc, graph, node). In the second place, so as to visualize at the same time the extracted structures but also their appearance in the data sources, we use "Graph Visualization Framework" [14] which proposes a set of java classes to visualize and handle the structures. This system, via an application nameded Royère, allows the display of the structures described by the GraphXML format.



```
<?xml version="1.0"?>
<!DOCTYPE GraphXML SYSTEM "file:GraphXML.dtd" [
<!ENTITY * admissibleProperties "
support CDATA #IMPLIED
nombreTid CDATA #IMPLIED
dateCalcul CDATA #IMPLIED
nomBase CDATA #IMPLIED ">
]>
<GraphXML>
<graph isDirected="true" isForest="true"
isAcyclic="true" "referredlayout=
"ReingoldTilford" id="Frequent" version="1.0"
vendor="ausms@lirmm.fr" isPlanar="true">
<support>85</support> <nombreTid>728</nombreTid>
<dateCalcul>031002</dateCalcul>
<nomBase>navire</nomBase>

<node name="1">
<label>Navire</label>
</node>
<node name="2">
<label>ClasseNavire</label>
</node>
<edge source="1" target="2">
</edge>
          ... ... ... ... ...
<node name="12" class="std">
<label>Profondeur</label>
</node>
<edge source="10" target="12">
</edge>
</graph> </GraphXML>
```

Fig. 6. Examples of extracted structures

Let us consider figure 6 which represents screenshots of visualized structures. We find at the left a frequent structure resulting from the frequent substructures search with at least 85% of a history of ships database displayed via Royère. On the right-hand side we have the same description within the GraphXML format.

3.4 Illustration

To present the usefulness of our approach, we illustrate below an application on the search for frequent substructures in a set of semi-structured documents. The data used result from a Canadian database located at the address http://daryl.chin.gc.ca:8001/basisbwdocs/sid/title1f.html. This database was created to meet the needs of the managers of the cultural resources in charge of information on archaeological wrecks. It contains information on ships which were registered in Canada or which navigate in Canadian water. It is divided into five sub-databases: Ships, Captains, Owners, Naval manufacturers, Travels. For each one of these databa-

ses, different structures exist. In a general way, the information contained in the database has a depth varying from 2 to 10. In the database, much of the information is incomplete, i.e. many of the fields are left empty.
To integrate this base into our prototype we initially wrote capable analyzers, after sources recovering, to convert this one into a database structure to which we can apply the principles presented in section 3. Thus, for a support of 85 %, we found 5 maximum size frequent substructures (one frequent sub-structure is presented in figure 6). For a support of 100% on this same database there is only one maximum size frequent substructure: *[Navire :{ClasseNavire ,Construction : {NumEnr}, DescriptionNavire{TypePoupe} ,NumCoque,NumMarineMilitaire}].* This schema represents information available for the totality of the ships. We learn for example that for the captains, only information relating to the name is available for the totality of those. It is only for one support of 60% that we will obtain two different maximum frequent sub-structures: *[Capitaine :{ Nom, Pays}]* and *[Capitaine : {Nom, NumeroCertificat}]*. The totality of information is indicated only for a support of 25%. To test the update of knowledge according to the modifications of the data sources, we simulated the addition with certain sources of new information such as for example the knowledge of the countries of origin of the captains. After update of the negative border elements, we found only one new maximum frequent substructure for a support of 60%: *[Capitaine :{ Nom, Pays, NumeroCertificat }].*

4. Related work

To the best of our knowledge, there is little research concerning the structural regularities in large data bases. Nevertheless, our approach is very close to that proposed in [20, 21] for the search for structural association in semi-structured data. The authors propose a very effective approach and solutions based on a new representation of the search space. Moreover, by proposing the optimizations based on strategies of prunings, they improve considerably the stage of generation of the candidates. In the same manner, the approach suggested in [18] is rather similar to the previous approach and uses a particular tree called tag tree patterns. In [4], the authors propose an algorithm called Find-Freq-

Trees which also uses an approach based on a search by level as in the algorithm Apriori [1] and extends the proposal by improving the technique of enumeration defined in [5] so as to discovering substructures in long sequences. Finally in [24], the author proposes two algorithms TreeMinerH and TreeMinerV for the search for frequent trees in a forest. TreeMinerH takes again the principle of the course in width of A-priori by improving the generation and counting of the candidates using the classes of equivalences, of a structure of prefixed tree and "scope list". Whereas in TreeMinerV, it proposes to see a tree like a vertical structure and associates this vision with a method of very effective in-depth course for the search for long sequences. In these two algorithms, the generation and the counting of the candidates are carried out by set operations on the "scope list", the prefixed structure makes it possible to reduce the number of transactions to be traversed in the database. In addition to the taking into account of the evolutions, our approach has however some differences. We are interested in the search for all the structures included in the base whereas they are interested only in search of tree-expression which are defined like trees going from the root to a final leaf of the tree. With this definition, they cannot find regularities of the form *[identity : {address : <street, zipcode>}]* which would be frequent but would be included in a longer transaction which is, itself, not frequent. The search algorithm for substructure used is based on a representation by vectors of bits which also enable us to work on long structures. Other search methods for structures of tree or graphs are also proposed but are not directly applicable to the excavation of semi-structured data. Thus, the authors of [23] propose an algorithm of discovery of approximate common structures and apply it to genomic applications discoveries. Dehaspe and al.[9] present an effective algorithm to solve the problem of frequent substructures discoveries in labellized graphs. Their approach is based on the use of ILP. In [15], an algorithm is also proposed to extract patterns from a directed graph.

With regard to the maintenance of the extracted frequent substructures, there does not exist, to our knowledge, works in this field. We showed that the search for substructures could approach that of sequential patterns. In the continuation of our work, we will thus examine the work carried out around this field. Near to the sequential patterns and the basis of many approaches, [8] proposes an algorithm called FUP, for an excavation of incremental data in the case of the rules of associations.

However, the problems of incremental updates within the framework of the sequential patterns are much more complex than that of the rules of associations insofar as the search space, i.e. the number of combinations is much larger. In [19], the authors propose an algorithm called ISM (Incremental Sequence Mining) which allows an update of the frequent sequences when new customers and new transactions are added to the data base. The suggested approach builds a lattice of sequence which contains all the frequent and negative border elements [17]. When new information arrives, they are added to this lattice. The problem of this approach is obviously the increasing size of the negative border which in our case is minimized, because based on vectors of bits. In [16], the ISE (Incremental Sequence Extraction) algorithm was proposed for the search for frequent patterns, it generates candidates in the entire database by attaching the sequences of the incremental database to those of the original base. This approach avoids keeping the sequences contained in the negative border and the recalculation of these sequences when the initial data base has been updated. However, by not preserving the negative border, it is necessary to more often traverse the base to seek the candidates. In [25] the algorithm proposed uses at the same time the concepts of negative border of the original data base and the concepts of suffixes and prefixes in the contrary of ISE. To control the size of this negative border, they introduce a minimum support for these elements thus reducing its size. Moreover this algorithm realizes an extension by prefix and suffix (using the negative border). The problem of this algorithm lies in the choice of the value of the minimum support for the negative border.

5. Conclusion

In this article, we proposed a functional architecture, AUSMS, of a system of extraction and maintenance of knowledge in bases of semi-structured objects. The originality of the approach lies in the implementation of effective algorithms to extract the frequent substructures in the base from semi-structured objects but also in the taking into account of the handled data. The use of bit vectors for the extraction and the management of the negative border also enable us to optimize the storage and the search of the structures. The tests which we carried out on bases resulting from the Web showed that the adopted approach was very

useful to help the end-user in the analysis of the various handled elements and offered solutions for the search for general information on the data sources, to contribute to the interrogation of bases containing semi-structured data and to help build views and indexes.
We are currently studying the application of AUSMS in data management resulting from Web servers (Web Mining Use) where we wish to analyze the complete behavior of the users. Even if currently, an input in a file "access log" is automatically added each time a request for a source reaches the server, it does not record certain behaviors of the user such as a frequent return or the recharging of a page when the pages are hidden by the navigator or Proxy. For example, the fact that a user is obliged to go back regularly can indicate poor design of the navigation of the server and such information is significant to improve the design of a site. For that, we developed a local user application which stores the user's behavior to transmit it to a database. The idea thus consists in coupling the information obtained in the file log of the navigator with the data base to be used as input with the process of AUSMS. Another research orientation that we are currently carrying out relates to the parallelization of the algorithms of extraction so as to optimize the search of the frequent substructures.

6. References

[1] R. Agrawal, T. Imielinski, and A. swami, "Mining Association Rules between Sets of Items in Large Databases ", Proceedings of SIGMOD'93, pp. 207-216, May 1993.

[2] R. Agrawal and R. Srikant, "Mining Sequential Patterns", Proceedings of International Conference on Data Engineering (ICDE'95), pp. 3-14, Tapei, Taiwan, March 1995.

[3] J. Ares, J. Gehrke, T. Yiu and J. Flannick, " Sequential Pattern Using Bitmap Representation ", Proceedings of PKDD'02, Edmonton, Canada, July 2002.

[4] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto and S. Arikawa, "Efficient substructure discovery from Large Semi-structured Data", Proceedings of the International Conference on Data Mining (ICDM'02), Washington DC, USA, April 2002.

[5] R. J. Bayardo Jr, "Efficiently Mining Long Patterns from Databases", Proceedings of SIGMOD'98, pp. 85-93, Seattle, USA, June 1998.
[7] S. Chawathe, S. Abiteboul and J. Widom, "Representing and Querying Changes History in Semistructured Data ", Proceedings of ICDE'98, Orlando, USA, February 1998.

[8] D. W. Cheung, J. Han, V. Ng and C. Y. Wong, “Maintenance of Discovered Association Rules in Large Databases: an Incremental Update Technique”, Proceedings of ICDE’96, pp. 116-114, New Orleans, USA, February 1996.

[9] L. Dehaspe, H. Toivonen and R. D. King, “Finding Frequent Substructures in Chemical-compounds”, Proceedings of KDD’98, pp. 30-36, New York, USA, August 1998.

[10] I. Herman and M.S. Marshall, “GraphXML An XML based graph interchange format”, Centre for Mathematics and Computer Sciences (CWI), Technical Report Amsterdam, 2000.

[11] P.A. Laur, F. Masseglia and P. Poncelet, “A General Architecture for Finding Structural Regularities on the Web”, Proceedings of the International Conference on Artificial Intelligence (AIMSA'00), September 2000.

[12] P.A. Laur et P. Poncelet. “AUSMS : un environement pour l’extraction de sous-structures fréquentes dans une collection d’objets semi-structurées (in french)”. Actes des Journées d’Extraction et Gestion des Connaissances (EGC’03), Lyon, France, 2003.

[13] P.A. Laur, « Mise à jour des motifs séquentiels : une approche basée sur la bordure négative » , rapport interne, LIRMM, Montpellier, 2002.

[14] M. Marshall, I. Herman and G. Melancon, “An Object-oriented Design for Graph Visualization”, Technical Report, Centre for Mathematics and CS, Amsterdam, 2000.

[15] T. Matsuda, T. Horiuchi, H. Motoda, T. Washio, K. Kumazawa and N. Arai. “Graph Based Induction for General Graph Structured Data”. In Proceedings of the DS’99 Conference, pp. 340-342, 1999.

[16] F. Masseglia, P. Poncelet and M. Teisseire, “Incremental Mining of Sequential Patterns in Large Database”, Actes des 16ièmes Journées Bases de Données Avancées (BDA'00), Blois, France, Octobre 2000.

[17] H. Mannila and H. Toivonen. « On an Algorithm for Finding all Interesting Sequences ». In Proceedings of the 13th European Meeting on Cybernetics and Systems Research, Vienna, Austria, April 1996.

[18] T. Miyahara, T. Shoudai, T. Uchida, K. Takahashi and H. Ueda, “Discovery of Frequent Tree Structured Patterns in Semistructured Web Documents “, Proceedings of PAKDD’01, pp. 47-52, Hong Kong, China, April 2001.

[19] S. Parthasarathy and M. J. Zaki, “Incremental and Interactive Sequence Mining”, Proceedings of the Conference on Information and Knowledge Management (CIKM’99), pp. 251-258, Kansas City, USA, November 1999.

[20] K. Wang and H. Liu, ”Schema Discovery for Semi-structured Data “, Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD’97), pp. 271-274., Newport Beach, USA, August 1997.

[21] K. Wang and H. Liu, “Discovering Structural Association of Semistructured Data”, In IEEE Transactions on Knowledge and Data Engineering , pp. 353-371, January 1999.

[23] J.TL. Wang, B.A. Shapiro, D. Shasha, K. Zhang and C.Y Chang, "Automated Discovery Structures" In Proceedings of KDD'96, pp. 70-75, 1996

[24] M. Zaki, "Efficiently Mining Frequent Trees in a Forest ", Proceedings of SIGKDD'02, Edmonton, Canada, July 2002.

[25] Q. Zheng, K. Xu, S. Ma and W. Lu, "The Algorithms of Updating Sequential Patterns", Proceedings of the International Conference on Data Mining (ICDM'02), April 2002.