Découverte de régularités pour l'intégration de données semi structurées

Pierre-Alain Laur, Xavier Baril

LIRMM – 161, Rue Ada
34392 Montpellier Cedex 5
{laur, baril}@lirmm.fr
http://www.lirmm.fr

Résumé. Cet article présente l'utilisation d'une technique de fouille de données pour aider à la spécification de vues sur des sources XML. Notre langage de vues permet d'intégrer des données XML provenant de sources hétérogènes. Cependant, la définition de motifs sur les sources permettant de spécifier les données à extraire est souvent difficile, car la structure des données n'est pas toujours connue. Nous proposons d'extraire les structures fréquentes dans les données des sources pour spécifier des motifs pertinents à utiliser dans la spécification des vues.

1 Introduction

L'objectif d'un système d'intégration de données est de fournir un accès unifié à différentes sources hétérogènes. Pour cela, on utilise généralement un mécanisme de vues et un langage commun (Halevy, 2003). Le choix d'XML est principalement motivé par deux raisons : (1) tout d'abord c'est un langage flexible qui permet de représenter des données provenant de différents modèles, (2) et ensuite de nombreuses applications actuelles permettent d'exporter leurs données en XML.

Pour intégrer des données XML, nous avons proposé VIMIX : *View Model for Integration of Xml sources* (Baril, 2003). Cependant, la définition de vues XML est souvent difficile car la structure des données des sources n'est pas toujours connue à l'avance. Cet article présente l'utilisation d'un algorithme de découverte de régularités pour aider à la spécification de vues XML. Nous avions présenté dans (Baril et al., 2001) une interface graphique pour la spécification de vues et un mécanisme d'aide qui ne tenait pas compte de la fréquence d'apparition des motifs dans les sources.

La recherche de régularités dans les bases de données a fait l'objet de nombreux travaux ces dernières années. La plupart des approches proposées s'intéressent à des structures plates ou fortement structurées. Cependant, contrairement aux bases de données traditionnelles où l'on décrit d'abord la structure des données (i.e. le type ou le schéma), les données XML peuvent ne pas être validées par une DTD ou un *XML-Schema*.

Un des objectifs de notre proposition est donc de découvrir les régularités structurelles existantes au sein d'un ensemble de documents semi structurés. Un exemple d'un tel ensemble est une source de données hétérogènes en XML. Nous considérons par la suite un arbre comme un graphe connecté acyclique et une forêt comme une collection d'arbres où

chaque arbre est un composant connecté de la forêt (arbre enraciné). Nous ne définirons pas de manière formelle, faute de place, l'inclusion d'une structure dans un arbre, nous conseillons au lecteur de se référer à (Laur et al., 2003).

Soit *DB* une base de données d'arbres aussi appelés structures, i.e. une forêt ou chaque arbre *T* est composé d'un identifiant et d'une structure incluse dans la forêt. Soit *supp (p)* le nombre d'occurrences de cette structure dans la base *DB*. Un arbre de la base de données *DB* contient *p* si et seulement si *p* est une sous structure de cet arbre. Le problème de la recherche de régularité dans les données semi structurées consiste à trouver toutes les structures fréquentes de DB, i.e. toutes les structures qui ont un nombre d'occurrences supérieur au support minimal choisi par l'utilisateur minSupp.

Notre approche de recherche de régularités structurelles est très proche de celle proposée dans (Wang et Liu, 1999). pour la recherche d'associations structurelles dans des données semi structurées. Les auteurs proposent une approche très efficace et des solutions basées sur une nouvelle représentation de l'espace de recherche. Dans (Zaki, 2002), l'auteur propose deux algorithmes TreeMinerH et TreeMinerV pour la recherche d'arbres fréquents dans une forêt. TreeMinerH reprend le principe du parcours en largeur de A-priori en améliorant la génération et le comptage des candidats. Quand à TreeMinerV, il propose de voir un arbre comme une structure verticale et associe à cette vision une méthode de parcours en profondeur très efficace. Cependant cette approche ne s'intéresse pas véritablement à la recherche des mêmes structures dans la mesure où le niveau d'imbrication n'est pas pris en compte. Dans notre algorithme, le principe de construction des candidats permet de minimiser le nombre de candidats générés étant donné que nous recherchons pour étendre uniquement les candidats de profondeur supérieure.

Cet article présente une technique de recherche de régularité dans des données semi structurées pour faciliter leur intégration. Il est organisé de la manière suivante. Dans la section 2, nous présentons l'architecture de notre système, le modèle et les grandes lignes du modèle VIMIX que nous utilisons pour l'intégration. La section 3 explique comment les motifs sur les sources sont générés. Enfin la section 4 contient la conclusion.

2 Architecture du système

La figure 1, illustre le système que nous avons mis en place. Les sources de données hétérogènes semi structurées, sous la forme de documents XML, sont dans un premier temps examinées pour permettre la construction automatique d'un *dataguide*. A partir de celui-ci l'utilisateur choisit un point d'entrée. Le système génère alors les différentes structures à étudier à partir de ce point d'entrée et des sources. L'algorithme de fouilles de données détermine alors pour différentes valeurs de support quels sont les différents motifs fréquents. L'utilisateur choisi parmi ces différents motifs, celui qui va permettre de construire une vue qui envoyée au module VIMIX permettra à l'utilisateur de visualiser les documents des sources après son application.

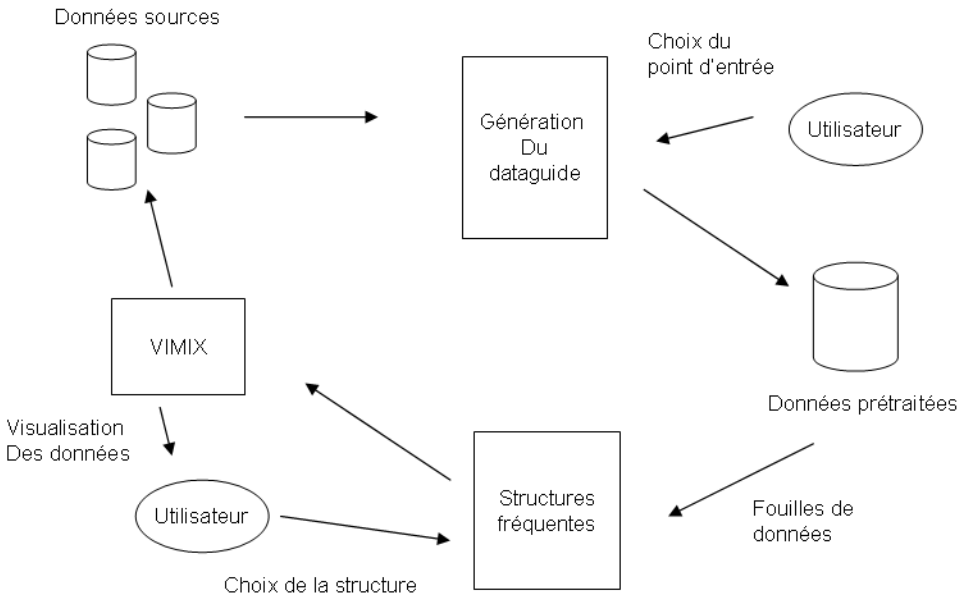


FIG 1 – Architecture

VIMIX (*VIew Model for Integration of Xml sources*) est un modèle de vues permettant de restructurer des données provenant de sources XML (Baril, 2003). La définition de vues VIMIX s'effectue en 3 grandes étapes, illustrées par la figure 2. La première étape intitulée « spécification des données à extraire » consiste à définir des motifs sur les sources qui permettent d'associer des variables aux données des sources. Dans cet article, nous présentons un mécanisme qui permet de générer des motifs sur les sources à partir de régularités découvertes dans les données des sources. Les deux autres étapes permettant de restructurer les données extraites et de spécifier le résultat sont présentées dans (Baril, 2003).

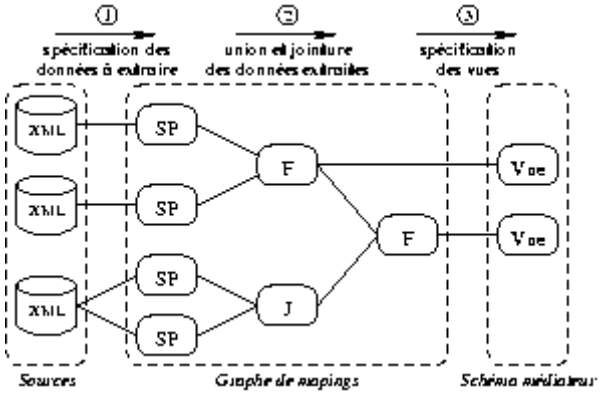
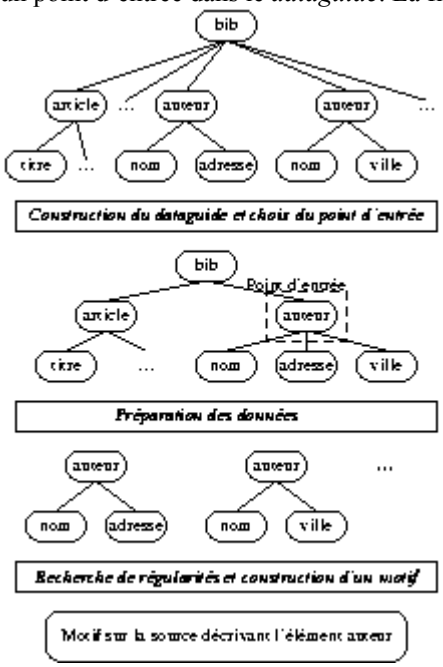


FIG 2 - Définition d'un schéma médiateur avec VIMIX

3 Génération de motifs sur les sources

La construction de motifs sur les sources est souvent difficile car elle nécessite de connaître la structure des données. Pour cela, nous avons proposé dans (Baril et al., 2001) un mécanisme d'aide basé sur la DTD d'une source (si elle existe), ou un *dataguide*. Le concept de *dataguide* a été proposé initialement pour les données OEM (Goldman et al., 1997). Nous avons adapté ce mécanisme pour notre modèle de données XML. Un *dataguide* est un resumé de la structure des données, dans lequel chaque chemin apparaît une seule fois. On peut donc utiliser ce *dataguide* (ou une DTD) pour proposer au concepteur de la vue tous les sous-éléments ou attributs possibles pour un élément donné.

La technique que nous proposons dans cet article permet de générer automatiquement des motifs sur les sources en utilisant un algorithme proposé dans (Laur et al., 2003). L'ensemble des structures à analyser est construit à partir d'une source de données XML en choisissant un point d'entrée dans le *dataguide*. La figure suivante illustre ce mécanisme.



L'exemple est basé sur une source de données bibliographiques. Le point d'entrée choisi dans le *dataguide* est l'élément auteur, pour générer un motif sur la source décrivant les auteurs.

Après avoir choisi un point d'entrée, l'ensemble des éléments contenus dans la source de données est transformé en une collection de structures. A partir de cette collection de structures, une étape de prétraitement permet de stocker les données extraites après filtrage de manière à éliminer celles qui ne sont pas utiles pour l'analyse (informations non pertinentes comme les informations de présentation du document XML). Suit alors, une étape de transformation qui permet de convertir les arbres en séquences (Laur et al., 2003).

Du fait de l'existence d'une bijection (Laur et al., 2000) entre la problématique de la recherche de sous structures et celle de la recherche de motifs séquentiels (Agrawal R. et Srikant R., 1995), pour rechercher les structures fréquentes dans la base obtenue lors de la phase précédente, nous utilisons un algorithme inspiré de ceux définis dans ce domaine. Les principes généraux de celui ci sont expliqués ci dessous :

Algorithme d'extraction
input : un support minimal (minSupp), une base de données DB
output : l'ensemble L des structures fréquentes maximales qui vérifient la contrainte de support minimal

k = 1 ;
C1 = {{i}/ i ∈ ensemble d'éléments atomiques transformés par la phase précédente}
while ($C_k \neq \emptyset$) **do**
    for each d ∈ D **do** VerifyCandidate (d,k) ;
        L_k = { c ∈C_k/support (c) ≥ minSupp} ;
        k += 1 ;
        CandidateGeneration (k) ;
return L^{DB} où L^{DB} est l'union de j=0 à k des L_j

Cet algorithme fonctionne de la manière suivante : à chaque étape *k*, la base *DB* est parcourue pour compter le support des candidats (procedure VerifyCandidate). A partir des candidats dont le nombre d'occurrences est supérieur au support minimal (minSupp), l'ensemble des structures fréquentes est construit : L^k. A partir de cet ensemble, de nouveaux candidats peuvent être construits (procedure CandidateGeneration). L'algorithme s'arrête quand la procédure de génération des candidats fournit un ensemble vide ou que la procédure VerifyCandidate retourne un ensemble vide de fréquents.

Pour appliquer cet algorithme, l'utilisateur spécifie la valeur de support minimal ou l'utilisateur peut choisir une plage de support et un pas. Le système effectuera les différentes recherches de structures fréquentes pour les supports de cette plage. Une fois les différents calculs terminés l'utilisateur sélectionne parmi les résultats un des motifs fréquents qui sera utilisé pour spécifier la vue.

Ce motif permettra après conversion de spécifier un motif sur la source pour définir une vue VIMIX. La phase de transformation, aisée, consiste à traduire la structure fréquente choisie par l'utilisateur dans la syntaxe du langage de spécification de vues VIMIX. Le motif ainsi généré peut être modifié par le concepteur de la vue. Le tableau suivant présente un exemple de motif fréquent et sa traduction en motif sur la source dans le langage VIMIX pour support de 80 %.

Support	Motif fréquent	Motif généré pour VIMIX
80%	{Navire: {NumCoque, Enregistrement:{DateEnr, PortEnr}, DescriptionNavire:{TypePoupe}}	<source-pattern name='sp-navires' source='navires.xml'> <search-axis function='children'> <source-node reg-expression='Navire'> <search-axis function='children'> <source-node reg-expression='NumCoque'/> <source-node reg-expression='Enregistrement'> <search-axis function='children'> <source-node reg-expression='DateEnr'/> <source-node reg-expression='PortEnr'/> </search-axis> </source-node> <source-node reg-expression='DescriptionNavire'> <search-axis function='children'> <source-node reg-expression='TypePoupe'/> </search-axis> </source-node> </search-axis> </source-node> </source-pattern>

4 Conclusion

Dans cet article, nous avons présenté un système, basé sur le modèle de vue VIMIX, qui permet d'intégrer des sources XML dont la structure n'est à priori pas connue. La définition de motifs sur les sources permet de spécifier les données à extraire, cette tâche difficile est facilité par l'utilisation d'un algorithme de fouille de données semi structurées qui recherche les régularités dans la structure d'un document XML. La coopération entre le système de vue VIMIX et un outil de fouilles de données semi structurées montre comment dans le cadre de volumes de données importantes nous pouvons simplifier le choix d'une vue pertinente.

Nous sommes actuellement en train de réaliser des expérimentations sur l'impact d'un tel système par deux types d'utilisateurs : l'un novice et l'autre avisé.

5 Références

Agrawal R. et Srikant R. (1995), Mining Sequential Patterns, Proceedings of the International Conference on Data Engineering (ICDE'95), pp. 3-14, Tapei, Taiwan.

Baril X. et Bellahsène Z. (2001), A Browser for Specifying XML Views, Proceedings of 7th International Conference on Object Oriented Information Systems, pp 164-174.

Baril X. (2003), Un modèle de vues pour intégrer des sources de données XML : VIMIX, Thèse de Doctorat de l'Université Montpellier II, à paraître.

Halevy H. (2003), Data Integration: A Status Report, Proceedings of BTW'2003, Invited paper, pp 24-29.

Laur P.A., Masseglia F. et Poncelet P. (2000), A General Architecture for Finding Structural Regularities on the Web, Proceedings of the International Conference on Artificial Intelligence (AIMSA'00).

Laur P.A., Teisseire M. et Poncelet P. (2003), AUSMS: an Environment for Frequent Sub-Structures Extraction in a Semi-Structured Object Collection, In proceedings of the 14th International Conference on Database and Expert Systems Applications (DEXA03), Prague, Czech Republic, pp. 38-45.

Goldman R. et Widom J. (1997), DataGuides: Enabled Query Formulation and Optimization in Semistructured Databases, Proceedings of 23th International Conference on Very Large Data Bases, pp 436-445.

Wang K. et Liu H. (1999), Discovering Structural Association of Semistructured Data, In IEEE Transactions on Knowledge and Data Engineering, pp. 353-371.

Zaki M. (2002), Efficiently Mining Frequent Trees in a Forest, Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD'02), Edmonton, Canada.

Summary

This paper introduces how to use a data mining approach to help building views over XML data. Our view language allows us to integrate XML data from heterogeneous sources. Meanwhile, the definition of sources patterns task allowing specifying data that have to be extracted is often a hard problem. Indeed, data structures are not always known. We propose an approach to extract frequent structures allowing relevant views specifications.