Vues et mises à jour de données semi-structurées

Une analyse de dépendances

Hicham IDABAL

Centre de Recherche en Informatique
Université Paris 1 Panthéon Sorbonne
90 rue de tolbiac, 75634 Paris Cedex 13
hidabal@univ-paris1.fr

RÉSUMÉ. *Dans ce papier nous étudions le problème classique de l'impact d'une mise à jour sur une vue, dans le cadre de données semi-structurées. Nous faisons les hypothèses suivantes: (i) le document source est modélisé par un arbre ordonné étiqueté par des symboles d'arité variable, (ii) une vue $\mathcal{V}$ est une requête arbre dont l'évaluation sur le document source fournit la vue partielle du document souhaitée (iii) une classe de mises à jour $\mathcal{C}$ est également donnée par une requête arbre sélectionnant les nœuds à modifier. Nous étudions alors le problème suivant: étant donné une requête de vue $\mathcal{V}$ et une classe de mise à jour $\mathcal{C}$ est-il possible de détecter si la vue $\mathcal{V}$ est indépendante de toute mise à jour q de $\mathcal{C}$? Nous montrons que le problème est en général NP-difficile. Nous exhibons une condition suffisante évaluable en temps polynomial assurant l'indépendance d'une vue $\mathcal{V}$ par rapport à une classe de mises à jour $\mathcal{C}$ ainsi que certaines sous-classes de requêtes de vues pour lesquelles le problème devient polynomial.*

ABSTRACT. *In this paper we study the classical problem of the impact of an update on a view defined over semi-structured data. We make the following assumptions: (i) the source document is modelized by an unranked, labeled, ordered tree, (ii) a view $\mathcal{V}$ is a tree query whose evaluation on the source document provides the desired partial view of the document, (iii) a class of updates $\mathcal{C}$ is also given by a tree query selecting the nodes to modify. We then study the following problem : given a view query $\mathcal{V}$ and a class of updates $\mathcal{C}$, is it possible to detect if the view $\mathcal{V}$ is independent of each update q in $\mathcal{C}$? We show that the problem is in general NP-hard. We propose a sufficient condition evaluable in polynomial time ensuring the independence of a view $\mathcal{V}$ with a class of updates $\mathcal{C}$. We then address some subclasses of view queries $\mathcal{V}$ for which the problem becomes polynomial.*

MOTS-CLÉS : *Données semi-structurées, XML, Requêtes , Mises à jour, Automates d'arbres.*

KEYWORDS: *Semi-structured data, XML, Queries, Updates, Tree Automata.*

1. Introduction

Motivés par le rôle clé joué par XML comme standard des échanges d'information sur le Web, les travaux portant sur la gestion des données semi-structurées foisonnent dans la littérature, et font de ce thème l'un des domaines les plus actifs de la recherche en informatique d'aujourd'hui. Dans ce papier nous nous intéressons au problème classique de l'impact d'une mise à jour sur une vue. Une vue permet de présenter partiellement les informations contenues dans un ou plusieurs documents sources afin de répondre par exemple à des besoins de personnalisation ou de confidentialité. Lorsque les données sources sont volumineuses, le coût de l'évaluation de la vue peut être élevé et il est donc important de ne pas recalculer la vue si cela ne s'avère pas nécessaire. Le problème de l'impact consiste précisément à détecter si, après une mise à jour des données sources, le résultat de l'évaluation de la requête de vue risque ou non d'être modifié. Dans le cas où l'on peut à priori obtenir une réponse négative à cette question, une nouvelle évaluation de la vue peut être évitée et par conséquent son coût économisé. Ce problème, classique dans le cadre des bases de données relationnelles ((Gupta *et al.,* 1993), (Griffin *et al.,* 1995) et (Vista, 1996)), a été étudié dans le cadre des bases de données objet ((Ali *et al.,* 2003) et (Scholl *et al.,* 1991)), des données semi-structurées ((Abiteboul *et al.,* 1998), (Zhuge *et al.,* 1998), (Liefke *et al.,* 2000) et (Quan *et al.,* 2000)), et également pour des langages comme XPath, XSLT et XQuery ((Onizuka *et al.,* 2005), (Sawires *et al.,* 2005) et (Dimitrova *et al.,* 2003)). Cependant dans la plupart d'entre eux, le problème est étudié en considérant que l'on dispose des documents sources sur lesquels la vue est évaluée et les méthodes mises en oeuvre utilisent ces documents sources.

Dans ce papier, nous adoptons un autre point de vue, selon lequel nous ne disposons pas à priori des documents sources mais seulement de la requête de vue $\mathcal{V}$ et de la mise à jour. Nous modélisons également la mise à jour sous la forme d'une requête $\mathcal{C}$, permettant de sélectionner les nœuds qui seront modifiés. Dans la mesure où nous ne précisons pas non plus la façon dont les nœuds sélectionnés par $\mathcal{C}$ seront modifiés, $\mathcal{C}$ représente en fait une classe de mises à jour possibles, à savoir toutes les mises à jour ne modifiant que les nœuds sélectionnés par $\mathcal{C}$. L'énoncé du problème que nous étudions est alors le suivant : "Étant donné une vue $\mathcal{V}$ et une classe $\mathcal{C}$ de mises à jour, déterminer si la vue $\mathcal{V}$ est indépendante de $\mathcal{C}$, c'est à dire si toute mise à jour q de $\mathcal{C}$ n'a aucun impact sur l'évaluation de $\mathcal{V}$, quelque soit le document source". Notre principal résultat est de fournir une condition suffisante pour qu'une vue $\mathcal{V}$ soit indépendante d'une classe de mise à jour $\mathcal{C}$. Cette condition est évaluable en temps polynomial par rapport à la taille des données $\mathcal{V}$ et $\mathcal{C}$. Nous montrons en utilisant un résultat de (Miklau *et al.,* 2004) que le problème général de l'indépendance de $\mathcal{V}$ par rapport à $\mathcal{C}$ est NP-difficile. Enfin nous exhibons des sous-classes de requêtes $\mathcal{V}$ pour lesquelles le problème devient polynomial (sans restriction sur le type de requête $\mathcal{C}$ définissant la classe de mises à jour). Dans (Raghavachari *et al.,* 2006), les auteurs abordent, comme dans ce papier, des problèmes similaires sans faire l'hypothèse que l'on dispose des documents sources. Cependant, ils se restreignent aux requêtes XPath de $P^{*,//,[]}$ ((XPath, Novembre 1999) et (Miklau *et al.,* 2004)). Le papier est organisé comme suit : dans la section (2) nous introduisons les concepts de requêtes de vue, de mise à jour et d'indépendance. Nous exhibons dans (3) les principaux résultats, et nous concluons dans (4).

2. Préliminaires

Dans ce travail les données semi-structurées (DSS) considérées sont des documents XML qui seront modélisés par des arbres ordonnés étiquetés sur un alphabet à arité variable Σ. Dans ce modèle, les valeurs textuelles des éléments XML sont ignorées, seuls les éléments définissant la structure du document sont représentés par les noeuds de l'arbre. Formellement, un document $\mathcal{T}$ est un couple $\mathcal{T}$=(D, λ) où D est un domaine d'arbre, (i.e. D est un sous-ensemble de $\mathbb{N}^*$ contenant le mot vide et vérifiant $\forall i \in \mathbb{N}, wi \in D \Rightarrow (w \in D$ and $wj \in D, \forall j < i)$) et λ associe une étiquette $a \in \Sigma$ à chaque noeud w de D. Dans la suite, le domaine D de l'arbre $\mathcal{T}$=(D, λ) sera noté $\mathcal{N}(\mathcal{T})$. Pour chaque noeud w de $\mathcal{N}(\mathcal{T})$, nous notons $\mathcal{T}(w)$, le sous-arbre issu de w dans $\mathcal{T}$ défini par $\mathcal{T}(w) = (D_w, \lambda_w)$ avec $D_w = \{wv/v \in \mathbb{N}^*\}$ et λ_w est la restriction de λ à D_w. Pour des raisons techniques, nous supposerons d'autre part, que le noeud racine est toujours étiqueté par le symbole '/' (voir Figure 1-(a)).

2.1. *Vue, Mises à jour et Indépendance*

Vue : dans le cas de documents XML l'exécution d'une vue se compose principalement de deux étapes : la première étape consiste à extraire du (ou des) document(s) source(s) un ensemble de nœuds contenant les informations souhaitées ; la seconde étape réorganise le résultat obtenu pour lui donner la structure personnalisée attendue et n'intervient donc pas dans l'étude de l'indépendance d'une vue par rapport à un ensemble de mises à jour. Dans ce travail, nous considérons que (1) les données sources sont constituées d'un unique document XML $\mathcal{T}$ et que (2) une requête de vue n-aire $\mathcal{V}$ exprime les conditions qui devront être satisfaites pour qu'un n-uplet de noeuds de $\mathcal{T}$ soit sélectionné par $\mathcal{V}$ pour l'extraction. Dans notre modèle, le résultat de l'extraction d'un noeud w de $\mathcal{T}$ est le sous-arbre, noté $\mathcal{T}(w)$, issu de w dans $\mathcal{T}$. Ainsi l'exécution d'une requête de vue n-aire $\mathcal{V}$ sur le document $\mathcal{T}$ retourne un ensemble de n-uplets de sous-arbres $(\mathcal{T}(w_1), ..., \mathcal{T}(w_n))$ correspondant à l'extraction des n-uplets de noeuds $(w_1, ..., w_n)$ sélectionnés par $\mathcal{V}$.

Exemple 1 : considérons le document semi- structuré $\mathcal{T}$ (Figure 1-(a)) et la requête de vue binaire $\mathcal{V}$ suivante "Donner les (Titre,Auteur) des articles publiés dans un journal". Son évaluation donnera : { $(t_1, s_1), (t_1, s_2)$ } avec $t_1 = \mathcal{T}(00120)$, $s_1 = \mathcal{T}(00121)$ et $s_2 = \mathcal{T}(00122)$ correspondant aux deux couples de sous-arbres construits à partir du titre et des deux auteurs de l'article associé au noeud 0012.

Mises à jour : dans ce travail nous modélisons une mise à jour sur un document XML source $\mathcal{T}$ par une opération qui (1) sélectionne un ensemble de noeuds de $\mathcal{T}$ à mettre à jour et (2) remplace le sous-arbre $\mathcal{T}(w)$ issu de chaque noeud w sélectionné par un nouvel arbre. Ainsi une mise à jour q d'un document semi-structuré est définie par la composition, $q = \sigma \circ \mathcal{C}$, de deux applications : l'application $\mathcal{C}$ sélectionne un ensemble de nœuds w à mettre à jour et l'application σ procède à la mise à jour par le remplacement des sous-arbres $\mathcal{T}(w)$ issus de ces noeuds w, par de nouveaux sous-arbres. L'application $\mathcal{C}$ représente en fait une classe de mises à jour : deux mises à jour q et q' font partie de la même classe de mises à jour si et seulement si elles sont définies à partir de la même application $\mathcal{C}$.

Exemple 2 : soit les deux mises à jour suivantes q_1 : modifier les auteurs d'articles en y ajoutant un N° de téléphone (Tel). Et q_2 : modifier les auteurs d'articles en changeant

Nom en (NomFamille, Prénom).
q_1 et q_2 appartiennent à la même classe $\mathcal{C}$ de mise à jour qui sélectionne les auteurs d'articles pour une modification.

Impact : nous dirons qu'une mise à jour q a un impact sur une vue $\mathcal{V}$ d'un document $\mathcal{T}$ si et seulement si les exécutions de $\mathcal{V}$ sur $\mathcal{T}$ avant et après la mise à jour q ne sont pas identiques. Formellement, si et seulement si : $\mathcal{V}(q(\mathcal{T})) \neq \mathcal{V}(\mathcal{T})$. Ainsi la requête de mise à jour q_1 de l'exemple 2 a clairement un impact sur la requête de vue $\mathcal{V}$ de l'exemple 1, puisqu'elle modifie chacun des sous-arbres s_1 et s_2 extraits.

Indépendance : l'objectif de ce travail est de détecter directement par l'analyse d'une requête de vue $\mathcal{V}$ et d'une mise à jour q l'impact de cette dernière sur le résultat de l'évaluation de $\mathcal{V}$. Il est important de noter ici que l'objectif est de détecter cet impact indépendemment de la donnée du document source qui n'est pas forcément disponible lors de l'analyse. D'autre part, dans ce contexte, la fonction de remplacement utilisée par la mise à jour q pouvant être de tout type, nous prenons l'option de nous focaliser sur la détection d'un impact éventuel sur une vue $\mathcal{V}$ d'une classe de mises à jour $\mathcal{C}$ plutôt que d'une mise à jour q. Le problème s'énonce donc formellement comme suit :
$\mathcal{V}$ est dite indépendante par rapport à la classe de mises à jour $\mathcal{C}$ si et seulement si :
$\forall\ \mathcal{T}$ document source, $\forall$ **q** mise à jour de $\mathcal{C}$, **q** n'a pas d'impact sur $\mathcal{V}$ dans $\mathcal{T}$.

Exemple 3 : (impact/non impact) sur le document $\mathcal{T}$ (Figure 1-(a)), soit la requête de vue $\mathcal{V}'$ suivante : "Donner les (Titre, Nom d'auteur) des articles publiés dans un journal". Cette requête diffère de la requête $\mathcal{V}$ de l'exemple 1 par le fait que seul le nom de l'auteur est extrait avec le titre de l'article. La mise à jour $q2$ (Exemple 2) a clairement un impact sur cette vue puisqu'elle modifie les noms d'auteurs. Ainsi la classe de mises à jour $\mathcal{C}$ a un impact sur la vue $\mathcal{V}'$ puisqu'il existe une mise à jour ($q2$) de la classe qui impacte la vue. Si on remplace la classe $\mathcal{C}$ par la classe de mise à jour $\mathcal{C}'$ qui modifie les auteurs de thèses, la vue $\mathcal{V}'$ est indépendante de la classe $\mathcal{C}'$.

Indépendance dans le contexte d'un schéma : dans de nombreux cas, il est raisonnable de supposer que le schéma contraignant les données sources est connu. Cette donnée supplémentaire peut alors permettre d'affiner l'analyse d'indépendance. Notons $\texttt{valide}(\mathcal{S}c)$ l'ensemble des documents valides par rapport à un schéma $\mathcal{S}c$. Dans ce contexte la définition de l'indépendance est modifiée comme suit : $\mathcal{V}$ est indépendante d'une classe de mises à jour $\mathcal{C}$ dans le contexte du schéma $\mathcal{S}c$ si et seulement si : $\forall\ \mathcal{T} \in \texttt{valide}(\mathcal{S}c)$, $\forall$ **q** mise à jour de $\mathcal{C}$ avec $\mathbf{q}(\mathcal{T}) \in \texttt{valide}(\mathcal{S}c)$, **q** n'a pas d'impact sur $\mathcal{V}$ dans $\mathcal{T}$.

2.2. *Des requêtes arbres pour sélectionner des tuples de noeuds*

Dans la suite nous modélisons le processus de sélection de noeuds, commun aux concepts de vue et de mise à jour, par la notion de requête arbre. Considérons la requête binaire $\mathcal{V}$ de l'exemple 1 représentée par la requête $\mathscr{R}$ (Figure 1-(b)) qui indique de manière schématique les conditions d'extraction d'un couple de nœuds (v, w) de $\mathcal{T}$: intuitivement, (v, w) est extrait de $\mathcal{T}$ si et seulement si il est possible de réaliser un plongement p de l'arbre $\mathscr{R}$ dans le document $\mathcal{T}$, de telle sorte que (1) les noeuds grisés 3 et 4 de $\mathscr{R}$ sont précisément associés par p aux noeuds v et w c'est à dire $p(3) = v$, $p(4) = w$ et (2) pour chaque arc (i_1, i_2) de la requête $\mathscr{R}$, il existe un chemin dans le document source de $p(i_1)$ à $p(i_2)$ dont la succession d'étiquettes

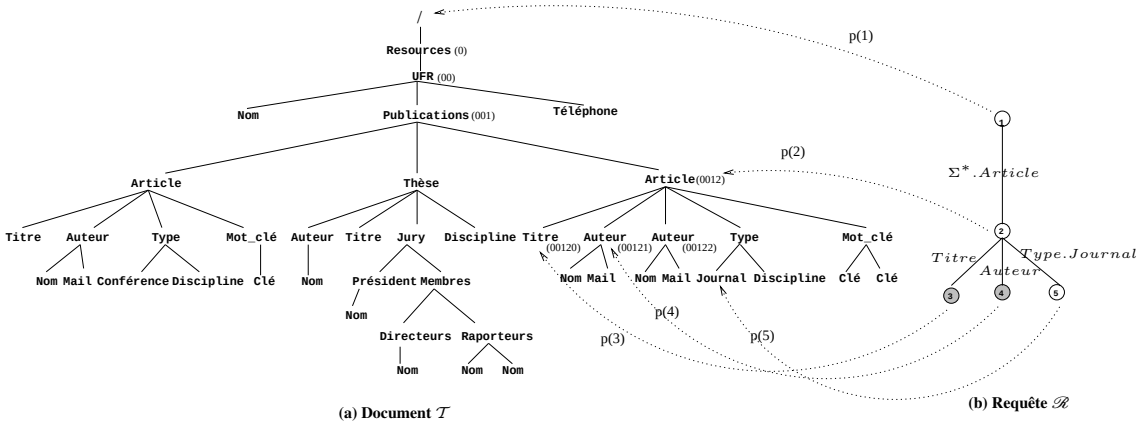

Figure 1. *(a) le document source $\mathcal{T}$, (b) la requête arbre $\mathscr{R}$*

satisfait les contraintes exprimées par l'expression régulière étiquetant l'arc (i_1, i_2) dans $\mathscr{R}$. Plus formellement :

Définition 1. *Soit Σ un alphabet fini d'étiquettes. Une requête arbre n-aire $\mathscr{R}$ sur Σ est définie par : $\mathscr{R} = (\Sigma, N, M, I, \mathcal{E})$ où (N, M) est un arbre (avec N ensemble fini de noeuds et M ensemble d'arcs), $\mathcal{E} : M \longrightarrow REG(\Sigma)$ est une application qui associe à chaque arc (i, j) de M une expression régulière de $REG(\Sigma)$ notée simplement $\mathcal{E}_{(i,j)}$, et I est un n-uplet de nœuds spécifiques $(i_1, ..., i_n)$ représentant les n-uplets de noeuds de sélection.*
La taille de $\mathscr{R}$, notée $|\mathscr{R}|$, est définie par :$|\mathscr{R}| = |N| + \Sigma_{e \in M} |m_e|$ où m_e un automate fini de mots associé à l'expression régulière $\mathcal{E}_e$.

Définition 2. ***a) Plongement*** *: un plongement de la requête arbre $\mathscr{R}$ dans $\mathcal{T}$, est une application p de N dans $\mathcal{N}(\mathcal{T})$ vérifiant : (i) L'image du nœud racine de $\mathscr{R}$ est le nœud racine de $\mathcal{T}$ (étiqueté par '/'). (ii) $\forall e = (i, j) \in M$, il existe un chemin allant de $p(i)$ à $p(j)$ dans $\mathcal{T}$ tel que la suite des étiquettes de ce chemin, excluant $p(i)$ et incluant $p(j)$, est un mot du langage régulier associé à l'expression $\mathcal{E}_{(i,j)}$. (iii) p est injective et préserve l'ordre des nœuds. (exemple de plongement Figure 1).*
b) Trace *: on appelle trace de $\mathscr{R}$ dans le document $\mathcal{T}$ selon le plongement p, le plus petit sous-arbre de $\mathcal{T}$ contenant l'image $p(N)$. On la note : $\texttt{trace}(\mathscr{R}, \mathcal{T})_p$.*
c) Évaluation *: soit $\mathcal{P}$ l'ensemble de tous les plongements de $\mathscr{R}$ dans $\mathcal{T}$. Le résultat, noté $\mathscr{R}_p(\mathcal{T})$, de l'évaluation de $\mathscr{R}$ sur $\mathcal{T}$ suivant le plongement p de $\mathcal{P}$ est le n-uplet d'arbres $(\mathcal{T}(p(i_1)), \ldots, \mathcal{T}(p(i_n)))$ où $(i_1, ..., i_n)$ est le n-uplet de nœuds de sélection I de $\mathscr{R}$. Le résultat, noté $\mathscr{R}(\mathcal{T})$, de l'évaluation de $\mathscr{R}$ sur $\mathcal{T}$ est $\bigcup_{p \in \mathcal{P}} \mathscr{R}_p(\mathcal{T})$.*

Pour des raisons techniques, nous nous restreignons dans ce travail au cas des classes de mises à jour $\mathcal{C}$ définie par des requêtes arbres unaires dont les noeuds de sélection sont des feuilles.

3. Résultats

3.1. *Problème NP-difficile*

Proposition 1. *Décider si $\mathcal{V}$ est dépendante de $\mathcal{C}$ est un problème NP-difficile.*

Démonstration. (Idée) Nous réduisons le problème de la non-inclusion des patterns XPath qui est NP-difficile (cf (Miklau *et al.,* 2004)) au problème de dépendance comme indiqué par la Figure 2. Par abus de notation, sur cette figure, on désigne par p et p' les requêtes arbres équivalentes aux pattern p et p'. Les deux sous arbres t_p et $t_{p'}$ contiennent respectivement une trace de p et une trace de p'. On montre que si $\mathcal{V}$ est dépendante de $\mathcal{C}$ alors $p \nsubseteq p'$: en effet, intuitivement dans le cas contraire (i.e $p \subseteq p'$), tout nœud sélectionné par $\mathcal{V}$ après (respectivement avant) la mise à jour l'est également avant (respectivement après) car tout arbre t_p contenant une trace de p contient une trace de p'. Inversement si $p \nsubseteq p'$, il existe un arbre t_p qui contient une trace de p mais qui ne contient pas de trace de p'. La mise à jour q de $\mathcal{C}$ qui consiste à supprimer le sous arbre issu du nœud étiqueté 'titre' sur l'arbre $\mathcal{T}$ de la Figure 2 démontre la dépendance de $\mathcal{V}$ par rapport à $\mathcal{C}$.

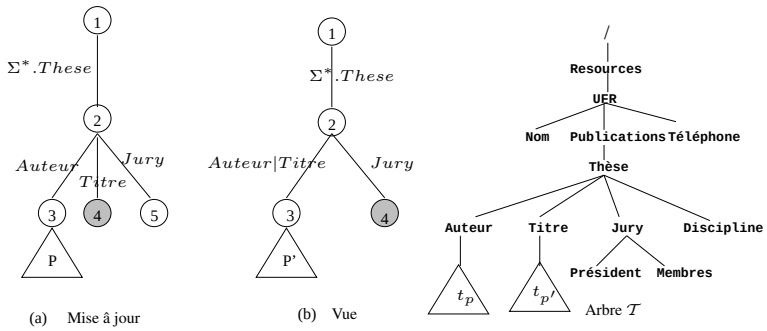

Figure 2.

3.2. *Un critère d'indépendance*

Selon les définitions de la section précédente, une requête de vue $\mathcal{V}$ est dépendante d'une classe de mises à jour $\mathcal{C}$ dans le contexte du schéma $\mathcal{S}c$ si et seulement si il existe $\mathcal{T} \in \texttt{valide}(\mathcal{S}c)$, il existe $q \in \mathcal{C}$ vérifiant $q(\mathcal{T}) \in \texttt{valide}(\mathcal{S}c)$ et il existe un n-uplet $\overrightarrow{n}$ de sous-arbres de $\mathcal{T}$ vérifiant : (1) $\overrightarrow{n} \in \mathcal{V}(\mathcal{T})$ *et* $\overrightarrow{n} \notin \mathcal{V}(q(\mathcal{T}))$
ou (2) $\overrightarrow{n} \notin \mathcal{V}(\mathcal{T})$ *et* $\overrightarrow{n} \in \mathcal{V}(q(\mathcal{T}))$. L'équation (1) signifie qu'un n-uplet $\overrightarrow{n}$ extrait avant la mise à jour, ne l'est plus après : donc il existe une trace, $trace_p(\mathcal{V}, \mathcal{T})$, de $\mathcal{V}$ dans $\mathcal{T}$ selon un plongement p vérifiant l'une des conditions suivantes :

- Un noeud de $trace_p(\mathcal{V}, \mathcal{T})$ a été modifié par q et le n-uplet $\overrightarrow{n}$ n'est plus extrait par $\mathcal{V}$.
- Un arbre de $\overrightarrow{n}$ a été modifié par q et $\overrightarrow{n}$ n'apparaît plus dans le résultat de $\mathcal{V}$.

L'équation (2) signifie qu'un nouveau n-uplet $\overrightarrow{n}$ est extrait après la mise à jour. C'est donc que la mise à jour par q d'un noeud dans $\mathcal{T}$ a créé une trace $trace_p(\mathcal{V}, q(\mathcal{T}))$ de $\mathcal{V}$ dans $q(\mathcal{T})$ selon un plongement p, permettant l'extraction de $\overrightarrow{n}$.

Définition 3. *Soit $\mathcal{L}$ l'ensemble des arbres $\mathcal{T}$ vérifiant :* (i) $\mathcal{T} \in \texttt{valide}(\mathcal{S}c)$
(ii) $\exists$ une trace de $\mathcal{V}$ dans $\mathcal{T}$ (existence d'un plongement p de $\mathcal{V}$ dans $\mathcal{T}$) (iii) $\exists$ une trace de $\mathcal{C}$ dans $\mathcal{T}$ (existence d'un plongement p' de $\mathcal{C}$ dans $\mathcal{T}$) (iv) $\mathcal{T}$ vérifie la condition :
$p'(I_c) \in \mathcal{N}(trace_p(\mathcal{V}, \mathcal{T})) \cup \mathcal{N}(\mathcal{V}_p(\mathcal{T}))$

Proposition 2. *Si $\mathcal{L} = \varnothing$ alors $\mathcal{V}$ est indépendante de $\mathcal{C}$ dans le contexte $\mathcal{S}c$.*

Démonstration. Supposons que $\mathcal{V}$ est dépendante de $\mathcal{C}$ dans le contexte $\mathcal{S}c$. L'assertion (1) ou (2) est donc vérifiée. Or l'assertion (1) implique l'existence d'un plongement p' de $\mathcal{C}$ dans $\mathcal{T}$ et un plongement p de $\mathcal{V}$ dans $\mathcal{T}$ tel que : $p'(I_c) \in \mathcal{N}(trace_p(\mathcal{V}, \mathcal{T})) \cup \mathcal{N}(\mathcal{V}_p(\mathcal{T}))$ d'où $\mathcal{L} \neq \varnothing$.
Si l'assertion (2) est vérifiée, il existe $\mathcal{T} \in \texttt{valide}(\mathcal{S}c)$, q dans $\mathcal{C}$ qui vérifie
$q(\mathcal{T}) \in \texttt{valide}(\mathcal{S}c)$, un plongement p de $\mathcal{V}$ dans $q(\mathcal{T})$ et un plongement p' de $\mathcal{C}$ dans $\mathcal{T}$. Puisque le noeud de mise à jour est une feuille de la requête arbre $\mathcal{C}$, le plongement p' de $\mathcal{C}$ dans $\mathcal{T}$ subsiste dans $q(\mathcal{T})$, i.e. il existe un plongement p'' de $\mathcal{C}$ dans $q(\mathcal{T})$ identique au plongement p' de $\mathcal{C}$ dans $\mathcal{T}$ avec $p'(I_c) = p''(I_c)$. Le plongement p'' vérifie $p''(I_c) \in \mathcal{N}(trace_p(\mathcal{V}, q(\mathcal{T})))$. Ceci implique : $q(\mathcal{T}) \in \mathcal{L}$ et donc $\mathcal{L} \neq \varnothing$.

3.3. *Test polynomial du critère d'indépendance*

La proposition 2 établit que la vacuité du langage $\mathcal{L}$ est une condition suffisante d'indépendance d'une requête de vue $\mathcal{V}$ par rapport à une classe de mises à jour $\mathcal{C}$ dans le contexte $\mathcal{S}c$. Dans cette section, on introduit un automate d'arbres $\mathcal{A}$ qui reconnaît l'ensemble des arbres de $\mathcal{L}$. Pour construire cet automate $\mathcal{A}$, on définit d'abord un automate $\mathcal{A}_{\mathscr{R}}$ qui reconnaît l'ensemble des arbres contenant une trace de la requête $\mathscr{R}$ en utilisant les automates de mots associés aux expressions régulières de $\mathscr{R}$. On modifie ensuite l'automate $\mathcal{A}_{\mathcal{V}}$ (respectivement $\mathcal{A}_{\mathcal{C}}$) pour identifier les noeuds de sélections (respectivement de mises à jour). On effectue le produit de ces deux automates modifiés et enfin le produit du résultat avec l'automate du Schéma $\mathcal{A}_{\mathcal{S}c}$.
L'algorithme proposé consiste alors à tester la vacuité du langage régulier d'arbres $\mathcal{L} = \mathscr{L}(\mathcal{A})$ dont la complexité polynomiale en la taille de $\mathcal{A}$ est connue.

Proposition 3. *(Taille de l'automate $\mathcal{A}$), si a_c et a_v désignent les arités maximales de $\mathcal{C}$ et $\mathcal{V}$ alors il existe une constante C tel que $|\mathcal{A}| \leq C|\Sigma|^2|\mathcal{A}_{\mathcal{S}c}||\mathcal{C}||\mathcal{V}|a_c a_v$.*

Proposition 4. *Le test de vacuité de $L(\mathcal{A})$ est polynomial (en $O(|\Sigma|^4|\mathcal{A}_{\mathcal{S}c}|^2|\mathcal{C}|^2|\mathcal{V}|^2 a_c^2 a_v^2)$) .*

3.4. *Requête arbre de vue linéaire*

Dans le cas où les requêtes de vue sont modélisées par un arbre linéaire, le critère d'indépendance devient une condition nécessaire et suffisante d'indépendance. En effet, dans ce cas, du fait que $\mathcal{V}$ est linéaire, le nœud de mise à jour de tout arbre $\mathcal{T}$ de $\mathcal{L}$ est soit un ancêtre soit un descendant du nœud de sélection. On montre alors qu'il est possible de définir une mise à jour q de $\mathcal{C}$ ayant un impact sur $\mathcal{V}$. Ainsi la condition $\mathcal{L} = \varnothing$ devient nécessaire.

Proposition 5. *Si $\mathcal{V}$ est linéaire alors $\mathcal{V}$ est indépendante de $\mathcal{C}$ dans le contexte $\mathcal{S}c$ si et seulement si $\mathcal{L}$ est vide.*

4. Conclusion

Dans ce travail nous avons utilisé un langage de spécification de requêtes, basé sur les arbres étiquetés par des expressions régulières. Ce langage est suffisamment général et permet en particulier de modéliser certains fragments de XPath notamment

$P^{*,//,[]}$ introduit en (Miklau *et al.,* 2004). Nos résultats peuvent ainsi être appliqués à ces fragments. Notre principale contribution est d'exhiber une condition suffisante d'indépendance entre une requête de vue et une classe de mises à jour, testable en temps polynomial, et qui est nécessaire et suffisante dans le cas particulier de requêtes de vue linéaires. Nous montrons également que le problème d'indépendance est en général NP-difficile. Notre analyse d'indépendance entre classes de mises à jour et requêtes de vue se ramène en fait à une analyse d'indépendance entre deux requêtes arbre et pourrait être utilisée dans d'autres contextes d'application, comme celui de l'étude de la commutativité entre deux requêtes de mises à jour. Ce type d'application a déjà été étudié dans (Ghelli *et al.,* 2007) et dans (Benedikt *et al.,* 2005) dans un but d'optimisation des mises à jour. Cependant les techniques utilisées sont des techniques d'analyse très différentes des notres et le langage de requête étudié incomparable avec celui des requêtes arbre. Ainsi nous pensons que notre approche et nos résultats devraient pouvoir trouver d'autres contextes d'application.

5. Bibliographie

Abiteboul S., McHugh J., Rys M., Vassalos V., Wiener J. L., « Incremental Maintenance for Materialized Views over Semistructured Data », *Very Large Data Bases,*, p. 38-49, 1998.

Ali M. A., Fernandes A. A., Paton N. W., « MOVIE : An incremental maintenance system for materialized object views », *Data & Knowledge Engineering*, vol. 47, p. 131-166, 2003.

Benedikt M., Bonifati A., Flesca S., Vyas A., « Verification of Tree Updates for Optimization. », *CAV*, vol. 3576, Springer, p. 379-393, 2005.

Dimitrova K., El-Sayed M., Rundensteiner E., « Order-sensitive view maintenance of materialized xquery views », ER Conference, 2003.

Ghelli G., Rose K. H., Siméon J., « Commutativity Analysis in XML Update Languages », *ICDT*, p. 374-388, 2007.

Griffin T., Libkin L., « Incremental maintenance of views with duplicates », *Proceedings of the 1995 ACM SIGMOD*, p. 328-339, 1995.

Gupta A., Mumick I. S., Subrahmanian V. S., « Maintaining views incrementally », *ACM SIGMOD international conference on Management of data*, ACM, p. 157-166, 1993.

Liefke H., Davidson S. B., « View Maintenance for Hierarchical Semistructured Data », *Data Warehousing and Knowledge Discovery*, p. 114-125, 2000.

Miklau G., Suciu D., « Containment and Equivalence for a fragment of XPath », *ACM*, vol. 51, p. 2-45, 2004.

Onizuka M., Chan F. Y., Michigami R., Honishi T., « Incremental Maintenance for Materialized XPath/XSLT Views », *14th international conference on World Wide Web*, ACM, 2005.

Quan L., Chen L., Rundensteiner E., « Efficient refresh in an XQL-based web caching system », Workshop on the Web and Databases, 2000.

Raghavachari M., Shmueli O., « Conflicting XML Updates », *Advances in Database Technology - EDBT*, vol. 3896, p. 552-569, 2006.

Sawires A., Tatemura J., Po O., Agrawal D., Candan K. S., « Incremental maintenance of path-expression views », *ACM SIGMOD*, p. 443-454, 2005.

Scholl M. H., Laasch C., Tresch M., « Updatable Views in Object-Oriented Databases », *Proc. 2nd Intl. Conf. on Deductive and Object-Oriented Databases (DOOD)*, number 566, 1991.

Vista D., Optimizing Incremental View Maintenance Expressions In Relational Databases, Thèse de doctorat, University of Toronto, 1996.

XPath W. C., « XML Path Language(XPath) Version 1.0 », Novembre 1999.

Zhuge Y., Garcia-Molina H., « Graph Structured Views and Their Incremental Maintenance », *Proc. 14th IEEE Conf. Data Engineering, ICDE*, IEEE Computer Society, p. 116-125, 1998.