Gestion de versions incertaines de documents XML

Mouhamadou Lamine Ba, Talel Abdessalem, Pierre Senellart

Institut Mines-Télécom
Télécom ParisTech ; CNRS LTCI
46 Rue Barrault, 75013 Paris, France
first.last@telecom-paristech.fr

RÉSUMÉ. *Les systèmes d'édition collaborative sur le Web permettent des interactions à large échelle entre contributeurs afin de faciliter l'enrichissement, l'échange et le partage de contenu. Cette collaboration n'est pas contrainte par une connaissance préalable du niveau d'expertise et de fiabilité des participants. La gestion de versions est donc cruciale pour tracer l'évolution du contenu partagé et la provenance des contributions. Dans de tels systèmes, l'incertitude est malheureusement omniprésente à cause des sources non fiables, des contributions incomplètes et imprécises, des éditions malveillantes et des actes de vandalisme possibles. Pour gérer cette incertitude, nous utilisons un modèle XML probabiliste. Chaque version d'un document partagé est représentée par un arbre XML et le document tout en entier, y compris ses versions, par un document XML probabiliste. Ceci résulte en une mesure d'incertitude sur chaque version et chaque partie du document. Nous montrons que les opérations classiques de gestion de versions peuvent être implantées directement comme opérations sur notre modèle ; son efficacité comparée aux systèmes de contrôle de versions classiques est démontrée sur des données réelles.*

ABSTRACT. *In order to ease content enrichment, exchange, and sharing, web-scale collaborative platforms such as Wikipedia or Google Drive enable unbounded interactions between contributors, without prior knowledge of their level of expertise and reliability. Version control is then essential for keeping track of the evolution of the shared content and its provenance. In such environments, uncertainty is ubiquitous due to the unreliability of the sources, the incompleteness and imprecision of the contributions, the possibility of malicious editing and vandalism acts, etc. To handle this uncertainty, we use a probabilistic XML model as a basic component of our version control framework. Each version of a shared document is represented by an XML tree and the whole document, together with its different versions, is modeled as a probabilistic XML document. We show that standard version control operations can be implemented directly as operations on the probabilistic XML model; efficiency with respect to deterministic version control systems is demonstrated on real-world datasets.*

MOTS-CLÉS : *XML, travail collaboratif, données incertaines, gestion de versions.*

KEYWORDS: *XML, collaborative work, uncertain data, version control.*

DOI:10.3166/ISI.19.4.9-34

1. Introduction

Dans de nombreuses plates-formes d'édition collaborative, au sein desquelles les contributions peuvent émaner de différents utilisateurs, la gestion du contenu est basée sur le contrôle de versions. Un système de contrôle de versions trace les versions aussi bien du contenu que des modifications. Un tel système facilite la résolution d'erreurs survenues durant le processus de révision, l'interrogation de versions antérieures, et la fusion de contenu provenant de contributeurs différents. Tels que résumés dans (Koc, Tansel, 2011 ; Altmanninger *et al.*, 2009), des efforts considérables liés à la gestion de versions des données ont été entrepris à la fois dans la recherche et dans les applications. Les premières applications furent le processus collaboratif de rédaction de documents, la conception assistée par ordinateurs et les systèmes de développement logiciels. Présentement, des outils de contrôle de versions puissants tels que Subversion (Pilato, 2004) et Git (Chacon, 2009) gèrent efficacement de très grands dépôts de code source et des systèmes de fichiers partagés.

Toutefois, les approches actuelles ne supportent pas la gestion de l'incertitude dans les données. C'est le cas de l'incertitude qui résulte de conflits. Les conflits sont fréquents dans les processus d'édition collaborative, en particulier lorsque le cadre est ouvert. Ils apparaissent dès lors que des éditions concurrentes tentent de modifier le même contenu. Les conflits conduisent, donc, à de l'ambiguïté dans la gestion des mises à jour sur les données. Les sources d'incertitudes dans un processus de contrôle de versions ne se limitent pas uniquement aux conflits. En effet, il existe des applications utilisant le contrôle de versions qui sont à la base incertaines. Parmi elles, on peut citer les plates-formes collaboratives Web à large échelle telles que Wikipedia [1] ou Google Drive [2] qui permettent des interactions sans restrictions entre un très grand nombre de contributeurs. Celles-ci se font sans une connaissance préalable du niveau d'expertise et de fiabilité des participants. Dans ces systèmes, le contrôle de versions est utilisé pour tracer l'évolution de contenus partagés et la provenance de chaque contribution. Au sein de tels environnements, l'incertitude est omniprésente à cause des sources non fiables, des contributions incomplètes et imprécises, des éditions malveillantes et des actes de vandalisme possible, etc. Par conséquent, une technique de contrôle de versions capable de manipuler efficacement l'incertitude dans les données pourrait être d'une grande aide pour ce type d'applications. Nous détaillons ci-après les cas d'utilisation possibles.

Les systèmes d'édition collaborative Web à large échelle comme Wikipedia ne définissent aucune restriction pour l'accès en écriture aux données. Il en résulte que les documents multi-versions contiennent des informations qui proviennent de différents utilisateurs. Comme esquissé dans (Voss, 2005), Wikipedia a connu un accroissement exponentiel du nombre de contributeurs et d'éditions par articles. Les caractères libre et ouvert conduisent à des contributions avec différents niveaux de fiabilité et de cohérence en fonction à la fois de l'expertise du contributeur (p. ex., novice ou expert)

1. http ://www.wikipedia.org/
2. https://drive.google.com/

et de la portée du sujet traité. En même temps, les guerres d'édition, les contributions malveillantes telles que les spams et les actes de vandalisme peuvent survenir à tout instant au cours de l'évolution d'un document partagé. Par conséquent, l'intégrité et la qualité de chaque article peuvent être fortement altérées. Les solutions préconisées contre ces problèmes critiques sont la révision des politiques d'accès pour les articles portant sur des sujets très sensibles, ou des solutions évaluant la qualité des contributions sur la base de la réputation des auteurs, de statistiques sur la fréquence de mise à jour des contenus, ou la confiance qu'un lecteur donné a sur l'information (Maniu *et al.*, 2011a ; Calzada, Dekhtyar, 2010 ; Adler, de Alfaro, 2007). Cependant, restreindre les éditions sur les articles Wikipedia à un groupe de contributeurs privilégiés ne résout pas le besoin de représenter et d'évaluer les incertitudes. En effet, ces articles peuvent demeurer incomplets, imprécis ou incertains, faire référence à des vues partielles, des informations fausses ou des opinions subjectives. La réputation des contributeurs ou le niveau de confiance sur les sources constituent des informations utiles pour une estimation quantitative de la qualité des versions, voire de chaque contribution. Cependant, une représentation efficace et préalable des incertitudes dans les versions de documents reste un pré-requis.

Par ailleurs, le filtrage et la visualisation de données constituent également des défis très importants dans les environnements collaboratifs. Les utilisateurs de Wikipedia, par exemple, ne sont pas uniquement des contributeurs, mais également des internautes intéressés par la recherche et la lecture d'articles sous contrôle de versions. Les systèmes actuels forcent les utilisateurs à visualiser soit la dernière version d'un article donné, même si cette dernière pourrait ne pas être la plus pertinente, soit une version à une date bien précise. Les utilisateurs, particulièrement dans les plates-formes de gestion de connaissance universelle comme Wikipedia, voudront sans doute accéder aisément aux versions les plus pertinentes ou celles des auteurs de confiance. Le filtrage de contenu est un des avantages de notre approche. Il peut être effectué de façon commode en cachant les contributions de sources non fiables, par exemple lorsqu'un acte de vandalisme est détecté, ou au moment de l'interrogation en vue de prendre en compte les préférences des utilisateurs et leurs degrés de confiance sur les contributeurs. Alternativement, pour lutter contre la désinformation, il semble utile de présenter aux utilisateurs les versions accompagnées d'informations sur leurs niveaux d'incertitude et celui de chaque partie de leurs contenus. Enfin, les utilisateurs devraient être capables, au moment de la visualisation, de demander un document qui correspond à la fusion de parties prises de différentes versions (p. ex., certaines d'entre elles peuvent être incomplètes, imprécises, et même incertaines prises à part). Nous proposons dans (Abdessalem *et al.*, 2011) une application mettant en évidence de nouvelles possibilités d'interaction et de sélection sur le contenu des pages Wikipedia. En particulier, le contenu d'une page n'est plus réduit à la derniére révision valide, mais correspond à la fusion de plusieurs révisions incertaines.

Vu que le contrôle de versions est incontournable dans les systèmes d'édition collaborative Web incertains à large échelle, la représentation et l'estimation des incertitudes au travers de la gestion de versions de données deviennent cruciales. Ceci dans l'optique d'améliorer la collaboration et de surmonter les problèmes comme la ré-

solution des conflits et la gestion de la fiabilité de l'information. Dans cet article, nous proposons un modèle de contrôle de versions XML incertaines pour les documents arborescents multi-versions dans les contextes d'édition collaborative ouverte. Les données que sont les documents bureautiques, les documents HTML ou XHTML, les formats wiki structurés, etc., manipulés au sein des scénarios d'application cités ont une structure arborescente ou peuvent être transcrits sous ce format ; XML est un encodage naturel pour les données arborescentes.

Les travaux liés à la gestion de versions de documents XML ont surtout porté sur la détection de changements (Rönnau, Borghoff, 2012 ; Lindholm *et al.*, 2006 ; Wang *et al.*, 2003 ; Khan *et al.*, 2002 ; Cobéna *et al.*, 2002). Seul certains, par exemple (Thao, Munson, 2011 ; Rönnau, Borghoff, 2009 ; Rusu *et al.*, 2005), ont proposé un modèle de données semi-structurées complet qui supporte le contrôle de versions. La gestion de l'incertitude a reçu une grande attention au sein de la communauté des bases de données probabilistes (Kimelfeld, Senellart, 2013 ; Suciu *et al.*, 2011), spécialement pour XML à des fins d'intégration de données. En effet, un ensemble de modèles de données XML probabilistes (Van Keulen *et al.*, 2005 ; Nierman, Jagadish, 2002 ; Abiteboul *et al.*, 2009 ; Kharlamov *et al.*, 2010) élaborés avec des sémantiques variées de distribution de probabilité sur les éléments de données, a été proposé. Ces modèles couvrent des techniques simplistes, par exemple (Van Keulen *et al.*, 2005), ne modélisant que des contraintes de dépendances locales (p. ex., l'exclusion mutuelle) entre nœuds et d'autres plus complexes qui prennent en compte une plage plus étendue de contraintes ainsi que des corrélations globales possibles. Un cadre général XML probabiliste, qui généralise tous les modèles XML incertains proposés dans la littérature, est défini par (Abiteboul *et al.*, 2009) et (Kharlamov *et al.*, 2010). Ce cadre introduit la notion de documents probabilistes (ou, plus court, p-documents).

Nous prenons en compte, dans notre cadre, l'incertitude dans les données au travers d'un modèle XML probabiliste comme élément de base de notre système de contrôle de versions. Chaque version d'un document partagé est représentée par un arbre XML. À un niveau abstrait, nous décrivons un document XML multi-version avec des données incertaines en utilisant des événements aléatoires, des scripts d'édition XML qui leur sont associés et un graphe acyclique orienté de ces événements. Pour une représentation concrète, le document tout entier, y compris ses différentes versions, est modélisé comme un document XML probabiliste correspondant à un arbre XML dont les branches sont annotées par des formules propositionnelles sur les événements aléatoires. Chaque formule propositionnelle décrit à la fois la sémantique des éditions incertaines (insertion et suppression) effectuées sur une partie donnée du document et sa provenance dans le processus de contrôle de versions. L'incertitude est évaluée à l'aide du modèle probabiliste et de la valeur de fiabilité associée à chaque source, chaque contributeur, ou chaque événement d'édition. Ceci résulte en une mesure d'incertitude sur chaque version et chaque partie du document. Le graphe acyclique orienté d'événements aléatoires maintient l'historique de l'évolution du document en traçant ses différents états et leurs liens de dérivation. Comme dernière contribution majeure de cet article, nous montrons que les opérations standard de contrôle de versions, en particulier les opérations de mise à jour et de fusion, peuvent

être implantées directement comme opérations sur le modèle XML probabiliste ; son efficacité en comparaison aux systèmes déterministes de contrôle de versions tels que Git et Subversion est démontrée sur des données réelles.

Après quelques préliminaires dans la section 2, nous revisitons le modèle XML probabiliste que nous utilisons dans la section 3. Nous détaillons le modèle de contrôle de versions XML probabiliste proposé et les principales propriétés sous-jacentes dans la section 4. Les sections 5 et 6 présentent la traduction des opérations de contrôle de versions usuelles, à savoir la mise à jour et la fusion respectivement, dans notre modèle. Dans la section 7, nous prouvons l'efficacité de notre modèle en le comparant à des systèmes de contrôle de versions déterministes au travers d'expériences sur données réelles. Nous esquissons par la suite (voir section 7.2) quelques-unes des capacités de filtrage de contenu inhérentes à l'approche probabiliste adoptée.

Les idées initiales qui ont conduit à ce travail ont été présentées comme un article de séminaire de doctorants dans (Ba *et al.*, 2011) ; le présent article est une extension (et traduction) de (Ba *et al.*, 2013b ; 2013a) auquel le lecteur peut se reporter pour les preuves de certains des résultats énoncés dans cet article.

2. Préliminaires

Nous présentons dans cette section les notions de base du contrôle de versions et la classe de documents XML semi-structurés qui sous-tendent notre proposition.

Un document multi-version fait référence à un ensemble de versions d'un même document dans un processus de contrôle de versions. Chaque version du document représente un état (instance) donné de l'évolution du document versionné. Un modèle de contrôle de versions classique est construit sur les notions fondamentales suivantes.

2.1. Concepts de base du contrôle de versions : versions et espace des versions

Par convention, une version désigne une copie d'un document sous contrôle de versions. Des opérations de dérivation lient les différentes versions d'un document. Une dérivation consiste à créer une nouvelle version en copiant d'abord une version existante avant de la modifier. Certaines versions, représentant des variantes, dérivent de la même origine. Les variantes (versions parallèles) caractérisent un historique d'édition non-linéaire avec plusieurs branches distinctes. Dans cet historique, une branche est une séquence linéaire de versions. Au lieu de stocker le contenu complet de chaque version, la plupart des techniques de contrôle de versions maintiennent uniquement les *diffs* entre les états, avec des méta-informations. Ces états (ou *commits* dans le monde Git (Chacon, 2009)) modélisent les différents scripts de mises à jour qui ont été explicitement validés à des instants distincts du processus de contrôle de versions. Un état vient aussi avec des informations sur le contexte (p. ex., auteur, date, commentaire) dans lequel ces modifications ont été faites. Il en résulte que chaque version est fonction de l'historique complet menant à un état donné. Nous adopterons ici la même approche pour modéliser les différentes versions d'un document.

Puisque le contenu de chaque version n'est pas complètement stocké, il doit y avoir un moyen de le retrouver en cas de besoin. L'espace des versions [3] représente l'histoire des éditions sur le document versionné (p. ex., l'historique des versions wiki comme décrit dans (Sabel, 2007)). Il contient toutes les information nécessaires liées aux versions et leurs dérivations. Comme indiqué ci-dessus, une dérivation implique au moins une version en entrée (plusieurs versions en cas de fusion) et une version en sortie. Nous décrivons sur cette base, et similairement à (Chacon, 2009), l'espace des versions de tout document multi-version comme un *graphe acyclique orienté*.

2.2. Modèle de documents XML non ordonnés : définition et mise à jour

Les applications qui motivent ce travail manipulent des données arborescentes. Nous considérons les données en présence comme des arbres XML non ordonnés, en conséquence. L'extension du modèle proposé aux arbres ordonnés est possible. Cela pourrait par exemple consister à restreindre l'ensemble des versions valides à celles se conformant à un ordre spécifique. Ce probléme est hors du champ de cet article et nous nous focalisons ici sur le cas des arbres non ordonnés. Considérons un ensemble fini $\mathscr{L}$ de chaînes de caractères (c'est-à-dire, étiquettes ou données textes) et un ensemble fini $\mathscr{I}$ d'identifiants tels que $\mathscr{L} \cap \mathscr{I} = \emptyset$. De plus, soient Φ et α respectivement une fonction d'étiquetage et une fonction d'identification. D'une manière formelle, nous définissons un document XML comme étant un arbre $\mathscr{T}$ non ordonné et étiqueté : les fonctions α et Φ associent chaque $x \in \mathscr{T}$ respectivement à un identifiant unique $\alpha(x) \in \mathscr{I}$ et à une chaîne de caractères $\Phi(x) \in \mathscr{L}$. L'arbre est non-borné, c'est-à-dire, le nombre de fils de chaque nœud dans $\mathscr{T}$ n'est pas fixé à l'avance. Par soucis de simplicité, nous supposerons que tous les arbres ont la même racine (même étiquette, même identifiant).

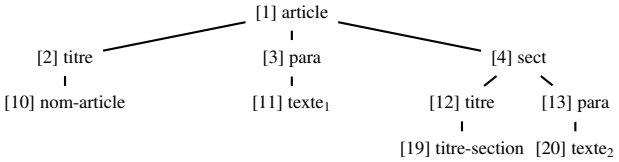


Figure 1. arbre XML $\mathscr{T}$: article Wikipedia

EXEMPLE 1. — La figure 1 montre un arbre XML $\mathscr{T}$ qui représente un article Wikipedia classique. Les identifiants de nœuds sont à l'intérieur des crochets précédant les chaînes de caractères de ses derniers. Le titre de cet article est donné par le nœud 10. Le contenu du document est structuré en sections (« sect ») avec leurs titres et paragraphes (« para ») renfermant les données textes. □

À l'aide des identifiants uniques, nous considérons deux types d'opérations d'édition sur le modèle de documents XML spécifié : *insertions* et *suppressions* de nœuds.

3. Noter que la notion d'espace des versions utilisée ici a une sémantique differente de celle du même nom définie dans le domaine de l'apprentissage des concepts (Mitchell, 1979).

Nous désignons une insertion par `ins(i, x)` dont la sémantique sur tout arbre XML consiste à ajouter le nœud x (nous supposons que x n'est pas déjà dans l'arbre) comme fils d'un certain nœud y tel que $\alpha(y) = i$. Si un tel nœud n'est pas trouvé dans l'arbre, l'opération ne fait rien. Observer qu'une insertion peut concerner un sous-arbre, et dans ce cas nous nous référons simplement par x à la racine de ce sous-arbre. De façon analogue, nous introduisons une suppression comme étant `supp(i)` où i est l'identifiant du nœud à supprimer. L'opération de suppression élimine le nœud cible, s'il existe, ainsi que ses descendants, de l'arbre XML. Nous concluons en définissant un script d'édition XML, $\Delta = \langle u_1, u_2, \ldots, u_i \rangle$, comme une séquence d'opérations d'édition élémentaires u_j (chaque u_j, avec $1 \leq j \leq i$, étant soit une insertion soit une suppression) à évaluer l'une après l'autre sur un document XML pour en créer un nouveau. Pour un arbre $\mathscr{T}$, nous notons le résultat de l'évaluation d'un script d'édition Δ sur $\mathscr{T}$ par $[\mathscr{T}]^{\Delta}$. Même si dans ce travail nous nous basons sur des identifiants persistants de nœuds pour définir nos opérations d'édition, la sémantique de ces opérations pourrait être étendue aux mises à jour exprimées par des requêtes, particulièrement utile dans les environnements d'édition collaborative distribués où les identifiants peuvent ne pas être facilement partageables.

3. XML probabiliste

Nous présentons brièvement dans cette section le système de représentation XML probabiliste que nous utilisons comme élément de base de notre système de contrôle de versions incertaines. Pour plus de détails, voir (Abiteboul *et al.*, 2009) pour le cadre général et (Kharlamov *et al.*, 2010) pour le modèle spécifique PrXML$^{\text{fie}}$ que nous employons. Ces modèles de représentation d'incertitude sont conçus à l'origine pour la gestion de données dans les domaines de l'intégration de données du Web et de l'extraction d'informations.

3.1. Les p-documents PrXML$^{\text{fie}}$ *: syntaxe et sémantique*

Un *système de représentation XML probabiliste* est une manière compacte d'encoder des distributions de probabilité sur des documents XML ; dans le cas d'intérêt ici, la distribution de probabilité est finie. Formellement, un espace de distribution XML probabiliste, ou px-espace, $\mathscr{S}$ sur une collection de documents XML est un couple (D, p) où D est un ensemble fini non vide de documents et $p : D \to]0,1]$ une mesure de probabilités qui affecte à chaque document d de D un nombre rationnel $p(d) \in]0,1]$ tel que $\Sigma_{d \in D} p(d) = 1$. Un *p-document*, ou *document XML probabiliste*, habituellement noté $\widehat{\mathscr{P}}$, définit un encodage compact d'un px-espace $\mathscr{S}$.

Nous considérons dans cet article une classe spécifique de p-documents, PrXML$^{\text{fie}}$ (Kharlamov *et al.*, 2010) (où *fie* est l'abréviation de *formula of independent events* ou *formule d'événements indépendants*) ; se restreindre à cette classe en particulier nous permet de donner une présentation simplifiée, se référer à (Abiteboul *et al.*, 2009 ; Kharlamov *et al.*, 2010) pour un cadre plus général. Supposons donné un ensemble de *variables aléatoires booléennes indépendantes*, ou *variables d'événements*, $b_1, b_2, \ldots, b_m$ et leurs probabilités d'existence respectives $Pr(b_1), Pr(b_2), \ldots, Pr(b_m)$. Un document PrXML$^{\text{fie}}$ est un arbre non ordonné, sans limite sur le nombre de fils

par nœud, et étiqueté, où chaque nœud (à l'exception de la racine) x peut être annoté avec une formule propositionnelle arbitraire $\mathit{fie}(x)$ sur les variables d'événements $b_1, b_2, \ldots, b_m$. Des formules différentes peuvent partager des événements communs, ce qui veut dire qu'il peut y avoir des corrélations entre formules, et le nombre de variable d'événements peut changer d'un nœud à l'autre.

Une affectation de valeurs de vérités ν des variables d'événements $b_1 \ldots b_m$ induit sur $\widehat{\mathscr{P}}$ un document XML particulier $\nu(\widehat{\mathscr{P}})$: le document où seuls les nœuds annotés avec des formules qui s'évaluent à vrai par ν sont conservés (les nœuds dont les formules s'évaluent à faux par ν sont supprimés de l'arbre, avec leurs descendants). Étant donné un p-document $\widehat{\mathscr{P}}$, les *mondes possibles* de $\widehat{\mathscr{P}}$, notés $\mathit{poss}(\widehat{\mathscr{P}})$, sont l'ensemble des tels documents XML. La *probabilité* d'un monde possible donné d de $\widehat{\mathscr{P}}$ est définie comme la somme des probabilités des affectations de valeurs de vérité qui donnent d. L'ensemble des mondes possibles, avec leurs probabilités, définit la *sémantique* de $\widehat{\mathscr{P}}$, le px-espace $[\![\widehat{\mathscr{P}}]\!]$ associé à $\widehat{\mathscr{P}}$.

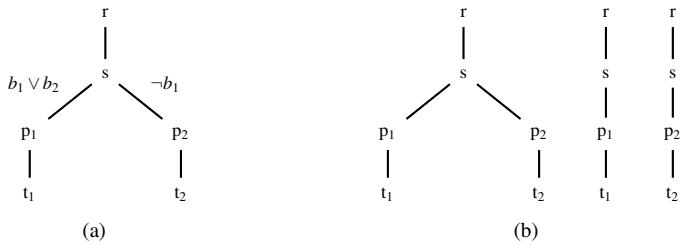


Figure 2. (a) un p-document PrXML$^{\text{fie}}$ *; (b) trois mondes possibles d_1, d_2 et d_3*

EXEMPLE 2. — La figure 2 montre (à gauche) un p-document PrXML$^{\text{fie}}$ concret $\widehat{\mathscr{P}}$ et (à droite) trois mondes possibles d_1, d_2 et d_3. Les formules annotant les nœuds sont indiquées au-dessus d'eux : $b_1 \vee b_2$ et $\neg b_1$ sont liés aux nœuds p_1 et p_2, respectivement. Les trois mondes possibles d_1, d_2 et d_3 sont obtenus en choisissant les valeurs de vérités de b_1 et b_2 suivantes : (i) faux et vrai ; (ii) vrai et vrai (ou vrai et faux) ; (iii) faux et faux. Pour chaque affectation de valeurs de vérités, on garde exactement les nœuds dont les formules s'évaluent à vrai. Si on fixe une distribution de probabilités sur les événements (grâce par exemple à un algorithme évaluant la réputation de la source de chaque événement), par exemple $Pr(b_1) = 0.4$ et $Pr(b_2) = 0.5$, on dérive la probabilité du monde possible d_1 comme $Pr(d_1) = (1 - Pr(b_1)) \times Pr(b_2) = 0.6 \times 0.5 = 0.3$. Il est possible de calculer la probabilité des autres mondes possibles de manière similaire. □

Par rapport aux autres systèmes de représentation XML probabiliste (Abiteboul *et al.*, 2009), PrXML$^{\text{fie}}$ est très succinct (car des formules propositionnelles arbitraires peuvent être utilisées, avec des corrélations arbitraires entre événements), c'est-à-dire, exponentiellement plus succinct que les modèles de (Van Keulen *et al.*, 2005 ; Nierman, Jagadish, 2002) ; il offre par ailleurs des insertions et suppressions en temps polynomial (Kharlamov *et al.*, 2010), un besoin clef pour notre modèle de contrôle de versions incertain. Cependant, un inconvénient non négligeable est que toutes les requêtes

(à motif d'arbre) non triviales sur ce modèle sont **#P**-difficile à évaluer (Kimelfeld *et al.*, 2009). Ce n'est pas nécessairement un problème ici, puisque nous privilégions dans notre cas des mises à jour efficaces et le fait de pouvoir extraire des mondes possibles, plutôt que l'exécution de requêtes arbitraires.

3.2. Provenance des données

La gestion de données XML incertaines basée sur le modèle PrXML$^{\text{fie}}$ bénéficie également des sémantiques multiples des variables d'événements en termes de description de l'information. En effet, en plus de la gestion de l'incertitude, ce modèle peut également supporter la conservation de l'information sur la *provenance des données* (ou *lignée*)[4] à l'aide des variables d'événements. La provenance des données consiste en des informations de traçabilité telles que la sémantique des changements, le responsable de ces changements, un estampillage temporel, etc., pour des données incertaines. Pour ce faire, il suffit d'utiliser la sémantique des variables d'événements comme représentant de l'information sur la provenance des données. Il est ainsi parfois utile d'utiliser des systèmes de représentation XML probabiliste même en l'absence de sources de probabilités fiables pour les événements individuels, au sens où ces événements peuvent être manipulés comme faisant partie d'un modèle de données incomplet (c'est-à-dire, on ne prend en compte que les mondes possibles et non leurs probabilités).

4. Modèle XML multi-version incertain

Dans cette partie, nous développons notre modèle de contrôle de versions XML incertain pour des documents arborescents édités de manière collaborative. Nous basons notre modèle sur trois concepts principaux : les événements de contrôle de versions, un p-document et un graphe orienté acyclique d'événements. Nous débutons par une formalisation des documents XML multi-version à travers une définition formelle de leur graphe d'espace de versions et de leur ensemble de versions. Le modèle proposé est ensuite introduit.

4.1. Documents XML multi-version

Considérons l'ensemble infini $\mathscr{D}$ de tous les documents XML ayant une racine avec étiquette et identifiant fixés. Soit $\mathscr{V}$ l'ensemble des *événements de contrôle de versions* $e_1 \dots e_n$. Ces événements représentent les différents états d'un arbre. Nous associons aux événements des informations contextuelles à propos des révisions (auteur, estampille temporelle, etc.). À chaque événement e_i est également associé un *script d'édition* Δ_i. En partant de ces bases, nous formalisons le graphe de l'espace de versions et l'ensemble des versions d'un document XML versionné comme suit.

L'espace des versions est un graphe orienté acyclique (*directed acyclic graph* ou DAG) $\mathscr{G} = (\mathscr{V} \cup \{e_0\}, \mathscr{E})$ où : (i) l'événement de contrôle de versions initial $e_0 \notin \mathscr{V}$,

4. Comme étudié dans (Zhang, Jagadish, 2013), garder des informations de provenance dans un processus de contrôle de versions pourrait être utile dans le cas de requêtes sur la provenance des données.

un événement spécial représentant le premier état de chaque arbre XML versionné, est la racine de $\mathscr{G}$; (ii) $\mathscr{E} \subseteq (\mathscr{V} \cup \{e_0\}) \times \mathscr{V}$, définissant les arêtes de $\mathscr{G}$, consiste en un ensemble de couples ordonnés d'événements de contrôle de versions. Chaque arête décrit implicitement une relation de dérivation orientée entre deux versions. Une *branche* de $\mathscr{G}$ est un chemin orienté impliquant un nœud initial e_i et un nœud final e_j. Ce dernier doit être atteignable depuis e_i en traversant un ensemble d'arêtes orientées de $\mathscr{E}$. Nous notons cette branche B_i^j. Une *branche enracinée* est une branche qui démarre à la racine du graphe.

Une version XML est le document de $\mathscr{D}$ qui correspond à un *ensemble* d'événements de contrôle de versions, l'ensemble des événements qui a donné lieu à cette version. Dans un système de contrôle de versions déterministe, cet ensemble correspond toujours à une branche enracinée du graphe de l'espace de versions. Dans notre système de contrôle de versions incertain, cet ensemble peut être arbitraire. Considérons l'ensemble $2^{\mathscr{V}}$ formé de toutes les sous-parties de $\mathscr{V}$. Cet ensemble de versions d'un document XML multi-version est donné par une fonction $\Omega : 2^{\mathscr{V}} \to \mathscr{D}$: à chaque ensemble d'événements correspond un arbre donné (ces arbres ne sont en général pas tous distincts). La fonction Ω peut être calculée à partir des scripts d'édition associés aux événements de la manière suivante :

— $\Omega(\emptyset)$ est l'arbre XML, formé d'une racine seule, de $\mathscr{D}$.
— Pour tout i, pour tout $\mathscr{F} \subseteq 2^{\mathscr{V}\setminus\{e_i\}}$ $\Omega(\{e_i\} \cup \mathscr{F}) = [\Omega(\mathscr{F})]^{\Delta_i}$.

On définit maintenant un document XML multi-version, $\mathscr{T}_{mv}$, comme une paire $(\mathscr{G}, \Omega)$ où $\mathscr{G}$ est un DAG d'événements de contrôle de versions et Ω est une fonction définissant l'ensemble des versions du document. Dans ce qui suit nous proposons une manière plus efficace de calculer la version correspondant à un ensemble d'événements, en utilisant un p-document comme modèle.

4.2. Document XML multi-version incertain : définition et gestion des incertitudes

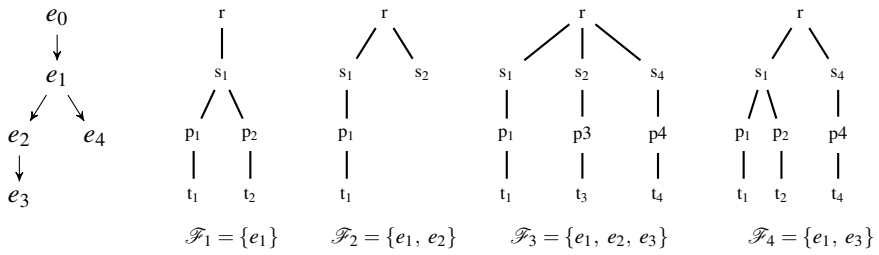
Un document multi-version est dit *incertain* si les événements de contrôle de versions, utilisés dans un processus de contrôle de versions, sont munis d'*incertitude*, comme dans des contextes collaboratifs ouverts. Quand nous parlons d'incertitude des événements de contrôle de versions, nous voulons dire que ce sont des événements aléatoires conduisant à des versions, et du contenu, incertains. Par conséquent, nous nous appuierons sur une *distribution de probabilité sur* $2^{\mathscr{V}}$ qui induira, en conjonction avec la fonction Ω, une distribution de probabilités sur $\mathscr{D}$.

Nous modélisons l'incertitude des événements en définissant maintenant un événement de contrôle de versions e_i de $\mathscr{V}$ comme une conjonction de variables booléennes aléatoires indépendantes $b_1 \ldots b_m$ avec les hypothèses suivantes : (i) une variable booléenne modélise une source donnée d'incertitude (p. ex., le contributeur) dans l'environnement de contrôle de versions ; (ii) les variables booléennes de chaque e_i sont deux à deux indépendantes ; (iii) une variable booléenne b_j réutilisée d'un événement à l'autre corrèle des événements de contrôle de version différents ; (iv) une variable booléenne particulière $b^{(i)}$, dite de *révision*, représentant plus spécifiquement l'incer-

titude dans la contribution, n'est pas partagée avec les autres événements de contrôle de versions et apparaît uniquement dans e_i.

Supposons donnée une distribution de probabilité sur les variables booléennes aléatoires b_j (cela vient typiquement d'une estimation de la confiance en un contributeur, ou en une contribution), qui induit une distribution de probabilité sur les formules propositionnelles sur les b_j de la manière usuelle (Kharlamov *et al.*, 2010). Nous obtenons maintenant la probabilité de chaque version (incertaine) d ainsi :

$$\Pr(d) = \Pr\Big(\bigvee_{\substack{\mathscr{F}\subseteq\mathscr{V}\\ \Omega(\mathscr{F})=d}} \mathscr{F}\Big) \text{ avec } \Pr(\mathscr{F}) = \Pr\left(\bigwedge_{e_j\in\mathscr{F}} e_j \wedge \bigwedge_{e_k\in\mathscr{V}\setminus\mathscr{F}} \neg e_k\right) \forall\, \mathscr{F}\subseteq\mathscr{V}. \quad (1)$$



(a) Graphe $\mathscr{G}$ (b) Versions possibles $\mathscr{T}_1$, $\mathscr{T}_2$, $\mathscr{T}_3$ et $\mathscr{T}_4$

Figure 3. (a) graphe d'espace de versions ; (b) 4 versions et leurs valeurs de vérité

EXEMPLE 3. — La figure 3 montre un document XML multi-version $\mathscr{T}_{mv}$ sujet à quatre événements de contrôle de versions. À gauche, nous avons l'espace des versions $\mathscr{G}$. La partie droite montre un exemple de quatre versions (incertaines) et leurs ensembles d'événements associés. On suppose que $\mathscr{T}_{mv}$ est initialement un document formé uniquement de la racine. Les trois premières versions correspondent aux versions couvertes par les systèmes déterministes de contrôle de versions, tandis que la dernière est engendrée en considérant que les changements accomplis par un événement de contrôle de versions intermédiaire, ici e_2, ne sont pas corrects. L'une des particularités de notre modèle est de fournir la possibilité de voir et de modifier ces types de versions incertaines qui représentent des versions virtuelles. Seules les éditions faites par les événements de contrôle de versions indiqués sont prises en compte dans le processus de production d'une version : dans $\mathscr{T}_4$, le nœud r et les sous-arbres enracinés en s_1, s_3 introduits respectivement par e_0, e_1 et e_3 sont présents, tandis que le sous-arbre p_3 ajouté par e_3 n'apparaît pas parce que son nœud père s_2 n'existe pas. Enfin, étant données des probabilités pour les événements de contrôle de versions, il est possible de mesurer la fiabilité de chaque version incertaine $\mathscr{T}_i$, pour $1 \leq i \leq 4$, en fonction de son ensemble d'événements correspondant $\mathscr{F}_i$ (et de tous les autres ensembles d'événements qui conduisent au même arbre). □

On observe directement, en particulier dans l'exemple de la figure 3, que le nombre de versions (incertaines) possibles d'un document multi-version incertain croît rapi-

dement (en fait, exponentiellement en le nombre d'événements). Le résultat en est que l'énumération et la gestion de toutes les possibilités avec la fonction Ω peut devenir difficile à partir d'un certain point. Pour résoudre ce problème, nous proposons une méthode efficace d'encodage de manière compacte des versions possibles, avec leurs valeurs de vérité. Intuitivement, un p-document $\mathsf{PrXML}^{\mathsf{fie}}$ modèle de manière compacte l'ensemble des versions possibles d'un document XML multi-version. Comme nous en avons discuté dans la section 3, un arbre probabiliste basé sur des formules propositionnelles fournit des caractéristiques utiles à notre contexte. Premièrement, il décrit de manière appropriée une distribution de valeurs de vérité sur un ensemble d'arbres XML incertains tout en fournissant un processus sensé pour retrouver une version donnée et sa probabilité. Deuxièmement, il fournit un système de représentation efficace pour les mises à jour, ce qui est crucial dans les environnements dynamiques tels que les applications basées sur le contrôle de versions.

4.3. Encodage XML probabiliste

Nous définissons un formalisme général de représentation de contrôle de versions XML incertain, habituellement noté $\widehat{\mathscr{T}_{mv}}$, comme un couple $(\mathscr{G}, \widehat{\mathscr{P}})$ où (a) $\mathscr{G}$ est, comme auparavant, un DAG d'événements, représentant l'espace de versions ; (b) $\widehat{\mathscr{P}}$ est un p-document $\mathsf{PrXML}^{\mathsf{fie}}$ avec des variables aléatoires booléennes $b_1 \dots b_m$ représentant efficacement l'ensemble des versions XML (incertaines) possibles et leurs valeurs de vérités correspondantes.

Nous définissons maintenant la sémantique d'un tel encodage comme un document multi-version incertain $(\mathscr{G}, \Omega)$ où $\mathscr{G}$ est le même et Ω est défini comme suit. Pour $\mathscr{F} \subseteq \mathscr{V}$, soit B^+ l'ensemble de toutes les variables aléatoires apparaissant dans l'un des événements de $\mathscr{F}$ et B^- l'ensemble de toutes les variables de révision $b^{(i)}$ pour e_i *non membre de* $\mathscr{F}$. Soit ν l'affectation de valeur de vérité de $b_1 \dots b_m$ qui associe aux variables de B^+ la valeur vrai, aux variables de B^- la valeur faux et aux autres variables une valeur arbitraire. Nous posons : $\Omega(\mathscr{F}) := \nu(\widehat{\mathscr{P}})$.

Le résultat suivant montre que cette sémantique est compatible avec la sémantique de px-espace des p-documents d'une part, et avec la distribution de probabilité définie par les documents multi-version incertains d'autre part.

THÉORÈME 4. — *Soit* $(\mathscr{G}, \widehat{\mathscr{P}})$ *un système de représentation de contrôle de versions incertain et* $(\mathscr{G}, \Omega)$ *sa sémantique. Nous supposons que toutes les formules de* $\widehat{\mathscr{P}}$ *peuvent s'exprimer comme des formules sur les événements de* $\mathscr{V}$ *(nous n'utilisons donc pas les* b_j *indépendamment des événements de contrôle de versions). Alors le px-espace* $[\![\widehat{\mathscr{P}}]\!]$ *définit la même distribution de probabilité sur* $\mathscr{D}$ *que* Ω.

La preuve est directe et s'appuie sur l'équation (1).

5. Mise à jour de XML multi-version incertain

Nous implantons la sémantique des opérations standards de mise à jour au-dessus de notre système de représentation XML probabiliste. Une mise à jour sur un document multi-version incertain correspond à l'évaluation d'éditions incertaines sur une

Entrées : $(\mathcal{G}, \widehat{\mathcal{P}})$, $\texttt{MÀJ}_{\Delta,\, e,\, e'}$
Sorties : mise à jour de $(\mathcal{G}, \widehat{\mathcal{P}})$ par $\texttt{MÀJ}_{\Delta,\, e,\, e'}$
$\mathcal{G} := \mathcal{G} \cup (\{e'\}, \{(e, e')\})$;
pour chaque *(opération d'édition u dans* Δ*)* **faire**
  si *u est une insertion* `ins` (i, x) **alors**
    $y :=$ `chercheNœudParId` $(\widehat{\mathcal{P}}, i)$;
    si `correspondanceTrouvée` $(\mathcal{T}_y, x)$ **alors**
      $fie_a(x) :=$ `obtientFieDuNœud` (x) ;
      `changeFieDuNœud` $(x, fie_a(x) \vee e')$;
    sinon
      `màjContenu` $(\widehat{\mathcal{P}},$ `ins` $(i, x))$;
      `changeFieDuNœud` (x, e') ;
  sinon si *u est une suppression* `supp` (i) **alors**
    $x :=$ `chercheNœudParId` $(\widehat{\mathcal{P}}, i)$;
    $fie_a(x) :=$ `obtientFieDuNœud` (x) ;
    `changeFieDuNœud` $(x, fie_a(x) \wedge \neg e')$;
retourner $(\mathcal{G}, \widehat{\mathcal{P}})$;

Algorithme 1. Algorithme de mise à jour (`MÀJPrXML`*)*

version (incertaine) donnée. Étant donné un triplet (Δ, e, e'), nous parlons de l'opération de mise à jour $\texttt{MÀJ}_{\Delta,\, e,\, e'}$ où Δ est un script d'édition, e est un événement de contrôle de versions existant pointant vers la version éditée et e' est un nouvel événement de contrôle de versions évaluant l'incertitude dans cette mise à jour. Nous formalisons $\texttt{MÀJ}_{\Delta,\, e,\, e'}$ sur $\mathcal{T}_{mv}$ ainsi :

$$\texttt{MÀJ}_{\Delta,\, e,\, e'}(\mathcal{T}_{mv}) := (\mathcal{G} \cup (\{e'\}, \{(e, e')\}),\ \Omega').$$

Une opération de mise à jour résulte donc en l'insertion d'un nouveau nœud et d'une nouvelle arête dans $\mathcal{G}$ et en l'extension de Ω en un Ω' que nous définissons maintenant. Pour chaque sous-ensemble $\mathcal{F} \subseteq \mathcal{V}'$ ($\mathcal{V}'$ est l'ensemble des nœuds de $\mathcal{G}$ après la mise à jour), on pose :

- si $e' \notin \mathcal{F}$: $\Omega'(\mathcal{F}) = \Omega(\mathcal{F})$;
- sinon : $\Omega'(\mathcal{F}) = [\Omega(\mathcal{F} \setminus \{e'\})]^{\Delta}$.

Ce qui précède donne une sémantique aux mises à jours sur des documents multi-version incertains ; cependant, la sémantique n'est pas utilisable en pratique car elle demande de considérer chaque sous-ensemble $\mathcal{F} \subseteq \mathcal{V}'$. Pour une solution plus commode, nous appliquons directement les mises à jour sur la représentation compact p-document du document multi-version. L'algorithme 1 décrit comment une telle opération de mise à jour $\texttt{MÀJ}_{\Delta,\, e,\, e'}$ est réalisée sur une représentation incertaine $(\mathcal{G}, \widehat{\mathcal{P}})$. Tout d'abord, le graphe est mis à jour comme auparavant. Ensuite, pour chaque opération u dans Δ, l'algorithme récupère le nœud cible de $\widehat{\mathcal{P}}$ avec `chercheNœudParId` (typiquement une opération en temps constant). Selon le type d'opération, il y a deux possibilités :

1. Si u est une insertion d'un nœud x, l'algorithme vérifie si x n'est pas déjà dans $\widehat{\mathcal{P}}$, par exemple en cherchant un nœud avec la même étiquette (la fonction `correspondanceTrouvée` recherche une correspondance pour x dans le sous-

arbre $\mathscr{T}_y$ enraciné en y). Si une telle correspondance existe, `obtientFieDuNœud` retourne sa formule courante $fie_a(x)$ et l'algorithme la met à jour en $fie_n(x) := fie_a(x) \vee e'$, spécifiant que x apparaît quand la mise à jour est valide. Sinon, `màjContenu` et `changeFieDuNœud` respectivement insère le nœud x dans $\widehat{\mathscr{P}}$ et met sa formule associée à $fie_n(x) = e'$.

2. Si u est une suppression d'un nœud x, l'algorithme obtient sa formule courante $fie_a(x)$ et la change en $fie_n(x) := fie_a(x) \wedge \neg e'$, indiquant que x doit être supprimé des mondes possibles quand cette mise à jour est valide.

Le reste de cette partie montre la correction et l'efficacité de notre approche. Tout d'abord, nous établissons que l'algorithme 1 respecte la sémantique des mises à jour. Ensuite, nous montrons que le comportement des systèmes de contrôle de versions déterministes peut être simulé en considérant uniquement un type spécifique d'ensemble d'événements. Finalement, nous caractérisons la complexité de l'algorithme.

THÉORÈME 5. — *L'algorithme 1, quand lancé sur un encodage XML probabiliste $\widehat{\mathscr{T}_{mv}} = (\mathscr{G}, \widehat{\mathscr{P}})$ d'un document multi-version $\mathscr{T}_{mv} = (\mathscr{G}, \Omega)$, avec une opération de mise à jour $\texttt{MÀJ}_{\Delta,e,e'}$, calcule une représentation $\texttt{MÀJ}_{\Delta,e,e'}(\widehat{\mathscr{T}_{mv}})$ du document multi-version $\texttt{MÀJ}_{\Delta,e,e'}(\mathscr{T}_{mv})$.*

La preuve de ce théoréme est détaillée dans (Ba *et al.*, 2013b).

La sémantique des mises à jour est donc la même, qu'elle soit énoncée sur les documents multi-version ou implantée par l'algorithme 1. Nous montrons maintenant que cette sémantique est compatible avec les opérations classiques de mise à jour des systèmes de contrôle de versions déterministes.

THÉORÈME 6. — *La définition formelle de mise à jour dans les documents multi-version incertains implante la sémantique des opérations standards de mise à jour dans les systèmes de contrôle de versions déterministes quand les ensembles d'événements sont restreints à des branches enracinéees.*

Nous terminons cette partie en énonçant le passage à l'échelle de notre algorithme :

THÉORÈME 7. — *L'algorithme 1 accomplit le processus de mise à jour sur la représentation d'un document XML multi-version en temps constant en la taille du document d'entrée. La taille de l'arbre probabiliste de sortie croît linéairement en la taille du script d'édition.*

6. Fusion de XML multi-version incertain

Nous détaillons dans cette section la transposition de l'opération de fusion XML usuelle dans notre modèle de contrôle de versions incertain. Nous présentons en premier lieu le processus de calcul des scripts d'édition à utiliser pour la fusion, et les scénarios de fusion les plus fréquents. Ensuite nous introduisons la sémantique de la fusion sur des versions incertaines d'un même document XML, ainsi qu'un algorithme efficace sur l'encodage XML probabiliste.

6.1. Stratégie de fusion : calcul de scripts d'édition et scénarios de fusion usuels

Une opération de fusion prend un ensemble de versions et fusionne leurs contenus dans une seule et nouvelle version. Nous nous concentrons ici sur la fusion de deux versions (le cas le plus usité dans les applications de tous les jours) à l'aide d'une approche *tripartite* (ou *three-way*), à savoir, une intégration de mises à jour provenant des entrées suivant leur version de base commune. L'intérêt principale de la fusion tripartite par rapport à celle bipartite (ou *two-way*) vient du fait qu'elle assure une meilleure correspondance de nœuds et détection de conflits. Nous supposons aussi que toutes les versions à fusionner sont uniquement générées à partir de mises à jour sur les versions de bases – cela, une fois de plus, à des fins de clarté dans la présentation. Le processus de fusion se divise en général en deux étapes : (i) une extraction des différents scripts d'édition qui ont mené aux versions en entrée et ; (ii) une génération du résultat de la fusion, en évaluant d'abord l'ensemble unifié des scripts extraits sur des données initiales, puis en résolvant les conflits éventuels. Supposons donné un document XML non ordonné sous contrôle de versions. Soient $\mathscr{T}_1$ et $\mathscr{T}_2$ deux versions arbitraires de ce document, ainsi que leur plus proche ancêtre commun $\mathscr{T}_a$. Nous détaillons ci-après les deux étapes du processus de fusion sur les versions $\mathscr{T}_1$ et $\mathscr{T}_2$.

Tout d'abord, nous déterminons le script d'édition spécifiant la fusion des versions $\mathscr{T}_1$ et $\mathscr{T}_2$ à l'aide de la fonction *diff3*$(\mathscr{T}_1, \mathscr{T}_2, \mathscr{T}_a)$. Cette fonction s'exécute sur des arbres non ordonnés dans lesquels les nœuds ont des identifiants uniques. Elle retournera un script avec seulement des insertions et des suppressions de nœuds comme opérations autorisées. De manière similaire à (Khanna *et al.*, 2007), notre *diff3* se base sur les fonctions *diff2*$(\mathscr{T}_a, \mathscr{T}_1)$ et *diff2*$(\mathscr{T}_a, \mathscr{T}_2)$, appelées algorithmes de diff *bipartite* (*two-way*). Les algorithmes bipartites ainsi définis apparient tout d'abord les nœuds de leurs entrées, et ensuite expriment les correspondances sous la forme de scripts d'édition Δ_1 et Δ_2, respectivement. Le résultat du *diff3*, que nous notons Δ_3, est égal à $\Delta_1 \cup \Delta_2$. Cette union peut résulter en trois types d'édition : éditions *équivalentes*, *conflictuelles* et *indépendantes*. Deux opérations $u_1 \in \Delta_1$ et $u_2 \in \Delta_2$ sont *équivalentes* si elles ont la même sémantique et les mêmes arguments. Seule une des deux est retenue dans Δ_3. À l'inverse, u_1 et u_2 sont *conflictuelles* lorsque u_1 est une insertion et u_2 est une suppression (ou inversement), et u_1 a ajouté des nœuds comme descendants de nœuds supprimés par u_2. Nous référons à l'ensemble de toutes les éditions conflictuelles dans Δ_3 par $\Delta^{\mathscr{C}}$. Nous disons qu'un nœud modifié par des éditions conflictuelles est un *nœud conflictuel*. Enfin, les éditions indépendantes sont celles de Δ_1 et Δ_2 qui n'appartiennent pas aux deux premières classes. L'ensemble des éditions équivalentes et indépendantes représentent les éditions *non conflictuelles* d'un algorithme de diff donné. Un nœud impacté par une édition non conflictuelle est un nœud *non conflictuel* pour une fusion.

La plupart des modèles de gestion de versions actuels propose trois scénarios de fusion qui prennent en compte la résolution de conflits éventuels (voir options de fusion, en particulier *mine-conflict* et *theirs-conflict*, dans Subversion (Pilato, 2004)). Rappelons que dans la grande majorité des cas, cette résolution est manuelle. Soit $\mathscr{T}_f$ la version résultante de la fusion de $\mathscr{T}_1$ et $\mathscr{T}_2$. Nous abstrayons ces cas de fusion possibles à l'aide des scripts et des versions en entrée comme suit :

1. $\mathscr{T}_f = [\mathscr{T}_1]^{\Delta_2 - \Delta^{\mathscr{C}}}$, c'est-à-dire, une fusion en considérant $\mathscr{T}_1$ et les éditions non conflictuelles provenant de Δ_1 ;
2. $\mathscr{T}_f = [\mathscr{T}_2]^{\Delta_1 - \Delta^{\mathscr{C}}}$ (ce cas est symétrique au précédent) ;
3. $\mathscr{T}_f = [\mathscr{T}_a]^{\Delta_3 - \Delta^{\mathscr{C}}}$, c'est-à-dire, une fusion sur la base de la version $\mathscr{T}_a$ et des éditions non conflictuelles provenant de Δ_3.

On peut noter (a) qu'il est facile de montrer que lorsque $\Delta^{\mathscr{C}} = \emptyset$, on obtient le même résultat pour les trois cas et ; (b) que la cas intuitif et naïf de fusion où l'utilisateur corrige les conflits peut être traité en choisissant en prime abord l'un des trois résultats ci-dessus avant d'effectuer ensuite les modifications souhaitées.

6.2. Fusion de versions XML incertaines

Nous considérons à présent notre abstraction de la fusion (couvrant au moins les scénarios abstraits ci-dessus) sur tout document XML multi-version incertain et l'algorithme correspondant sur son encodage XML probabiliste. Un contexte incertain induit une fusion à la base incertaine ; les versions en entrée et les scripts sont fournis avec de l'incertitude. Considérons un document XML multi-version incertain $\mathscr{T}_{mv} = (\mathscr{G}, \Omega)$ sujet à n événements de contrôle de versions et $\widehat{\mathscr{T}_{mv}} = (\mathscr{G},\ \widehat{\mathscr{P}})$ son encodage XML probabiliste.

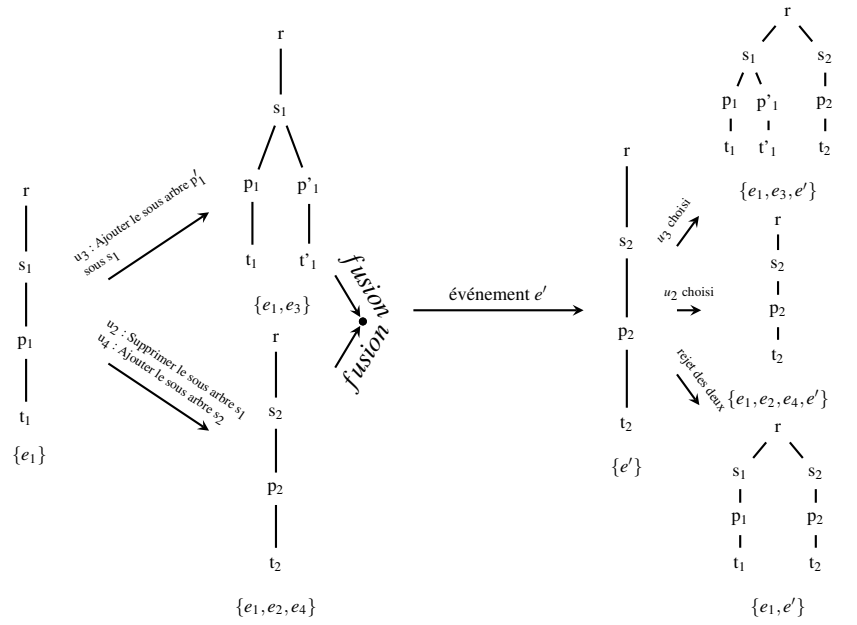
Étant donné un triplet $(e_1,\ e_2,\ e')$, nous parlons de l'opération de fusion avec de l'incertitude $\texttt{Fusion}_{e_1,\ e_2,\ e'}$ où e_1 et e_2 pointent vers les deux versions à être fusionné et e' est un nouvel événement évaluant l'incertitude dans cette fusion. Nous formalisons la sémantique de $\texttt{Fusion}_{e_1,\ e_2,\ e'}$ sur $\mathscr{T}_{mv}$ comme suit :

$$\texttt{Fusion}_{e_1,\ e_2,\ e'}(\mathscr{T}_{mv}) := (\mathscr{G} \cup (\{e'\}, \{(e_1,\ e'), (e_2,\ e')\}),\ \Omega').$$

D'une part, cette évaluation ajoute un nouvel événement et deux arêtes dans l'espace des versions $\mathscr{G}$. D'autre part, elle génère une nouvelle distribution Ω' laquelle étend Ω avec de nouvelles versions possibles et ensembles d'événements. Soit $\mathscr{A}_{e_1}$ et $\mathscr{A}_{e_2}$ l'ensemble de toutes les ancêtres strictes dans $\mathscr{G}$ de e_1 et e_2 respectivement. Nous notons l'intersection par $\mathscr{A}_s = \mathscr{A}_{e_1} \cap \mathscr{A}_{e_2}$. Pour tout $\mathscr{F} \in 2^{\mathscr{V} \cup \{e'\}}$, formellement nous établissons :

- Si $e' \notin \mathscr{F} : \Omega'(\mathscr{F}) := \Omega(\mathscr{F})$;
- Si $\{e_1, e_2, e'\} \subseteq \mathscr{F} : \Omega'(\mathscr{F}) := \Omega(\mathscr{F} \setminus \{e'\})$;
- Si $\{e_1, e'\} \subseteq \mathscr{F}$ et $e_2 \notin \mathscr{F} : \Omega'(\mathscr{F}) := [\Omega((\mathscr{F} \setminus \{e'\}) \setminus (\mathscr{A}_{e_2} \setminus \mathscr{A}_s))]^{\Delta_2 - \Delta^{\mathscr{C}}}$;
- Si $\{e_2, e'\} \subseteq \mathscr{F}$ et $e_1 \notin \mathscr{F} : \Omega'(\mathscr{F}) := [\Omega((\mathscr{F} \setminus \{e'\}) \setminus (\mathscr{A}_{e_1} \setminus \mathscr{A}_s))]^{\Delta_1 - \Delta^{\mathscr{C}}}$;
- Si $\{e_1, e_2\} \cap \mathscr{F} = \emptyset$ et $e' \in \mathscr{F} : \Omega'(\mathscr{F}) := [\Omega((\mathscr{F} \setminus \{e'\}) \setminus ((\mathscr{A}_{e_1} \setminus \mathscr{A}_s) \cup (\mathscr{A}_{e_2} \setminus \mathscr{A}_s)))]^{\Delta_3 - \Delta^{\mathscr{C}}}$.

Les scripts ci-dessus sont tous calculés à l'aide de la fonction tripartite décrite dans la section 6.1. Cette fonction prend en entrée dans chacun des cas requis les versions arbitraires $\mathscr{T}_1 = \Omega((\mathscr{F} \setminus \{e'\} \cap \mathscr{A}_{e_1}) \cup \{e_1\})$, $\mathscr{T}_2 = \Omega((\mathscr{F} \setminus \{e'\} \cap \mathscr{A}_{e_2}) \cup \{e_2\})$, et $\mathscr{T}_a = \Omega(\mathscr{F} \setminus \{e'\} \cap \mathscr{A}_s)$ où $\mathscr{F}$ est un sous-ensemble d'événements dans $\mathscr{V} \cup \{e'\}$ considérés comme valides.



(a) version de base $\mathscr{T}_a$, variantes $\mathscr{T}_1$ et $\mathscr{T}_2$; scripts $\langle u_2, u_4 \rangle$ et $\langle u_3 \rangle$

(b) génération de la fusion : d'abord, validation de u_4 puis résolution du conflit entre u_2 et u_3

Figure 4. stratégie de fusion : (a) versions incertaines et (b) résultat de la fusion

EXEMPLE 8. — La figure 4 décrit le processus de fusion de deux versions possibles $\mathscr{T}_1$ and $\mathscr{T}_2$ de l'exemple de la figure 3 connaissant leur version de base $\mathscr{T}_a$. Dans notre modèle, cette opération est simplement prise en compte avec la fusion des événements e_3 et e_4 lesquels pointent sur les versions en entrée. À gauche de la figure, nous avons les versions $\mathscr{T}_1$, $\mathscr{T}_2$ et $\mathscr{T}_a$ ainsi que les scripts $\langle u_2, u_4 \rangle$ et $\langle u_3 \rangle$ qui ont conduit aux variantes. Ces scripts sont extraits avec les fonctions diff de la section 6.1. À droite, nous explicitons le processus de fusion (avec l'événement de fusion e' estimant l'incertitude dans l'opération) comme suit : (i) Avant tout, toutes les éditions non conflictuelles dans les scripts, c'est-à-dire, ici uniquement u_4, sont validées pour obtenir la partie de la fusion (ou résultat intermédiaire) qui est certaine avec la validité de e' ; (ii) Ensuite, l'ensemble des fusions possibles est engendré en énumérant les différentes possibilités de traitement des éditions conflictuelles u_2 et u_3. Les deux premières versions possibles sont obtenues en propageant respectivement u_2 et u_3 sur le résultat intermédiaire. Concrètement, notre stratégie de fusion retournera les mêmes documents fusionnés en prenant $\mathscr{T}_1$ et $\mathscr{T}_2$ et en les mettant à jour avec les éditions non conflictuelles provenant respectivement de $\langle u_3 \rangle$ et $\langle u_2, u_4 \rangle$. Enfin, la dernière version possible est produite en annulant toutes les éditions conflictuelles, à savoir, en ajoutant les nœuds conflictuels de la version de base dans le résultat intermédiaire. □

L'opération de fusion incertaine ainsi formalisée ci-dessus ne passe cependant pas à l'échelle puisqu'elle exige d'évaluer toute version possible pour calculer le résultat global de la fusion. Dans ce qui suit, nous proposons une manière plus efficace de réaliser cette fusion.

6.3. Fusion sur l'encodage XML probabiliste

Nous introduisons Algorithme 2 (`FusionPrXML`) comme étant une méthode plus efficiente sur l'encodage XML probabiliste $\widehat{\mathcal{T}_{mv}} = (\mathcal{G},\ \widehat{\mathcal{P}})$ de réaliser la fusion de XML multi-version incertain. La définition de cette algorithme requiert toutefois la connaissance des nœuds conflictuels et non conflictuels pour la fusion. Ainsi nous définissons d'abord la notion de *nœuds conflictuels* étant donné une fusion $\texttt{Fusion}_{e_1,\, e_2,\, e'}$ sur $\widehat{\mathcal{T}_{mv}}$, puis nous détaillons l'algorithme proprement dit. Nous nous appuierons sur les formules associées aux nœuds pour détecter ceux qui sont conflictuels.

Considérons les affectations de valeurs de vérité suivantes sur les événements dans $\mathcal{G}$: (i) ν_s initialisant les événements dans $\mathcal{A}_s$ à vrai et les variables de révision de toutes les autres événements à faux ; (ii) ν_1 affectant vrai à toutes les événements dans $\mathcal{A}_{e_1} \cup \{e_1\}$ et faux aux variables de révision des autres événements ; et enfin (iii) ν_2 initialisant les événements dans $\mathcal{A}_{e_2} \cup \{e_2\}$ à vrai et les variables de révision des autres événements à faux. Nous introduisons en premier lieu la lignée (ou provenance) d'un nœud incertain dans le p-document $\widehat{\mathcal{P}}$.

DÉFINITION 9. — *(Lignée d'un nœud) La lignée d'un nœud donné $x \in \widehat{\mathcal{P}}$, notée $fie^{\uparrow}(x)$, est la formule propositionnelle résultante de la conjonction de la formule de ce nœud x avec celles associées à tous ses nœuds ancêtres dans $\widehat{\mathcal{P}}$.*

Au contraire de sa formule [5], la lignée d'un nœud dans le p-document encode l'histoire toute entière des éditions, à partir de l'événement initial, sur le chemin menant à ce nœud. En conséquence, nous pouvons définir les nœuds conflictuels dans le p-document en utilisant leurs lignées comme suit.

DÉFINITION 10. — *(Nœud conflictuel) Sous l'encodage XML probabiliste $\widehat{\mathcal{T}_{mv}}$, nous disons qu'un x donné dans $\widehat{\mathcal{P}}$ est un nœud conflictuel au regard de la fusion des événements e_1 et e_2 lorsque sa lignée satisfait les conditions suivantes. a) $fie^{\uparrow}(x) \models \nu_s$; b) $fie^{\uparrow}(x) \not\models \nu_1$ (ou $fie^{\uparrow}(x) \not\models \nu_2$) et ; c) $\exists y \in \widehat{\mathcal{P}}$, $\texttt{desc}\,(x, y)$: $fie^{\uparrow}(y) \not\models \nu_s$ et $fie^{\uparrow}(y) \models \nu_2$ (ou $fie^{\uparrow}(y) \models \nu_1$) où $\texttt{desc}\,(x, y)$ veut dire que y est un descendant de x.*

THÉORÈME 11. — *La définition 10 est cohérente avec celle des nœuds conflictuels donnée dans la section 6.1.*

La preuve du théorème 11 est une conséquence directe de la façon dont `MÀJPrXML` procède.

Un nœud conflictuel dans $\widehat{\mathcal{P}}$ produit des descendants conflictuels. Nous nous référons aux nœuds conflictuels dans $\widehat{\mathcal{P}}$ suivant la fusion des événements e_1 et e_2 avec

5. La formule d'un nœud décrit juste la sémantique des éditions à partir de l'événement où il a été inséré pour la première fois.

Entrées : $(\mathscr{G}, \widehat{\mathscr{P}})$, e_1, e_2, e'
Sorties : fusion dans $(\mathscr{G}, \widehat{\mathscr{P}})$ avec $\texttt{Fusion}_{e_1,\, e_2,\, e'}$
$\mathscr{G} := \mathscr{G} \cup (\{e'\}, \{(e_1, e'), (e_2, e')\})$;
pour chaque ***nœud non conflictuel*** *x dans* $\widehat{\mathscr{P}} \setminus \widehat{\mathscr{P}}_{|\mathscr{C}_{\{e_1,\, e_2\}}}$ **faire**
  `remplace` (*fie*(x), e_1, $(e_1 \vee e')$) ;
  `remplace` (*fie*(x), e_2, $(e_2 \vee e')$) ;
retourner $(\mathscr{G}, \widehat{\mathscr{P}})$

Algorithme 2. Algorithme de fusion (`FusionPrXML`*)*

la restriction $\widehat{\mathscr{P}}_{|\mathscr{C}_{\{e_1,\, e_2\}}}$. C'est sur cette base que nous déduisons ci-après l'ensemble des nœuds non conflictuels.

DÉFINITION 12. — *(Nœud non conflictuel) Pour la fusion des événements e_1 et e_2, nous définissons un nœud non conflictuel x comme un nœud dans $\widehat{\mathscr{P}} \setminus \widehat{\mathscr{P}}_{|\mathscr{C}_{\{e_1,\, e_2\}}}$ dont la formule fie(x) satisfait à une des conditions suivantes :*

- *fie*$(x) \models \nu_s$, *fie*$(x) \not\models \nu_1$ *et fie*$(x) \not\models \nu_2$;
- *fie*$(x) \not\models \nu_s$, *fie*$(x) \models \nu_1$ *et fie*$(x) \models \nu_2$;
- *fie*$(x) \models \nu_s$, *fie*$(x) \models \nu_1$ *et fie*$(x) \not\models \nu_2$;
- *fie*$(x) \models \nu_s$, *fie*$(x) \not\models \nu_1$ *et fie*$(x) \models \nu_2$;
- *fie*$(x) \not\models \nu_s$, *fie*$(x) \models \nu_1$ *et fie*$(x) \not\models \nu_2$;
- *fie*$(x) \not\models \nu_s$, *fie*$(x) \not\models \nu_1$ *et fie*$(x) \models \nu_2$.

THÉORÈME 13. — *La définition 12 est cohérente avec celle des nœuds non conflictuels donnée dans la section 6.1.*

La preuve du théorème 13 est triviale.

Nous poursuivons cette section en détaillant d'abord l'algorithme de fusion, puis en énonçant sa correction.

L'algorithme 2 prend en entrée l'encodage XML probabiliste $(\mathscr{G}, \widehat{\mathscr{P}})$ de $\mathscr{T}_{mv}$, les événements e_1 et e_2 dans $\mathscr{G}$, et le nouvel événement e' modélisant à la fois les éléments de la fusion et leur degré d'incertitude. L'algorithme met d'abord à jour $\mathscr{G}$ comme indiqué dans la section 6.2. Ensuite, la fusion dans $\widehat{\mathscr{P}}$ résulte en une modification mineure des formules associées aux nœuds non conflictuels dans $\widehat{\mathscr{P}} \setminus \widehat{\mathscr{P}}_{|\mathscr{C}_{\{e_1,\, e_2\}}}$ au travers la fonction `remplace`. Cette fonction substitue à ces formules toutes les occurrences de e_1 et e_2 par $(e_1 \vee e')$ et $(e_2 \vee e')$ respectivement. L'intuition est que chaque fusion possible, laquelle est valide avec l'existence de e' indépendamment de celle des autres événements, doit contenir au moins les nœuds non conflictuels dans $\widehat{\mathscr{P}}$ supposés certains avec e_1 et e_2. Les nœuds non conflictuels dont l'existence est indépendante de e_1 et e_2, dépendront uniquement des valeurs de vérité de leurs événements ancêtres au sein de chaque sous-ensemble d'événements incluant l'événement de fusion. Enfin, l'existence des nœuds conflictuels dans le résultat de la fusion repose sur une idée subjective de priorité implicite accordée à e_1 ou e_2 lorsque e' est vrai. Dans le scénario où e' et e_1 sont vrais et e_2 est faux, nous supposons que e_1 est plus probable que e_2 pour la fusion ; dans ce cas seuls les nœuds conflictuels valides avec $\mathscr{A}_{e_1} \cup \{e_1\}$

sont sélectionnés. La situation inverse fonctionne de la même manière. Par contre tout nœud conflictuel sera supprimé lorsqu'aucune priorité n'est définie, c'est-à-dire, pour une affectation mettant e' à vrai et les variables de révision de e_1 et e_2 à faux.

Soient $\mathscr{T}_{mv} = (\mathscr{G}, \Omega)$ et $\widehat{\mathscr{T}_{mv}} = (\mathscr{G}, \widehat{\mathscr{P}})$ respectivement un document XML multi-version incertain et son encodage XML probabiliste. Nous supposons, en plus, définie la fonction sémantique $[\![.]\!]$ telle que $[\![\widehat{\mathscr{T}_{mv}}]\!] = (\mathscr{G}, [\![\widehat{\mathscr{P}}]\!])$ avec $[\![\widehat{\mathscr{P}}]\!]$ renvoyant la même distribution de probabilité que Ω. Étant donné une fusion $\texttt{Fusion}_{e_1,\, e_2,\, e'}$, nous énonçons à présent que `FusionPrXML` respecte la sémantique de l'opération de fusion définie dans la section 6.2.

THÉORÈME 14. — *La définition de l'algorithme 2 est correcte au regard de la sémantique de l'opération de fusion sur un document XML multi-version incertain. En d'autres mots, le diagramme suivant commute :*

$$\begin{array}{ccc} \widehat{\mathscr{T}_{mv}} & \xrightarrow{\quad [\![.]\!] \quad} & [\![\widehat{\mathscr{T}_{mv}}]\!] \\ \Big\downarrow \texttt{FusionPrXML}_{(e_1, e_2, e')} & & \Big\downarrow \texttt{Fusion}_{e_1,\, e_2,\, e'} \\ \widehat{\mathscr{T}'_{mv}} & \xrightarrow[\quad [\![.]\!] \quad]{} & [\![\widehat{\mathscr{T}'_{mv}}]\!] \end{array}$$

Nous concluons cette section en énonçant le passage à l'échelle de `FusionPrXML`.

THÉORÈME 15. — *L'algorithme 2 exécute la fusion sur l'encodage XML probabiliste de tout document XML multi-version incertain en temps proportionnel à la taille des formules des nœuds impactés par les mises à jour dans les branches fusionnées.*

Pour la preuve des théorèmes 14 et 15, nous référons à (Ba *et al.*, 2013a).

7. Expérimentation et évaluation

Nous présentons, dans cette section, une évaluation expérimentale de notre modèle. Nous nous intéressons en premier aux opérations de création et de lecture de versions, et comparons l'efficacité de notre modèle sur ces deux opérations par rapport à celle de deux systèmes de gestion de versions de référence que sont Git et Subversion. Ensuite, nous présentons les avancées en termes de possibilité de filtrage de contenus qu'offre notre modèle. Toutes les mesures de temps montrées dans cette section correspondent à des temps de calcul en mémoire centrale (temps CPU). Les données sont préalablement chargées en mémoire centrale pour éviter le surcoût des accès disque. Les tests ont été réalisés dans les mêmes conditions matérielles pour les trois systèmes comparés.

7.1. Analyse de performances : jeux de données, implantation et analyse de coûts

Nous avons mesuré le temps d'exécution des opération de création (ou *commit*) et de restitution (ou *checkout*) de versions, sur Git, Subversion et l'implantation de notre modèle (PrXML). L'objectif est de montrer la faisabilité de notre proposition.

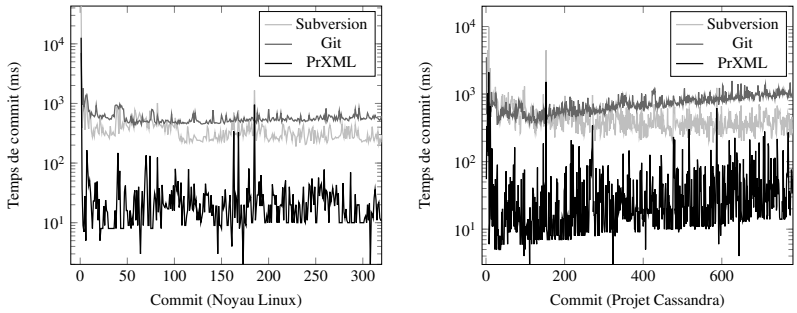


Figure 5. évaluation de la durée du commit (échelle logarithmique pour l'axe des ordonnées)

Vu que nos systèmes de référence sont tous déterministes, nous précisons que les tests pour l'analyse de performances ont été réalisés sur des données déterministes. Le principal intérêt de notre modèle demeurant sa capacité à gérer des versions de données incertaines, nous abordons et évaluons cet aspect dans la section 7.2.

Les jeux de données utilisés pour les tests ont été constitués à partir des branches principales des projets de développement du noyau Linux [6], et de Apache Cassandra [7]. Les données représentent l'organisation hiérarchique de deux systèmes de fichiers assez larges, et constituent deux exemples de données arborescentes partagées dans un environnement collaboratif. Le projet de développement du noyau Linux utilise Git comme système de gestion de versions. Nous avons obtenu une copie de l'historique du système de fichiers en clonant la branche principale de développement du projet, et nous l'avons maintenue à jour en propageant régulièrement sur notre copie locale les dernières mises à jour effectuées sur la branche originale. Nous avons suivi la même démarche pour les données du projet Cassandra, qui utilise Subversion comme système de gestion de versions. Au total, nous avons dans chaque branche locale plus de dix mille versions (ou commits), chacune matérialisant un ensemble de mises à jour. Dans nos expérimentations, nous nous sommes focalisé sur l'évolution de l'arborescence des systèmes de fichiers des projets considérés, en ignorant les changements réalisés sur le contenu plat des fichiers. Nous avons tracé les commits et les opérations de dérivation de versions à partir des logs de Git et de Subversion. Nous avons représenté la hiérarchie de chacun des deux systèmes de fichiers sous forme d'arbre XML sur lequel nous avons appliqué les changements successifs observés. À chaque insertion, respectivement suppression, d'un fichier ou d'un répertoire dans le système de fichiers est appliquée une insertion, respectivement suppression, d'un nœud dans l'arbre XML correspondant.

6. https://www.kernel.org/
7. http://cassandra.apache.org/

L'implantation de notre modèle de gestion de versions (PrXML) a été réalisée en Java. Nous nous sommes aussi basés sur les interfaces de programmation SVNKit [8] et JGit [9] de Java pour mettre en place les opérations standards de manipulation de versions dans Subversion et Git respectivement. Le but poursuivi est de réaliser toutes nos expérimentations dans les mêmes conditions. Le système Subversion mémorise dans ses fichiers de traces (logs) les changements effectués sur l'arborescence du système de fichiers à chaque commit. Chaque fichier log contient une liste d'expressions de chemins (*paths*), des identifiants de fichiers ou de répertoires dans le système de fichiers, et les mises à jours qui leurs sont associées. Dans le cas de Git, l'évolution du système de fichiers est représentée sous forme d'objets Git représentant les différents états de l'arborescence du système. Chaque objet correspond à un état du système de fichiers obtenu à la suite d'un commit.

Les figures 5 et 7 comparent le coût, en termes de temps d'exécution, des opérations de commit et de checkout de versions dans Git, Subversion et notre système PrXML. Les résultats montrent que le temps d'exécution de ces opérations est souvent plus rapide sur notre système et, au pire des cas, comparable à celui obtenu sur Git ou Subversion. En particulier, la figure 5 montre clairement que l'opération de commit est plus rapide sur notre système. Les mesures effectuées sur PrXML tiennent, bien évidemment, compte du temps de calcul additionnel des scripts de mises à jour (*deltas*) dans Git et Subversion.

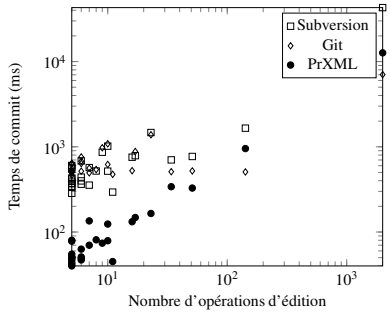


Figure 6. coût du commit en fonction du nombre d'opérations de mise à jour (scripts $\geq$ 5 opérations)

Une analyse en profondeur des résultats montre que le temps de commit est fonction de la taille du script d'édition associé à chaque version (voir figure 6), ce qui confirme la proposition 7. Cependant, PrXML reste efficace, à l'exception de quelques commits qui correspondent à des versions obtenues à la suite d'un grand nombre de mises à jour (plus d'une centaine d'opérations de mise à jour). Dans notre modèle, la création de versions se traduit par la mise à jour de l'arbre XML probabiliste représentant le document multi-version. Le script d'édition associé à chaque nouvelle version

8. http://svnkit.com/
9. http://www.eclipse.org/jgit/

est transposé sur l'arbre XML probabiliste, ce qui rend le coût du commit dépendant du nombre d'opérations de mises à jour associées à chaque version. En revanche, Git hache les fichiers et stocke la valeur de hachage (SHA-1) obtenue, indexée par le nom du répertoire ou du fichier. Le système Subversion trace dans un fichier log les mises à jour associées aux chemins concernés dans l'arborescence du système de fichiers. Ainsi, l'insertion d'un sous-arbre (ensemble de fichiers organisés en répertoires et sous-répertoires) dans le système de fichiers peut se traduire par une simple opération dans Git et Subversion, alors qu'elle peut nécessiter une série d'insertions de nœuds dans notre modèle.

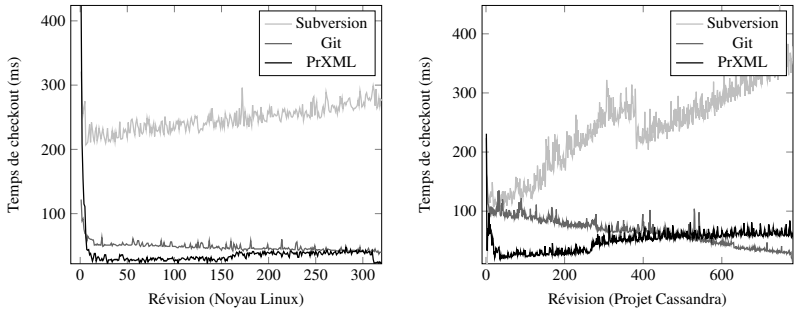


Figure 7. coût du checkout (restitution de version)

La figure 7 montre les résultats obtenus pour la mesure du temps d'exécution de l'opération de checkout de versions sur Git, Subversion et PrXML. Nous nous intéressons en particulier aux versions dérivées de façon séquentielle les unes des autres. Dans ce cas, le calcul de la version numéro n se traduit souvent par une cascade de calculs intermédiaires sur les versions précédentes, ce qui peut alourdir le temps d'exécution du checkout. Dans la figure 7, le numéro de version est précisé sur l'axe des abscisses et le coût (en ms) du checkout sur l'axe des ordonnées. Les résultats montrent que le checkout est plus rapide sur notre système que sur Subversion, pour les deux jeux de données considérés (noyau Linux et Cassandra). Par rapport à Git, notre système a de meilleures performances pour les premières versions créées (celles précédées par un faible nombre de versions dans la hiérarchie de dérivation), alors qu'il devient moins performant sur les dernières versions qui découlent d'une série importante de versions successives. Ceci vaut pour les tests effectués sur les deux jeux de données. Il faut préciser que certains systèmes de gestion de versions utilisent des diffs réversibles (Rusu *et al.*, 2005) pour accélérer le calcul des versions, lorsque celles-ci sont obtenues à la suite d'une longe série de dérivations linéaires.

7.2. Évaluation de la gestion des incertitudes : capacités de filtrage des données

L'évaluation de l'incertitude et le filtrage automatique des données non fiables sont deux défis importants pour les systèmes d'édition collaborative à large échelle. L'évaluation de l'incertitude est utile car les documents partagés sont produits par différents contributeurs, avec différents niveaux d'expertises et de fiabilité. Cette fiabilité peut être estimée de différentes manières, en s'appuyant par exemple sur les indicateurs

de confiance ou la réputation de chaque contributeur (voir (Adler, de Alfaro, 2007 ; Maniu *et al.*, 2011b ; 2011a)).

Dans le cas de plates-formes comme Wikipedia, notre approche peut permettre une gestion automatique des conflits entre contributeurs. Étant donné le nombre élevé de ces contributeurs et la fréquence des conflits, sur certains sujets sensibles, une gestion automatique des conflits est très utile. Notre approche permet de filtrer de façon transparente, et sans blocage des pages conflictuelles, les spams et les contributions d'utilisateurs peu fiables ou malveillants. Dans PrXML, des variables booléennes sont associées aux sources des données (les contributeurs dans le cas d'une plate-forme comme Wikipédia). Le fait d'affecter la valeur faux à une variable permet de filtrer les données produites par la source correspondante. Ceci peut être fait de façon automatique, lorsqu'un acte de vandalisme est détecté, ou lors de l'accès aux données par un utilisateur. Dans ce derniers cas, le système peut adapter le contenu en fonction des préférences de l'utilisateur et de son opinion (ou confiance) vis à vis des différents contributeurs. Nous avons montré dans (Abdessalem *et al.*, 2011) une application de ces possibilités d'interaction et de filtrage sur des contenus de la plate-forme Wikipedia. Une page Wikipedia est vue comme une fusion d'un ensemble de contributions incertaines et n'est plus limitée à la dernière version valide. Le degré d'incertitude associé à chaque partie d'une page (une section par exemple) est déduit de l'incertitude associée à chacune des révisions qui ont affecté cette partie. L'utilisateur peut ainsi isoler le contenu correspondant à une révision donnée, éliminer les parties introduites par une révision ou un contributeur, ou limiter le contenu de la page aux seules parties introduites par quelques révisions ou quelques contributeurs de confiance.

Nous avons aussi testé la capacité de notre approche à gérer les actes de vandalisme, en prenant comme exemple les pages les plus vandalisées dans Wikipedia (voir most vandalized Wikipedia pages [10]). Nous avons réussi à reconstituer le contenu des pages dans l'état où il aurait du se trouver si les actes de vandalisme n'avaient pas été enlevés. Nous avons vu qu'il est facile avec notre modèle d'avoir la vision souhaitée (avec ou sans vandalisme) d'une page, en jouant sur les valeurs de vérité attribuées aux variables booléennes. Notons que Wikipedia élimine systématiquement les versions vandalisées de ses pages, alors qu'il serait utile, pour différentes raisons, de les laisser accessibles aux utilisateurs qui le souhaitent. Nous avons pu détecter les actes de vandalisme sur les pages choisies aussi bien que peuvent le faire les robots de Wikipedia, sans élimination des versions vandalisées et en laissant le choix aux utilisateurs de tenir en compte ou pas ces versions.

8. Conclusion

Nous avons présenté dans cet article un modèle de gestion de versions de documents XML spécialement adapté aux environnements d'édition collaborative à large échelle. Ce travail fait partie des premières applications concrètes des travaux théoriques sur le XML probabiliste (Nierman, Jagadish, 2002 ; Van Keulen *et al.*, 2005 ;

10. http://en.wikipedia.org/wiki/Wikipedia:Most_vandalized_pages

Abiteboul *et al.*, 2009 ; Kimelfeld *et al.*, 2009 ; Kimelfeld, Sagiv, 2008 ; Kharlamov *et al.*, 2010 ; Kimelfeld, Senellart, 2013). Notre modèle a été implanté et évalué sur des jeux de données réels. Nous présentons dans cet article les résultats obtenus qui montrent l'efficacité de notre approche. Nous avons également rappelé les avantages de notre modèle et les possibilités inédites de filtrage sur les données incertaines qu'il permet de réaliser. Toutefois, une utilisation de notre modèle dans un environnement réel, combiné avec un algorithme évaluant la réputation des participants et la possibilité de connaître les préférences des lecteurs, devrait permettre d'avoir un aperçu plus détaillée des avantages introduites pour les aspects probabilistes de notre système.

Bibliographie

Abdessalem T., Ba M. L., Senellart P. (2011, Mar). A probabilistic XML merging tool. In *EDBT*, p. 538–541. Uppsala, Sweden, ACM. (Démonstration)

Abiteboul S., Kimelfeld B., Sagiv Y., Senellart P. (2009, Oct). On the expressiveness of probabilistic XML models. *VLDB Journal*, vol. 18, n° 5, p. 1041–1064.

Adler B. T., de Alfaro L. (2007, May). A content-driven reputation system for the Wikipedia. In *WWW*, p. 261–270. Banff, Alberta, Canada, ACM.

Altmanninger K., Seidl M., Wimmer M. (2009). A survey on model versioning approaches. *IJWIS*, vol. 5, n° 3, p. 271-304.

Ba M. L., Abdessalem T., Senellart P. (2011, Nov). Towards a version control model with uncertain data. In *PIKM*, p. 43–50. Glasgow, Scotland, ACM.

Ba M. L., Abdessalem T., Senellart P. (2013a, Sep). Merging uncertain multi-version XML documents. In *DChanges.* Florence, Italy, CEUR-WS.org.

Ba M. L., Abdessalem T., Senellart P. (2013b, Sep). Uncertain version control in open collaborative editing of tree-structured documents. In *DocEng*, p. 27–36. Florence, Italy, ACM.

Calzada G. de la, Dekhtyar A. (2010, Apr). On measuring the quality of Wikipedia articles. In *WICOW*, p. 11–18. Raleigh, North Carolina, USA, ACM.

Chacon S. (2009). *Pro Git* (1st éd.). Berkely, CA, USA, Apress.

Cobéna G., Abiteboul S., Marian A. (2002, Feb – Mar). Detecting Changes in XML Documents. In *ICDE*, p. 41-52. San Jose, CA, USA, IEEE Computer Society.

Khan L., Wang L., Rao Y. (2002, May). Change detection of XML documents using signatures. In *Real world RDF and semantic web applications.* Honolulu, Hawai.

Khanna S., Kunal K., Pierce B. C. (2007, Dec). A formal investigation of Diff3. In *FSTTCS*, p. 485–496. New Delhi, India, Springer-Verlag.

Kharlamov E., Nutt W., Senellart P. (2010, Mar). Updating Probabilistic XML. In *Updates in XML.* Lausanne, Switzerland, ACM.

Kimelfeld B., Kosharovsky Y., Sagiv Y. (2009, Oct). Query evaluation over probabilistic XML. *VLDB Journal*, vol. 18, n° 5, p. 1117–1140.

Kimelfeld B., Sagiv Y. (2008, Dec). Modeling and querying probabilistic XML data. *SIGMOD Rec.*, vol. 37, n° 4, p. 69-77.

Kimelfeld B., Senellart P. (2013). Probabilistic XML: Models and complexity. In Z. Ma, L. Yan (Eds.), *Advances in probabilistic databases for uncertain information management*, vol. 304, p. 39–66. Springer-Verlag.

Koc A., Tansel A. U. (2011, Mar). A survey of version control systems. In *ICEME.* Orlando, Florida USA, IIS.

Lindholm T., Kangasharju J., Tarkoma S. (2006, Oct). Fast and simple XML tree differencing by sequence alignment. In *DocEng*, p. 75–84. Amsterdam, The Netherlands, ACM.

Maniu S., Cautis B., Abdessalem T. (2011a, Jun). Building a signed network from interactions in Wikipedia. In *DBSocial*, p. 19–24. Athens, Greece, ACM.

Maniu S., Cautis B., Abdessalem T. (2011b, Mar). Casting a web of trust over Wikipedia : an interaction-based approach. In *WWW (companion volume)*, p. 87–88. Hyderabad, India, ACM.

Mitchell T. M. (1979). *Version Spaces : An approach to concept learning.* Thèse de doctorat non publiée, Stanford, CA, USA.

Nierman A., Jagadish H. V. (2002, Aug). ProTDB : probabilistic data in XML. In *VLDB*, p. 646–657. Hong Kong, China, VLDB Endowment.

Pilato M. (2004). *Version Control With Subversion*. Sebastopol, CA, USA, O'Reilly & Associates, Inc.

Rönnau S., Borghoff U. (2009, Jul). Versioning XML-based office documents. *Multimedia Tools and Applications*, vol. 43, n° 3, p. 253–274.

Rönnau S., Borghoff U. (2012, May). XCC : change control of XML documents. *Computer Science - Research and Development*, vol. 27.

Rusu L. I., Rahayu W., Taniar D. (2005, Nov). Maintaining versions of dynamic XML documents. In *WISE*, p. 536–543. New York, USA, Springer Berlin Heidelberg.

Sabel M. (2007, Oct). Structuring wiki revision history. In *WikiSym.* Montréal, Québec, Canada, ACM.

Suciu D., Olteanu D., Ré C., Koch C. (2011). *Probabilistic databases*. Morgan & Claypool Publishers.

Thao C., Munson E. V. (2011, Sep). Version-aware XML documents. In *DocEng*, p. 97-100. Mountain View, California, USA, ACM.

Van Keulen M., Keijzer A. de, Alink W. (2005, Apr). A Probabilistic XML Approach to Data Integration. In *ICDE*, p. 459-470. Tokyo, Japan, IEEE Computer Society.

Voss J. (2005, Jul). Measuring Wikipedia. In *ISSI.* Stockholm, Sweden, In Press.

Wang Y., DeWitt D. J., Cai J.-Y. (2003, Mar). X-Diff : An Effective Change Detection Algorithm for XML Documents. In *ICDE*, p. 519-530. Bangalore, India, IEEE Computer Society.

Zhang J., Jagadish H. V. (2013, Apr). Revision provenance in text documents of asynchronous collaboration. In *ICDE*, p. 889-900. Brisbane, Australia, IEEE Computer Society.