Contrôle de version incertain dans l'édition collaborative ouverte de documents arborescents

M. Lamine Ba
Institut Mines–Télécom; Télécom ParisTech; LTCI
Paris, France
mouhamadou.ba@telecom-paristech.fr

Talel Abdessalem
Institut Mines–Télécom; Télécom ParisTech; LTCI
Paris, France
talel.abdessalem@telecom-paristech.fr

Pierre Senellart
Télécom ParisTech & The University of Hong Kong
Paris, France & Hong Kong
pierre.senellart@telecom-paristech.fr

En vue de faciliter l'enrichissement, l'échange et le partage de contenu, les plates-formes collaboratives Web telles que Wikipedia ou Google Docs permettent des interactions à large échelle entre un grand nombre de contributeurs. Cette collaboration ne requiert pas une connaissance préalable du niveau d'expertise et de fiabilité des participants. La gestion de version est donc essentielle pour garder une trace de l'évolution du contenu partagé et de la provenance des contributions. Dans de tels environnements, l'incertitude est malheureusement omniprésente à cause des sources non fiables, des contributions incomplètes et imprécises, des éditions malveillantes et des actes de vandalisme possible, etc. Pour gérer cette incertitude, nous utilisons un modèle XML probabiliste comme élément de base de notre système de contrôle de version. Chaque version d'un document partagé est représenté par un arbre XML et le document tout en entier, incluant toutes ses différentes versions, est modélisé en un document XML probabiliste. L'incertitude est évaluée via le modèle probabiliste et la mesure 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. Nous démontrons que les opérations classiques de gestion de version peuvent être implémentées directement comme opérations sur le modèle XML probabiliste ; son efficacité comparée aux systèmes de contrôle de version déterministes est démontrée sur des données réelles.

Mots-clés
XML, travail collaboratif, données incertaines, gestion de version, documents arborescents

1 Introduction

Version Control in Open Environments In many collaborative editing systems, where several users can provide content, content management is based on version control. A version control system tracks the versions of the content as well as changes. Such a system enables fixing error made in the revision process, querying past versions, and integration of content from different contributors. As surveyed in [12, 27], much effort related to version control has been carried out both in research and in applications. The prime applications were collaborative document authoring process, computer-aided design, and software development systems. Currently, powerful version control tools, such as Subversion [19] and Git [16], efficiently manage large source code repositories and shared filesystems.

However, existing approaches leave no room for uncertainty handling, for instance, uncertain data resulting from conflicts. Conflicts are common in collaborative editing tasks, in particular in an open environment. They arise whenever concurrent edits attempt to change the same content. As a result, conflicts introduce some ambiguities in content change management. But sources of uncertainties in the version control process are not only due to conflicts. Indeed, there are inherently uncertain applications using version control, such as web-scale collaborative platforms: Platforms such as Wikipedia [6] or Google Docs [2] enable unbounded interactions between a large number of contributors, without prior knowledge of their level of expertise and reliability. In these systems, version control is used 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. Therefore, a version control technique able to properly manipulate uncertain data may be very helpful in this kind of applications. We detail application scenarios next.

Uncertainty in Wikipedia Versions Some web-scale collaborative systems such as Wikipedia have no write-access restrictions over documents. As a result, multi-version documents include data from different users. As shown in [39], Wikipedia has known an exponential growth of contributors and editions per articles. The open and free features lead to contributions with variable reliability and consistency depending both on the contributors' expertise (e.g., novice or expert) and the scope of the debated subjects. At the same time, edit wars, malicious contributions like spams, and vandalism acts can happen at any time during document evolution. Therefore, the integrity and the quality of each article may be strongly altered. Suggested solutions to these critical issues are reviewing access policies for articles discussing hot topics, or quality-driven solutions based on the reputations of authors, statistics on frequency of content change, or the trust a given reader has on the information [10,21,30]. But restricting editions on Wikipedia articles to a certain group of privileged contributors does not suppress the necessity of representing and assessing uncertainties. Indeed, edits may be incomplete, imprecise or uncertain, showing partial views, misinformations or subjective opinions. The reputation of contributors or the confidence level on sources are useful information towards a quantitative evaluation of the quality of versions and even more of each atomic contribution. However, a prior efficient representation of uncertainty across document versions remains a prerequisite.

User Preference at Visualization Time Filtering and visualizing content are also important features in collaborative environments. In Wikipedia, users are not only contributors, but also consumers, interested in searching and reading information on multi-version articles. Current systems constrain the users to visualize either the latest revision of a given article, even though it may not be the most relevant, or the version at a specific date. Users, especially in universal knowledge management platforms like Wikipedia, may want to easily access more relevant versions or those of authors whom they trust. Filtering unreliable content is one of the benefits of our approach. It can be achieved easily by hiding the contributions of the offending source, for instance when a vandalism act is detected, or at query time to fit user preferences and trust in the contributors. Alternatively, to deal with misinformation, it seems useful to provide versions to users with information about their amount of uncertainty and the uncertainty of each part of their content. Last but not least, users at visualization time should be able to search for a document representing the outcome of combining parts (e.g., some of them might be incomplete, imprecise, and even uncertain taken apart) from different versions. We demonstrate in [7] an application of these new modes of interaction to Wikipedia revisions: an article is no longer considered as the last valid revision, but as a merge of all possible (uncertain) revisions.

Approach Since version control is primordial in uncertain web-scale collaborative systems, representing and evaluating uncertainties throughout data version management becomes crucial for enhancing collaboration and for overcoming problems such as conflict resolution and information reliability management. In this paper, we propose an uncertain XML version control model tailored to multi-version tree-structured documents in open collaborative editing contexts. Data, that is, office documents, HTML or XHTML documents, structured Wiki formats, etc., manipulated within the given application scenarios are tree-like or can be easily translated into this form; XML is a natural encoding for tree-structured data. Work related to XML version control has focused on change detection [18, 22, 28, 33, 40]. Only some, for instance [32, 34, 36], have proposed an extensive semi-structured data model aware of version control; see Section 6 for details. Uncertainty management in XML has received a great attention in the probabilistic database community, especially for data integration purposes. A set of elaborate uncertain (probabilistic) XML data models [9, 23, 31, 38] with several distinct semantics of probability distributions over data items, has been proposed. [9] and [23] follow a general probabilistic XML representation system defining the concept of probabilistic documents (abbr. p-documents) which generalizes previously proposed uncertain XML models.

In our model, we handle uncertain data through 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. At the abstract level, we consider a multi-version XML document with uncertain data based on random events, XML edit scripts attached to them and a directed acyclic graph of these events. For a concrete representation the whole document, with its different versions, is modeled as a probabilistic XML document representing an XML tree whose edges are annotated by propositional formulas over random events. Each propositional formula models both the semantics of uncertain editions (insertion and deletion) performed over a given part of the document and its provenance in the version control process. Uncertainty is evaluated

using the probabilistic model and the reliability measure associated to each source, each contributor, or each editing event, resulting in an uncertainty measure on each version and each part of the document. The directed acyclic graph of random events maintains the history of document evolution by keeping track of its different states and their derivation relationships. As last major contribution of this paper, we show that standard version control operations, in particular update operation, can be implemented directly as operations on the probabilistic XML model; efficiency with respect to deterministic version control systems like Git and Subversion is demonstrated on real-world datasets.

Outline After some preliminaries in Section 2, we review the probabilistic XML model we use in Section 3. We detail the proposed probabilistic XML version control model and some strong properties thereof in Section 4. In Section 5, we demonstrate the efficiency of our model with respect to deterministic version control systems through measures on real-world datasets, and we describe some of the content filtering capabilities (Cf. Section 5.2) of our approach. Finally, we review some related work in Section 6. Initial ideas leading to this work were presented as a PhD workshop article in [13]; the description of the model, with translations of version control operations into operations on the probabilistic XML model, proofs of translation correctness, and experimental validation, are fully novel.

This work is accepted for publication at the ACM DocEng 2013 conference [14].

2 Preliminaries

In this section, we present some basic version control notions and the semi-structured XML document model underlying our proposal. A *multi-version document* refers to a set of versions of the same document handled within a version control process. Each version of the document represents a given state (instance) of the evolution of this versioned document. A typical version control model is built on the following common notions.

Document version A version is a conventional term that refers to a document copy in document-oriented version control systems. The different versions of a document are related by derivation operations. A derivation consists of creating a new version by first copying a previously existing one before performing modifications. Some versions, representing variants, are in a derivation relationship with the same origin. The variants (parallel versions) characterize a non-linear editing history with several distinct branches of the same multi-version document. In this history, a branch is a linear sequence of versions. Instead of storing the complete content of each version, most version control approaches only maintains *diffs* between states, together with meta-information on states. These states (or commits in Git world [16]) model different sets of changes that are explicitly validated at distinct stages of the version control process. A state also comes with information about the context (e.g., author, date, comment) in which these modifications are done. As a consequence, each version depends on the complete history leading up to a given state. We will follow here the same approach for modeling the different versions of a document within our framework.

Version Space Since the content of each version is not fully saved, there must be manner to retrieve it when needed. The version space represents the editing history over a versioned document (e.g., wiki version history as given in [35]). It maintains necessary information related to the versions and their derivations. As mentioned above, a derivation relationship implies at least one input version (several incoming versions for merge operations) and an output version. Based on this, we model similarly to [16] a version space of any multi-version document as a *directed acyclic graph*.

Unordered XML Tree Documents Our motivating applications handle mostly tree-structured data. As a result, we consider data as unordered XML trees. Note that the proposed model can be extended to ordered trees (this may require restricting the set of valid versions to those complying with a specific order, we leave the details for future work); we choose unordered trees for convenience of exposition given that in many cases order is unimportant. Let us assume a finite set $\mathscr{L}$ of strings (i.e., labels or text data) and a finite set $\mathscr{I}$ of identifiers such that $\mathscr{L} \cap \mathscr{I} = \emptyset$. In addition, let Φ and α be respectively a labeling function and an identifying function. Formally, we define an *XML document* as an *unordered*, *labeled* tree $\mathscr{T}$ over identifiers in $\mathscr{I}$ with α and Φ mapping each node $x \in \mathscr{T}$ respectively to a unique identifier $\alpha(x) \in \mathscr{I}$ and to a string $\Phi(x) \in \mathscr{L}$. The tree is unranked, i.e., the number of children of each node in $\mathscr{T}$ is not assumed to be fixed. Given an XML tree $\mathscr{T}$, we define $\Phi(\mathscr{T})$ and $\alpha(\mathscr{T})$ as respectively the set of its node strings and the set of its node identifiers. For simplicity, we will assume all trees have the same root node (same label, same identifier).

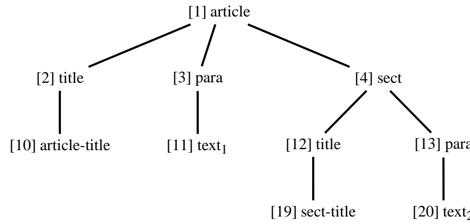

Figure 1: Example XML tree $\mathscr{T}$: Wikipedia article

Example 2.1. *Figure 1 depicts an XML tree $\mathscr{T}$ representing a typical Wikipedia article. The node identifiers are inside square brackets below node strings. The title of this article is given in node* 10. *The content of the document is structured in sections ("sect") with their titles and paragraphs ("para") containing the text data.*

XML Edit Script Based on unique identifiers, we consider two basic edit operations over the specified XML document model: node *insertions* and *deletions*. We denote an insertion by $\texttt{ins}^{i,\,x}$ whose semantics over any XML tree consists of inserting node x (we suppose x is not already in the tree) as a child of a certain node y satisfying $\alpha(y) = i$. If such a node is not found in the tree, the operation does nothing. Note that an insertion can concern a subtree, and in this case we simply refer with x to the root of this subtree. Similarly, we introduce a deletion as $\texttt{del}^{i}$ where i is the identifier of the node to suppress. The delete operation removes the targeted node, if it exists, together with its descendants, from the XML tree. We conclude

by defining an XML edit script, $\Delta = < u_1, u_2, \ldots, u_i >$, as a sequence of a certain number of elementary edit operations u_j (each u_j, with $1 \leq j \leq i$, being either an insertion or a deletion) to carry out one after the other on an XML document for producing a new one. Given a tree $\mathscr{T}$, we denote the outcome of applying an edit script Δ over $\mathscr{T}$ by $[\mathscr{T}]^{\Delta}$. Even though in this work we rely on persistent identifiers on tree nodes to define edit operations, the semantics of these operations could be extended to updates expressed by queries, especially useful in distributed collaborative editing environments where identifiers may not be straightforward to share.

3 Probabilistic XML

We briefly introduce in this section the probabilistic XML representation system we use as a basis of our uncertain version control system. For more details, see [9] for the general framework and [23] for the specific $\mathsf{PrXML}^{\mathsf{fie}}$ model we used. These representation systems are originally intended for XML-based applications such as Web data integration and extraction. For instance, when integrating various semi-structured Web catalogs containing personal data, some problems such as overlapping or contradiction are frequent. Typically, one can find for the same person name two distinct affiliations in different catalogs. A probabilistic XML model is used to automatically integrate such data sources by enumerating all possibilities: (a) the system considers each incoming source; (b) it maps its data items with the existing items in the probabilistic repository to find correspondences and; (c) giving that, it represents the matches as a set of possibilities. The resolution of conflicts is thus postponed to query time, where each query will return a set of possibilities together with their probabilities. The intuition is that resolving semantic issues before an effective integration is unfeasible in this situation. On one hand, it is often a tedious and error-prone resolution process. On the other hand, there might not be any certain knowledge about the reliability of the sources, and data completeness.

p-Documents A *probabilistic XML representation system* is a compact way of representing probability distributions over possible XML documents; in the case of interest here, the probability distribution is finite. Formally, a probabilistic XML distribution space, or px-space, $\mathscr{S}$ over a collection of uncertain XML documents is a couple (D,p) where D is a nonempty finite set of documents and $p : D \to (0,1]$ is a probability function that maps each document d in D to a rational number $p(d) \in (0,1]$ such that $\Sigma_{d \in D} p(d) = 1$. A *p-document*, or *probabilistic XML document*, usually denoted $\widehat{\mathscr{P}}$, defines a compact encoding of a px-space $\mathscr{S}$.

$\mathsf{PrXML}^{\mathsf{fie}}$: Syntax and Semantics We consider in this paper one specific class of p-documents, $\mathsf{PrXML}^{\mathsf{fie}}$ [23] (where *fie* stands for *formula of independent events*); restricting to this particular class allows us to give a simplified presentation, see [9, 23] for a more general setting. Assume a set of *independent random Boolean variables*, or *event variables* in short, $b_1, b_2, \ldots, b_m$ and their respective probabilities $Pr(b_1), Pr(b_2) \ldots, Pr(b_m)$ of existence. A $\mathsf{PrXML}^{\mathsf{fie}}$ p-document is an unordered, unranked, and labeled tree where every node (except

for the root) x may be annotated with an arbitrary propositional formula $\mathit{fie}(x)$ over the event variables $b_1, b_2, \ldots, b_m$. Different formulas can share common events, i.e., there may be some correlation between formulas and the number of event variables in the formulas may vary from one node to another.

A valuation ν of the event variables $b_1 \ldots b_m$ induces over $\widehat{\mathscr{P}}$ one particular XML documents $\nu(\widehat{\mathscr{P}})$: the document where only nodes annotated with formulas valuated to true by ν are kept (nodes whose formulas are valuated to false by ν are deleted from the tree, along with their descendants). Given a p-document $\widehat{\mathscr{P}}$, the *possible worlds* of $\widehat{\mathscr{P}}$, denoted as $\mathit{pwd}(\widehat{\mathscr{P}})$ is the set of all such XML documents. The *probability* of a given possible world d of $\widehat{\mathscr{P}}$ is defined as the sum of the probability of the valuations that yield d. The set of possible worlds, together with their probabilities, defines the *semantics* of $\widehat{\mathscr{P}}$, the px-space $[\![\widehat{\mathscr{P}}]\!]$ associated to $\widehat{\mathscr{P}}$.

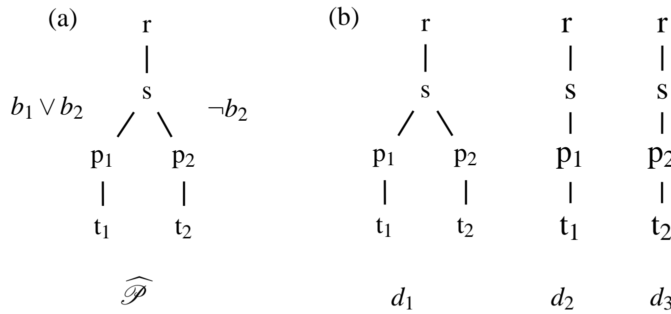

Figure 2: (a) $\mathsf{PrXML}^{\mathsf{fie}}$ p-document $\widehat{\mathscr{P}}$; (b) Three possible worlds d_1, d_2 and d_3

Example 3.1. *Figure 2 sketches on the left-side a concrete* $\mathsf{PrXML}^{\mathsf{fie}}$ *p-document* $\widehat{\mathscr{P}}$ *and on the right-side three possible worlds* d_1*,* d_2 *and* d_3*. Formulas annotating nodes are shown just above them:* $b_1 \vee b_2$ *and* $\neg b_2$ *are bound to nodes* p_1 *and* p_2 *respectively. The three possible worlds* d_1*,* d_2 *and* d_3 *are obtained by setting the following valuations of* b_1 *and* b_2*: (a)* true *and* false*; (b)* true *and* true *(or* false *and* true*); (c)* false *and* false*. At each execution of the random process, the distributional node chooses exactly the nodes whose formulas are evaluated at* true *given the valuation specified over event variables. Assuming a probability distribution over events, for instance* $Pr(b_1) = 0.4$ *and* $Pr(b_2) = 0.5$*, we derive the probability of the possible world* d_1 *as* $Pr(d_1) = Pr(b_1) \times (1 - Pr(b_2)) = 0.4 \times (1 - 0.5) = 0.2$*. We can compute similarly the probabilities of all other possible worlds.*

With respect to other probabilistic XML representation systems [9], $\mathsf{PrXML}^{\mathsf{fie}}$ is very succinct (since arbitrary propositional formulas can be used, involving arbitrary correlations among events), i.e., exponentially more succinct than the models of [31,38], and offers tractable insertions and deletions [23], one key requirement for our uncertain version control model. However, a non-negligible downside is that all non-trivial (tree-pattern) queries over this model are **#P**-hard to evaluate [24]. This is not necessarily an issue, here, since we favor in our application efficient updates and retrieval of given possible worlds, over arbitrary queries.

Data Provenance Uncertain XML management based on the PrXML$^{\text{fie}}$ model also takes advantage of the various possible semantics of event variables in terms of information description. Indeed, besides uncertainty management, the model also provide support for keeping information about *data provenance* (or lineage) based on the event variables. Data provenance is information of traceability such as change semantics, responsible party, timestamp, etc., related to uncertain data. To do so, we only need to use the semantics of event variables as representing information about data provenance. As such, it is sometimes useful to use probabilistic XML representation systems even in the absence of reliable probability sources for individual events, in the sense that one can manipulate them as incomplete data models (i.e., we only care about possible worlds, not about their probabilities).

4 Uncertain Multi-version XML

In this section we elaborate on our uncertain XML version control model for tree-structured documents edited in a collaborative manner. We build our model on three main concepts: version control events, a p-document, and a directed acyclic graph of events. We start by formalizing a multi-version XML document through a formal definition of its graph of version space and its set of versions. Then, we formally introduce the proposed model.

4.1 Multi-Version XML Documents

Consider the infinite set $\mathscr{D}$ of all XML documents with a given root label and identifier. Let $\mathscr{V}$ be a set of *version control events* $e_1, \dots, e_n$. These events represent the different states of a tree. We associate to events contextual information about revisions (authorship, timestamp, etc.). To each event e_i is further associated an *edit script* Δ_i. Based on this, we formalize the graph of version space and the set of versions of any versioned XML document as follows.

Graph of version space The *version space* is a rooted directed acyclic graph (DAG) $\mathscr{G} = (\mathscr{V} \cup \{e_0\}, \mathscr{E})$ where: (i) the initial version control event $e_0 \notin \mathscr{V}$, a special event representing the first state of any versioned XML tree, is the root of $\mathscr{G}$; (ii) $\mathscr{E} \subseteq \mathscr{V}^2$, defining the edges of $\mathscr{G}$, consists of a set of ordered couples of version control events. Each edge implicitly describes a directed derivation relationship between two versions. A *branch* of $\mathscr{G}$ is a directed path that implies a start node e_i and an end node e_j. The latter must be reachable from the former by traversing a set of ordered edges in $\mathscr{E}$. We refer to this branch by B_i^j. A *rooted branch* is a branch that starts at the root of the graph.

XML versions An XML version is the document in $\mathscr{D}$ corresponding to a *set* of version control events, the set of events that made this version happen. In a deterministic version control system, this set always corresponds to a rooted branch in the version space graph. In our uncertain version control system, this set may be arbitrary. Let us consider the set $2^{\mathscr{V}}$ comprising all sub-parts of $\mathscr{V}$. The set of versions of a multi-version XML document is given by a mapping $\Omega : 2^{\mathscr{V}} \to \mathscr{D}$: to each sets of events corresponds a given tree (these trees are

typically not all distinct). The function Ω can be computed from edit scripts associated with events as follows:

- $\Omega(\emptyset)$ maps to the root-only XML tree of $\mathscr{D}$.
- For all i, for all $\mathscr{F} \subseteq 2^{\mathscr{V}\setminus\{e_i\}}$ $\Omega(\{e_i\} \cup \mathscr{F}) = [\Omega(\mathscr{F})]^{\Delta_i}$.

A multi-version XML document, $\mathscr{T}_{mv}$, is now defined as a pair $(\mathscr{G},\Omega)$ where $\mathscr{G}$ is a DAG of version control events, whereas Ω is a mapping function specifying the set of versions of the document. In the following we propose a more efficient way to compute the version corresponding to a set of events, using a p-document for storage.

4.2 Uncertain Multi-Version XML Documents

A multi-version document will be *uncertain* if the version control events, staged in a version control process, come with *uncertainty* as in open collaborative contexts. By version control events with uncertainty, we mean random events leading to uncertain versions and content. As a consequence, we will rely on a *probability distribution over* $2^{\mathscr{V}}$, that will, together with the Ω mapping, imply a probability distribution over $\mathscr{D}$.

Uncertainty modeling We model uncertainty in events by further defining a version control event e_i in $\mathscr{V}$ as a conjunction of semantically unrelated random Boolean variables $b_1,\dots,b_m$ with the following assumptions: (i) a Boolean variable models a given source of uncertainty (e.g., the contributor) in the version control environment; (ii) all Boolean variables in each e_i are independent; (iii) a Boolean variable b_j reused across events correlates different version control events; (iv) one particular Boolean *revision* variable $b^{(i)}$, representing more specifically the uncertainty in the contribution, is not shared across other version control events and appears positively in e_i.

Probability Computation We assume given a probability distribution over the Boolean random variables b_j's (this typically comes from a trust estimation in a contributor, or in a contribution), which induces a probability distribution over propositional formulas over the b_j's in the usual manner [23]. We now obtain the probability of each (uncertain) version d of as follows: $\Pr(d) = \Pr(\bigvee_{\substack{\mathscr{F}\subseteq\mathscr{V}\\ \Omega(\mathscr{F})=d}} \mathscr{F})$ with the probability of each set of events $\mathscr{F} \subseteq \mathscr{V}$ given by:

$$\Pr(\mathscr{F}) = \Pr\left(\bigwedge_{e_j\in\mathscr{F}} e_j \wedge \bigwedge_{e_k\in\mathscr{V}\setminus\mathscr{F}} \neg e_k\right). \tag{1}$$

Example 4.1. *Figure 3 sketches an uncertain multi-version XML document $\mathscr{T}_{mv}$ with four staged version control events. On the left-side, we have the version space $\mathscr{G}$. The right-side shows an example of four possible (uncertain) versions and their associated event set. We suppose that $\mathscr{T}_{mv}$ is initially a root-only document. The three first versions correspond to versions covered by deterministic version control systems, whereas the last one is generated by considering that the changes performed at an intermediate version control event, here e_2, as incorrect. One feature of our model is to provide the possibility for viewing and modifying*

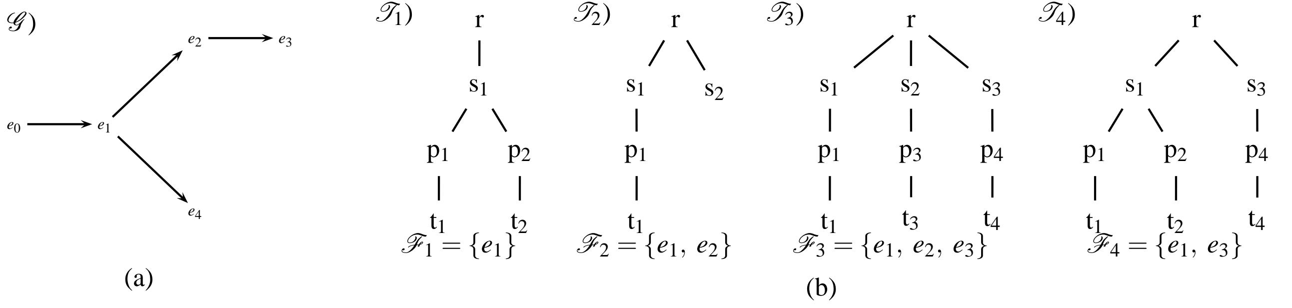

Figure 3: (a) Graph of Version Space; (b) Four versions and their mapping truth-values

these kinds of uncertain versions representing virtual versions. Only edits performed at the specified version control events are taken into account in the process of producing a version: in $\mathscr{T}_4$, the node r and the subtrees rooted at s_1, s_3 respectively introduced at e_0, e_1 and e_3 are present, while the subtree p_3 added at e_3 does not appear because its parent node s_2 cannot be found. Finally, given probabilities of version control events, we are able to measure the reliability of each uncertain version $\mathscr{T}_i$, for each $1 \leq i \leq 4$, based on its corresponding event set $\mathscr{F}_i$ (and all other event sets that map to the same tree).

We straightforwardly observe, for instance with the simple example in Figure 3, that the amount of possible (uncertain) versions of any uncertain multi-version document may grow rapidly (indeed, exponentially in the number of events). As a result, the enumeration and the handling of all the possibilities with the function Ω may become tedious at a certain point. To address this issue, we propose an efficient method for encoding in a compact manner the possible versions together with their truth values. Intuitively, a $\mathsf{PrXML}^{\mathsf{fie}}$ p-document compactly models the set of possible versions of an uncertain multi-version XML document. As stressed in Section 3, a probabilistic tree based on propositional formulas provides interesting features for our setting. First, it describes well a distribution of truth values over a set of uncertain XML trees while providing a meaningful process to find back a given version and its probability. Second, it provides an update-efficient representation system, which is crucial in dynamic environments such as version-control–based applications.

4.3 Probabilistic XML Encoding

We introduce a general uncertain XML version control representation framework, denoted by $\widehat{\mathscr{T}_{mv}}$, as a couple $(\mathscr{G}, \widehat{\mathscr{P}})$ where (a) $\mathscr{G}$ is as before a DAG of events, representing the version space; (b) $\widehat{\mathscr{P}}$ is a $\mathsf{PrXML}^{\mathsf{fie}}$ p-document with random Boolean variables $b_1 \dots b_m$ representing efficiently all possible (uncertain) XML tree versions and their corresponding truth-values.

We now define the semantics of such an encoding as the uncertain multi-version document $(\mathscr{G}, \Omega)$ where $\mathscr{G}$ is the same and Ω is defined as follows. For all $\mathscr{F} \subseteq \mathscr{V}$, let B^+ be the set of all random variables occurring in one of the events of $\mathscr{F}$ and B^- be the set of all revision variables $b^{(i)}$'s for e_i *not in* $\mathscr{F}$. Let ν be the valuation of $b_1 \dots b_m$ that sets variables of B^+ to true, variables of B^- to false, and other variables to an arbitrary value. We set $\Omega(\mathscr{F}) := \nu(\widehat{\mathscr{P}})$.

The following shows that this semantics is compatible with the px-space semantics of p-documents on the one hand, and the probability distribution defined by uncertain multi-version documents on the other hand.

Proposition 4.1. *Let* $(\mathcal{G}, \widehat{\mathcal{P}})$ *be an uncertain version control representation framework and* $(\mathcal{G}, \Omega)$ *its semantics as just defined. We further assume that all formulas occurring in* $\widehat{\mathcal{P}}$ *can be expressed as formulas over the events of* $\mathcal{V}$ *(i.e., we do not make use of the* b_j*'s independently of version control events). Then the px-space* $[\![\widehat{\mathcal{P}}]\!]$ *defines the same probability distribution over* $\mathcal{D}$ *as* Ω*.*

The proof is straightforward and relies on Equation (1).

4.4 Updating Uncertain Multi-Version XML

We implement the semantics of standard update operations on top of our probabilistic XML representation system. An update over an uncertain multi-version document corresponds to the evaluation of some uncertain edits on a given (uncertain) version. With the help of a triple (Δ, e, e'), we refer to an update operation as $\texttt{updOP}_{\Delta,\ e,\ e'}$ where Δ is an edit script, e is an existing version control event pointing to the edited version and e' is an incoming version control event evaluating the amount of uncertainty in this update. We formalize $\texttt{updOP}_{\Delta,\ e,\ e'}$ over $\mathcal{T}_{mv}$ as below.

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

An update operation thus results in the insertion of a new node and a new edge in $\mathcal{G}$, and an extension of Ω with Ω' that we now define. For any subset $\mathcal{F} \subseteq \mathcal{V}'$ ($\mathcal{V}'$ is the set of nodes in $\mathcal{G}$ after the update), we have:

- if $e' \notin \mathcal{F}$: $\Omega'(\mathcal{F}) = \Omega(\mathcal{F})$;
- otherwise: $\Omega'(\mathcal{F}) = [\Omega(\mathcal{F} \backslash \{e'\})]^{\Delta}$.

What precedes gives a semantics to updates on uncertain multi-version documents; however, the semantics is not practical as it requires considering every subset $\mathcal{F} \subseteq \mathcal{V}'$. For a more usable solution, we perform updates directly on the p-document representation of the multi-version document. Algorithm 1 describes how such an update operation $\texttt{updOP}_{\Delta,e,e'}$ is performed on top of an uncertain representation $(\mathcal{G}, \widehat{\mathcal{P}})$. First, the graph is updated as before. Then, for each operation u in Δ, the algorithm retrieves the targeted node in $\widehat{\mathcal{P}}$ using `findNodeById` (typically this is a constant-time operation). According to the type of operation, there are two possibilities.

1. If u is an insertion of a node x, the algorithm checks if x does not already occur in $\widehat{\mathcal{P}}$, for instance by looking for a node with the same label (the function `matchIsFound` searches a matching for x in the subtree $\mathcal{T}_y$ rooted at y). If such a matching exists, `getFieOfNode` returns its current formula $\mathit{fie}_o(x)$ and the algorithm updates it to $\mathit{fie}_n(x) := \mathit{fie}_o(x) \vee e'$, specifying that x appears when this update is valid. Otherwise, `updContent` and `setFieOfNode` respectively inserts the node x in $\widehat{\mathcal{P}}$ and sets its associated formula as $\mathit{fie}_n(x) = e'$.

```
Input: (𝒢, 𝒫̂), updOP_{Δ,e,e'}
Output: updating 𝒯_mv in 𝒯̂_mv
𝒢 := 𝒢 ∪ ({e'}, {(e, e')});
foreach (u in Δ) do
    if u = ins^{i, x} then
        y := findNodeById(𝒫̂, i) ;
        if matchIsFound(𝒯_y, x) then
            fie_o(x) := getFieOfNode(x) ;
            setFieOfNode(x, fie_o(x) ∨ e');
        else
            updContent(𝒫̂, ins^{i, x});
            setFieOfNode(x, e');
    else
        x := findNodeById(𝒫̂, i) ;
        fie_o(x) := getFieOfNode(x) ;
        setFieOfNode(x, fie_o(x) ∧ ¬e');
return (𝒢, 𝒫̂);
```

Algorithm 1: Update algorithm

2. If u is a deletion of a node x, the algorithm gets its current formula $fie_o(x)$ and sets it to $fie_n(x) := fie_o(x) \wedge \neg e'$, specifying that x must be removed from possible worlds where this update is valid.

The rest of this section shows the correctness and efficiency of our approach: First, we establish that Algorithm 1 respects the semantics of updates. Second, we show that the behavior of deterministic version control systems can be simulated by considering only a specific kind of event set. Third, we characterize the complexity of the algorithm.

Proposition 4.2. *Algorithm 1, when ran on a probabilistic XML encoding* $\widehat{\mathcal{T}_{mv}} = (\mathcal{G}, \widehat{\mathcal{P}})$ *of a multi-version document* $\mathcal{T}_{mv} = (\mathcal{G}, \Omega)$, *together with an update operation* $\mathtt{updOP}_{\Delta,e,e'}$, *computes a representation* $\mathtt{updOP}_{\Delta,e,e'}(\widehat{\mathcal{T}_{mv}})$ *of the multi-version document* $\mathtt{updOP}_{\Delta,e,e'}(\mathcal{T}_{mv})$.

Proof. Let: $\begin{cases} \mathtt{updOP}_{\Delta,e,e'}(\widehat{\mathcal{T}_{mv}}) = (\mathcal{G}', \widehat{\mathcal{P}'}) \\ \mathtt{updOP}_{\Delta,e,e'}(\mathcal{T}_{mv}) = (\mathcal{G}', \Omega') \end{cases}$ (it is clear that the version space DAG is the same in both cases). We need to show that Ω' corresponds to the semantics of $\widehat{\mathcal{P}'}$; that is, if we note the semantics of $(\mathcal{G}', \widehat{\mathcal{P}'})$ as $(\mathcal{G}', \Omega'')$, we need to show that $\Omega' = \Omega''$. By definition, for $\mathcal{F} \subseteq \mathcal{V}'$, $\Omega'(\mathcal{F}) = \Omega(\mathcal{F})$ if $e' \notin \mathcal{F}$, and $\Omega'(\mathcal{F}) = [\Omega(\mathcal{F} \backslash \{e'\})]^{\Delta}$ otherwise. Let us distinguish these two cases.

In the first scenario implying subsets $\mathcal{F}$ which do not contain e', we have $\Omega'(\mathcal{F}) = \Omega(\mathcal{F})$. Since $\mathcal{T}_{mv}$ is the semantics of $\widehat{\mathcal{T}_{mv}}$, we know that $\Omega(\mathcal{F}) = \nu(\mathcal{F})$ for a valuation ν that sets the special revision variable b' corresponding to e' to false. Now, let us look at the document $\nu(\widehat{\mathcal{P}'})$. By construction the update algorithm does not delete any node from $\widehat{\mathcal{P}}$ but just

inserts new nodes and modifies some formulas. Suppose that there exists a node $x \in \nu(\widehat{\mathscr{P}})$ such that $x \notin \nu(\widehat{\mathscr{P}'})$. Since $x \in \nu(\widehat{\mathscr{P}})$, x cannot be a new node in $\widehat{\mathscr{P}'}$. Thereby, its new formula $\mathit{fie}_n(x)$ after the update is either $\mathit{fie}_o(x) \vee e'$ or $\mathit{fie}_o(x) \wedge \neg e'$. In both cases, $\mathit{fie}_n(x)$ satisfies ν, because $\mathit{fie}_o(x)$ satisfies ν and ν sets b' (and therefore e') to false. This leads to a contradiction and we can conclude that for all node $x \in \nu(\widehat{\mathscr{P}})$, we have $x \in \nu(\widehat{\mathscr{P}'})$. Similarly, if a node x is in $\mathscr{F}(\widehat{\mathscr{P}'})$, because ν sets e' to false, x will also be in $\nu(\widehat{\mathscr{P}})$. Combining the two, $\Omega''(\mathscr{F}) = \nu(\widehat{\mathscr{P}'}) = \nu(\widehat{\mathscr{P}}) = \Omega(\mathscr{F})$.

The second scenario concerns subsets $\mathscr{F}'$ in which e' appears. We obtain a version $\Omega'(\mathscr{F}')$ by updating $\Omega(\mathscr{F}'\backslash\{e'\})$ with Δ. Let us set $\mathscr{F} = \mathscr{F}'\backslash\{e'\}$. There exists a valuation ν such that $\nu(\widehat{\mathscr{P}}) = \Omega$ (and thus, $\Omega'(\mathscr{F}') = [\nu(\widehat{\mathscr{P}})]^{\Delta}$) with ν setting all variables of events in $\mathscr{F}$ to true, and making sure that all other events are set to false. Let ν' be the extension of ν where all variables of e' are set to true. It suffices to prove that $[\nu(\widehat{\mathscr{P}})]^{\Delta} = \nu'(\widehat{\mathscr{P}'})$. First, it is clear that the nodes in $\nu(\widehat{\mathscr{P}})$ which are not modified by Δ are also in $\nu'(\widehat{\mathscr{P}'})$. Indeed, their associated formulas do not change in $\widehat{\mathscr{P}'}$, and hence the fact these satisfy ν are sufficient for selecting them in $\widehat{\mathscr{P}'}$ with the valuation ν'. Suppose now an operation u in Δ involving a node x: u either adds x as a child of a certain node y or deletes x. In the former case, if y exists in $\nu(\widehat{\mathscr{P}})$, then its formula satisfies ν and x is added in the document when it does not already exist. With Algorithm 1, u is interpreted in $\widehat{\mathscr{P}'}$ by the existence of x under y with an attached formula being either $\mathit{fie}_n(x) = e'$ (newly added) or $\mathit{fie}_n(x) = \mathit{fie}_o(x) \vee e'$ (reverted node). As a consequence, $\nu'(\widehat{\mathscr{P}'})$ selects x as in both possible expressions of $\mathit{fie}_n(x)$. Let us analyze the case where u is a deletion of x. If x is not present in $\nu(\widehat{\mathscr{P}})$, i.e., u changes nothing in this document. Through Algorithm 1, u results in a new associated formula set to $\mathit{fie}_n(x) = \mathit{fie}_o(x) \wedge \neg e$ for the node x in $\widehat{\mathscr{P}'}$. Obviously, we can see that x will not be in $\nu'(\widehat{\mathscr{P}'})$ because the satisfiability of $\mathit{fie}_n(x)$ requires the falseness of e' whose condition does not hold in $\mathscr{F}$. Now, if x is found in $\nu(\widehat{\mathscr{P}})$, u deletes the node, as well as its children, from the document. As a result, the outcome does not contain x, which is conform to the fact that $x \notin \nu'(\widehat{\mathscr{P}'})$. We have proved that for all node x in $[\nu(\widehat{\mathscr{P}})]^{\Delta}$, x is also in $\nu'(\widehat{\mathscr{P}'})$. By similar arguments, we can show that the converse is verified, i.e., for all node x in $\nu'(\widehat{\mathscr{P}'})$, x belongs to $[\nu(\widehat{\mathscr{P}})]^{\Delta}$. □

The semantics of update is therefore the same, whether stated on uncertain multi-version documents, or implemented as in Algorithm 1. We now show that this semantics is compatible with the classical update operation of version control systems.

Proposition 4.3. *The formal definition of updating in uncertain multi-version documents implements the semantics of the standard update operation in deterministic version control systems when sets of events are restricted to rooted branches.*

Proof. (Sketch) The update in our model changes the version space $\mathscr{G}$ similarly to a deterministic version control setting. As for its evaluation over the set of versions, we only need to show that the operation also produces a new version by updating the version mapping B_0^i (with e the ith version control event in $\mathscr{G}$) with Δ as in a deterministic formalism. For building the resulting version set, the operation as given above is defined such that for all subset $\mathscr{F} \subseteq \mathscr{V}$

with $e \in \mathscr{F}$, we carry out Δ on $\Omega(\mathscr{F})$ for producing a new version $\Omega'(\mathscr{F} \cup \{e'\})$. Amongst all the subsets satisfying this condition, obviously there is at least one which maps to B_0^i. □

We conclude by showing our algorithm is fully scalable:

Proposition 4.4. *Algorithm 1 performs the update process over the representation of any uncertain multi-version XML document with a constant time complexity with respect to the size of the input document. The size of the output probabilistic tree grows linearly in the size of the update script.*

Proof. The first part of the algorithm consists in updating $\mathscr{G}$. This is clearly a constant-time operation, which results in a single new node and a single new edge in $\mathscr{G}$ for every edit script. As for the second part of the algorithm, i.e., the evaluation of the update script over the probabilistic tree, let $|\widehat{\mathscr{P}}|$ and $|\Delta|$ be respectively the size of the input probabilistic document $\widehat{\mathscr{P}}$ and the length of Δ. By implementing $\widehat{\mathscr{P}}$ as an amortized hash table, we execute a lookup of nodes in $\widehat{\mathscr{P}}$ based on `findNodeById` or `matchIsFound` in constant time. (`matchIsFound` requires storing hashes of all subtrees of the tree, but this data structure can be maintained efficiently – we omit the details here.) The upper bound of Algorithm 1 occurs when Δ consists only of insertions. Since the functions `getFieOfNode`, `updContent` and `setFieOfNode` also have constant execution costs, we can state that the overall running time of Algorithm 1 is only a function of the number of operations in Δ. As a result, we can conclude that the update algorithm performs in $O(1)$ with respect to the number of nodes in $\widehat{\mathscr{P}}$ and $\mathscr{G}$.

At each execution, Algorithm 1 will increase the input probabilistic tree by a size bounded by a constant for each update operation, together with the size of all inserts. To sum up, the size increase is linear in the size of the original edit script. □

5 Evaluation of the model

This section describes the experimental evaluation of the proposed model, based on real-world applications. We first present a comparative study of our model with two popular version control systems Git and Subversion, in order to prove its efficiency. Then we describe the advances in terms of content filtering offered by our model.

All times shown are CPU time, obtained by running in-memory tests, avoiding disk I/O costs by putting all accessed file systems in a RAM disk. Measures have been carried out using the same settings for all three systems.

5.1 Performance analysis

We measured the time needed for the execution of two main operations: the commit and checkout of a version. The tests where conducted on Git, Subversion, and the implementation of our model (PrXML). The goal is to show the feasibility of our model rather than to prove that it is more efficient than the mentioned version control systems. We stress that, though for comparison purposes our system was tested in a deterministic setting, its main interest relies in the fact that it is able to represent uncertain multi-version documents, as we illustrate further in Section 5.2.

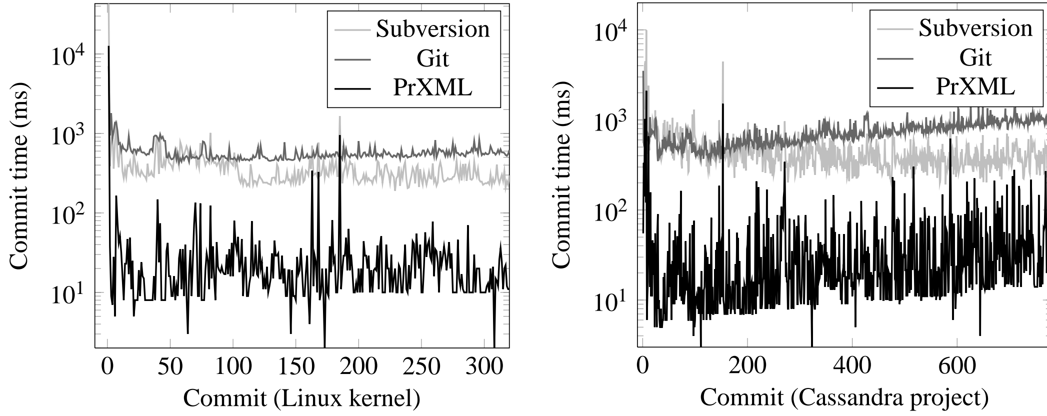

Figure 4: Measures of commit time over real-world datasets (logarithmic y-axis)

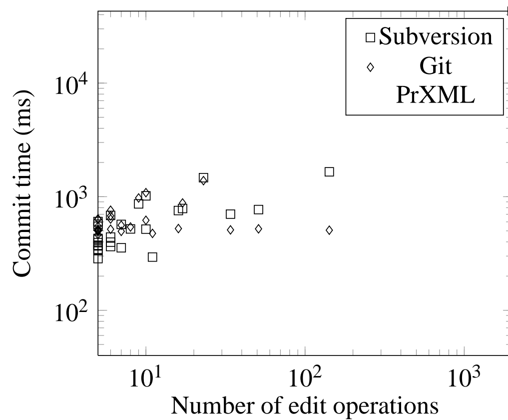

Figure 5: Commit time vs number of edit operations (for edit scripts of length ≥ 5)

Datasets and Implementation. As datasets, we used the history of the master branches of the *Linux kernel development* [4] and the *Apache Cassandra project* [1] for the tests. These data represent two large file systems and constitute two good examples of tree-structured data shared in an open and collaborative environment. The Linux kernel development natively uses Git. We obtained a local copy of its history by cloning the master development branch. We maintained up-to-date our local copy by pulling every day the latest changes from the original source. We followed a similar process with the Cassandra dataset (a Subversion repository).

In total, each local branch has more than ten thousand commits (or revisions). Each commit materializes a set of changes, to the content of files or to their hierarchy (the file system tree). In our experiments, we focused on the commits applied to the file system tree and ignored content change. We determined the commits and the derivation relationships from Git and Subversion logs. We represented the file system in an XML document and we transposed the atomic changes to the file system into edit operations on the XML tree. To each insertion, respectively deletion, of a file or a directory in the file system corresponds an insertion, respectively a deletion, of a node in the XML tree.

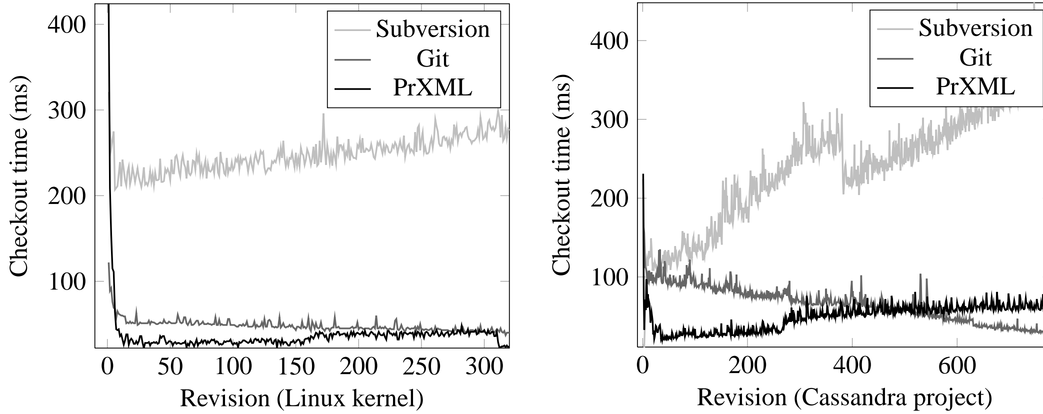

Figure 6: Measures of checkout time over real-world datasets (linear axes)

We implemented our version control model (PrXML) in Java. We used the Java APIs SVNKit [5] and JGit [3] to set up the standard operations of Subversion and Git. The purpose was to perform all the evaluations in the same conditions. Subversion uses a set of log files to track the changes applied to the file system at the different commits. Each log file contains a set of paths and the change operations associated to each path. As for Git, it handles several versions of a file system as a set of related Git tree objects represented by the hashes of their content. A Git tree object represents a snapshot of the file system at a given commit.

Cost analysis. Figures 4 and 6 compare the cost of the *commit* and the *checkout* operations in Subversion, Git, and PrXML. The commit time indicates the time needed by the system to create a version (commit), whereas the checkout time corresponds to the time necessary to compute and retrieve the sought version. The obtained results show clearly that PrXML has good performance with respect to Git and Subversion systems. The experiments were done using the datasets obtained from the Linux Kernel and Cassandra projects, as indicated above. For both datasets, we observe in Figure 4 that our model has in general a low commit cost (note that the y-axes are logarithmic on Figure 4).

An in-depth analysis of the results show that the commit costs depend in our model on the number of edit operations associated to the commits (see Figure 5), as implied by Proposition 4.4. However, PrXML remains efficient compared to the other systems, except for some few commits characterized by a large number of edits (at least one hundred edit operations). This can be explained by the fact that our model performs the edit operations over XML trees, whereas Git stores the hashes of the files indexed by the directory names, and Subversion logs the changes together with the targeted paths in flat files. An insertion of a subtree (a hierarchy of files and directories) in the file system can be treated as a simple operation in Git and Subversion, whereas it requires a series of node insertions in our model.

Our model is able to generate linear versions (corresponding to event sets that are rooted branches) as well as arbitrary ones. However, traditional version control systems are only able to produce linear versions. As a consequence, in this paper we focused our experiments on retrieving linear versions for comparison purposes. Figure 6 shows the measures obtained for

the checkout of successive versions in PrXML, Git and Subversion. The x-axis represents version numbers. Retrieving a version number n requires the reconstruction of all previous versions (1 to $n-1$). The results obtained show that our model is significantly more efficient than Subversion for both datasets (Linux Kernel and Cassandra projects). Compared to Git, PrXML has a lower checkout cost for initial versions, while it becomes less efficient in retrieving recent versions for the Cassandra dataset. Note that, traditional version control models mostly use reversible diffs [34] in order to speed up the process of reconstructing the recent versions in a linear history.

5.2 Filtering capabilities

Efficient evaluation of the uncertainty and automatic filtering of unreliable contents are two key issues for large scale collaborative editing systems. Evaluation of uncertainty is needed because a shared document can result from contributions of different persons, who may have different levels of reliability. This reliability can be estimated in various ways, such as an indicator of the overall reputation of an author (possibly automatically derived from the content of contributions, cf. [10]) or the subjective trust a given reader has in the contributor. For popular collaborative platforms, like Wikipedia, an automatic management of conflicts is also necessary because the number of contributors is often very large. This is especially true for documents related to hot topics, where the number of conflicts and vandalism acts can evolve rapidly and compromise document integrity.

In our model, filtering unreliable contents can be done easily by setting to false the Boolean variables modeling the corresponding sources. This can be done automatically, for instance when a vandalism act is detected, or at query time to fit user preferences and opinion about the contributors. A shared document can also be regarded as the merge of all possible worlds modeled by the generated revisions. We demonstrate in [7] an application of these new filtering and interaction capabilities to Wikipedia revisions: an article is no longer considered as the last valid revision, but as a merge of all possible (uncertain) revisions. The overall uncertainty on a given part of the article is derived from the uncertainty of the revisions having affected it. Moreover, the user can view the state of a document at a given revision, removing the effect of a given revision or a given contributor, or focusing only on the effect of some chosen revisions or some reliable contributors.

We also tested the possibility for the users to handle more advanced operations over critical versions of articles such as vandalized versions. We chose the most vandalized Wikipedia articles (as given by *Wikipedia:Most_vandalized_pages*), and we used our model to study the impact of considering as reliable some versions affected by vandalism. We succeeded in reconstructing the chosen articles as if the vandalism had never been removed; obtaining this special version of the article is very efficient, since it consists in applying a given valuation to the probabilistic document, which is a checkout operation whose timing is comparable to what is shown in Figure 6. Note that in the current version of Wikipedia, the content of vandalized versions is systematically removed from the presented version of an article, even if some users may want to visualize them for various reasons. Our experiments have shown that we can detect the vandalism as well as Wikipedia robots do, and automatically manage it in PrXML, keeping all uncertain versions available for checkout.

6 Related work

Our previous work. We present in [7, 13] initial studies towards the design of an uncertain XML version control system: [7] is a demonstration system focusing on Wikipedia revisions and showing the benefits of integrating an uncertain XML version control approach in web-scale collaborative platforms; [13] is a PhD workshop paper with early ideas behind modeling XML uncertain version control.

Version Control Systems. While a lot of work was carried out on version control in object-oriented systems (e.g., [8, 11, 15, 20]), recent research and tools are focusing on document-oriented models. Many products, seen as *general-purpose systems*, are used for version control over different kind of documents. *Subversion, ClearCase, Git, BitKeeper*, and *Bazaar* are some examples of them. In general, the considered approaches do not take into account the semantics of the changes represented by the successive versions. The concern is the reconstruction of the committed versions, rather then the understanding of the evolution of the modeled world. In Subversion [19] and similar systems, version control is based on edit distance algorithms designed for flat text, whereas the Git family [16] of tools uses cryptographic approaches. For XML and structured documents, both techniques are inadequate because the semantics of the changes is crucial in this case. A lot of work was done on change detection on XML documents, and different *XML diff* tools have been developed [18, 28, 33]. An in-depth analysis of the proposed approaches can be found in [17]. Besides that, XML version control models such as [34] and [36] store all versions in the same XML document, and extend the XML schema of the latter with some elements used for the identification of each version. However, the drawback of these approaches is the redundancy of the content shared between different versions and the cost of the updates operations.

Probabilistic XML. Uncertainty handling in XML was originally associated to the problem of automatic Web data extraction and integration. In this context, uncertainty may have different origins: the extraction process, the unreliability of the data sources, the incompleteness of the data, etc. Several efforts have been made and some probabilistic approaches have been proposed (see [29] for a survey), especially the work of van Keulen et al. [37, 38]. Then a representation system that generalizes all the existing models was proposed in [9] and [23]; we refer to [26] for a survey of the probabilistic XML literature.

7 Conclusion

We presented in this paper an uncertain XML version control model tailored to multi-version tree-structured documents, in open collaborative editing contexts. This is one of the first actual work focusing on concrete applications of the existing literature on probabilistic XML [9, 23–26, 31, 38]. The comparison of our model to the most popular version control systems, done on real-world data, shows its efficiency. Moreover, our model offers new filtering and interaction capabilities which are crucial in open collaborative environments, where the data sources, the contributors and the shared content are inherently uncertain. The main

direction for future developments is the support of more complex version control operations, notably *merging*. Similarly to insertions and deletions, it is possible to implement merging by directly modifying the p-document, leading to an efficient management of uncertain versions. At last, the model could be extended to also support other kinds of edit operations like *moves* of intermediate nodes in XML.

8 Acknowledgements

This work was partially supported by the Île-de-France regional DROD project, and the French government under the STIC-Asia program, CCIPX project. We would like to thank the anonymous reviewers for their valuable suggestions on improving this paper.

References

[1] Cassandra Project. `http://cassandra.apache.org/`.

[2] Google Drive. `https://drive.google.com/`.

[3] Java Git. `http://www.eclipse.org/jgit/`.

[4] Linux Kernel. `https://www.kernel.org/`.

[5] [Sub]Versioning for Java. `http://svnkit.com/`.

[6] Wikipedia Platform. `http://www.wikipedia.org/`.

[7] T. Abdessalem, M. L. Ba, and P. Senellart. A probabilistic XML merging tool. In *EDBT*, 2011. Demonstration.

[8] T. Abdessalem and G. Jomier. VQL: A query language for multiversion databases. In *DBPL*, 1997.

[9] S. Abiteboul, B. Kimelfeld, Y. Sagiv, and P. Senellart. On the expressiveness of probabilistic XML models. *VLDB Journal*, 18(5), 2009.

[10] B. T. Adler and L. de Alfaro. A content-driven reputation system for the Wikipedia. In *WWW*, 2007.

[11] A. Al-Khudair, W. A. Gray, and J. C. Miles. Dynamic evolution and consistency of collaborative configurations in object-oriented databases. In *Proc. TOOLS*, 2001.

[12] K. Altmanninger, M. Seidl, and M. Wimmer. A survey on model versioning approaches. *IJWIS*, 5, 2009.

[13] M. L. Ba, T. Abdessalem, and P. Senellart. Towards a version control model with uncertain data. In *PIKM*, 2011.

[14] M. L. Ba, T. Abdessalem, and P. Senellart. Uncertain version control in open collaborative editing of tree-structured documents. In *Proc. DocEng*, 2013.

[15] W. Cellary and G. Jomier. Consistency of versions in object-oriented databases. In *VLDB*, 1990.

[16] S. Chacon. Git Book. `http://book.git-scm.com/`.

[17] G. Cobéna and T. Abdessalem. A comparative study of XML change detection algorithms. In *Services and Business Computing Solutions with XML: Applications for Quality Management and Best Processes*. IGI Global, 2009.

[18] G. Cobéna, S. Abiteboul, and A. Marian. Detecting Changes in XML Documents. In *ICDE*, 2002.

[19] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato. *Version Control with Subversion*. O'Reilly Media, 2008.

[20] R. Conradi and B. Westfechtel. Towards a uniform version model for software configuration management. In *System Configuration Management*, 1997.

[21] G. de la Calzada and A. Dekhtyar. On measuring the quality of Wikipedia articles. In *WICOW*, 2010.

[22] L. Khan, L. Wang, and Y. Rao. Change detection of XML documents using signatures. In *Real World RDF and Semantic Web Applications*, 2002.

[23] E. Kharlamov, W. Nutt, and P. Senellart. Updating Probabilistic XML. In *Updates in XML*, 2010.

[24] B. Kimelfeld, Y. Kosharovsky, and Y. Sagiv. Query evaluation over probabilistic XML. *VLDB Journal*, 18(5), 2009.

[25] B. Kimelfeld and Y. Sagiv. Modeling and querying probabilistic XML data. *SIGMOD Rec.*, 37(4), 2009.

[26] B. Kimelfeld and P. Senellart. Probabilistic XML: Models and complexity. In Z. Ma and L. Yan, editors, *Advances in Probabilistic Databases for Uncertain Information Management*. Springer-Verlag, 2013.

[27] A. Koc and A. U. Tansel. A survey of version control systems. In *ICEME*, 2011.

[28] T. Lindholm, J. Kangasharju, and S. Tarkoma. Fast and simple XML tree differencing by sequence alignment. In *DocEng*, 2006.

[29] M. Magnani and D. Montesi. A survey on uncertainty management in data integration. *J. Data and Information Quality*, 2, 2010.

[30] S. Maniu, B. Cautis, and T. Abdessalem. Building a signed network from interactions in Wikipedia. In *DBSocial*, 2011.

[31] A. Nierman and H. V. Jagadish. ProTDB: probabilistic data in XML. In *VLDB*, 2002.

[32] S. Rönnau and U. Borghoff. Versioning XML-based office documents. *Multimedia Tools and Applications*, 43, 2009.

[33] S. Rönnau and U. Borghoff. XCC: change control of XML documents. *CSRD*, 2010.

[34] L. I. Rusu, W. Rahayu, and D. Taniar. Maintaining versions of dynamic XML documents. In *WISE*, 2005.

[35] M. Sabel. Structuring wiki revision history. In *WikiSym*, 2007.

[36] C. Thao and E. V. Munson. Version-aware XML documents. In *DocEng*, 2011.

[37] M. van Keulen and A. de Keijzer. Qualitative effects of knowledge rules and user feedback in probabilistic data integration. *VLDB Journal*, 18, 2009.

[38] M. Van Keulen, A. de Keijzer, and W. Alink. A Probabilistic XML Approach to Data Integration. In *ICDE*, 2005.

[39] J. Voss. Measuring Wikipedia. In *ISSI*, 2005.

[40] Y. Wang, D. J. DeWitt, and J.-Y. Cai. X-Diff: An Effective Change Detection Algorithm for XML Documents. In *ICDE*, 2003.