Towards a Version Control Model with Uncertain Data

M. Lamine Ba
Université Cheikh Anta Diop
DMI; FST
Dakar, Senegal
mouhamadou84.ba@ucad.edu.sn

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

Pierre Senellart
Institut Télécom; Télécom ParisTech; CNRS LTCI
Paris, France
pierre.senellart@telecom-paristech.fr

ABSTRACT

Content-based online collaborative platforms and office applications are widely used for collaborating and exchanging data, in particular in the form of XML-based electronic documents. Usually, a version control system is built-in in these applications to support collaboration and to properly manage document evolution. However, most version control models require error-prone and time-consuming manual management of conflicts and uncertainties, thus heavily affecting collaborative work. Since in collaborative contexts conflicts are of a more semantic nature, i.e., due to uncertainty on what is really true, we focus on a version control model that deals with uncertain data. We introduce an XML version control model to assess uncertainty in data and to automatically resolve conflicts. Office documents and those of content-based online collaborative platforms are described in XML. We give features and requirements of our targeted model by identifying important characteristics of versioning systems, in particular XML version control systems. As first results, we propose a version-space formalism for collaborative data and we put forward an uncertainty management approach by translating the problem in the framework of XML data integration.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design—*Data models*; I.7.1 [**Document and Text Processing**]: Document and Text Editing—*Version control*

General Terms

Theory, Design

Keywords

XML, collaborative work, uncertain data, version control

PIKM'11, October 28, 2011, Glasgow, Scotland, UK.

1. INTRODUCTION

Collaborative editing and data sharing is widespread in content-based applications. Office applications and collaborative content-based Web environments are prime examples of this. Electronic documents such as OpenDocument and OpenXML have emerged as effective means for exchange of documentation and cooperation in the creation of content, both in personal and business settings [3]. For example, employees of world-wide companies with shared projects can simultaneously contribute to the redaction of a document using an office application. To manage document evolution, such a collaboration requires an efficient version control model able to resolve conflicts in an automatic manner and to potentially assess uncertainties in the produced content, two features that are especially relevant in large-scale open collaborative systems such as Wikipedia and similar content-based Web platforms. Lacking that, participants need to resort to involved and time-consuming manual management of conflicts and uncertainty.

Version control is one of the major factors that promotes collaborative editing and data sharing at large scale. In the literature, version control has been extensively studied in the two contexts of object-based databases [4, 5, 7, 8] and file-based systems [6, 17, 22, 23]. Abstracting out the differences between both settings, the common problem is to manage changes in documents shared by large teams (e.g., software deliverables, text documents, etc.). Thus, several techniques covering multiple document formats (in particular text, binary, and XML formats) have been proposed [6, 13, 16–18, 23]. Nevertheless, although effective version control tools [6, 17] are widely used, the majority of them cannot automatically resolve conflicts and do not address uncertainty. This heavily affects collaborative editing works because it is mostly limited to dealing with conflicting and uncertain data.

Some systems such as the Wikipedia platform are based on the concept of free contributions [21]. Anyone can participate in the writing of any article, no matter one's expertise on the topic. These systems include version control capabilities to manage document versions during the editing process. The conflicts, that correspond to simultaneous contributions, are frequent especially in the case of documents related to hot subjects. Ideally, the version control technique used would automatically resolve conflicts, making error-prone and time-consuming manual intervention unnecessary. This automation can be an alternative to banning the editing of a given page, as often happens in Wikipedia when some controversial topic leads to *edit wars*. Moreover,

conflicts can be of a more semantic nature, i.e., due to uncertainty on what is really true.

Because contributions come from different sources, their reliability must be managed. In online content-based systems like Wikipedia, knowledge about a given topic differs from one author to another. Moreover, vandalism acts and spams result in irrelevant content, which highly affects the integrity (or quality) of documents. Thus, the content of versions has variable quality and its reliability heavily depends on the reputation of its contributors. Furthermore, in some cases, a person might prefer to visualize the contributions performed by authors whom she trusts. She may also want to only see contributions that respect some given reliability constraints (e.g., all contributions having a reliability value greater than a given threshold). The overall value of reliability of a specific part of a document can be derived from the uncertainty of versions having affected it and therefore be computed from reputation and trust values of the contributors. These values of trust can in some cases be automatically determined, for example by analyzing past behavior of the contributors, and especially recording interaction between contributors within an online community such as proposed by authors in [15] in the case of Wikipedia.

XML tends to be a standard way for representing and exchanging data in many applications. Many electronic formats for information sharing are based on XML, such as OpenXML, OpenDocument, RSS, DocBook, etc. Moreover, Web data is inherently semi-structured and its presentation format (HTML, XHTML) is akin to XML. For these reasons, we focus on an XML data model for documents manipulated in a collaborative fashion. The modeling of uncertainty in XML is supported by the literature [2, 11], which will allow us to introduces some facilities for automated resolution of conflicts. In addition, XML change control techniques [13, 16, 24] enable detecting structural modifications between two XML document versions in an efficient manner.

Studies on XML version control have concentrated many efforts [10, 23, 25]. Whereas earlier proposals have only focused on XML change detection [10], recently some works go towards full XML version control systems [23]. The XML differencing algorithm constitutes the heart of the version control system. Existing *diff* tools such as [13, 16, 24] allow efficiently managing versions of an XML document depending of their application contexts. In particular, [24], intended for versioning office documents [23, 25], supports basic operations such as detecting changes, computing differences between versions, merging non conflictual versions, and so on. Furthermore, this system is not built on the assumption that versions are linearly generated, which makes it convenient in many real cases (e.g., when documents are concurrently edited). We will further discuss these XML diff algorithms according to the requirements of version control.

In this paper we present initial directions towards a version control model that deals with uncertain data. The automatic resolution of conflicts is tied to uncertainty management in version control model. Indeed, conflicts are simply representations of several possibilities, each of them having a certain level of reliability. Our goal is to build a model based on a *directed acyclic graph* (DAG) of *random variables* (also called *probabilistic events*) and a *probabilistic XML document*. The DAG represents the derivation graph of versions where nodes are probabilistic events. Probabilistic events model uncertain data and each generated version is associated with a unique probabilistic event. A probabilistic XML document is defined as an ordered tree whose edges are annotated with probabilistic information (formula using probabilistic events). It is the probabilistic representation of the merging of all XML document versions. In a content-based collaborative system, a given version of an XML document contains incomplete information, which can be enriched by subsequent updates that generate new versions. Thus, we treat the problem of uncertainty in XML version control using the concepts of uncertainty management defined in probabilistic XML data integration systems [29]. Probability values may be generated using statistics, similarity functions, or trust measures. Note that, in the absence of reliable sources of probability, the model can also be used as an incomplete data model representing all possibilities.

Analogically, the updates of the uncertain versions of a XML document in a collaborative environment are considered to be similar to the merge of different data sources within a probabilistic XML data integration system. In a previous work [1], we have demonstrated a probabilistic XML merging algorithm that integrate all successive revisions of a given Wikipedia article. Each part of the result of the merge is attached corresponding reliability value. We plan to extend the algorithm from [1] in the definition of our targeted version control model with uncertain data by assuming that versions are not necessary sequentially generated.

We describe the first steps towards a XML version control model that deals with uncertain data in Section 5, by giving its main requirements and our first results. In Section 4, we review state-of-the-art version control, with a special emphasis on version control models of XML documents. In Section 3, we give an overview of managing uncertain data with XML. We first present a motivating example based on the probabilistic approach of version control for collaborative data.

2. MOTIVATING EXAMPLE

Figure 1 displays an application scenario of four versions of a document resulting from a collaborative process in a content-based open environment. The version v_1 (respectively, v_2) is derived from the initial version v_0 by removing the paragraph p_1 (respectively, p_2). The last version v_3 is generated from v_2 by adding a new paragraph p_3 with the text node $text_3$. This simple example obviously shows that the contributors do not agree on the exact content of the document, so an assessment of the reliability of each contribution can be useful in the versioning process.

We now sketch how this can be represented in our probabilistic approach to version control. Figure 2 shows the derivation graph of the versions and the output of the merge process of all versions using the algorithm of in [1]. The Boolean random variables e_0, e_1, e_2, e_3, which are used to assess uncertainties and to keep information about data provenance, are associated respectively to versions v_0, v_1, v_2, and v_3. Their probability value can be computed using, e.g., reputation scores of contributors. The document resulting from the merge contains all contributions, each of them with their corresponding value of reliability. It represents the state of the edited document in the versioned system after a certain number of contributions.

We explain in Section 5.2 the entire process that produces

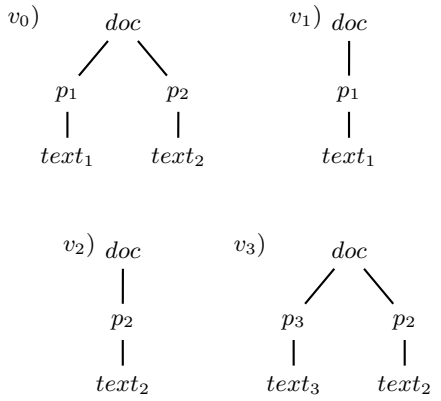

Figure 1: Uncertain versions of a XML document

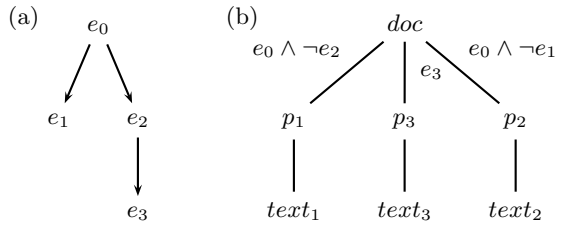

Figure 2: Representation of versions

the probabilistic document and also the practical use of the graph structure of version derivation during the merge.

3. UNCERTAINTY AND XML

Managing uncertainty in XML is mostly motivated by data integration systems. These aim at allowing users a uniform access to a set of data sources through a unified view. The merge of data (e.g., Web data or scientific data) from different sources often comes down to resolving semantic problems or conflicting data resulting from the fact that different or contradictory versions of the same data simultaneously coexist. In some contexts, these data are considered to be inherently uncertain due to the imprecise data extraction tasks, the unreliability of sources or the incompleteness of data. Consequently, there is no longer only one truth, but the result of the merge process describes several possible views on the data, each of them with a certain level of probability. Several efforts for data integration with probabilistic means have been realized, especially the works of van Keulen et al. [9, 28, 29]. The probabilistic approach allows to achieve an automated data integration scenario with the ability to assess the quality of the manipulated data in terms of trust value, reliability, etc. It might be particularly suitable for systems, e.g., collaborative environments, where the relevance of data sources about a given subject is very important.

Several probabilistic XML models [2] describing uncertain data with different semantics have been proposed. Figure 3 is a simplified example of the outcome of the merge of two documents with the model of *possible worlds* defined in [29]. Documents may be shared over a distributed system and independently updated. During a synchronization phase between two peers, the system merges documents by enumerating all possibilities, each with a probability value representing the likelihood with which the documents refer to the same real world objects or not. The output in Figure 3 states that documents are semantically related with a probability equal to 0.8. The model based on *formulas of probabilistic events* [2] (illustrated in Figure 2) is more general than that of possible worlds and enables both assessing uncertainty in data and keep information about data provenance (data lineage). This model is suitable to version control with uncertain data where one of basic requirements may be the ability to track provenance of data and to assess uncertainty. Finally, there are many probabilistic integration systems (see [14] for a survey), and we focus only on those [1, 9] directly related to the merge of versions of an XML document manipulated in a collaborative manner.

Provenance and Uncertainty in XML. In the modeling of uncertain XML data, probabilistic events, also seen as random Booleans variables, are manipulated to describe both data lineage (the source of data) and uncertain data. In this context, uncertainty in data is assumed to be closely related to data provenance. Numerous semantics can be associated to the probabilistic events, for example they can represent the contributors within a collaborative editing system. In addition, complex dependencies between data (e.g., correlation between data items) are kept in the data model.

The document resulting from the integration of several versions of the same document contains formula annotations on the edges connecting nodes. Each logical formula f is an arbitrary propositional formulas built on probabilistic events $e_0, e_1, \ldots, e_j$. A random process over this document uses a valuation ξ that sets some variables $e_i, \ldots, e_k$ to *true* and the others to *false*, generates throughout three steps a *deterministic document* (see [2, 11] for more details).

Uncertain Data Integration Tools. The uncertain data integration systems in [1, 9] build a probabilistic XML document by merging several versions of the same document. The merging algorithm is divided in two steps: first a *matching of versions* is done and second the *matches are merged*. The matching phase corresponds to finding the changes introduced between two successive versions in terms of added, deleted, and shared nodes. This is similar to the generation of a set of edit operations by applying a differencing algorithm in version control field. The merge of matches updates the previous probabilistic (or incomplete) document by adding new contents and by recomputing the probability expressions (qualitative or quantitative expressions).

The data integration system in [9] developed by de Keijzer and van Keulen deals with local dependencies such as mutual exclusivity between two choices as described in [29]. It uses numerical values as probability values that are associated with particular nodes in the result of the merge whose quality might be improved, leveraging of knowledge rules and user feedback [28]. The tool of [1], based on probabilistic events, is designed for properly managing the reliability of documents in the social online platform Wikipedia. It enables also to accomplish some operations, proper to version control systems (e.g., select a specific revision), on

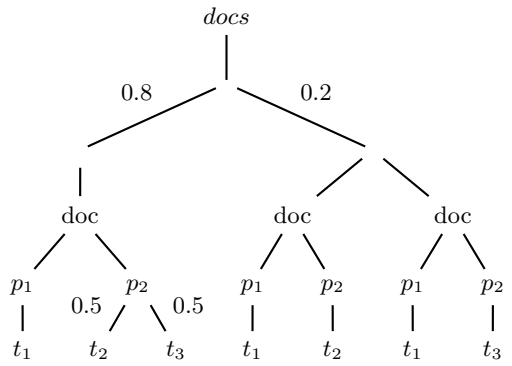

Figure 3: Model of possible worlds

Wikipedia documents. We plan to use in our targeted model this probabilistic approach, which seems to be more flexible than the previous one in terms of uncertainty management in XML for version control.

4. SURVEY ON VERSION CONTROL

The field of version control has been extensively investigated both in research and industrial communities [4–8, 12, 17, 22, 23]. In general, first systems [4, 5, 7, 8, 12, 22] are based on object-oriented models, whereas recent research or industrial tools [6,17,23] are focused on file-oriented models. Abstracting out the differences between both settings, the common problem is to manage changes on documents, e.g., source codes, text documents, etc., shared by large teams.

Object-oriented models are used in software engineering environments (i.e., CAD systems[1] and CASE[2]). In these environments, features and behaviors of objects are regarded to be the same as those defined in classical object-oriented approaches in databases or in programming languages. Objects are described as classes with attributes and methods. In the same way, concepts like inheritance and generalization are used to represent the structural constraints between objects. Aspects about branching are also taken into account enabling modifications over objects happening in parallel along multiple branches, which is very important in particular in distributed and cooperative development environments. Collaboration and distribution can increase the number of different designs of the same object within the system. As storage support, these object-oriented version control systems are built on top of existing object-oriented database systems. Therefore, versions of a same object coexist simultaneously in the same place. As a consequence, most research efforts, e.g., those in [4, 5, 22], have been concentrated on consistency management of versions and configurations, i.e., how to identify the versions that compose a consistent configuration or, more generally, a consistent state of the modeled world. For example using *global version stamps* [5] or *descriptors of consistency* [4]. In case of large software engineering environments, version control systems based on file-oriented models (also called *source code management tools*) manage source codes, documentation, and configuration files.

Many systems intended for version control of documents, covering multiple formats (e.g., text, binary and XML formats), have been proposed both in research [10,13,16,18,23] and in industry [6, 17]. A large majority among them, seen as *general-purpose version control systems*, by default only manipulate text or binary formats. However, in general they propose options for integrating unsupported formats like XML format. *Subversion, ClearCase, Git, BitKeeper* and *Bazaar* forms a non exhaustive list of them and a more detailed survey can be found in [23]. These tools are either built on line-based differencing algorithms like GNU diff [18], or use simply cryptographic hashing techniques [6]. Both methods are not convenient for XML documents, because the semantics of changes is not described meaningfully leading to a possible hide of the collaborative editing aspects. So, XML differencing algorithms [10, 13, 16, 24] are approached to detect structural changes between two document versions. Furthermore, version control tools support the concurrent access to the same file, which enables collaboration and data sharing over distributed systems. They implement mechanisms of locking (generally, unreserved locking) for concurrency management and include some merging features. Under versioning, a complete history of versions of files is maintained and each versions is stored in a compressed form or using a delta. Typically, the version space is represented sequentially or using a graph structure. Directories handling is also available within some system, e.g. in [6]. Additionally, conflicts appear sometimes in merging phase and their detection depend on the application. Unfortunately, many file-oriented version control systems require human intervention to resolve these conflicts before a final validation of the merge. Resolution consists in keeping only one possibility even though it is not the most relevant. This can not be efficient in content-based editing environments like Wikipedia. The overall features of a file-based version control systems can be implemented in general as a virtual filesystem over the filesystem of an existing operating system as in [6, 17].

XML version control [10, 13, 16, 23, 24] has emerged with the increase use of XML data. Some focus on XML change control [10, 13, 16] whereas the system proposed by Rönnau and Borghoff [23, 24] is intended for versioning office documents (e.g., OpenDocument and OpenXML documents).

Version Control in XML-Based Systems

XML version control aims at detecting structural changes between two XML document versions. Modifications affect nodes or subtrees in XML document. Thus, change detection techniques for flat documents that are often line-based are no longer appropriate to the context of XML.

Existing XML version control systems are based on XML differencing algorithms, which are designed mostly by resolving the tree-to-tree editing problem. Generally, these algorithms are classified regarding to several criteria such as the complexity in time and space, quality of the delta, delta representation, tree model considered, version control facilities, etc. Detailed comparative studies according to their criteria has been proposed in [10, 19, 23–25]. As a result, tools proposed in [13, 16, 24] and based on an ordered tree model seem to be very effective for computing the so-called *delta* (difference) between two versions. An ordered tree model is inherent to XML and describes well the structure of many documents such as Web documents and some electronic documents (OpenDocument and Office OpenXML [24]) which are currently manipulated in collabo-

[1]Computer Aided Design Systems
[2]Computer Aided Software Environments

rative editing environments. The *XCC* [24], *Faxma* [13], and *XyDiff* [16] algorithms run in linear time[3] without drastically affecting the quality of the resulting output. They use a state-based approach and a two-way diffing technique that are both common to control version systems in general. A two-way approach considers only the two successive versions as inputs unlike to a three-way approach that takes in addition the common base version. Moreover in general, they only assume equality between nodes (or subtrees) in two versions according to the ordered tree model of XML, using indexing mechanisms (e.g., hashing functions) on nodes or subtrees.

XML Diff algorithms and Delta Models. The XML differencing problem corresponds to finding the minimum set of operations that transforms one tree into another, or an approximation thereof. The tree model and delta model on which the differencing technique is based are important factors. They both influence the accuracy of the differencing operation and its complexity. The tree model refers either to an ordered tree or an unordered tree. The order in XML documents within collaborative environments is semantically very important. Delta models give the supported operations, the targeted elements and the format of the obtained delta.

XCC [24], *Faxma* [13], and *XyDiff* [16] all assume an ordered tree model, which is convenient for their corresponding application contexts (office documents and XML data warehouses) or for the ordered structure of XML itself. The model of XML consist of element nodes having a list of children or attributes, and text nodes. In addition to that, nodes have unique identifiers, for example their postfix positions [16]. Moreover, delta models in these three tools consider the operations of *insertion*, *deletion*, *move*, and *update*. The first three are defined on nodes or sub-trees (eventually, sequence of trees) and the last on all types of nodes. These operations describe meaningfully the semantics of all possible changes on XML documents. This design is chosen with respect to change patterns of XML documents in each application (e.g., office documents in [24]). For example, in office documents, authors in [24] have established that often a move operation affects an entire subtree and a modifications (update) on documents concern in most case text nodes. Additionally, *XyDiff* use persistent identifiers (XID) for representing the exact location of a change within a delta whereas *Faxma* gives the absolute path as reference of node. Both addressing schemes works well when versions are sequentially generated, but fail in other cases. *XCC* change representation is intended for documents manipulated in a collaborative manner. Its addressing scheme is based on the context of the change, i.e., it represents the target of change by including its neighboring elements using a hashing function.

The delta between two versions is generated in two phases. Firstly, a correspondence between nodes belonging to two versions is done and then the delta is computed from the result of the matches. *XyDiff* and *XCC* follow the same matching algorithm by searching the LCS (longest common subsequence) of two XML trees in a bottom-up traversal of trees. Unlike the tools mentioned above, Faxma match sequences of tokens encoding the input XML documents. The problem of trees matching is transposed into the domain of sequences alignment, which allows to easily use string matching algorithms (e.g., LCS algorithm and algorithms supporting block moving). In both matching approaches, heuristic algorithms or dynamic programming techniques are used to speed up the process and reduce the overall required time without affected drastically the quality of the result. The generated delta contains a set of edit operations [16, 24] or a transformation script [13] and its format can be described in XML. A transformation script allows just to obtain a specific version knowing its predecessor.

Merging Versions and Conflicts Detection. *XyDiff* and *Faxma* are basically XML change control tools. They are both restricted to a linear document evolution model, so their corresponding delta model can only reconstruct versions in a linear manner. This method seems inefficient in collaboration systems when parallel editions are frequent. In addition, they do not support merging changes which is crucial in collaborative editing systems. In contrast, *XCC* is intended to XML document versions in collaborative contexts. The obtained delta can be applied (*patch* operation) on a version for which it was not computed for. Moreover, it supports the merge of changes with its context-oriented delta model that is based on the so-called *fingerprint* [20] concept. The *fingerprint* keeps the context of a node by taking into account its neighbors on a radius r and using a hashing technique for addressing nodes. Furthermore, *XCC* tool includes mechanisms to detect conflicts during merging phase, but their resolution is manual. Finally, the XML version control for office documents proposed in [23] by Rönnau and Borghoff is based on this diff algorithm.

5. VERSIONING WITH UNCERTAINTY

In this section we elaborate on our version control system that deals with uncertain data. We first present the main features of the intended versioning model and their underlying basic requirements. We then put forward a model and explain how it can be used for versioning.

5.1 Main Ideas and Basic Requirements

At large scale, most content-based applications rely on efficient version control mechanisms to support collaboration and data sharing between peers. Uncertainty in data is common to numerous collaborative editing environments, like Wikipedia and similar content-based Web platforms, because the contributors are unreliable, which leads to contributions inherently uncertain, sometimes resulting in conflicts. Thus, an efficient management of this uncertainty is crucial, especially in view of the tedious and error-prone manual tasks that would otherwise be required. Nevertheless, existing version control models do not address uncertain data.

For enabling uncertainty management within the versioning process, we aim to develop a version control model with uncertain data. Our main idea is to use a probabilistic approach for version control of XML data in collaborative open environments. Data manipulated in these environments are mostly described in XML and uncertainty modeling in XML data is supported in the literature as stated in Section 3. With a probabilistic approach of version control for XML documents, we assume that each version of a document

[3]Most XML diff algorithms run at least in quadratic time to produce optimal results.

comes with a *probabilistic event* (or random variable) that captures both data provenance and uncertainty in data. The state of the versioned document, edited in a collaborative manner, at a certain version v_k, is no longer represented by the content of this version but the merge of all generated versions $v_0, v_1, \ldots, v_k$ using a probabilistic merging algorithm. Each part in the result of the merge, which is seen as a *probabilistic document*, has a logical formula attached to it. This logical formula represents the level of reliability of the corresponding content. As a preliminary result, we propose further in this section a formalism for the graph structure of version derivation and the merging algorithm that generates a probabilistic XML document. Obviously, this is only one step of the entire work.

We will probably need a suitable XML change control algorithm that may depend on our application domains. Performance evaluation on real data (e.g., from Wikipedia) will be required to estimate the efficiency of the overall merging algorithm. In addition, we also imagine to transpose the traditional manipulation operations of document versions (create, delete, read, and merge of versions) to our model, specifying their semantics and their implementations. A special focus could be made on new functionalities that can enable the use of probabilities (for instance, remove the versions having a probability less than a given threshold, find the versions that have a fixed probability, find the versions that have a probability higher that a given threshold, etc.).

In order to achieve our research targets, we have identified a set of basic requirements in the side of version control and we discuss some about them in the following.

Modeling of Version Space. We consider modeling the version space with a *directed acyclic graph* (DAG), consisting of random variables as nodes and edges for revisions and variants relationships. A graph structure corresponds effectively to the evolution of documents shared by a large team as stated both in object-oriented context [4,5,7,8] and in file-oriented context [6, 23]. It describes very well the parallel edition of documents, which is an important feature in most collaborative environments. As an illustration, Sabel [27] has demonstrated its conformity with the structure describing document evolution in Wikipedia. The random variables implicitly describe document versions, thus they can be use to access to these latter. A version can have an arbitrary number of revisions and variants. It has at least one parent version. If a version is not derived from another version, it corresponds to the initial version of the document (eventually, an empty document). Versions with no outgoing revisions are current versions of their corresponding branches.

Managing Changes. As the version control model is supposed to be based on an XML data model, we need an XML differencing algorithm which seems to be more convenient for detecting structural change between two XML document versions. Its output can be used for storage, merging operations, versions reconstruction, documentation purposes, etc. In our context, the differencing algorithm can be considered as the first stage of the probabilistic merge process. Basically, it must be able to give as result the set of shared, deleted and added nodes between two versions with an acceptable running time. We do not necessarily need to detect the moved nodes or subtrees, since the semantics of move operation can be seen as a delete and then an add. However, we will study the impact of move operation on documents manipulated in collaborative environments.

As described in Section 4, the differencing algorithms proposed in [13, 16, 24] seem to be efficient on office documents and Web data. We plan to evaluate in a deeper manner these algorithms in a real-life context of data manipulated in large-scale collaborative platforms like Wikipedia. Furthermore, requirements on differencing algorithms can benefit to the fact that our model is not concerned about problems related to conflicts detection. Conflict detection leads to the use of sophisticated techniques for producing a diff as correct as possible. Some trade-off could be found. Moreover, the representation of the diff may impact on the complexity of the merge step that consists to the update of the probabilistic document. Indeed, for example new nodes can be added at the correct location in the p-document. Thus, the location of nodes in XML trees must be efficiently defined by the differencing algorithm.

Querying on Versions. Existing version control systems enable basic operations like reverting to a previous version, comparing two given versions, and selecting a specific version according to its author or date of edition. Usually, users can manipulate them by interacting with a GUI interface or via command lines. These basic operations must also be possible in our targeted version control model. As each version comes with a probabilistic event and all generated versions are merged in a p-document, then a possibility for answering queries on versions is to translate their semantics on probabilistic events and to execute the resulting query on the p-document. As a simple example, we can answer a query $\mathcal{Q}$ selecting version v_0 in our running example (see Figures in Section 2), by defining a valuation ξ that sets e_0 to *true* and e_1, e_2, e_3 to *false*. Furthermore, several semantics can be associated to probabilistic events, e.g., they can be related to contributors or dates of edition. Consequently, operations on versions involving contributors or dates of edition should be easily implemented. If the p-document is seen as an ordinary document with formulas attached to nodes as attributes, the previous queries can be answered using a subset of the XPath query language. Finally, a special emphasis should be put on operations that will manipulate directly uncertainty. For example, a person may want to visualize the state of the document if certain contributions or unreliable authors have never happened. The semantics of these kinds of query need to be formalized and implemented, because they are very important in particular when only relevance contributions have to be returned within the collaborative environment.

5.2 Versioning with Uncertain Data

We assume $v_0, v_1, \ldots, v_n$ to be the different versions (i.e., n ordinary XML documents) of a given document $\mathcal{D}$ handled in a collaborative and open manner by a large community. Our targeted framework takes as input a set of XML document versions, which is given in a linear order (e.g., as in Wikipedia revisions) or in graph like structure, and gives as output the graph derivation of versions and the probabilistic document resulting from the merge of all the initial versions.

We formalize the graph $\mathcal{G}$ of version derivation and the probabilistic merging algorithm that builds the probabilistic document (p-document) $\mathcal{P}$ representing the content of the versioned document at each version.

Graph of Version Derivations. With our version control model, we consider that each version v_i comes with a probabilistic event (or random variable) e_i. A random variable models the validity (existence) of its associated version. We decide to use it for describing a version. During the evolution of a document, except for the initial version, a given version is generated from another version (or possibly several, in the case of a merge), leading to dependency relationships between these two versions.

As a result, we represent the corresponding graph of version derivation with a DAG structure. The DAG is defined by a pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of nodes consisting of random variables $e_0, e_1, \ldots, e_n$ and $\mathcal{E}$ is the set of edges that represent the direct dependencies between these variables. In this representation, a random variable implicitly describes a specific version of the document. A Bayesian Network $\mathcal{B}$ can be defined on this DAG structure $\mathcal{G}$. Formally, we have $\mathcal{B} = (\mathcal{G}, \Theta)$ where Θ denotes a set of conditional probabilities over random variables $e_0, e_1, \ldots, e_n$, i.e., if a version v_i derived from an another version v_j, the conditional probability $\Pr(e_i \mid e_j)$ is introduced as a parameter, while $\Pr(e_j \mid e_i) = 1$. Such a Bayesian network is used to represent knowledge about an uncertain domain [26], and is therefore suitable for our context.

In real applications, this graph structure can be derived from the implemented version control system (e.g., Git) or constructed by evaluating the similarity between generated versions. For content-based Web collaborative platforms like Wikipedia, successive revisions of a given document are maintained although the evolution of a document is not necessarily linear. This means the corresponding graph structure of version derivation can be inferred by computing the similarity between revisions. For instance, Sabel [27] proposes a method to represent revision history of a Wikipedia page as a tree of versions by comparing texts of the versions.

Probabilistic XML Document. We construct a probabilistic document $\mathcal{P}$ corresponding to the merge of all versions of a given XML document. For this purpose, we extend a merging algorithm demonstrated in a previous work [1] by assuming that versions are not necessary sequentially generated. Basically, the algorithm is built on the following assumptions:

- Initially, $\mathcal{P}$ corresponds to the initial version of the document which can be a root-only document.
- A mapping function α gives the correspondence between nodes x of versions and those $\alpha(x)$ of the p-document. This mapping function is maintained by computing diffs between successive documents.
- Event formulas $f_{old}(x)$ and $f_{new}(x)$ are associated to a node $x \in \mathcal{P}$ respectively before and after the merge; $f_{new}(x)$ is expressed in terms of $f_{old}(x)$ and the random variable associated to the new version. We refer to [1] for details. In addition, conditions on nodes are not repeated on its children.

In addition to that, we define a set $\mathcal{L}$ containing couples of versions related by a derivation relationship. Formally, for all versions v_i and v_j, with $0 \leq i < j \leq n$, associated respectively with the random variables e_i and e_j; we have

$$\mathcal{L} = \{(v_i, v_j) \mid (e_i, e_j) \in \mathcal{G} \text{ for each } 0 \leq i < j \leq n\}$$

For merging the k current versions of a document, our algorithm successively performs the matching of couples of versions in $\mathcal{L}$ and merges the result in $\mathcal{P}$. The merge consists of updating $\mathcal{P}$ with nodes and events describing the matching step. The details of the entire procedure of the merge and expressions of event formulas is stated in [1].

By applying the algorithm on the running example of versions in the Figure 1, we deduce from the graph derivation of versions (see Figure 2(a)) $\mathcal{L} = \{(v_0, v_1), (v_0, v_2), (v_2, v_3)\}$. The result of merging these four versions, depicted in Figure 2(b), is obtained as follows:

- At the beginning, we assume that the p-document corresponds to v_0 with formula $f = e_0$ associated to nodes p_1 and p_2. First, versions v_0 and v_1 are matched leading to the result that the subtree rooted at node p_2 is removed. As a result, the formula of p_2 become $f_{p_2} = e_0 \wedge \neg e_1$ in the p-document for indicating that the subtree rooted at this node is not present in v_1. The formula associated to node p_1 does not change since according to $\mathcal{G}$ the existence of v_1 leads to that of v_0 and the subtree rooted at p_1 is shared by these two versions.

- In the second step, the matching of versions v_0 and v_2 gives as result that the subtree rooted at p_1 is removed whereas the subtree rooted at p_2 remains unchanged. Analogically to the previous merge, when updating the p-document, the formula associated to p_1 become $f_{p_1} = e_0 \wedge \neg e_2$ and that associated to p_1 does not change. The p-document at this state contains two subtrees: a subtree rooted at p_1 with the text node $text_1$ and the subtree rooted at p_2 with the text node $text_2$.

- Finally, versions v_2 and v_3 are matched. The subtree rooted at node p_3 is a newly added one in the version v_3. The two versions share the subtree rooted at p_2. In the merging phase, the subtree rooted at p_3 is added to the root of the p-document and the formula $f_{p_3} = e_3$ is associated to the node p_3. Similarly, according to the derivation dependency in $\mathcal{G}$ between versions v_2 and v_3, the formula associated to the node p_2 remains unchanged.

The overall output of the merge process contains three subtrees and formulas $f_{p_1} = e_0 \wedge \neg e_2$, $f_{p_2} = e_0 \wedge \neg e_1$, $f_{p_3} = e_3$ associated respectively to each subtree. The probability values of e_0, e_1, e_2 and e_3 are inferred, e.g., from a trust system stating that authors of versions v_0, v_3 have reputation values equal to 0.5 and authors of versions v_1, v_2 respectively have a reputation of $0.4, 0.7$.

Updating the p-document constitutes a crucial step during the integration of a new version. This phase does not have a heavy impact on the running time of the entire process, since all kind of update operations over p-documents can be made in polynomial-time data complexity [11] (the trade-off being that the probability of a query becomes costly to compute). Although a practical evaluation of the execution time of the merging has not been done yet, empirical analysis of the application implemented for Wikipedia articles in [1] give promising results.

6. CONCLUSION AND FURTHER WORKS

We have presented initial directions towards an XML version control model with uncertain data for collaborative open environments like Wikipedia. So far, we have given a formalism for a version space of collaborative data, a sketch of the probabilistic approach, and some promising features of our targeted model. In order to reach our research goals, we need to further implement an appropriate XML differencing algorithm and formally specify the set of possible operations on the version model with uncertain data. An extensive experimentation on real data (e.g., from Wikipedia) could also lead to interesting insight. The probabilistic merging algorithm, which is the core of the studied model, needs to incorporate a change control algorithm in the first step of its process. The three diff algorithms proposed in [13,16,24] run in linear time on large documents such as office documents or Web documents. In particular, [24] is intended for collaborative contexts, and may be a promising basis. The only problem is that these algorithms detect *move* operations that are currently not supported in our model. We aim at evaluating each of these algorithms in our setting to determine the most suitable one. Some modifications may be needed to obtain a convenient XML differencing algorithm. In addition, semantics of basic operations on traditional version control models must be possible within our proposed model. A possible approach is to implement these operations using probabilistic events. Moreover, we also want to support queries on information about uncertainty. We must also evaluate the overall complexity of our probabilistic merging algorithm. It also seems interesting to compare the efficiency between representing all versions in a p-document and maintain successive deltas stored with an integral content of a specific version, which is what is commonly done in existing version control models. Finally, a possible research direction is to study the reliability of detecting controversial topics in Wikipedia by using the proposed compact representation of versions of a same document.

7. REFERENCES

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

[2] S. Abiteboul, B. Kimelfeld, Y. Sagiv, and P. Senellart. On the expressiveness of probabilistic XML models. *VLDB Journal*, 18:1041–1064, 2009.

[3] A. Adler, J. C. Nash, and S. Noël. Evaluating and implementing a collaborative office document system. *J. Interacting with Computers*, 18:665–682, 2006.

[4] 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.

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

[6] S. Chacon. The Git community book. `http://book.git-scm.com/index.html`.

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

[8] G. D. David. Palimpsest: A data model for revision control. In *Proc. Collaborative Hypermedia Systems*, 1994.

[9] A. de Keijzer and M. van Keulen. IMPrECISE: Good-is-good-enough data integration. In *Proc. ICDE*, 2008.

[10] C. G, A. T, and H. Y. A comparative study for XML change detection. Technical report, INRIA Rocquencourt, 2002.

[11] E. Kharlamov, W. Nutt, and P. Senellart. Updating probabilistic XML. In *Proc. Updates in XML*, 2010.

[12] Kögel and Maximilian. Towards software configuration management for unified models. In *Proc. Comparison and versioning of software models*, 2008.

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

[14] M. Magnani and D. Montesi. A survey on uncertainty management in data integration. *J. Data and Information Quality*, 2:1–33, 2010.

[15] S. Maniu, B. Cautis, and T. Abdessalem. Building a signed network from interactions in Wikipedia. In *Databases and Social Networks*, 2011.

[16] A. Marian. Detecting changes in XML documents. In *Proc. ICDE*, 2002.

[17] C. Michael Pilato, B. Collins-Sussman, and W. F. Brian. *Version Control with Subversion*. O'Reilly Media, 2008.

[18] E. W. Myers. An O(ND) difference algorithm and its variations. *Algorithmica*, 1:251–266, 1986.

[19] L. Peters. Change detection in XML trees: a survey. In *TSIT Conference*, 2005.

[20] M. O. Rabin. Fingerprinting by random polynomials. Technical report, Center for Research in Computing Technology, Harvard University., 1981.

[21] S. Rafaeli and Y. Ariel. *Online Motivational Factors : Incentives for Participation and Contribution in Wikipedia*, pages 243–267. Cambridge University Press, 2008.

[22] C. Reichenberger and S. Kratky. Object-oriented version control: Use inheritance instead of branches, 2009.

[23] S. Rönnau and U. Borghoff. Versioning XML-based office documents. *Multimedia Tools and Applications*, 43:253–274, 2009.

[24] S. Rönnau and U. Borghoff. XCC: change control of XML documents. *Computer Science - Research and Development*, pages 1–17, 2010.

[25] S. Rönnau, J. Scheffczyk, and U. M. Borghoff. Towards XML version control of office documents. In *Proc. DocEng*, 2005.

[26] F. Ruggeri, F. Faltin, and R. Kenett. Bayesian networks. *Encyclopedia of Statistics in Quality and Reliability, Wiley and Sons*, 2007.

[27] M. Sabel. Structuring Wiki revision history. In *Proc. WikiSym*, 2007.

[28] M. van Keulen and A. de Keijzer. Qualitative effects of knowledge rules and user feedback in probabilistic data integration. *VLDB Journal*, 18:1191–1217, 2009.

[29] M. van Keulen, A. de Keijzer, and W. Alink. A probabilistic XML approach to data integration. In *Proc. ICDE*, 2005.