TNTBase – a Versioned XML Database

Vyacheslav Zholudev and Christoph Lange

Computer Science, Jacobs University Bremen,
{v.zholudev,ch.lange}@jacobs-university.de

Abstract. A huge amount of documents is created and changed in our everyday life, so that Version Control Systems like Git or SVN are tightly integrated with documents workflows. On the other hand, XML has come of age as a basis for document formats, and even though XML as a text-based format is suitable for version control in principle, the fact that version control systems work on files makes the integration of fragment access techniques like XPath or XQuery difficult.

In this paper, we present the state of the art of TNTBase, a versioned XML database based on Berkeley DB XML and Subversion. Thus, the system integrates versioning and fragment access needed for fine-granular document content management. It is intended as a basis for collaboratively editing and sharing XML documents, and also provides an infrastructure for specialization towards specific applications and their document formats, such as validation, format-specific "XML-database views" and human-oriented presentation.

1 Introduction

A tremendous number of documents is being created and managed nowadays all over the world. We observe the development of a deep web (web content stored in databases), from which the surface web (what we see in our browsers) is generated. With the merging of XML fragment access techniques (most notably URIs [6] and XPath [3]) and database techniques and the ongoing development of XML-based document formats, we are seeing the beginnings of a deep web of XML documents, where surface documents are assembled, aggregated and mashed up from background information in XML databases by techniques like XQuery [7], and document (fragment) collections are managed by XQuery Update [8]. At the same time, the web is constantly changing. Therefore, we need an infrastructure for managing changes in the XML-based deep web. Unfortunately, Version Control systems like Git or Subversion [26] which have heavily influenced collaboration workflows in software engineering are deeply text-based (w. r. t. diff/patch/merge) and do not integrate well with XML databases and XQuery. On the other hand, some XML databases address temporal aspects, but their versioning possibilities cannot be compared to traditional Version Control systems. Moreover, the latter do not really provide format-specific features based on format schemas or semantics.

During the last years there have been a lot of attempts towards versioning XML. In this paper, we present the state of the art defined by our own approach: the TNTBase system, a versioned XML database that is built on top of a tight integration of Berkeley DB XML [5] into the Subversion Server. The system is intended as an enabling technology that provides a basis for future XML-based document management systems that support collaborative editing and sharing by integrating versioning and fragment access, both of which are required for fine-granular document content management. We also present how a general TNTBase installation can be extended to meet additional requirements of specific XML-based languages and their applications.

The TNTBase system was born in the context of OMDoc (Open Mathematical Documents [15]), an XML-based representation of the structure of mathematical knowledge and communication. Correspondingly, the development requirements for the TNTBase come out of OMDoc-based applications and their storage needs. But TNTBase, as described here, is independent from all of these and is not biased towards mathematical content. However, in Sect. 5, we briefly describe how TNTBase is being used for certain mathematical applications. TNTBase is available under an open source license from `http://tntbase.org`.

Sections 2 and 3 describe two major components of TNTBase: the core and the application-specific layer. In Sect. 4, we separately discuss the most powerful part of the application layer – Virtual Documents. Sect. 5 shows the features described so far in use by summarizing the real-life scenarios in which TNTBase is currently being used. Sect. 6 concludes the paper. Throughout the paper, a collection of Computer Science lecture notes in OMDoc will serve as a running example. We will show how different features of TNTBase help us to manage this material and extract necessary information from it.

2 TNTBase Core

2.1 Related Projects

The core of the TNTBase system is based on the tight integration of two widely used open source projects: Subversion and Berkeley DB XML. We provide a short description of those aspects of the systems that are relevant to TNTBase and discuss what is missing for versioned XML storage.

Subversion (SVN) is a centralized version control system [26]. For our work the server part is relevant, which maintains the version history of documents and directories in a repository. Unfortunately, like any other version control system, SVN does not focus on the structure of documents inside a repository: its storage algorithms only distinguish between text and binary files. In particular, SVN does not support XML database features, most importantly querying via XQuery, indexing, and updating via XQuery Update. Another limitation of SVN is the fact that the smallest versioned entity in its repository is a file. But for some applications it is desirable to abstract away from the notion of files, and work with XML objects like sections in technical manuals in the DocBook format, or

theorems or proofs in mathematical documents. The Virtual Documents, which we will introduce in Sect. 4, abstract even further from the notion of a file.

Berkeley DB XML (DB XML) is an open source, embedded native XML database built on top of Berkeley DB [4]. The latter is also one possible SVN storage backend (in Sect. 2 we will see the consequences of that fact). DB XML inherits the advantages and features (e. g. portability, ACID[1] transactions, replication, scalability, easy deployment, etc.) from Berkeley DB and extends it with the typical features of native XML databases: XQuery-based access to documents (including XQuery Update), content-based indexing for improving the performance of queries, XQuery extension functions written in other programming languages, XQuery debugging, etc. Unfortunately, DB XML does not support versioning, which is becoming more and more crucial when managing collections of XML documents. Some XML databases (e. g. [11, 19, 24]) support primitive versioning, but they have a number of limitations in comparison to ordinary version control systems like SVN or Git (e. g. no branching, no merging, no computation of minimal differences between revisions). Also, DB XML does not have a notion of a file system, which becomes a serious shortcoming when managing huge collections of documents.

2.2 xSVN – an XML-enabled Repository

xSVN, a modified SVN server integrated with DB XML, is one of the two foundations of TNTBase (see [30] for details). Its design is motivated by the observation that both the SVN server and the DB XML library are based on Berkeley DB (BDB). The SVN server uses it to store repository information[2], and DB XML uses it for storing the raw bytes of the XML documents and for supporting consistency, recoverability and transactions. Moreover, transactions can be shared between BDB and DB XML, making the integration more natural.

We have extended the SVN backend storage module so that it stores the latest (HEAD) revisions of XML documents[3] in DB XML (that happens to be physically another BDB table that stores XML nodes in a special format). Non-XML data like PDF, images or LaTeX source files, differences between revisions, directory entry lists and other repository information are retained in BDB. From an end-user perspective there is no difference to SVN: all the SVN commands are still available and have the same behavior. Thus, for non-XML files the workflow of xSVN is absolutely the same as in SVN. Thereby we are also able to store plain text or binary data in xSVN that can supplement the collection of XML files (e. g. licensing information or PDF documents generated from XML). Moreover, we can commit XML and non-XML files in the same transaction.

[1] atomicity, consistency, isolation, durability

[2] Alternatively, SVN can use a filesystem storage backend, but we use BDB.

[3] xSVN treats a file as XML if its extension is *.xml* or its *svn:mime-type* property is either set to *text/xml*. This behaviour can be tweaked in the server-side configuration files, or by instructing the SVN client to automatically set the right *svn:mime-type* property.

Keeping XML documents in DB XML *allows* us to access those files not only via any SVN client, but also through the DB XML API, with all benefits mentioned in Sect. 2.1: efficient querying and update, indexing, and transactions.

xSVN's deltification algorithms, which compute the difference between a new revision and the previous one, are retained unchanged from normal SVN, as the are very complex. The benefits of an XML-aware differencing are, however, evident, both in terms of smaller and less invasive deltas, and more informative conflict resolution strategies. Therefore, we are currently working on bringing XML-aware diff into the server side as an additional feature (cf. Page 5).

In conclusion, xSVN offers versioned XML storage. However, without additional modules it is not yet really useful as the only difference from SVN is that it refuses to commit ill-formed XML documents. In the next section we describe how we take advantage of DB XML integrated into xSVN.

2.3 XML-related features

The second foundation of the TNTBase core is a Java library called *DB XML Accessor*, which communicates with xSVN's DB XML bypassing all other repository information stored in BDB. A web application sitting on top of that library exposes its services to end-users via HTTP. The DB XML Accessor extends DB XML's XQuery syntax with the following TNTBase-specific functions: **Querying path-based collections of documents:** As mentioned in Sect. 2.1, DB XML does not have a notion of a file system: every document has just a unique name. But in order to make access methods compliant with xSVN's repository structure, we introduced several XQuery functions that allow to address documents based on their file system path. The trick here is that xSVN augments XML documents with metadata fields that are supported by DB XML – including path and revision information –, which allows us to traverse the filesystem structure without looking into other parts of xSVN backend storage. A user can address a single document by the XQuery function `tnt:doc($path as xs:string) as document-node()`, where `$path` is a path of a file in a repository. To obtain a collection of documents, the `tnt:collection ($pattern as xs:string) as document-node()*` function can be used, where `$pattern` is the template for a filesystem path that may contain wildcards (*, ?) and //, representing an arbitrary chain of child directories. For example, the query `tnt:collection ('/lectures//math/*set*.omdoc')` will return all lecture documents regarding set theory in all math sub-folders of lectures. **Querying previous revisions:** By default xSVN only stores the HEAD revisions of XML documents, but it is also possible to access previous revisions in queries. There are two options:

1. DB XML Accessor can *cache* a particular subset of any revision in DB XML on request. That subset can then be retrieved by the `tnt:collection ($pattern, $revision-number)` XQuery function. Note that only those documents will be returned that have been *cached* and comply with the given filesystem path pattern. This requires additional coordination from the user side but was deemed more reasonable than keeping all revisions in DB XML by default.

2. Using the tnt:doc($path, $revision) function. Unless the document under $path has been *cached* or is a HEAD revision, it is retrieved by the *SVNKit Adapter* module of DB XML Accessor. SVNKit Adapter utilizes the SVNKit Java library [27] that implements an SVN client. This method works without prior caching, but queries against that document will typically be typically slower, as the current method does not take advantage of DB XML indexes.

Modifying via XQuery Update: Apart from modifying documents e. g. in an XML editor and committing them using an SVN client, TNTBase takes advantage of DB XML's XQuery Update facilities, and, in contrast to pure DB XML, modified documents are versioned, i. e., a new revision is committed to xSVN whenever on every update operation. Note that DB XML's XQuery Update support could not be used out of the box, as it modifies documents without preserving their history. Such modifications would not only lose information but also confuse xSVN, because it guarantees that files are retrieved absolutely the same as they were committed. TNTBase introduces counterparts of the XQuery Update functions that make changes consistent with history preservation [28]. For example, in order to replace the node /omdoc/theory of the document /lectures/math/sets.omdoc, one has to use tnt:replace-node(tnt:doc('/lectures/math/sets.omdoc') /omdoc/theory[1], $new-theory-element). That substitutes the theory element, commits a new revision and returns the commit information as a string.

XML diff: We are working on bringing an XML-aware diff as another feature to the TNTBase core. So far, we have implemented a simple XML diff function in XQuery, which compares two XML elements and returns a simple XML output describing the differences. It ignores whitespaces differences, comments and attribute order. It can be used for comparing different revisions of documents and finds an internal application in editing of Virtual Documents (Sect. 4).

2.4 Interfaces

TNTBase provides two different communication interfaces: (i) The xSVN interface behaves like the normal SVN interface – the *mod_dav_svn* Apache module serves requests from remote SVN clients – with one exception: If one of the committed XML files is ill-formed, then xSVN will abort the whole transaction. (ii) The RESTful [13] interface exposes the DB XML Accessor features described in the previous subsection for XML operations [32].

3 TNTBase Application Layer

Our case studies (cf. Sect. 5) showed that many tasks specific to particular XML formats can be done by TNTBase. That was a reason to derive a separate layer on top of the TNTBase core and augment this layer with **F**ormat-specific functionalities (see Fig. 1). TNTBase provides facilities to integrate format-specific validation (e. g. against RelaxNG schemas [9]) and presentation (e. g. via XSL Transformations). But often a format requires more specific functionality, for instance, extracting knowledge to different representations (e. g. RDF) upon commit and caching it, or complex rendering tasks (such as rendering MathML

formulæ when presenting our lecture notes as XHTML). Such functionality can be supplied as additional modules and injected into the application layer via the TNTBase plugin API (see [28] for details).

Often document workflows involve both core components of TNTBase, xSVN and the web application on top of the DB XML Accessor library (cf. Sect. 2), and require interactions between them. For instance, when one commits a new version of some XML documents and wants their human-oriented XHTML presentation to be cached, this involves committing files in xSVN, figuring out what files the presentation should be generated for, sending the corresponding RESTful request to the F-specific layer of TNTBase, generating the presentation and finally saving it in DB XML. We omit the technical details of these interactions, as they mainly concern the (painful) integration of C++ (xSVN) and Java (DB XML).

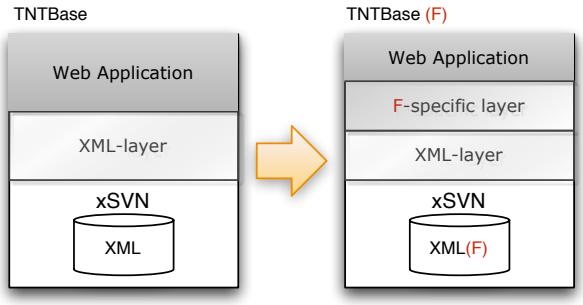

Fig. 1. TNTBase for specific applications

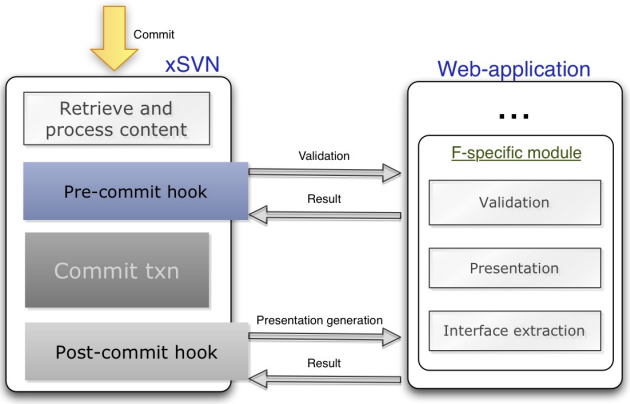

Fig. 2. Interactions between xSVN and Web-application

For our implementation, we utilize the standard SVN pre-commit and post-commit hook mechanisms for figuring out what subset of committed files is subject to further processing, such as validation or generation of presentation (see Fig. 2). If so, a pre-/post-commit hook sends requests to the TNTBase RESTful interface, where the actual processing is executed. The return codes and error output stream are used to notify the committer about the result (e. g. validation errors). This mechanism is surprisingly flexible and naturally fits into the xSVN approach, since it is inherited from SVN. In the future, we are going to increase further scalability and manageability, but our approach shows that that the workflows described in this paper naturally work inside the TNTBase architecture.

These workflows are configured in the repository itself via specialized XML configuration files in a pre-defined *admin* directory. Thus, the configuration can be edited offline. A workflow is enabled in the three steps:

1. Provide methods (units that comprise a workflow) identifiers and additional information. In case of schema validation if may be the location of a schema (that may reside in the repository itself). If an advanced validation that is performed by a 3rd party plugin is needed, the Java class name that plugin has to be provided. Once the configuration files have been committed, TNTBase will reserve a unique URL of the form http://<tntbase_host>/restful/integration/validation/<workflow_name>/validate?path=/path/to/file1.omdoc&path=/path/to/file2.omdoc, which can be used for manually validating a set of files.
2. If validation or presentation generation should be performed on commit, the user may set the tntbase:validate SVN property on a file or directory. The format of this property is $w_1+w_2+\ldots+w_n$, a list of workflow names separated by "+".
3. If step 2 takes place, then, finally, the user has to generate a pre-/post-commit hook via a special TNTBase hook generator, providing the workflow name. The installation of this hook will lead to processing (like validation or presentation) of the affected by tntbase:validate property files.

Let us consider the following example (see [31] for elaborated information). Suppose we want to validate lecture notes on the topic of graphs against a RelaxNG schema and forbid commits of invalid files. Moreover, we intend to perform a structural validation, one of whose subtasks is checking that there are no cyclic imports (suppose this structural validation is performed by a TNTBase plugin), but in case of failure we only want to notify the user, but not reject the commit. First of all, we create a configuration file /admin/validation/methods.xml in our repository:

```
<methods xmlns="http://tntbase.mathweb.org/ns">
  <schema name="omdocRNG" location="tntbase:/admin/validation/omdoc.rng" type="rng"/>
  <java name="structural" class="info.kwarc.tntbase.plugins.StructuralValidationPlugin"/>
</methods>
```

We define schema validation named omdocRNG, whose schema is located in the same repository (note the tntbase: prefix). Also, we define the structural validation named structural and performed by a Java class that implements the TNTBase plugin interface. In order to expose OMDoc files to a validation workflow in our scenario, we set the tntbase:validate property on a folder /lectures/graphs with value *.omdoc omdocRNG+structural, which means that all files with extension omdoc in the folder /lectures/graphs will be validated against a schema named omdocRNG, followed by the validation step named structural. The last step in our setting is to generate a pre-commit hook for the omdocRNG method via a bundled script, and a post-commit hook for the structural method. After the creation of these repository hooks, our desired scenario is installed. These steps need some coordination on the part of the user but provide a powerful and flexible framework for manipulating files, notifying users and controlling commit time behavior.

4 Virtual Documents – Views on XML Documents

Virtual Documents (VDs) are a general framework for *integrating XQueries into XML documents as computational devices* and processing them efficiently. As a rough approximation, VDs are "XML database views" analogous to views in relational databases; these are virtual tables in the sense that they are the results of SQL queries computed on demand from the explicitly represented database tables. Similarly, VDs are the results of XQueries computed on demand from the XML files explicitly represented in TNTBase, presented to the user as entities (files) in the TNTBase filesystem. Like views in relational databases TNTBase, VDs are editable and become very useful abstractions in the interaction with versioned XML storage. In this section we introduce a reader to the concept of VDs. For further information refer to [31] and [29] for theoretical and practical aspects, respectively.

4.1 VDs Introduction by Example

VDs are first class citizens in the TNTBase filesystem. Whereas they are internally quite different from usual documents, they look like normal files for a user: one can browse them, validate, apply stylesheets, query and even modify them. VDs are essentially a tight mix of static XML parts with XQueries and instructions in XML form that instruct TNTBase how to arrange the XQuery results inside a VD. Let us start with the simple example that we want to have a joined list of mathematical exercises with information about their authors. We would like to have the root element and the elements that embrace authors and exercises. XQueries that select necessary data will augment our document.

Consider the following *VD Specification* (VD Spec) – a definition of a VD (see the bottom left of Fig. 3):

Listing 1.1. Example of a VD Spec

```
<tnt:virtualdocument xmlns:tnt="http://tntbase.mathweb.org/ns">
  <tnt:skeleton xml:id="exercises">
    <omdoc xmlns:dc="http://purl.org/dc/elements/1.1/">
      <dc:title>Exercises for Computer Science lectures</dc:title>
      <dc:creator>Michael Kohlhase</dc:creator>
      <omdoc>
        <dc:title>Acknowledgements</dc:title>
        <omtext>
          The following individuals have contributed material to this document:
          <tnt:xqinclude query="tnt:collection('/exercises//*.omdoc')//dc:creator">
            <tnt:return><tnt:result/></tnt:return>
          </tnt:xqinclude>
        </omtext>
      </omdoc>
      <omdoc>
        <dc:title>Exercises</dc:title>
        <tnt:xqinclude>
          <tnt:query name="exercises.xq"/>
          <tnt:return><tnt:result/></tnt:return>
        </tnt:xqinclude>
      </omdoc>
    </omdoc>
  </tnt:skeleton>
```

```
25    <tnt:query name="exercises.xq">
        for $t in $topics return
          tnt:collection(concat('/exercises/', $t, '/*.omdoc'))//exercise
      </tnt:query>
    </tnt:virtualdocument>
```

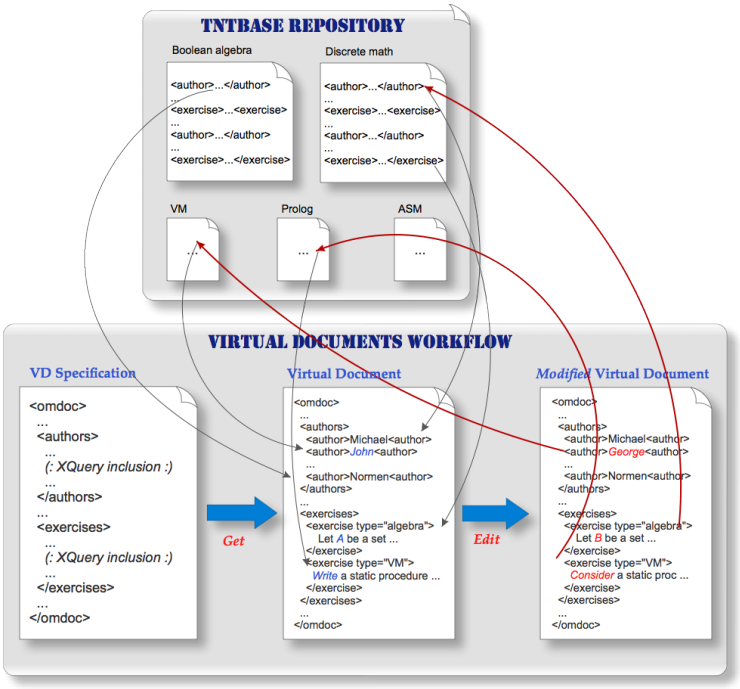

Fig. 3. A complete VD workflow

Conceptually, the VD Spec consists of the top-level parts of the intended exercises document, where some document fragments have been replaced by embedded XQueries that generate them. In general, queries are embedded into a VD Spec in the form <**tnt:xqinclude**><**tnt:query**>Q</**tnt:query**><**tnt:return**>R</**tnt:return**></**tnt:xqinclude**>, where Q is an XQuery and R consists of a result expression, in which tnt:result elements will be replaced with the results of Q. The query in line 10–12 has been abbreviated to <tnt:xqinclude query="Q">R</tnt:xqinclude>, as it does not contain embedded elements. The result of this particular query would be a list of author names wrapped in an *omtext* element. The query in lines 17–20 uses another useful feature: a reference to a shared, reusable external query (see below). Queries in VDs can also reference the physical files in a repository, or include other XQuery modules, which enables further reusability.

VDs are created simply by committing a VD Spec to the xSVN repository and executing an additional RESTful method of TNTBase that takes as an input

the path of a virtual document specification and the path of the new virtual document. Thus, we can create multiple virtual documents from a single virtual document specification. For instance, the above virtual document specification XQueries can be changed so that the VD would contain not more than a certain amount of exercises. This can be done by creating a new VD Spec that will reference a skeleton from List. 1.1 and *overrides* the exercises query:

Listing 1.2. Reusing a VD Skeleton

```
<tnt:virtualdocument xmlns:tnt="http://tntbase.mathweb.org/ns">
  <tnt:skeleton href="/specs/exercises-spec.xml"/>
  <tnt:query name="exercises.xq">
    for $t in $topics return
      tnt:collection(concat('/exercises/', $t, '/*.omdoc'))//exercise[position() le $max]
  </tnt:query>
  <tnt:params>
   <tnt:param name="max">
     <tnt:value>10</tnt:value>
   </tnt:param>
  </tnt:params>
</tnt:virtualdocument>
```

Thus we write a skeleton once, and may use it for creating VDs with different content. Note the parameter `$max`, which declared with a default value below the query element. There is one more undefined parameter `$topics` that can be defined in another VD Spec that references the current one, or when creating a VD file system entity via a RESTful method (see [33, 32] for the technical details). Thus the reference-based setup caters for a wide variety of reuse scenarios, as we can see in List. 1.2. One may think of method overriding in Java or C++. It is also possible to have an empty `tnt:query` element as the specification. Then it becomes an *abstract* specification, and can be compared to abstract methods in Java or pure virtual functions in C++.

Virtual Document are can be retrieved via a RESTful request (see middle bottom picture of Fig. 3), or via XQuery (see below).

4.2 Other Features of VDs

Retrieving the content of a VD already gives us read-only "XML database views", but we can do more with VDs:

Querying Like usual documents, VDs can also be queried via the TNTBase XQuery function `tnt:vdoc($path as xs:string)`. It can be mixed with other elements of a query, or even can be used to constitute queries of other VD Specs. If a VD Spec is self-contained (i. e. it has all referenced variables and queries defined), then its generated content can be also retrieved without the need to have a VD file system entity associated with this VD Spec: `tnt:vd-spec($spec-path as xs:string)`.

Materializing We can turn the content of a VD into a regular file in a repository, i. e. *materialize* a VD. That makes sense, for instance, when computing a VD takes a considerable amount of time, whereas the content of the sources queried by the VD changes rarely, or when a user intends to fix the content of the source document (cf. the ontology refactoring use case in Sect. 5) and make it versioned: In our example, when a user is satisfied with the list of exercises he obtained

from the VD, he might want to make it persistent by adding it to repository under a certain path. If there is already a file in that place, materializing results in a new revision of that file. Thus we can even version VDs, with the possibility of rollback if required. **Editing** One of the most advanced features of is the ability to *edit* VDs and *commit* them back TNTBase via the RESTful interface: changed parts of a VD that came from files in a repository will be transparently propagated back to the sources, while the repository history is preserved, i. e. a new revision is created. At the bottom right of Fig. 3, we can see the final phase of a VD workflow: editing and submitting it back. The modified parts (marked in red) are propagated back to their origin. All changes are performed in a single xSVN transaction, of which only those files will be part that were implicitly affected by editing the VD. Editing static parts of a VD is not allowed; otherwise TNTBase will not allow to commit a VD. The XML diff discussed on Page 5 takes identified allowed and prohibited changes. The strong advantage of editing VDs is that users can abstract away from the physical documents in the repository and work with semantically consistent objects (like theorems or exercises) focusing only on relevant information aggregated into one logical unit.

5 Applications

TNTBase is constantly evolving becoming mature for integration into other projects or ready to be used as a stand-alone application. In this section we describe the use cases and projects where TNTBase has found its application so far.

Repository for Computer Science Lectures at Jacobs University: Our research group has accumulated a collection of more than 2000 slides of lecture notes and homework problems. They are originally written in STEX, a LATEX-based input syntax for OMDoc. Besides being compiled to PDF for presenting and reading, they can be translated to OMDoc using the LATEXML TEX→XML converter[20]. The JOMDoc library [12] can render the OMDoc semantic markup to human-readable XHTML, plus MathML for mathematical formulæ [1]. These steps had been performed semi-automatically in the time before TNTBase. Beyond that, we wanted to be able to query the document collection (e. g. "what exercises deal with structural induction?"), to generate exams automatically (e. g. by selecting exercises that cover certain topics, have not been used last year, and sum up to one hour), and to make the lecture notes browsable interactively (e. g. by showing the definition of a mathematical symbol in a popup, when the user clicked on an occurrence of it in a formula). These features were enabled by maintaining the lecture notes in TNTBase and integrating plugins for certain tasks. The LATEXML plugin is now triggered after committing an STEX document and converts it automatically to OMDoc. These OMDoc documents can then be queried, but we have also implemented automated exam generation as VDs that we materialize to publish final versions of exams. The Krextor RDF extraction plugin (cf. [17]), which is run afterwards, extracts structural outlines and metadata from the latest version of an OMDoc document into an

RDF representation in order to (i) provide semantic querying possibilities beyond XQuery[4], (ii) to be able to enrich the rendered documents with semantic annotations, to which interactive services can attach (e. g. in-place lookup of relevant linked information), (iii) and to publish the contents of a repository as Linked Data, so that external semantic search engines and mashups can make use of them. [10]. In order to store RDF scalably, to answer SPARQL queries, and to publish RDF as Linked Data, we put the RDF extracted by Krextor into a Virtuoso RDF database ("triple store") [23], which is connected to TNTBase by another plugin. Finally, there is a plugin that integrates the JOMDoc library, which renders OMDoc human-readable XHTML+MathML documents and now also enriches them with semantic RDFa annotations [10].

Logic Atlas: In the LATIN project (Logic Atlas and Integrator), we deal with almost 300 highly modularized and interlinked logical theories that have been formalized in OMDoc. Similarly to the lecture note scenario, these documents have to be rendered to human-readable presentations. This is realized by the MMT (Module system for Mathematical Theories) web server application, which obtains the formalized documents as well as instructions for how to display formal mathematical symbols from different locations in an underlying TNTBase repository. Additionally, validation of the mathematical theories was required – a structural validation (in terms of Sect. 3) that extends beyond XML schema validation. For that purpose, the MMT library has been integrated into TNTBase as another validation plugin. After schema validation, it extracts the formal logical structure of an OMDoc document to an RDF-like representation – representing, for example, whether one mathematical symbol is defined in terms of other symbols, whether other symbols occur in its type declaration, or whether a mathematical theory imports (= reuses) other theories. Based on that information, it checks the logical *well-formedness*. If, for example, a definition uses symbols that have not been imported before, this definition and thus the whole document is not well-formed. We refer to [16] for further details.

Ontology Engineering and Refactoring: Ontology engineering is a discipline closely related to software engineering. Usually, engineers collaborate on a project. Support for integrated testing as well as refactoring, i. e. restructuring the (internal) structure of an ontology without changing its external behavior, is essential. We utilize VDs for refactoring ontologies that are written in the OWL 2 Web Ontology Language [25] and serialized as XML [22]. In the lifecycle of an ontology, it is not always obvious whether a refactoring step is appropriate, and it is not always granted that it does not have any effects on external modules (e. g. software, or annotated documents) depending on the ontology. Therefore, we allow for *previewing* refactorings via VDs. The specifications of these VDs

[4] The advantage of querying RDF (in the SPARQL language) is that an RDF representation can abstract from different structural dimensions of knowledge being represented in syntactically different ways in XML. We have detailed that in a case study with Software Engineering documents containing structures in dimensions as diverse as mathematical models, project organization and responsibilities, document layout, and revision histories[14].

are maintained in the same repository as the ontologies and can collaboratively be refined by the ontology developers. We have so far implemented VD specifications for renaming entities, factoring out or merging modules, rewriting axioms, lowering expressivity and stripping axiom annotations. Once such a VD specification has been applied to an ontology [module] – physical ontology documents or another VD, the resulting view on the refactored ontology can be downloaded and tested in an ontology development environment. Once the developers are satisfied with the result of a refactoring, they can materialize the VD, thus obtaining the refactored ontology as a physical document. We envisage TNTBase as a backend to be integrated with ontology engineering tools in order to complement their functionality. Besides translation and refactoring support, we envisage automated testing support [21]: When a user commits an ontology, a pre-commit method feeds it into a reasoner (integrated as a validation plugin) in order to check its consistency – quite similar to MMT's logical well-formedness check mentioned above. We refer to [18] for further details about refactoring OWL ontologies.

6 Conclusion and Future Work

We have presented the TNTBase, a versioned XML database, in all of its facets, from the technical core to its current applications. The core, formed from the integration of the two reliable and scalable systems Subversion and Berkeley DB XML, combines the document collection management capabilities of version control systems (branching, merging, history view, collaborative editing) with the fine-grained fragment access and querying capabilities of XML databases. Much of TNTBase's core functionality is accessible via a normal Subversion interface, which allows for a gentle migration from legacy Subversion-based workflows towards applications that take more advantage of the added XML functionality, which is accessible via a RESTful interface. TNTBase can be adapted to applications that require advanced XML document management workflows in various ways: The plugin architecture allows for the integration of application-specific components, e. g. for validation beyond the XML schema, knowledge extraction, and human-friendly presentation. Virtual Documents (VDs) provide flexible views on XML documents.

Over the last two years, TNTBase has matured enough for being used in the application scenarios that we have summarized in Sect. 5. The largest-scale application, the lecture note repository, has so far demanded the highest extent of customization. It showcases all varieties of application-specific extensions: validation and translation of committed TEX sources, rendering of human-readable web documents with integrated interactive lookup services, knowledge extraction for semantic query answering and integration into the Semantic Web, as well as generation of exams via VDs. The logic atlas and ontology engineering applications further demonstrate the diverse applicability and customizability of TNTBase. In the course of using TNTBase for these applications, we have significantly increased stability and performance, taking care of multi-user en-

vironments and scalability. Furthermore, one is able to mirror TNTBase repositories to normal SVN repositories by replication functionality adopted from Subversion. This possibility once more justifies the decision of combining the two systems as it allows not to worry about TNTBase failures that may cause data corruption (actually, it never happened to us): in this case we can easily restore them from a replicated SVN repository with all history preservation.

Future work on TNTBase will concentrate on providing further high-level XML processing functionality, from which all of our current application scenarios will benefit. We are working on integrating a semantic XML differencing algorithm [2] into TNTBase. It will allow users to compare XML trees based on declarative equivalence models that describe what XML nodes in a particular language are considered to be equal. For example, we might consider two theory elements in OMDoc equal if their formal parts are equal, ignoring narrative structures only written for human readers, and ignoring the order of examples. Again, the equivalence models will be collaboratively editable in the same repository where XML files reside.

References

[1] *Mathematical Markup Language 3.0*. Candidate Recommendation. W3C, 2009. URL: http://w3.org/TR/2009/CR-MathML3-20091215.

[2] S. Autexier and N. Müller. "Semantics-Based Change Impact Analysis for Heterogeneous Collections of Documents". In: *10th ACM Symposium on Document Engineering*. under submission. 2010.

[3] *XML Path Language (XPath) 2.0*. Recommendation. W3C, 2007. URL: http://w3.org/TR/2007/REC-xpath20-20070123/.

[4] *Berkeley DB*. URL: http://oracle.com/technology/products/berkeley-db/.

[5] *Berkeley DB XML*. URL: http://oracle.com/database/berkeley-db/xml/.

[6] *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986. IETF, 2005. URL: http://www.ietf.org/rfc/rfc3986.txt.

[7] *XQuery: An XML Query Language*. Recommendation. W3C, 2007. URL: http://w3.org/TR/xquery/.

[8] *XQuery Update Facility 1.0*. Candidate Recommendation. W3C, 2008. URL: http://w3.org/TR/xquery-update-10/.

[9] *RELAX NG Specification*. Tech. rep. OASIS, 2001. URL: http://www.relaxng.org/spec-20011203.html.

[10] C. David, M. Kohlhase, C. Lange, F. Rabe, N. Zhiltsov, and V. Zholudev. "Publishing Math Lecture Notes as Linked Data". In: *ESWC*. LNCS 6089. Springer, 2010. arXiv: 1004.3390.

[11] *Ipedo XML Database*. URL: http://ipedo.com/html/ipedo_xml_db.html.

[12] *JOMDoc Project — Java Library for OMDoc documents*. URL: http://jomdoc.omdoc.org.

[13] *JSR 311: JAX-RS: The Java API for RESTful Web Services*. URL: https://jsr311.dev.java.net/nonav/releases/1.0/.

[14] A. Kohlhase, M. Kohlhase, and C. Lange. "Dimensions of Formality: A Case Study for MKM in Software Engineering". In: *Intelligent Computer Mathematics*. LNAI. Springer, 2010. arXiv: 1004.5071.

[15] M. Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. LNAI 4180. Springer, 2006. URL: http://omdoc.org.

[16] M. Kohlhase, F. Rabe, and V. Zholudev. "Towards MKM in the Large: Modular Representation and Scalable Software Architecture". In: *Intelligent Computer Mathematics*. LNAI. Springer, 2010. URL: http://kwarc.info/kohlhase/papers/mkm10-scalable.pdf.

[17] C. Lange. "Krextor – An Extensible XML→RDF Extraction Framework". In: *Scripting and Development for the Semantic Web (SFSW)*. 2009. URL: http://kwarc.info/projects/krextor/pubs/sfsw09-krextor.pdf.

[18] C. Lange and V. Zholudev. "Previewing OWL Changes and Refactorings Using a Flexible XML Database". In: *Ontology Repositories and Editors*. CEUR. 2010. URL: http://kwarc.info/clange/pubs/ores2010-tntbase.pdf.

[19] *MarkLogic Server*. URL: http://www.marklogic.com/product/marklogic-server.html.

[20] *LaTeXML: A LaTeX to XML Converter*. URL: http://dlmf.nist.gov/LaTeXML/.

[21] D. Misev. "Integrating SUMO and OMDoc". Bachelor's Thesis. Computer Science, Jacobs University, Bremen, 2010.

[22] *OWL 2: XML Serialization*. Recommendation. W3C, 2009. URL: http://w3.org/TR/2009/REC-owl2-xml-serialization-20091027/.

[23] OpenLink Software. *OpenLink Universal Integration Middleware – Virtuoso Product Family*. URL: http://virtuoso.openlinksw.com.

[24] *Oracle XML DB*. URL: http://oracle.com/technology/tech/xml/xmldb/.

[25] *OWL 2: Document Overview*. Recommendation. W3C, 2009. URL: http://w3.org/TR/2009/REC-owl2-overview-20091027/.

[26] *Subversion*. URL: http://subversion.tigris.org/.

[27] *SVNKit*. URL: http://svnkit.com/.

[28] *TNTBase TRAC*. URL: https://tntbase.org.

[29] V. Zholudev and M. Kohlhase. "Scripting Documents with XQuery: Virtual Documents in TNTBase". 2010. URL: http://kwarc.info/kohlhase/papers/balisage10.pdf.

[30] V. Zholudev and M. Kohlhase. "TNTBase: a Versioned Storage for XML". In: *Balisage*. Mulberry, 2009. URL: http://kwarc.info/vzholudev/pubs/balisage.pdf.

[31] V. Zholudev, M. Kohlhase, and F. Rabe. "A [insert XML Format] Database for [insert cool application]". In: *XML Prague*. 2010. URL: http://kwarc.info/vzholudev/pubs/XMLPrague.pdf.

[32] *TNTBase – RESTful API*. URL: https://tntbase.org/wiki/restful.

[33] *TNTBase – Virtual Documents*. URL: https:/tntbase.org/wiki/VD.