The TNTBase System and Validation of XML Documents

Vyacheslav Zholudev
Jacobs University of Bremen
D-28759, Bremen, Germany
v.zholudev@jacobs-university.de

Abstract

TNTBase is an open-source versioned XML database obtained by integrating Berkeley DB XML into the Subversion Server. The system is intended as a basis for collaborative editing and sharing XML-based documents. It integrates versioning and fragment access needed for fine-granular document content management.

Nowadays well-formedness of electronic documents plays a giant role in the contemporary document workflows and applications have to provide domain-specific validation mechanisms for documents they work with. On another hand, XML is coming of age as a basis for document formats, and even though there are a lot of schema-based validation formats and software available for XML, domain-specific directions still remain unfilled.

In this paper we present the TNTBase system in general and its validation support for XML documents.

1 Introduction

With the rapid growth of computers and Internet resources the communication between humans became much more efficient. The number of electronic documents and the speed of communication are growing rapidly. We see 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 [BLFM98] and XPath [CD99; BBC$^+$07]) 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, validated and mashed up from background information in XML databases by techniques like XQuery [XQu07] and document (fragment) collections are managed by XQuery Update [XQU08].

The Web is constantly changing — it has been estimated that 20% of the surface Web changes daily and 30% monthly [CGM00; FMNW03]. While archiving services like the `Wayback Machine` try to get a grip on this for the surface level, we really need an infrastructure for managing and validating changes in the XML-based deep web.

Unfortunately, support for this has been very frugal. Version Control systems like CVS and Subversion [SVN08] which have transformed collaboration workflows in software engineering are deeply text-based (wrt. diff/patch/merge) and do not integrate well with XML databases and validators for different schema languages for XML, like RelaxNG [Rel] or XML Schema [W3C06]. Some relational databases address temporal aspects [DDL02], but this does not seem to have counterparts in the XML database or XQuery world. Wikis provide simple versioning functionalities, but these are largely hand-crafted into each system's (relational) database design.

In this paper we describe in short the TNTBase system, an open-source versioned XML database obtained by integrating Berkeley DB XML [Ber09b] into the Subversion Server [SVN08]. 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 the enabling technologies of versioning and fragment access needed for fine-granular document content management. Also we discuss our vision of how the validation of XML documents should be done in a way that is conformant to Subversion philosophy and how it fits to the TNTBase system.

The TNTBase system is developed in the context of the OMDoc project (Open Mathematical Documents [OMD; Koh06]), an XML-based representation format for the structure of mathematical knowledge and communication. Correspondingly, the development requirements for the TNTBase come out OMDoc-based applications and their storage needs. We are experimenting with a math search engine [KŞ06], a collaborative community-based reader panta rhei [pan], the semantic wiki SWiM [Lan08], the learning system for mathematics ActiveMath [Act08], and a system for the verification of statements about programs VeriFun [Ver08].

In the next section we will summarize what the TNTBase system is and what it does. Then we will be ready to cover validation mechanisms (see Section 3) offered by TNTBase and explain some design decisions. Section 4 will detail ideas for future work regarding validation of OMDoc documents, and Section 5 concludes the paper.

2 The TNTBase System

2.1 Overview

In this section we provide an overview of TNTBase to allow a better understanding of Section 3. Details can be found at [ZK09].

A slightly simplified view of the TNTBase architecture is presented in Figure 1. The core of TNTBase is the xSVN library developed by the author. The main difference between xSVN and the SVN server it replaces is that the former stores the youngest revisions of XML

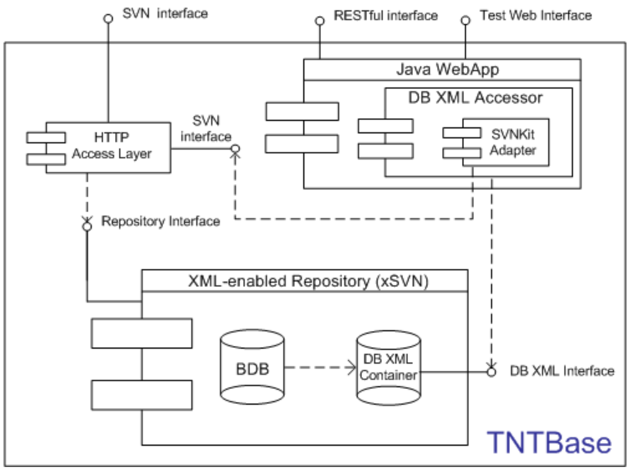

Figure 1: TNTBase architecture

files and also other revisions (on user requests) in Berkeley DB XML (DB XML) instead of Berkeley DB [Ber09a]. This gives us a possibility to employ the DB XML API for querying and modifying XML files via XQuery and XQuery Update facilities. Also such a substitution helps us to keep XML files well-formed and, optionally, conformant to an XML Schema.

In TNTBASE XSVN is managed by Apache's `mod_dav_svn` module or accessed by DB XML ACCESSOR (a Java library which provides a high-level access to DB XML on top of its API) locally on the same machine. Apache's `mod_dav_svn` module exposes an HTTP interface exactly like it is done in SVN. Thereby a TNTBASE user can work with TNTBASE repository exactly in the same way as with a normal SVN repository via HTTP protocol including Apache's SVN authentication via `authz` and `groups` files. In other words any SVN client is able to communicate with TNTBASE. The non-XML content can be managed as well in TNTBASE, but only via an XSVN's HTTP interface.

The DB XML ACCESSOR module can work directly with XML content in an XSVN repository by utilizing the DB XML API. All indispensable information needed for XML-specific tasks is incorporated in a DB XML container using additional documents or metadata fields of documents. SVNKITADAPTER (a Java library which employs SVNKit [SVN07]) comes into play when the revision information needs to be accessed, and acts as a mediator between an XSVN repository and DB XML ACCESSOR. And in turn when DB XML ACCESSOR intends to create a new revision in a XSVN repository it also exploits SVNKITADAPTER functionality.

The DB XML ACCESSOR realizes a number of useful features, but is able to access an XSVN repository only locally. To expose all its functionality to the world TNTBASE provides a RESTful interface, see [ZK09; TNT09b]. We use the Jersey [Jer09] library to implement a RESTful interface in TNTBASE. Jersey is a reference implementation of JAX-RS (JSR 311), the Java API for RESTful Web Services [JSR09] and has simplified our implementation considerably.

TNTBASE provides a test web-form that allow users to play with a subset of the TNTBASE functionality. Also an XML-content browser is available online which shows the TNTBASE file system content including virtual files. United authentication for all interfaces is a subject for a

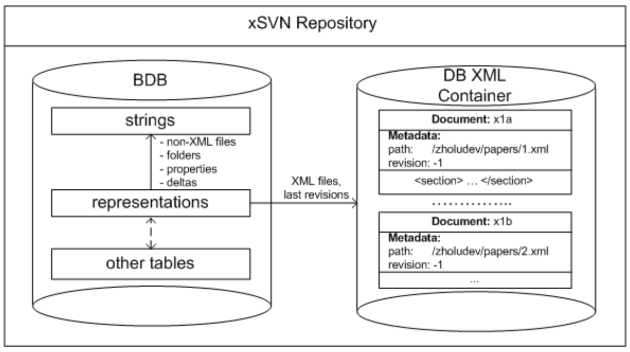

Figure 2: XSVN repository

future work.[1]

Currently readers can try out an online TNTBASE test instance (see [TNT09a]), for additional information refer to [TNT09b].

2.2 XSVN, an XML-enabled Repository

Since XSVN is a core of TNTBASE and will be referred to in Section 3, we will cover it here in detail. The architecture of XSVN and thus TNTBASE is motivated by the following observation: Both the SVN server and the DB XML library are based on Berkeley DB (BDB) [Ber09a]. The SVN server uses it to store repository information[2], and DB XML for storing raw bytes of XML and for supporting consistency, recoverability and transactions. Moreover, transactions can be shared between BDB and DB XML. Let us look at the situation in more detail.

The SVN BDB-based file system uses multiple tables to store different repository information like information about locks, revisions, transactions, files, and directories, etc. The two important tables here are *representations* and *strings*. The *strings* table stores only raw bytes and one entry of this table could be any of these:

1. a file's contents or a delta (a difference between two versions of the same entity (directory entry lists, files, property lists) in a special format) that reconstructs file contents
2. a directory entry list in special format called *skel* or a delta that reconstructs a directory entry list skel
3. a property list skel or a delta that reconstructs a property list skel

From looking at a *strings* entry alone there is no way to tell what kind of data it represents; the SVN server uses the *representations* table for this. Its entries are links that address entries in the *strings* table together with information about what kind of *strings* entry it references, and — if it is a delta — what it is a delta against. Note that the SVN server stores only the youngest revision (called the **head revision**) explicitly in the *strings* table. Other revisions of whatever entity (a file, a directory or a property list) are recomputed by recursively applying inverse deltas from the head revision.

To extend SVN to XSVN (an *XML-enabled repository*), we have added the DB XML library to SVN and add a new type of entry in the *representations* table that points to the last version of that document in the DB XML *container*

[1] see Ticket https://trac.mathweb.org/tntbase/ticket/3

[2] In fact SVN can also use a file-system based storage back end (SVN FS), but this does not affect TNTBASE.

(see Figure 2). Containers are entities in DB XML which are used for storing XML documents. Literally, a container is a file on disk that contains all the data associated with your documents, including metadata and indices. For every xSVN repository we use only one container located in the same folder as BDB tables, and therefore it allows us to share the same BDB environment exploited by an SVN back end.

From an end-user perspective there is no difference between SVN and xSVN: all the SVN commands are still available and have the same behavior. But for XML documents the internals are different. Assume that we commit a newly added XML file[3]. Now its content does not go to the *strings* table, but instead a file is added to DB XML container with a name which is equal to the reference key stored in the also newly created *representations* entry of *DB XML full-text* type. Note when we commit a set of files, and even one of XML files is not well-formed then the commit fails and no data are added into an xSVN repository, which conforms to the notion of a transaction in SVN and DB XML. When we want to checkout or update a working copy, xSVN knows what files are stored in DB XML and those files are read from a DB XML container. Another important thing is the scenario when we commit another version of an XML file. The older revision is deleted from DB XML, the newer revision is added to DB XML and a delta between these revisions are stored in the *strings* table. This delta has the normal SVN format and the SVN deltification algorithms have not been changed in xSVN. Thus we are still able to retrieve older revisions of XML documents. Concerning non-XML files the workflow of xSVN is absolutely the same as in SVN: data are stored in the same BDB tables, and the code behaves entirely in the same way. Thereby we are also able to store text or binary data in xSVN which can supplement the collection of XML files (e.g. licensing information or PDFs generated from XML). And moreover we can add or commit XML and non-XML files in the same transaction.

In conclusion: xSVN already offers a versioned XML storage, but without additional modules it is useless as the only difference to SVN is that it refuses to commit ill-formed XML documents. The detailed description of additional services built on top of xSVN is out of scope of this paper (refer to [ZK09] for such information).

3 XML Validation

In this section we will discuss only the xSVN part of TNTBase and will explain how validation is realized in an SVN-compatible way. Although addition of content is also allowed via RESTful interface of TNTBase, the most convenient and manageable way of doing this is utilizing an SVN client, and therefore validation should be implemented on the xSVN server and should be managed by any SVN client.

[3]By default, xSVN considers a file as an XML document if its extension is *.xml* or its *svn:mime-type* property is set to either *text/xml* or *application/xml*. This behavior can be easily adapted, for instance, by checking if a file starts with `<?xml`. Even now an SVN user can benefit from using automated property setting in SVN, i.e. associate certain file extensions with *text/xml svn:mime-type* property. For example, **.xslt* or **.xsd* would obtain *text/xml* mime-type on adding to a working copy and therefore will be treated as XML files for xSVN.

3.1 Why Relax NG?

As we mentioned above, integrating DB XML to the SVN server automatically gives us inspection of XML for well-formedness and conformance to W3C XML Schema associated with a particular XML document. While XML Schema successfully tackled problems with DTDs, it also exposed better opportunities for defining XML languages. But using XML Schema for validation of documents is not always convenient or enough. For instance, an official schemata for OMDoc and MathML [ABC+09] formats are in the Relax NG syntax. Other XML languages may also have no official XML Schema, or a developer of a new XML language may prefer Relax NG. There are a lot of supporters of XML Schema as well as of Relax NG, and there are plenty of disputes about which format is better, but one fact is unquestionable: TNTBase should support Relax NG validation as well since the language our system focuses on is OMDoc.

There could be two ways of avoiding Relax NG validation in TNTBase:

1. Make XML Schema as a primary format for describing OMDoc language. Thus the necessity of having Relax NG validation disappears.
2. Use Relax NG as a primary format for OMDoc, but every time OMDoc Relax NG schema is changed, regenerate XML Schema out of it and use the latter in TNTBase.

The first item does not suit us since XML Schema format has problems that are not presented in Relax NG. Let us enumerate the most significant of them (for more information refer to [XSD09]):

1. XML Schema is hard to read and may be interpreted incorrectly by users not experienced enough.
2. XML Schema provides very weak support for unordered content.
3. XML Schema's support for attributes provides no advance over DTDs.
4. The XML Schema Recommendation is hard to read and understand.

The generation XML Schema out of OMDoc's Relax NG Schema does not work, since even the best converter which the author explored so far — Trang [Tra09] — is not able to convert it. The reason for this is that Trang does not support nested grammars, but they are used in OMDoc's Relax NG.

Thus, the decision was to implement Relax NG validation in TNTBase.

3.2 Relax NG Validation in xSVN

As was mentioned in the beginning of this section, the validation should be realized in xSVN which is implemented in C/C++ programming languages. The latter fact complicates the integration of many Relax NG validator engines since most of them are written in Java. Most notable of them are: Jing [Jin09] and MSV [MSV09]. The only decent Relax NG validator for C/C++ which the author managed to find so far is Libxml2 [Vei] library. But the serious disadvantage of Libxml2 is its ambiguous and not well-designed error message system. Therefore the decision was to refuse Libxml2 library and make use of one of Java Relax NG validators, namely *Jing*. For integration between C++ and Java *The Java Native Interface (JNI)* [JNI09] has been employed. Due to awkwardness of JNI and, in particular, Java's method invocation from C/C++, the Jing library has been changed in a way that it simplifies Relax NG validation inside xSVN and returns nice-looking error messages back (if any occured). Thus the combination of a modified Jing library and a part of C++ code which invokes methods of this library comprises the xSVN module responsible for Relax NG validation.

3.3 How to Tell xSVN What to Validate?

When we were trying to answer on the question "How to tell xSVN what to validate", we wanted to keep an SVN client unchanged and modify only the server side of xSVN. The ultimate solution has two aspects: client and server. On the client side a user has to provide the `tntbase:validate` property for a file he intends to expose for validation. Also this property can be set on a folder recursively, then all files in that folder (and its subfolders) will be validated by xSVN. The value of this property should be a name of a schema. The names of all user-available schemata are stored on a server side in an ad-hoc schema configuration file (SCF) `schemata.xml` which is situated in `db` folder of your repository. The template file is generated automatically during creation of a new repository. An SCF may look like this:

Listing 1: Schemata configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<schemata xmlns="http://tntbase.mathweb.org/ns">
    <schema name="omdoc1.2"
      path="/home/OMDoc/omdoc-1.2/omdoc.rnc" type="rnc"/>
    <schema name="omdoc1.6"
      path="/home/OMDoc/omdoc.rng"/>
    <schema name="docbook1.5"
      path="/vzholudev/Papers/Balisage/balisage-1-1.rng"/>
</schemata>
```

So as we can see for each name we have a file system path which represents a schema. An xSVN administrator is responsible for setting this up. If a user sets a schema name that is not in an SCF, then the files to be validated against this schema are considered to be invalid and the whole xSVN transaction is aborted. Also if during a commit even one file turned out to be invalid then the whole xSVN transaction is aborted as well, i.e. no files get committed. This perfectly reflects the notion of an SVN transaction. Furthermore, it is not necessary to set up the `tntbase:validate` property right after addition of a file (files). A user can do it at any point of time, e.g. after the revision *21* has been committed. Then after any commit of that file, it will be validated until the `tntnase:validate` property is removed. On Listing 1.1 we can see the use of the attribute `type` that denotes the type of a schema. Currently it could be either `rnc` or `rng` that represents Compact or XML syntaxes of Relax NG respectively. If the `type` attribute is omitted, then the type of Relax NG is calculated depending on the extension of a file in the `path` attribute. If this calculation failed to be done, then validation fails with the corresponding error.

3.4 Versioned Schemata

Often development of documents is accompanied by development of a corresponding schema. Sometimes new types or elements are added or old ones are being evolved in an XML language, and such changes should be reflected in a schema as well. Thus we want to keep a schema in a repository and validate documents against its last version (head revision). In the approach considered in the previous section schemata are meant to be in a file system but not in a repository. It would be relatively easy to change or enhance the format of the `schemata.xml` file in such a way that a schema name points to the path in a repository, and when documents are about to be validated, retrieve the appropriate schema file from a repository. However, things are getting more complicated when the main schema file contains links to secondary schemata. In this case the schema validator — in our case Jing — will not be able to resolve references to other schemata since it does not know where to search for them. There validation will fail even though an arbitrary document is well-formed. One of the solutions would be to implement special entity resolver in Jing which would know how to retrieve schemata by path from a repository.

However, the faster and more elegant solution exists. Assume that we store our schemata in a repository under the path `/main/schemata`. On a server side we checkout this path to a working copy to some place, e.g. `/var/www/schemata`, and update this working copy from a post-commit hook. So our schemata folder is always up-to-date and we can easily link this folder from the `schemata.xml`. The only overhead of this approach is that our data are duplicated: in a repository and in a local file system. The next step would be to place the `schemata.xml` file into repository and allow clients to manage this file remotely, but this is a subject for a future work[4].

3.5 Managing Validation Properties

Let us go back to our xSVN working copies. Setting up a validation property (`tntbase:validate`) for every single file we added might be somewhat cumbersome. Setting a property recursively for the whole directory may reduce our efforts. However, when we add a file later to the directory that has been exposed to the validation property, we do not get such a property for the newly added file. Probably, this behaviour is not what we want to achieve. We want the validation property to adhere to every added file whose parent directory has this property. In general we saw three ways of attaining this that differ in complexity and flexibility:

SVN client approach On the client side leverage the automated property setting which is offered by any SVN client. This method is quite straightforward and has nothing to do with the xSVN server. That is we can associate different validation properties with file extensions. For example, `*.omdoc` files will automatically get `omdoc1.6` validation property on addition to a working copy. For this in the per-user or system-wide configuration area (see Chapter "Runtime Configuration Area" in SVN Book [CSFP04]) one should modify `config` file by setting property `enable-auto-props` to `yes` and add the following line to the `auto-props` section: `*.omdoc = tntbase:validate omdoc1.6`.

The serious limitation of this method is that we can not set up different validation properties for the files with the same extensions, but which are located in different folders. For instance, if we have two folders for OMDoc documents, one is for version 1.2 and one is

[4]See Ticket `https://trac.mathweb.org/tntbase/ticket/54`

for version 1.6, then we can not associate schemata for different versions of OMDoc for these folders. Moreover, the administration of such a feature is done on the system or user level. That means that automated property setting will be applied for all repositories and all working copies. That might be not desirable in cases when a user works with multiple XSVN repositories that contain documents in different XML languages, but with the same extension, like `*.xml`.

SVN server approach This approach consists in implementing the pre-commit hook which checks the validation property of a parent folder of the committed item, and if the former owns one, then the same validation property is set to the committed item as well. This approach is more flexible than the previous one, but needs additional repository administration efforts (creating and managing hooks). Also it would be impossible to protect a single file in a folder against validation if a validation property has been set on a parent directory.

Combined approach The most scalable solution described here takes advantage of the first two methods. Each file may or may not have a `tntbase:validate` property. If it is presented, then it contains the name of a schema (like we discussed before). If it is not presented, then parent folders are taken into consideration. Each folder also may have a `tntbase:validate` property, but in a different format given by the following BNF: `tntbase:validate ::= (FILE_EXTENSION SCHEMA_NAME)* DEFAULT_SCHEMA_NAME`, i.e. every validation property, if presented, should have the value $ext_1\ s_1\ ext_2\ s_2\ ...\ ext_n\ s_n\ s_{def}$. Thus the files with the extension ext_i are validated against the schema s_i, all other files are validated against schema s_{def}. There is a special reserved schema name `none`, which tells that this file should not be validated. To sum up, when the file f with the extension ext is committed, it is validated against the schema s, where s is determined in the following order:

- If f has a `tntbase:validate` property, then s is extracted from it. Schema name might also be `none`.
- If f does not own a validation property, then the f's extension ext is being searched in the parent folder's validation property. If there is no entry $ext\ s$ in there, then s is s_{def}. If there is also no default schema name s_{def}, then we repeat this step for the parent folder of the f's parent folder.
- If we achieved the root of a repository and still did not find a schema name for f, then s becomes `none`.

This mechanism is fairly simple and gives an extreme flexibility and scalability. Moreover it does not require further repository administration — everything (apart from defining an SCF) is managed on the client side.

The third method has been implemented in TNTBase as the most sophisticated mechanism for managing `tntbase:validate` properties and defining independent islands of validation.

4 Towards High-Level Format-Specific Validation

We can distinguish tree stages of document validation (most languages exhibit the same problems, but we will use OMDoc language as an example):

1. **XML validation**. Implies well-formedness checking and validity according to a schema (if presented).
2. **Structural validation**. For example, all theorems have proofs, all used symbols are defined and are in scope.
3. **Semantic validation**. On this stage we should check that, for instance, all expressions are well-typed, all proofs are correct, examples contain material about entities they are linked to, etc.

Each stage is stricter than the previous one, and eventually all three should be performed in context of a single system like TNTBase that utilizes auxiliary libraries to achieve a validation goal.

The first stage is already implemented in TNTBase (see Section 3), just expansion of supported schema languages would be a possible task to do in this direction.

Sometimes we need even deeper validation then that is allowable by Relax NG schema (or any other schema language for XML). For example we might be willing to check whether all symbols are visible in an OMDoc document, i.e. each symbol has been defined locally (i.e. in the same file) or has been defined in those documents that are imported in the initial document (and so on recursively). Also we might want to check for redundant or cyclic imports. Consider the following parts of OMDoc documents:

Listing 2: arith1.omdoc

```
<?xml version="1.0" encoding="utf−8"?>
<omdoc xml:id="arith1−omdoc" version="1.6"
        modules="CD"
        xmlns:dc="http: // purl .org/dc/elements /1.1/ "
        xmlns:cc="http: //creativecommons.org/ns"
        xmlns="http: //omdoc.org/ns">
  ...
  <theory xml:id="arith1">
    <symbol name="plus" xml:id="plus">
      <metadata>
        ...
      </metadata>
      <type system="sts.omdoc#sts">
          ...
          <!−− definition goes here −−>
          ...
      </type>
    </symbol>
        ...
  </theory>
</omdoc>
```

Listing 3: alg1.omdoc

```
<?xml version="1.0" encoding="utf−8"?>
<omdoc xml:id="arith1−omdoc" version="1.6"
        modules="CD"
        xmlns:dc="http: // purl .org/dc/elements /1.1/ "
        xmlns:cc="http: //creativecommons.org/ns"
        xmlns="http: //omdoc.org/ns">
  ...
  <theory xml:id="alg1"  cdbase="http: //www.openmath.org/cd">
    ...
    <imports xml:id="alg1−imports−arith1" from="arith1.omdoc#arith1"/>
    ...
    <assertion  xml:id="zero−prop−1" type="lemma">
      ...
      <FMP>
        <OMOBJ xmlns="http://www.openmath.org/OpenMath">
          ...
            <OMS cd="arith1" name="plus"/>
          ...
```

```
            </OMOBJ>
          </FMP>
        </assertion>
        ...
      </theory>
    </omdoc>
```

On Listing 1.3 we can see the use of the symbol `plus` in the document `alg1.omdoc`. So sticking to our example, we must check that the symbol `plus` is defined in the scope. For this we must check the `imports` statement and then see whether the theory `arith1` of `arith1.omdoc` contains the definition of the `plus`, and it indeed does (see Listing 1.2). Thus if we are able to successfully check all the symbols in `alg1.omdoc`, then we say that this document is structurally valid (in our example). To be precise, we also have to check the absence of cyclic and redundant imports.

Such kind of validation is already available in JOMDoc [JOM] library and is referred to the second stage of document validation (see above). This type of validation provided by JOMDoc should be integrated into TNTBASE as the latter is positioned as an intelligent storage for OMDOC. Currently there is a stub in the XSVN validation engine which allows to have a different values of the `type` attribute in an SCF. So in the future if a schema name is associated with *jomdoc* type of validation then the more intelligent validity check will be performed. This validation usually involves multiple documents to be explored, and therefore the links between those documents should be resolved inside an XML container of XSVN. The special *imports* resolver should be implemented in JOMDoc in order to be able to find entities inside an XSVN's container (but not only in a file system or on Internet) that are referenced via *import* statements of OMDOC language. When this task is completed, we can start integrating the JOMDoc validation mechanism into the XSVN's validation engine. Other types of second stage validation are planned to be incorporated into JOMDoc library (see [Rab08] for more details).

Finally, the third stage is the most complicated one. For more details about this validation could be found at [Rab08]. Currently this is a subject for a future work, and ideas how to accomplish that are not formed clearly enough.

5 Conclusion

We have presented an overview of the TNTBASE system, a versioned XML database system that can act as a storage solution for an XML-based deep web, and discussed the validation mechanisms it exposes or will expose in the future. The implementation effort has reached a state, where the system has enough features to be used in experimental applications. TNTBASE may significantly ease implementation and experimentation of XML-based applications, as it allows to offload the storage layer to a separate system. Moreover users that require only versioning functionality may use TNTBASE as a version control system whereas more exigent users can experiment with additional features of the system. Even those users that need only versioning, can benefit from validating XML documents stored in the repository. That may help to keep a collection of documents more consistent and valid from different perspectives.

Acknowledgments

I would like to thank my supervisor Michael Kohlhase and my colleague Florian Rabe for providing use cases for the system and shooting at me their ideas. Their desire to have a system that totally satisfies their needs caused the considerable evolution of ideas devoted to the management of validation properties.

References

[ABC+09] Ron Ausbrooks, Bert Bos, Olga Caprotti, David Carlisle, Giorgi Chavchanidze, Ananth Coorg, Stéphane Dalmas, Stan Devitt, Sam Dooley, Margaret Hinchcliffe, Patrick Ion, Michael Kohlhase, Azzeddine Lazrek, Dennis Leas, Paul Libbrecht, Manolis Mavrikis, Bruce Miller, Robert Miner, Murray Sargent, Kyle Siegrist, Neil Soiffer, Stephen Watt, and Mohamed Zergaoui. Mathematical Markup Language (MathML) version 3.0. W3C Working Draft of 4. June 2009, World Wide Web Consortium, 2009.

[Act08] ACTIVEMATH, seen September 2008. web page at `http://www.activemath.org/`.

[BBC+07] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernandez, Michael Kay, Jonathan Robie, and Jerome Simeon. XML Path Language (XPath) Version 2.0. W3C recommendation, The World Wide Web Consortium, January 2007.

[Ber09a] Berkeley DB, seen January 2009. available at `http://www.oracle.com/technology/products/berkeley-db/index.html`.

[Ber09b] Berkeley DB XML, seen January 2009. available at `http://www.oracle.com/database/berkeley-db/xml/index.html`.

[BLFM98] Tim Berners-Lee, Roy T. Fielding, and Larry. Masinter. Uniform Resource Identifiers (URI), Generic Syntax. RFC 2717, Internet Engineering Task Force, 1998.

[CD99] James Clark and Steve DeRose. XML Path Language (XPath) Version 1.0. W3C recommendation, The World Wide Web Consortium, November 1999.

[CGM00] J. Cho and H. Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *Proc. of the 26th International Conference on Very Large Databases*, pages 200–209, 2000.

[CSFP04] Ben Collins-Sussman, Brian W. Fitzpatrick, and Michael Pilato. *Version Control With Subversion*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2004.

[DDL02] C.J. Date, Hugh Darwen, and Nikos Lorentzos. *Temporal Data & the Relational Model*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 2002.

[FMNW03] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet Wiener. A large-scale study of the evolution of web pages. In *WWW2003*. ACM Press, 2003.

[Jer09] Reference Implementation for building RESTful Web services, seen April 2009. available at `https://jersey.dev.java.net/`.

[Jin09] Jing — Relax NG Validator in Java, seen May 2009. available at `http://www.thaiopensource.com/relaxng/jing.html`.

[JNI09] The Java Native Interface, seen May 2009. available at `http://java.sun.com/docs/books/jni/`.

[JOM] JOMDoc Project — Java Library for OMDoc documents.

[JSR09] JSR 311: JAX-RS: The Java API for RESTful Web Services, seen April 2009. available at `https://jsr311.dev.java.net/nonav/releases/1.0/index.html`.

[Koh06] Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. Number 4180 in LNAI. Springer Verlag, 2006.

[KŞ06] Michael Kohlhase and Ioan Şucan. A search engine for mathematical formulae. In Tetsuo Ida, Jacques Calmet, and Dongming Wang, editors, *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2006*, number 4120 in LNAI, pages 241–253. Springer Verlag, 2006.

[Lan08] Christoph Lange. SWiM: A semantic wiki for mathematical knowledge management. web page at `http://kwarc.info/projects/swim/`, seen October 2008.

[MSV09] The Sun Multi-Schema XML Validator, seen May 2009. available at `https://msv.dev.java.net/`.

[OMD] OMDoc. web page at `http://omdoc.org`.

[pan] The panta rhei Project. seen March 2009.

[Rab08] Florian Rabe. *Representing Logics and Logic Translations*. PhD thesis, Jacobs University Bremen, 2008.

[Rel] A Schema Language for XML. available at `http://www.relaxng.org/`.

[SVN07] SVNKit - The only pure Java Subversion library in the world!, seen September 2007. available at `http://svnkit.com/`.

[SVN08] Subversion, seen June 2008. available at `http://subversion.tigris.org/`.

[TNT09a] TNTBase Demo, seen June 2009. Available at `http://alpha.tntbase.mathweb.org:8080/tntbase/lectures/`.

[TNT09b] TNTBase Home Page, seen June 2009. Available at `https://trac.mathweb.org/tntbase/`.

[Tra09] Trang — Multi-format schema converter based on RELAX NG, seen May 2009. available at `http://www.thaiopensource.com/relaxng/trang.html`.

[Vei] Daniel Veillard. The XML c parser and toolkit of gnome; libxml. System Home page at `http://xmlsoft.org`.

[Ver08] VeriFun: A verifier for functional programs, seen February 2008. system homepage at `http://www.verifun.de/`.

[W3C06] XML Schema. `http://www.w3.org/XML/Schema`, 2006. Seen July 2006.

[XQu07] XQuery: An XML Query Language, seen December 2007. available at `http://www.w3.org/TR/xquery/`.

[XQU08] XQUpdate: XQuery Update Facility 1.0, seen February 2008. available at `http://www.w3.org/TR/xquery-update-10/`.

[XSD09] XML Schema vs. RELAX NG, seen May 2009. available at `http://www.webreference.com/xml/column59/index.html`.

[ZK09] Vyacheslav Zholudev and Michael Kohlhase. TNTBase: a versioned storage for XML. accepted at BALISAGE 2009, available at `http://kwarc.info/vzholudev/pubs/balisage.pdf`, 2009.