Towards Collaborative Content Management And Version Control For Structured Mathematical Knowledge

Michael Kohlhase[1], Romeo Anghelache[2]

[1] Computer Science, Carnegie Mellon University, kohlhase+@cs.cmu.edu
[2] Albert Einstein Institute, Golm, Germany, romeo@psyx.org

Abstract. We propose an infrastructure for collaborative content management and version control for structured mathematical knowledge. This will enable multiple users to work jointly on mathematical theories with minimal interference.
We describe the API and the functionality needed to realize a CVS-like version control and distribution model. This architecture extends the CVS architecture in two ways, motivated by the specific needs of distributed management of structured mathematical knowledge on the Internet. On the one hand the one-level client/server model of CVS is generalized to a multi-level graph of client/server relations, and on the other hand the underlying change-detection tools take the math-specific structure of the data into account.

Versioning is a can of worms.
But what good is a can of worms if you never open it?
Norm Walsh on the `www-tag` *mailing list, 11 Sep. 2002*

1 Introduction

In the last years we have seen the birth of a new research area: "Mathematical Knowledge Management" (MKM), which is concerned with representation formalisms for mathematical knowledge, such as MathML [CIMP01], OpenMath [CC98] or OMDoc [Koh00], mathematical content management systems [FK00,ABC$^+$02,APCS01], as well as publication and education systems for mathematics. The perceived interest in the domain of general knowledge management tools applied to mathematics is that mathematics is a very well-structured and well-conceptualized subject. The main focus of the MKM techniques is to recover the content/semantics of mathematical knowledge and exploit it for the application of automated knowledge management techniques, with an emphasis on web-based and distributed access to the knowledge.

In this paper, we extend the focus of MKM techniques from the distributed *access* to mathematical knowledge to the *creation process* of mathematical knowledge, which is — for the most part — a distributed and collaborative process. After all, even if mathematicians often develop individual contributions alone (e.g.

in single-authored papers), the progress of a mathematical theory or sub-field involves a multitude of authors — communicating via meetings, messages and publications. Moreover, in contrast to the "knowledge access" scenario, where the mathematics is relatively static, the "knowledge creation" scenario involves managing the change of resources. We claim that MKM techniques have the potential of supporting this scenario as well, and that the "knowledge creation" scenario is potentially even more important for applications, as knowledge can only be accessed after it has been created. In fact, we expect the implementations of techniques like the ones presented in this paper, to play a similarly facilitating role in the development of open repositories of formal mathematical knowledge as the code management systems like the CVS system [CVS] have had for the creation of the wealth of open-source software we know today.

Currently, MKM systems either support simple monotonic addition of mathematical content or are specialized to particular applications, e.g. the Maya system [AHMS02] which is specialized to formal software engineering and verification. The "development graph" model for a management of theory change [Hut00] employed in this system uses a rich set of relations among theories to trace logical dependencies among mathematical objects and propagate/limit the effects of changes to the theories.

Our own MKM system MBase [FK00,KF01] is currently a member of the first class, but it can communicate with the Maya system via the joint interface language OMDoc [Koh00]. As an effect, MBase/Maya support theory management on the fragment of OMDoc that corresponds to the Maya development graph. In fact, in [KF01] we have proposed to distinguish two kinds of MBases, different in their data changing policies.

– An archive MBase which is epitomized by the Journal MBase MJ in our scenario below, it archives unchanging mathematical knowledge and is referenced by many other MBases.
– A scratch-pad MBase like the personal MBases MR and MR', that do not have any dependents and are primarily used for theory development.

To get a feeling of the requirements for the functionality addressed in this paper, let us take a look at a likely research communication scenario: We will first describe the communication pattern in a neutral way — say as it could have happened in the era of mathematics done with pen and paper (around 2001), and then model it using distributed MKM (about 2010).

classical, see Figure 1 Researcher R works on theory T together with his colleague R' at institute I. The theory T is a body of mathematics laid down in an article A published in journal J. Now, R extends theory T by a new definition D (say for a mathematical object O), proves a set P of theorems about O, and calls the resulting extended theory E. After that, R tells her colleague R' at I about D and P (say by circulating a memo in I), who gets interested and proves a set P' of useful properties of O. Together, R and R' put the theory E into final form F, and submit it to journal J. This accepts F and publishes it.

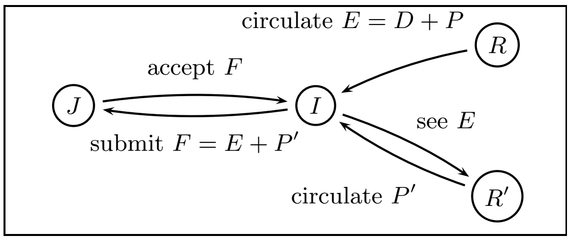

Fig. 1. Classical Research Cooperation

with MKM, see Figure 2 In 2005, the publisher of journal J has established an MBase server MJ for J which now contains theory T. Furthermore, the institute has its own departmental MBase MI and the researchers R and R' have the personal MBases MR and MR'. Now R develops the formalization

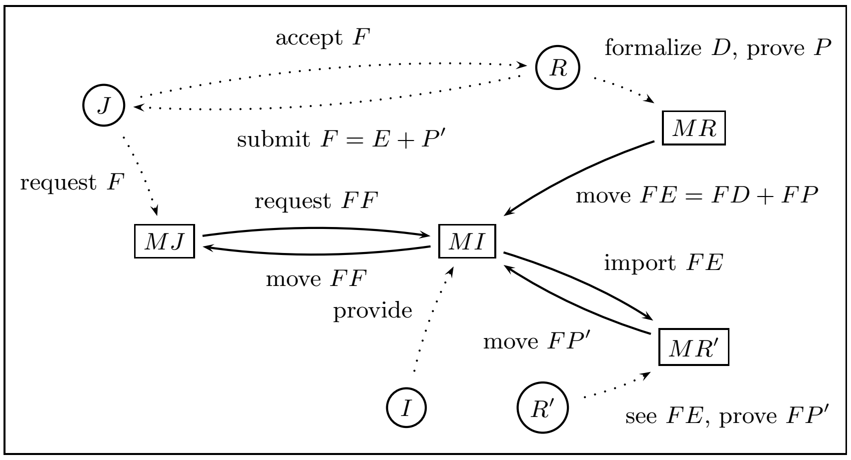

Fig. 2. Research Cooperation with distributed MKM

FD of O, stores it in MR and formalizes the set P of theorems by formalizing them and formally proving them[1] (yielding FP in MR). Instead of sending around an internal note about D and P in I, R moves their formalizations FD and FP into the institute MBase server MI, from where R' can import them into his personal MBase MR'[2]. On this basis R' formally proves FP', and adds it to theory FE, yielding FF the formal version of theory F. Then R and R' submit F to journal J, who evaluates it (possibly via his own

[1] To do so, R may need to revise the initial version of D several times in order to be able to prove the desired theorems (reproving the already obtained results that depended on a previous version of D every time). This process is supported by MBase/Maya based on techniques presented in [AHMS02]

[2] Alternatively, R could leave FD and FP in MR and tell R' personally about them, allowing him to import them from MR into MR'; but this is a matter of institute policy, which we will not address here.

personal MBase) it and finally accepts F. To publish F on MJ, it requests FF from MI, which moves it there.

1.1 Contribution of this paper

As we have seen in the scenario above, a strict division into archive and scratch-pad knowledge bases is unrealistic, since it does not reflect the current and anticipated nature of scientific communication and publication: Collaboration and theory change occurs at every level and should be supported by an infrastructure for collaborative content management and version control which enables multiple users to work jointly on mathematical theories with minimal interference.

We will develop a general architecture for a collaborative content management extending the CVS architecture and specialize it for mathematical knowledge by taking into account the structure of mathematical documents. For the second task we will build on both the work on structural `diff/patch/merge` utilities in Xml, as well on the semantic management of change in the Maya system [AHMS02].

Even though the work reported in this paper is motivated by the MBase system, it is much more general, since it only depends on the communication format used by the system. The methods are not even specific to the OMDoc format, we will only assume that the knowledge base systems use a similar Xml-based format for communication and provide a way to re-create the original interface documents. This would for instance cover the the HELM system [APCS01], which employs a lightweight infrastructure based mainly on Xml documents and XslT stylesheets for MKM.

2 An Infrastructure for Managing distributed mathematical knowledge cooperatively

The proposed infrastructure for collaborative theory management is largely based on the CVS (**C**oncurrent **V**ersions **S**ystem [CVS]) architecture. This system is widely used to support collaborative software development, since it combines software versioning with controlled concurrent access to the resources under CVS control. We will briefly review the basic notions of CVS, and describe our multi-level architecture with reference to it.

2.1 Cooperative Version Control in CVS

CVS is a server-based system for concurrent version control, used mainly for software development. The CVS server provides a so-called CVS repository R, which keeps a representation of all committed versions (called **revision**s) of the software together with logging information.

A CVS client C can then **check out** a **working copy** of the software and work on it. Let us for simplicity assume that C checks out the most recent revision in the repository, the so-called **head**. After completing the development task, C can **commit** the changes Δ to the repository, creating a new (current) revision in the repository. She will usually accompany the commit with a short description of the changes; this is also logged in the repository, eventually adding up to a changelog for the software development.

It is a distinguishing feature of CVS that the repository is not locked when a working copy is checked out. So another client C' can also have active working copies of the software and work on them. When C commits, the working copy of C' which was based on the (old) head, is no longer **up to date** with the repository. As a consequence, the changes C' has made to the software cannot be committed to the repository. C' can not simply check out a new working copy from R, since she would lose her work; therefore (upon C's request) CVS merges the changes Δ into C's working copy to keep it up to date with respect to the head of R. Now (after resolving any conflicts introduced by the merge) C' can commit her changes to the repository. Even though conflicts can occur in the merging operation, they are sufficiently infrequent in practice.

We have seen above that version control in CVS protocol is based on the computation, communication and management of differences (changes) to files. CVS uses the `unix` utilities

`diff` for determining the changes in a working copy to be committed to the repository
`patch` for updating old revisions
`merge` for merging changes into a working copy to keep it up to date with the repository.

To facilitate the functionality described above, the CVS server represents committed non-head revisions of files internally as reverse `diff`s from the head revision (which is stored explicitly). Thus the head revision can be served immediately, whereas older revisions can be computed by applying the respective reverse diffs. In this model, a version can be represented as a specific sequence of transformations (edit scripts).

2.2 A multi-level Client/Server Architecture

CVS has a one-level client-server architecture, i.e. all the CVS clients can only communicate with a dedicated CVS server. In the distributed MKM settings like in Figure 2, we have a knowledge base MI that acts both as a repository for MR and MR' and as a client for MJ.

We will say that a knowledge base A is **downstream** from an knowledge base B, iff A is a CVS client of B or any knowledge base C that is downstream from B. The relation of being **upstream** is the converse relation to **downstream**. In Figure 2, MR, MR', and MI are downstream from MJ. Note that commit actions push information upstream and update actions pull information downstream.

A multi-level client-server architecture has inherent advantages: it can, for instance simulate CVS branching: In CVS a branch is used, if a set of clients want to make changes to software that are either too disruptive or too extensive for the usual update/commit cycle. In essence a branch acts as a virtual repository for the development and allows controlling revisions without disturbing the main development (the so-called **trunk**).

In our multi-level architecture, a branch in repository A for clients $S^1, \ldots, S^n$ can be simulated by creating a new knowledge base B downstream from A and upstream from the S^i. B is initialized by checking out a working copy from A, and the S^i can track their revisions in B and eventually commit the result to B. Closing the branch corresponds to deleting the knowledge base B and updating the S^i.

2.3 An Atomized Version Control Relation

The CVS protocol is based on the file system hierarchy for grouping and anchoring user interaction. For instance, update and commit commands issued without reference to a particular file will be applied to all registered files in the current directory.

The file system hierarchy is replaced with a document-centered (given by `omgroup` in the OMDoc representation) or semantic hierarchies (given by theories or development graphs). The notion of a file (or equivalently of an OMDoc document) is only a secondary concept — if present at all — in the conceptual hierarchy of mathematical knowledge management systems. In particular, the level of a file is not the lowest level of an object under version control. This role is taken up by the notion of a mathematical object represented by a top-level OMDoc element. As a consequence, the client/server relation is atomized to mathematical objects instead of files. We speak of the **version control relation** that relates working copies of mathematical objects with their repository instances. Of course this relation must be acyclic.

Just as a file system can contain working copies from multiple repositories, a knowledge base can contain objects that are working copies checked out different repositories, though for each mathematical object, the version control relation is a tree, i.e. every object has at most one server it can be committed to and updated from. Intuitively, a math object is — for the parent element — as a file for a directory; and files have attributes like creation time, modification time, permissions; so should the math objects, which can be stored in the Dublin Core metadata of OMDoc elements.

2.4 Interaction of Version Control with Distribution and Knowledge Base Consistency

In [KF01] we have identified four tasks necessary for distributing mathematical knowledge bases: caching, moving, changing, and deleting mathematical objects. Before we give them interpretations in our architecture, let us re-examine the assumptions we based the analysis on; they include (paraphrased):

A3 all mathematical elements have a unique "**defining**" realization in the network of knowledge bases.

A4 mathematical objects are never changed.

Assumption **A3** is directly related to distribution: every object has a unique description: a pair consisting of the URL of the knowledge base and the unique identifier of the object there. All other copies of the object are just cached copies of it.

Assumption **A4** was useful for distribution, since it makes caching and maintenance very simple. Relaxing **A4** — which is the task at hand in this paper — has two aspects: How do we ensure consistency in situations where e.g. a definition or theorem that other mathematical objects depend on are changed in mathematically significant ways[3]. We will not deal with this problem here, since it is already studied in great detail in the development graph model [Hut00].

The question we will address in this paper is purely at a protocol level: it can largely be framed in terms of the interaction between **A3** and version control. We will study this with respect to the three distribution tasks identified in [KF01].

Caching Mathematical Objects: We used assumption **A4** to allow trivial caching. In the new architecture, we identify the caching relation to be the version control relation: to cache a copy of a mathematical object, it is simply checked out from the repository as a working copy. Note that objects that are working copies can never be defining instances of mathematical objects in our model. In the new model cache-consistency is a well-understood problem, since an object can always be updated from its repository. The ensuing conflicts can be resolved by the standard three-way merge methods described e.g. in [Lin01].

Moving Mathematical Objects One of the most basic procedures is that of moving objects between knowledge bases, e.g. of the theory FF from MI to MJ after the submission described in our scenario. This action can be modeled by adding FF as a defining instance to MJ, deleting FF in MI, and checking out FF from MJ to MI, which acts as a CVS client for MJ for this object. Note that with this construction, we can only move mathematical objects upstream, which is the natural direction.

Deleting and Changing Mathematical Objects: Since we leave the question of maintaining knowledge base consistency to the development graph techniques which entail re-examining mathematical objects that depend on the changed ones, augmenting the "pull" technology of our CVS-like architecture with a "push" component seems advantageous. Note that mathematical objects are always upstream from ones that logically depend upon them. Therefore a knowledge base $\mathcal{M}$ keeps a record of all the upstream knowledge bases, so that these can be notified of any changes and trigger propagation of the change. Apart from notification of dependents this information can be used for optimizations like the following: Whenever $\mathcal{M}$ moves the defining instance of an object $\mathcal{O}$ to

[3] Of course changes like correcting typos or changing explanatory text are unproblematic from a consistency point of view.

some knowledge base $\mathcal{M}''$, then it can send the new location of $\mathcal{O}$ to all upstream knowledge bases, asking them to update their reference objects and thus shielding itself from future requests to $\mathcal{O}$.

3 Computing Differences and Managing Change

In this section we will describe the computational utilities underlying our collaboration architecture. CVS uses the line-based `diff/patch/merge` utilities to compute differences between versions, update files, and merge differences into modified working copies. In applications like ours, where we know more about the structure of the data, we can do better, and arrive at more compact, less intrusive edit scripts[4]. For instance, if we know that whitespace carries no meaning in a document format, two documents are considered equal, even if they differ (with respect to the distribution of whitespace characters) in every single line; as a consequence, the computed difference would be empty.

We will look at different document models and their impact on computing differences between documents in this section. Before we do this, let us briefly clarify what we mean by a document model by comparison to mathematical models. In mathematics, when we define a class of mathematical objects (e.g. vector spaces), we have to say which objects belong to this class, and when they are to be considered equal (e.g. vector spaces are equal, iff they are isomorphic). For document models, we do the same, only that the objects are documents. XML supports the first task by allowing us to specify a document type definition (DTD) or an XML Schema, which can be used for mechanical document validation, but leaves the second to be clarified in the (informal) format specifications.

Listing 1. An OMDoc definition.

```
<definition id="comm-def" for="comm">
  <CMP xml:lang="en">An operation <OMOBJ id="op"><OMV name="op"/></OMOBJ>
     is called commutative, iff
    <OMOBJ id="comm1">
      <OMA><OMS cd="relation1" name="eq"/>
        <OMA><OMV name="op"/><OMV name="X"/><OMV name="Y"/></OMA>
        <OMA><OMV name="op"/><OMV name="Y"/><OMV name="X"/></OMA>
      </OMA>
    </OMOBJ> for all <OMOBJ id="x"><OMV name="X"/></OMOBJ>
    and <OMOBJ id="y"><OMV name="Y"/></OMOBJ>.
  </CMP>
  <CMP xml:lang="de">
    Eine Operation <OMOBJ xref="op"/> heißt kommutativ, falls
    <OMOBJ xref="comm1"/> für alle <OMOBJ xref="x"/> und <OMOBJ xref="y">.
  </CMP>
```

[4] Compactness of edit scripts is important for storage and query efficiency in MKM systems, while minimal intrusiveness (patching does not disrupt document structure) is important for humans to track and understand changes.

```
</definition>
```

Of course, the stronger the equality modulo which differences are computed, the better the edit scripts become. The conceptual core of the MBASE data model is given by the OMDOC format [Koh00,OMD], which is also used as an interface representation for communication between MBASEs and their clients. We will base our discussion in this section concretely on the OMDOC document model, building up to it by discussing the underlying XML document model. We will discuss generalizations to other document formats for MKM in section 3.4.

Let us call two documents $\mathcal{M}$-equal, iff they are equal with respect to the document model $\mathcal{M}$, analogously we will call an algorithm an $\mathcal{M}$-`diff` algorithm, iff it computes differences modulo $\mathcal{M}$-equality. In the rest of this section, we will use the OMDOC element in Listing 1 as a running example.

3.1 Using the tree structure of XML Documents

As OMDOC is an XML application, we can make use of the generic tree structure of XML documents. For instance, XML specifies that the order of attribute declarations in XML elements is immaterial, double and signle quotes can be used interchangeably for strings, XML comments (<!−−...−−>) are ignored, and whitespace characters in the UniCode serialization is only meaningful in text nodes. As a consequence, the serialization in Listing 2 is XML-equal to the one in Listing 1, but not to the one in Listing 4.

Listing 2. An XML-equal serialization for Listing 1

```
<definition for="comm"                    id="comm-def" >
  ...
 <CMP xml:lang='de'> <!-- note the unabbreviated empty element -->
   Eine Operation <OMOBJ xref="op"></OMOBJ> heißt kommutativ, falls
  <OMOBJ xref='comm1'/> für alle <OMOBJ xref="x"/> und <OMOBJ xref='y'>.
 </CMP>
</definition>
```

There is a large body of work on using the XML tree structure to compute differences of XML documents modulo XML-equality (see e.g. [WDC02]). The algorithms (see [CRGMW96] for an introduction) compute partial tree matchings[5] and express these as so-called “edit scripts” that add and delete XML elements and attributes in the source tree to arrive at the target tree. The work has been mainly concerned with finding algorithms for optimal (least-cost) edit scripts and complexity issues. Formats like `XUpdate` [LM00] (see Listing 3 for an example) use XPATH [Cla99] expressions to identify the elements the instructions act upon.

The central problem of finding corresponding nodes in trees critically depends on the notion of tree-similarity employed. If the document is strongly keyed (e.g.

[5] Which nodes correspond to each other modulo a given notion of tree similarity?

all elements have unique ID attributes, which cannot be changed by the user[6] or the knowledge management system employs some node numbering system like the one proposed in [CTZZ01]), then the key structure gives a very natural notion of node correspondence, and differencing becomes relatively simple. For the un-keyed case, only the notion of structural isomorphism and of ordered and un-ordered trees has been considered e.g. in [CRGMW96].

Listing 3. An XUpdate edit script (partly) updating Listing 1 to Listing 4

```
<xu:modifications xmlns:xu="http://www.xmldb.org/xupdate">
  <xu:variable name="c" select="definition/CMP[0]/OMOBJ[@id='comm1']"/>
  <xu:remove select="definition/CMP[0]/OMOBJ[@id='comm1']/@xref"/>
  <xu:append select="definition/CMP[0]/OMOBJ[@id='comm1']" child="1">
    <xu:value-of select="$c"/>
  </xu:append>
  <xu:remove select="definition/CMP[0]/OMOBJ[@xref='comm1']/*"/>
  <xu:update select="definition/CMP[0]/OMOBJ[@xref='comm1']/@xref">
   <xu:value-of select="'comm1'"/>
  </xu:update>
</xu:modifications>
```

3.2 The OMDoc Document Model

Let us now take a look at how the OMDoc document model can be used for more *semantic* differencing (OMDoc-diff[7]).

Listing 4. An OMDoc-equal representation for Listings 1 and 2

```
<definition id="comm-def" for="comm">
  <CMP xml:lang="de">Eine Operation <OMOBJ xref="op"/> heißt kommutativ, falls
    <OMOBJ id="comm1">
      <OMA><OMS cd="relation1" name="eq"/>
        <OMA><OMV name="op"/><OMV name="X"/><OMV name="Y"/></OMA>
        <OMA><OMV name="op"/><OMV name="Y"/><OMV name="X"/></OMA>
      </OMA>
    </OMOBJ> für alle <OMOBJ xref="x"/> und <OMOBJ xref="y">.
  </CMP>
  <CMP xml:lang="en">An operation <OMOBJ id="op"><OMV name="op"/></OMOBJ>
    is called commutative, iff <OMOBJ xref="comm1"/> for all
    <OMOBJ id="x"><OMV name="X"/></OMOBJ> and
```

[6] The action of changing keys in the data, can lead to un-intuitive and computationally sub-optimal edit scripts, but does not compromise the method per se.

[7] Note that we are *not* proposing to use mathematical equality here, which would make the formula $X + Y = Y + X$ (the OMOBJ with id="comm1" in Listing 4 instantiated with addition for op) mathematicallly equal to the trivial condition $X + Y = X + Y$, obtained by exchaning the right hand side $Y + X$ of the equality by $X + Y$, which is mathematically equal (but not OMDoc-equal).

```
    <OMOBJ id="y"><OMV name="Y"/></OMOBJ>.
  </CMP>
</definition>
```

The OMDoc document model extends the XML document model in various ways. For instance[8], the order of `CMP` children of an `omtext` element does not matter, and the distribution of whitespace is irrelevant even in text nodes. More generally, as OMDoc documents have both formal and informal aspects, they can contain *data-set-based* as well as *document-structured* information. At one extreme an OMDoc document contains a formalization of a mathematical theory, as a reference for automated theorem proving systems. There, logical dependencies play a much greater role than the order of serialization in mathematical objects. We call such documents **data set based** and specify the value `DataSet` in the `Type` element of the OMDoc metadata for such documents. On the other extreme we have human-oriented presentations of mathematical knowledge, e.g. for educational purposes, where didactic considerations determine the order of presentation. We call such documents **document-structured** and specify this by the value `Text`. Note that since OMDoc allows to specify Dublin Core metadata [WKLW99] at many levels, document-structured and data set based parts can interleave in the same document, allowing OMDoc-`diff` algorithms to take this into account.

Moreover OMDoc uses a variant of OpenMath objects [CC98] that can be represented as directed acyclic graphs (DAGs; using `ID`/`IDREF` links) rather than regular trees: an empty element with an `xref` attribute is OMDoc-equal to the element that carries the corresponding `id` attribute. As a consequence, the representations in Listings 1 and 2 are OMDoc-equal to the one in Listing 4, and an OMDoc-`diff` algorithm must generate the empty edit script between all three, while an XML-`diff` algorithm should generate an extension of the `XUpdate` script in Listing 3.

In particular, the process of exploding the DAG to a tree representation or sharing a tree to a DAG should not result in a difference computation. The same applies to the OMDoc representation of proofs, where an additional level of structure sharing is possible. A case where the underlying structure of the data is not tree-like, that is, not based on structure-sharing, is the development graph itself, which can even be cyclic. Here, first steps for defining a correspondence relation and for determining changes have been taken in [AHMS00] and implemented in the Maya system.

3.3 Challenges for OMDoc-`diff` Algorithms

As we have shown, taking advantage of OMDoc-equality in computing differences leads to more concise edit-scripts, which is essential in an environment where document processing applications manipulate mathematical content by acting on internal data structures and generate target documents from these. In

[8] As an introduction to the OMDoc format is beyond the scope of this paper, we will assume a basic knowledge of [Koh00] and the material at [OMD].

such situations, it is impossible to predict which of the possibly many OMDoc-equal representations will be generated. Since in a CVS-like collaborative protocol any `diff` can lead to a conflict that will require human intervention for resolution, the availability of such algorithms will be crucial for adoption.

Of course extending XML-equality to OMDoc-equality in computing differences breaks the underlying assumptions of the algorithms described in section 3.1. For instance, the DAG-nature of OMDoc documents requires the differencing algorithms to (virtually) expand the objects to tree form while processing them[9].

It seems that techniques from [BKTT02] can be used to get around the obvious computational difficulties involved in differencing modulo equality. [BKTT02] trivialize the tree matching problem by assuming that all tree representations are "strongly keyed", employing a generalized notion of data base keys to determine element correspondence in XML documents. They claim that sensible data formats are almost always strongly keyed up to data in XML text nodes. We have not verified this for OMDoc yet, but for instance even though `CMP` nodes do not have `ID` attributes, they are keyed, since they have `xml:lang` attributes, which must be unique among their siblings. However, `CMP` content however is not keyed, since it is generic text data (which is trivially un-keyed) mixed with representations of mathematical object represented as content MathML or OpenMath objects (this also caused some addressing problems in the `XUpdate` script in Listing 3). Note that in OMDoc documents managed by MKM systems (as opposed to directly written by hand), the OMDoc `mid` attributes can be used for keying, alleviating the higher computational costs of the un-ordered algorithms somewhat.

Obviously, we need a combination of the XML tree-based un-keyed algorithms with key-sensitive techniques for our application; such algorithms have been requested, but to the author's knowledge not been reported on so far.

3.4 Modular $\mathcal{M}$-`diff` Algorithms

Given that most of the OMDoc document model is rather standard (DAGs vs. trees, sets vs. lists of children, etc.), it is appealing to develop general $\mathcal{M}$-`diff` algorithms, where the notion of $\mathcal{M}$-equality is specified externally, e.g. by extending the document schema to a full document model.

Note that XML Schema so far only specifies full document models (i.e. including equality) for so-called **data types** (e.g. "`100`" and "`1.0E2`" are equal as members of the data type `float`). Thus we could define the notion of XML-Schema-`diff`, which would take these into account, but this is only marginally relevant for our problem here, since it only concerns the leaves of the trees we are dealing with.

Listing 5. Specifying Order in XML Schema using `xs:appinfo`

```
<xs:complexType name="omtextType">
```

[9] In the file system metaphor, this would correspond to following symbolic links

```
    ...
    <xs:sequence>
      <xs:annotation><xs:appinfo><mdiff:unordered/></xs:appinfo></xs:annotatic
      <xs:element name="CMP" type="inCMPtype" maxOccurs="unbounded"/>
      <xs:element name="FMP" type="FMP" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    ...
  </xs:complexType>
```

A more promising avenue seems to be to make use of the `xs:appinfo`[10] element to specify document models for complex types in XML Schemata — as opposed to just content models for validation. Based on the examination of the OMDoc document model in section 3.2, it seems plausible to assume that we could go a long way by specifying

document order e.g. by an element `mdiff:unordered` in Listing 5, and
link semantics e.g. as in Listing 6 where we specify that the `xref` attribute of an OpenMath object means that it represents a copy of the object that carries the corresponding `id` attribute.

Listing 6. Specifying DAG attributes in XML Schema using `xs:appinfo`

```
<xs:attributeGroup name="DAG.attrib">
  <xs:attribute name="xref" type="xs:anyURI" use="optional">
    <xs:annotation><xs:appinfo><mdiff:dag-source/></xs:appinfo></xs:annotatior
  <xs:attribute>
  <xs:attribute name="id" type="xs:ID" use="optional">
    <xs:annotation><xs:appinfo><mdiff:dag-target/></xs:appinfo></xs:annotatior
  <xs:attribute>
</xs:attributeGroup>
```

So, if an XML document (fragment) is an instance of a schema that containts document model specifications like the ones in Listings 5 and 6, then a modular `diff` algorithm can read the schema and customize — multiple times during the parsing process if necessary — the comparison criteria used by the algorithm.

4 Conclusion

We have laid down first ideas for a collaborative version control model for MKM systems, based on the OMDoc format. We have sketched the overall architecture, and determined some of the requirements for OMDoc-`diff` algorithms that come from the respective structural invariants of the data in MKM systems.

We have seen that the architecture can be kept quite close to that of the well-known CVS system[11], and interacts well with the requirements for distribution

[10] The `xs:appinfo` is introduced in XML Schema expressly for such purposes.

[11] Actually, [BKTT02] propose a repository organization that is not `diff`-based, which would be interesting to experiment with, but integrating it into a *collaborative* version control environment is not trivial

identified in [KF01], which is encouraging from an implementation point of view. In particular, we are currently experimenting with the idea to annotate all information necessary for a CVS-like file-based formalism in the `metadata` elements of mathematical objects. We could for instance use the existing Dublin Core `Date` and `Identifier` element for timestamping, and keeping version information. Further information, such as pointers to the repository in working copy objects can be kept in the `metadata/extradata` element provided by OMDoc expressly for this purpose. We will experiment with a HELM [APCS01]-like setup based on OMDoc files on web-servers and implement merging by server-side XslT processing.

The main item for further research is an OMDoc-`diff` algorithm as described in section 3.2. In the literature on version management in Xml, we often hear the argument that difference-computation is not needed in practice, since documents are generated by Xml structure editors, but this only moves the burden from an independent postprocess (implement once) to a module in every editor. Moreover, this would penalize authors for using general Xml editors, since they could only incorporate Xml-`diff` algorithms. Finally, the actual editing process employed by the user may not correspond to the optimal edit script.

Given a good difference computation algorithm, merging can be obtained by relatively simple extensions, especially since our CVS-like architecture allows the usage of the so-called three-way merge (see [Lin01]), where two revisions are compared with respect to a known base revision, from which they have been created. Here, edit scripts for the changes from the base can be computed for both revisions. These can be analyzed and combined to a joint edit script which updates the base revision to the merged revision. [Man01,Lin01] present algorithms for three-way merge of Xml documents and there are even commercial implementations (e.g. the one described in [LF02]). Since the merge operation only depends on the edit scripts which act on the generic Xml structure, and not on the particular structure of the OMDoc format, we can use these algorithms and implementations off the shelf.

Acknowledgements The first author was supported by a Heisenberg stipend of the German Research Agency (DFG). The authors would like to thank Serge Autexier, Andreas Franke, Dieter Hutter, and Bernd Krieg-Brückner for stimulating discussions in the early stages of this work.

References

[ABC$^+$02] Stuart Allen, Mark Bickford, Robert Constable, Richard Eaton, Christoph Kreitz, and Lori Lorigo. FDL: A prototype formal digital library – description and draft reference manual. Technical report, Computer Science, Cornell, 2002. `http://www.cs.cornell.edu/Info/Projects/NuPrl/html/FDLProject/02cucs-fd%l.pdf`.

[AHMS00] Serge Autexier, Dieter Hutter, Heiko Mantel, and Axel Schairer. Towards an evolutionary formal software-development using CASL. In C. Choppy and D. Bert, editors, *Proceedings Workshop on Algebraic Development Techniques, WADT-99*. Springer, LNCS 1827, 2000.

[AHMS02] Serge Autexier, Dieter Hutter, Till Mossakowski, and Axel Schairer. The development graph manager MAYA (system description). In Hélene Kirchner, editor, *Proceedings of 9th International Conference on Algebraic Methodology And Software Technology (AMAST'02)*. Springer Verlag, 2002.

[APCS01] Andrea Asperti, Luca Padovani, Claudio Sacerdoti Coen, and Irene Schena. HELM and the semantic math-web. In Paul B. Jackson Richard. J. Boulton, editor, *Theorem Proving in Higher Order Logics: TPHOLs'01*, volume 2152 of *LNCS*, pages 59–74. Springer, 2001.

[BKTT02] Peter Buneman, Sanjeev Khanna, Keishi Tajima, and Wang Chiew Tan. Archiving scientific data. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2002.

[CC98] Olga Caprotti and Arjeh M. Cohen. Draft of the Open Math standard. The Open Math Society, `http://www.openmath.org`, 1998.

[CIMP01] David Carlisle, Patrick Ion, Robert Miner, and Nico Poppelier. Mathematical Markup Language (MathML) version 2.0. W3C recommendation, World Wide Web Consortium, 2001. Available at http://www.w3.org/TR/MathML2.

[Cla99] XML path language (XPath) Version 1.0. W3C recommendation, The World Wide Web Consortium, 1999. available at `http://www.w3.org/TR/xpath`.

[CRGMW96] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change detection in hierarchically structured information. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 493–504, 1996.

[CTZZ01] Shu-Yao Chien, Vassilis J. Tsotras, Carlo Zaniolo, and Donghui Zhang. Storing and querying multiversion XML documents using durable node numbers. In *Proc. of The 2nd International Conference on Web Information Systems Engineering (WISE)*, 2001.

[CVS] Concurrent versions system: The open standard for version control. Web site at `http://www.cvshome.org`.

[FK00] Andreas Franke and Michael Kohlhase. System description: MBase, an open mathematical knowledge base. In David McAllester, editor, *Automated Deduction – CADE-17*, number 1831 in LNAI, pages 455–459. Springer Verlag, 2000.

[Hut00] Dieter Hutter. Management of change in structured verification. In *Proceedings Automated Software Engineering (ASE-2000)*. IEEE Press, 2000.

[KF01] Michael Kohlhase and Andreas Franke. MBase: Representing knowledge and context for the integration of mathematical software systems. *Journal of Symbolic Computation; Special Issue on the Integration of Computer algebra and Deduction Systems*, 32(4):365–402, 2001.

[Koh00] Michael Kohlhase. OMDoc: An open markup format for mathematical documents. Seki Report SR-00-02, Fachbereich Informatik, Universität des Saarlandes, 2000. `http://www.mathweb.org/omdoc`.

[LF02] Robin La Fontaine. Merging XML files: A new approach providing intelligent merge of xml data sets. In *XML Europe 2002 - Conference Proceedings*, 2002.

[Lin01] Tancred Lindholm. A 3-way merging algorithm for synchronizing ordered trees – the 3DM merging and differencing tool for XML. Master's thesis, Helsinki University of Technology, 2001. `http://www.cs.hut.fi/~ctl/3dm/`.

[LM00] Andreas Laux and Lars Martin. `XUpdate` - XML update language. Working Draft of the XML:DB Initiative, 2000. `http://www.xmldb.org/xupdate/xupdate-wd.html`.

[Man01] Gerald W. Manger. A generic algorithm for merging SGML/XML-instances. In *XML Europe 2001 - Conference Proceedings*, 2001.

[OMD] The OMDoc repository. web page at `http://www.mathweb.org/omdoc`.

[WDC02] Yuan Wang, David J. DeWitt, and Jin-Yi Cai. X-Diff: An effective change detection algorithm for XML documents, 2002. Submitted, `http://www.cs.wisc.edu/~yuanwang/xdiff.html`.

[WKLW99] S. Weibel, J. Kunze, C. Lagoze, and M. Wolf. Dublin Core Metadata Element Set, Version 1.1: Reference Description. DCMI Recommendation, 1999. `http://dublincore.org/documents/1999/07/02/dces/`.