Versioned Links

Andrea Kohlhase[1] and Michael Kohlhase[2]

[1] German Research Center for Artificial Intelligence (DFKI)
Andrea.Kohlhase@dfki.de
[2] Computer Science, Jacobs University Bremen
m.kohlhase@jacobs-university.de

Abstract. Recently consistency in mutable knowledge collections can be supported by change management systems that draw on the specified semantics for knowledge objects and their relations. But even with machine support a seemingly minor change can easily cascade into a major adaptation task. In this paper we argue that mutable knowledge collections can be supported by **versioned links**: Links as first-class elements defined by a triple of *versioned* elements (`subject`/`predicate`/`object`). The main idea explored here is that changes *need not* be propagated to linked elements if those still reference the originally linked object. We give a model for versioned links that is easy to embed in existing MKM systems.

1 Introduction

MKM formats explicitly represent relations between objects to compute related objects and predict the way changes affect them; see [Hut09,Mül10,AM10] for recent progress in this field. Nevertheless, this process can cascade a seemingly minor change into a major adaptation task if aiming at overall consistency as it is e.g. suggested in [GS07].

In contrast, we suggest to aim for local consistency. The purpose of local consistency is to avoid situations, where the *meaning* or *functionality* of an object changes unintentionally due to a change in an object it references; e.g. a mathematical theorem may be invalidated, if the definition of a concept it uses is changed. The main idea explored in this paper is that changes *need not* be propagated to linked objects but can continue to reference the originally linked object. Intuitively, the original reference to the definition insulates the theorem against the change. Thus, we like to introduce a solution to the "**late binding problem**"[3] (see Section 2). To enable the conservation of local consistency, we explore in this paper the new concept of '*versioned links*' (introduced in Section 3). We assume that document collections in MKM are stored in version control systems as a concession to change in knowledge management processes, that induce a notion of 'versioned objects'. Next we elevate the relations between

[3] This is called in analogy to the technique of "late binding" (also called dynamic/name binding) in programming languages, where the method being called upon an object is looked up by its name at runtime.

versioned objects to first-class citizens and obtain versioned links as triples (`subject/predicate/object`). Naturally, they are versioned as well, so that links and their components carry an independent revision number. Section 4 concludes the paper.

2 Versioning and Late Binding Problems

Late binding problems are well-known in systems that reuse objects, ranging from *i)* library incompatibilities in software systems (also known as "DLL hell") over *ii)* version conflicts of shared macro packages in LaTeX document collections and *iii)* hyperlinks in Wikipedia to *iv)* the regressions in theorem prover libraries as tactics and lemmata evolve.

In Software Engineering the late binding problems in *i)* are treated by links to *versioned packages*. Packages are collections of inter-related files intended for re-use in other packages, their internal references are kept in sync. When a new version of a package is released, dependent packages may be ported to it. In the packages approach links are treated as package metadata and deployment problems are mitigated by specialized package management systems building on this metadata.

With respect to *ii)* in LaTeX the situation is somewhat different: Macros are commonly organized as packages, but used inside individual documents, so that a metadata-driven approach of package management systems is less useful. Therefore LaTeX supports in-document version metadata and links to them: The `\usepackage` macro allows an optional date argument: `\usepackage{foo}[2001/06/13]` loads the package `foo`, if the package date provided by the `\ProvidesPackage` statement in `foo.sty` is younger than June 13^{th}, 2001, and raises an error otherwise. We can see this behavior as a weak operationalization of versioned links in the absence of versioned package retrieval. Stronger operationalizations are available in some systems like MikTeX, which will update the installed packages. Note that the adequacy of this is ensured by the soft constraint that LaTeX packages are only supposed to grow monotonically in functionality. Note as well that these versioned links only cover the 'functionality aspect' of LaTeX seen as a software system with reusable macros. There is no provision for references to text fragments in or between documents, that is, inter-document references on the 'meaning aspect' are not available.[4] As a consequence change management in LaTeX document collections is virtually nonexistent. In other document formats the situation is similar.

In *Wikipedia*, which does have inter-document links and even an underlying versioning system, the situation with respect to *iii)* is uneven. On the one hand, Wikipedia strongly advises using versioned references for citing articles [Wik11], on the other hand all links between articles are unversioned (head links).

[4] Though it should be possible to build them into the LaTeX `\label`/`\ref` mechanism using similar ideas, e.g. in a framework like STeX, which already extends the `\label`/`\ref` mechanism by inter-document links [Koh10], possibly aided by the `svninfo` package to allow revisions instead of dates.

The situation of *theorem prover libraries* in *iv)* is more fine-granular, references occur deeply embedded in the mathematical structures — e.g. in proofs — and point to fine-grained objects — e.g., definitions or lemmata. To the best of our knowledge, none of the theorem prover libraries support versioned links or have package management systems, but rely on manual resolution of conflicts during regression testing, e.g. through institutions like the Mizar Library committee.

> "*Moreover, we need to make sure that dependencies are always consistent. Files in the database can depend on each other, sometimes in an indirect (transitive) way. First of all, we want to require all saved files to be valid (compilable); they can still contain incomplete proofs terminated with the Coq Admitted keyword or its equivalents for other provers. For a valid saved file we want to ensure that the current version remains valid after changes to the files it relies on. Some provers already include compatibility verification mechanisms. Coq stores the checksums of files to ensure binary compatibility between compiled proofs. To solve the problem, we have to consider the static and the dynamic approach. The dynamic approach is the convention that a file always refers to the latest versions of other files. It means that saving any change to a file will induce a costly recompilation of all files it depends on. Another problem is that changing definitions deep inside a library will make many developments incompatible and thus correct files will stop working. Saving only valid files does not solve this problem since the objects they contain (their interface) might be modified too. This approach also makes older versions of existing files immediately obsolete. The opposite approach is static linking, where a saved file always refers to the same version of other files. In other words, we never change a file, but rather add a new version of that file, with a fresh name. This means that the user will have to manually update the version number of files that are referred to if newer versions of those become available. The main advantage of this approach is that of integrity: provided you can safely assign new version numbers, you can enable concurrent access. Moreover, changing a file will never break another file. However, when changing a file deep in the library, one has to manually modify all the files in the dependency chain between that file and the files in which the changes should be reflected, which can sometimes be quite heavy. [CK07, p. 228]*" "*[...]We believe that the static approach is a more adequate way to store older (historical) versions of a given file, whereas up-to-date files should use the dynamic approach towards dependency. This way, older versions of files still make sense by statically referring to older versions of files they depend on. The latest versions can remain up-to-date with their immediate dependencies by being dynamically linked to them, i.e. recompiled when new versions of those files are saved. It might happen that such a file might not be valid anymore because of changes made to its dependencies: to keep validity we have to make it link statically to the suitable previous version. To help with this version compatibility issue, we propose a three-colour scheme: A file is labelled as red (i.e outdated) if it depends statically on an olderthan- latest version of another file. A file is labelled as yellow (i.e tainted) if it depends only on the latest versions*

of other files, and one or more of those files have a yellow or red status. Yellow status thus tracks the files which are indirectly lagging behind. A file is labelled as green (i.e. up-to-date) if it depends only on the latest versions of other files, and all those files are also labelled as up-to-date (green status). The separation between the yellow and red files comes from the fact that red files have to be manually updated to become green again (i.e. by creating a new version of them), whereas yellow files might be fixed by updating the red files that taint them. The switching to red status can be automated by rewriting Require statements on-the-fly to make them refer statically to the last suitable version of the file depended on. This means that fixing a red file can give red status to yellow files that it was tainting, thus pushing the problem upwards in the dependency tree. If the user wants to export a file together with its dependencies from the repository, a mechanism can be used to convert long file names (with version number) to short ones. The case might arise where a file would refer, directly or by transitivity, to an old version of itself. We can either forbid this or generate fresh file names using standard suffixing techniques." [ibd.]

To conclude this little survey of the use of versioned links in Software Engineering and MKM, we note that even though various versioning extensions of links are in use to solve late binding problems, (versioned) links are not treated as first-class objects.

3 MKM with Versioned Links

In this paper, we suggest to make use of two facts: *i)* Revision control systems (RCS) give access to old revisions, in particular, access to objects in old revisions to which other objects are consistently linked.[5] *ii)* The advent of versioned query interfaces like [ZK09,FFK11] enable access to versioned objects. In particular, the (platonic) concept of "the" object with identifier $\mathcal{O}$ is refined to the set of (concrete) objects $\mathcal{O}$ with distinct revision numbers, therefore links can point to such versioned objects.

To make the discussion more precise, we will now define the concepts involved more formally, starting out with a simplified version of the notion of *fs*-trees and version control systems developed in [Mül10], which we will review now.

3.1 *fs*-Trees and RCS Repositories

We will use *fs*-trees as a unifying notion of file system trees and semi-structured (XML) documents that abstracts from particular file system implementations and encodings.

In a nutshell, an ***fs*-tree** is an ordered, typed, labeled tree, whose edges are labeled with (directory/element) names and its leaves with strings (which either

[5] In this paper we assume a concept of global revisions as employed e.g. in the Subversion system [PCSF08]. There, any commit to the repository increments the revision number.

correspond to text files or text nodes in XML)footnoteThe original *fs*-trees had the notion of symbolic links and repository externals which we do not need here.. The node types distinguish nodes into directories, files, XML elements, and XML attributes, and carry constraints that make them faithful models of file systems and XML files.

The main property we will use in this paper is that any node/subtree in an *fs*-tree $\mathcal{T}$ can be addressed by a unique **name path**, i.e., a sequence $\pi = a_1/\cdots/a_k$ of names a_i. We write $\mathcal{T}/\pi$ for the *fs*-subtree of $\mathcal{T}$ rooted at π. Note that for a given *fs*-tree $\mathcal{T}$, a subtree $\mathcal{T}/\pi$ is either a directory, a file, or an XML fragment/subtree. Note furthermore, that name paths in *fs*-trees directly map to (file) URIs with XPath fragment identifiers. Thus any name path π is of the form δ/ρ, where δ is a **directory path** (i.e., a name path where all names are directory names) and ρ is an **fragment identifier** (i.e., a name path, where all names are XML element names); we call δ the **directory path of** π and ρ the **fragment identifier of** π.

If we denote the set of all *fs*-trees by $\mathcal{FS}$, we can represent a version control **repository** as a partial function $\mathcal{R}\colon \mathbb{N} \rightharpoonup \mathcal{FS}$ that maps **revision identifiers** (without loss of generality an initial segment of $\mathbb{N}$) to *fs*-trees. For a repository $\mathcal{R}$ and $n \in dom(\mathcal{R})$, where $dom(\mathcal{R})$ is the domain of $\mathcal{R}$, we call the *fs*-tree $\mathcal{R}(n)$ the **revision** n **of** $\mathcal{R}$; this notion extends to *fs*-subtrees: We say that a subtree of $\mathcal{R}(n)$ is **at revision** n. Finally, we say that $\mathcal{D}$ is a **document in** $\mathcal{R}$, if $\mathcal{D} = \mathcal{R}(n)/\delta$ for some directory path δ and revision n.

Given this vocabulary, the correspondence to mathematical knowledge management in the large can be seen as in Figure 1. We employ repositories to model the development of document collections over time, where the collection $\mathcal{C}$ at a concrete time point corresponds to a revision $\mathcal{R}(n)$ of the repository $\mathcal{R}$, and document collections, documents, and objects are realized as *fs*-tree fragments (subtrees of the revision) $\mathcal{R}(n)/\pi$. From now on we will use the concepts in Fig. 1 modulo the correspondence relation given by the dotted lines.

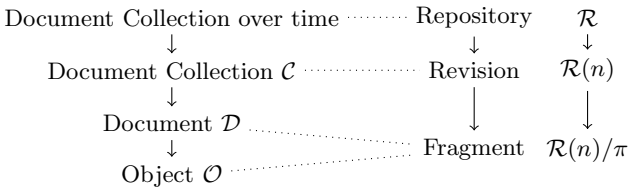

Fig. 1. A Realization of Document Collections over Time

3.2 Links in MKM Formats

Before we can define the concept "versioned link", we need to think about the status of links in MKM representation formats.

Definition 1. Let $\mathcal{T}$ be an *fs*-tree, then we define an (unversioned) **link in** $\mathcal{T}$ to be an RDF triple (see [MM04]) where `subject`, `predicate`, and `object` are

name paths in $\mathcal{T}$. We speak of a **file system link** if the `object` is a document or directory node, otherwise of a **fragment link**. In the latter case we distinguish **intra-document links**, where `subject` and `object` are in the same document, from **inter-document links** where they are not.

For the time being we will disregard links outside of a document collection $\mathcal{C} = \mathcal{R}(n)$ for some n; as all practical revision control systems encode file paths as URIs, our notion of links is a special case of RDF triples, if we assume that the `predicate`s are documented in the collection, which we can without loss of generality.

The set of links induced by a document collection $\mathcal{C}$ is determined by the representation format of the documents in $\mathcal{C}$. Instead of making this formal, we will appeal to the intuition of the reader by giving some examples:

i) `\input` statements in TEX/LATEX induce document links for the `predicate` "input", which tells the formatting engine to replace `\input{`⟪fileURI⟫`}` with a file referenced by ⟪fileURI⟫.

ii) `<a href="`⟪URI⟫`"`⟪attribs⟫`>`⟪link text⟫`</a>` in HTML induces a link for the `predicate` "display", which tells the browser to display the fragment referenced by ⟪URI⟫ in the browser when the user left-clicks ⟪link text⟫ (details specified by ⟪attribs⟫).

iii) `<proof for="`⟪URI⟫`">...</proof>` in OMDoc induces a link for the `predicate` "proves" whose `subject` is the `proof` object itself and whose `object` is the `assertion` element referenced by ⟪URI⟫. Note that the meaning of "proves" is not operational but given by the OMDoc ontology (cf. [Koh06] for the specification and [Lan11] for a formalization). In this case we even have a system that extracts all links from a document.

Note that all of these links rely on name paths in $\mathcal{C}$ (realized as URIs) for identification of resources (nodes in $\mathcal{C}$). Note furthermore, that all of these induce links whose `predicate` is pre-determined by the document format, i.e., given by the special syntax and induces a URI referencing a relation from the document ontology and whose `subject` is the resource containing the syntax that induces the link. We will call such link-inducing syntax in an MKM format $\mathcal{F}$ an **$\mathcal{F}$-reference**. Even though $\mathcal{F}$-references dominate in MKM formats, we will also cover proper links represented in any RDF representation format; they sometimes exist as standoff markup in MKM systems.

3.3 Versioned Links

In *fs*-trees it is straightforward to define versioned links.

Definition 2. Let $\mathcal{R}$ be a repository, $n \in dom(\mathcal{R})$ a revision identifier, and $\pi \in \mathcal{R}(n)$ a name path, then we call a pair $\langle \pi, n \rangle$ a **versioned name path** in $\mathcal{R}$.

Versioning systems usually reserve a special, intensional "revision identifier" for the respective youngest revision, which is called the **head revision** and denoted with $\uparrow$.

Definition 3. We call a versioned name path a **head path**, iff it is of the form $\langle \pi, \uparrow \rangle$ for some name path π.

Note that $\langle \pi, n \rangle$ identifies a resource in a repository $\mathcal{R}$. Building on this, we can finally define the concept of a versioned link.

Definition 4. For a given repository $\mathcal{R}$ we call an RDF triple a **versioned link in** $\mathcal{R}$, iff its `subject`, `predicate`, and `object` are versioned name paths in $\mathcal{R}$. **Versioned $\mathcal{F}$-references** are defined accordingly.

Note that versioned links generally involve four revisions: The revisions of the `subject`, `predicate`, and `object` as well as the revision of the link itself (e.g. given by the revision of the file that contains the representation of the RDF triple). For versioned $\mathcal{F}$-references this revision variety is restricted by their special syntactic structure. In particular, the revisions of `subject` and link are necessarily identical, and the revision of the `predicate` is given by the format $\mathcal{F}$, it is therefore uniform over the document. Note that this observation has an implication on the design of document formats: If we want to escape the version identifications of links, we need to use standoff links.

Definition 5. We call a versioned link a **head link**, iff all of its three versioned name paths are head paths. We call a versioned $\mathcal{F}$-reference a **head reference**, iff its `object` is a head path.

Definition 6. We call a versioned link an **inter-revision** link/reference, iff it involves at least two different revisions and an **intra-revision** link otherwise.

4 Conclusion

We have presented the concept of versioned links as a tool for managing change in mathematical document collections and knowledge repositories. Essentially the introduction of versioned links allows to move parts of the problem of consistency management in the exploration phase of MKM into one of coherence management in the codification phase. However, we contend that this allows for much more flexible and natural workflows.

We loosely built our discussion on the model of a *centralized RCS* like Subversion. At first glance, one may be tempted to think that *distributed RCS* (**DRCS**) like Git or Mercurial already support the practices versioned links are designed for (to get an overview of their differences see e.g. [O'S09]). Indeed, one can see and use each local repository in a distributed network as such an "island of consistency", and the practice of pulling changes from local repositories as a coherence management process. But note that this approach only supports the equivalent of *file-level versioned links* and is therefore too coarse-granular for mathematical knowledge which requires *object-level links*. Incidentally, programming languages mainly support file-level links, so DRCS fit the versioned packages development model in Software Engineering. We conjecture that in this case, a repository network is essentially isomorphic to a flattened repository with

versioned links. It seems possible to mimic versioned links in DRCS, if we are willing to break apart MKM documents into object-size files using an inclusion technique like XInclude, but this seems a larger intervention than the introduction of versioned links we propose. We used the centralized model in this paper, since we have the TNTBase system that offers efficient access to versioned XML objects, given a similar XML-fragment-access-enabled DRCS, studying the interaction of versioned links with distribution will probably lead to even more natural workflows.

References

[AM10] Serge Autexier and Normen Müller. Semantics-based change impact analysis for heterogeneous collections of documents. In Michael Gormish and Rolf Ingold, editors, *Proceedings of the 10th ACM symposium on Document engineering*, DocEng '10, pages 97–106, New York, NY, USA, 2010. ACM.

[CK07] Pierre Corbineau and Cezary Kaliszyk. Cooperative repositories for formal proofs. In Manuel Kauers, Manfred Kerber, Robert Miner, and Wolfgang Windsteiger, editors, *Towards Mechanized Mathematical Assistants. MKM/Calculemus*, number 4573 in LNAI, pages 221–234. Springer Verlag, 2007.

[FFK11] Ghislain Fourny, Daniela Florescu, and Donald Kossmann. A time machine for XML. Technical report, ETH Zürich, Switzerland, 2011. available at `http://www.dbis.ethz.ch/research/publications/timemachinexml.pdf`.

[GS07] Adam Grabowski and Christoph Schwarzweller. Revisions as an essential tool to maintain mathematical repositories. In Manuel Kauers, Manfred Kerber, Robert Miner, and Wolfgang Windsteiger, editors, *Towards Mechanized Mathematical Assistants*, volume 4573 of *Lecture Notes in Computer Science*, pages 235–249. Springer Berlin / Heidelberg, 2007.

[Hut09] Dieter Hutter. Semantic management of heterogeneous documents (invited talk). In *Proceedings of the Mexican International Conference on Artificial Intelligence (MICAI-2009)*, number 5845 in LNAI, pages 1–14. Springer, 2009.

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

[Koh10] Michael Kohlhase. `sref.sty`: Semantic crossreferencing in LaTeX. Self-documenting LaTeX package, Comprehensive TeX Archive Network (CTAN), 2010.

[Lan11] Christoph Lange. *Enabling Collaboration on Semiformal Mathematical Knowledge by Semantic Web Integration*. PhD thesis, Jacobs University Bremen, 2011. submitted January 31, defended March 11.

[MM04] Frank Manola and Eric Miller. RDF Primer. W3C Recommendation, World Wide Web Consortium (W3C), February 2004.

[Mül10] Normen Müller. *Change Management on Semi-Structured Documents*. PhD thesis, Jacobs University Bremen, 2010.

[O'S09] Bryan O'Sullivan. Making sense of revision-control systems. *Communications of the Association for Computing Machinery (CACM)*, 52(9):57–62, 2009.

[PCSF08] C. Michael Pilato, Ben Collins-Sussman, and Brian W. Fitzpatrick. *Version Control With Subversion*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2 edition, 2008.

[Wik11] Wikipedia. Citing wikipedia — Wikipedia, the free encyclopedia, 2011. [Online; accessed 05-Jan-2011].

[ZK09] Vyacheslav Zholudev and Michael Kohlhase. TNTBase: a versioned storage for XML. In *Proceedings of Balisage: The Markup Conference 2009*, Balisage Series on Markup Technologies. Mulberry Technologies, Inc., 2009. available at `http://kwarc.info/vzholudev/pubs/balisage.pdf`.