Maintaining Islands of Consistency via Versioned Links

Andrea Kohlhase
Jacobs University Bremen
Campus Ring 1
28759 Bremen, Germany
a.kohlhase@jacobs-university.de

Michael Kohlhase
Jacobs University Bremen
Campus Ring 1
28759 Bremen, Germany
m.kohlhase@jacobs-university.de

ABSTRACT

One of the core tasks of technical communication and knowledge management is maintaining the internal and external consistency of document collections. The design of (technical) communication infrastructures has to take this into account from the start. Consistency of static collections is enforced by format constraints (e.g. specified in a schema and validated grammatically). Recently, consistency in mutable knowledge collections can be supported by change management systems, that draw on 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 the practice of maintaining "islands of consistency" in 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. With this concept a major adaptation task can be put under user-friendly impact management. We give a model for versioned links that is easy to embed in existing systems and show how this concept supports impact management workflows.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods

General Terms

Documentation, Theory, Management

Keywords

Change Management, Consistency Management, Knowledge Management, Links, References

SIGDOC'11, October 3–5, 2011, Pisa, Italy.

1. INTRODUCTION

The generally accepted model for knowledge management (KM) assumes that explicit knowledge is represented in some form of document collections, which we will call **knowledge base**s. These knowledge bases range from a project's collection of documents in various formats as used in [13], over semantic DITA [5] files, to organizational wikis as suggested in [19]. It is an implicit assumption that each knowledge base $\mathcal{K}$ is **consistent**, i.e., satisfies validity conditions of the representation format and additional semantic conditions. In particular, knowledge in $\mathcal{K}$ may not lead to contradictions. In a semantic format like DITA some of the integrity conditions involved in determining consistency may be expressed in $\mathcal{K}$ in the form of links between document fragments. In contrast, a knowledge base $\mathcal{K}$ is a **correct** description of a situation $\mathcal{S}$, iff all statements that can be derived from $\mathcal{K}$ hold in $\mathcal{S}$. The consistency assumption for $\mathcal{K}$ is largely independent of the notion of correctness, since it cannot be defined as a property of $\mathcal{K}$ alone. Clearly, correctness is a very strong desideratum for a knowledge base, but consistency is a precondition, since inconsistencies in $\mathcal{K}$ render the very notion of correctness and must therefore be maintained at all times.

However, knowledge management processes naturally involve changes — either because the situation $\mathcal{S}$ changes or to improve correctness and/or completeness. Changes can range from addition of new material, over correction of errors and refactoring of concepts, to formalization, and deletions (see e.g. [11]). In the presence of the validity and semantic constraints mentioned above, even small changes may induce the necessity to adapt other (related) knowledge items (possibly in other documents) to restore consistency. This process can cascade a seemingly minor change into a major adaptation task. As a consequence the KM community has developed elaborate mechanisms for "change management" to maintain consistency. These make use of the fact that semantic formats often explicitly represent the relations between objects to compute related objects and predict the way changes affect them; see [12, 15, 3] for recent progress in this field.

But even with such systems in place, establishing consistency after each change in the collection remains a daunting prospect, which we will elaborate on in Sect. 2. To enable the conservation of consistency within typical workflows nevertheless, we explore in this paper the new concept of 'versioned links' (introduced and discussed in Sect. 3). We assume that document collections are stored in a revision control system as a concession to change, inducing a notion of 'versioned object'. Next we elevate the relations

between versioned objects to first-class citizens and obtain versioned links as triples (`subject/predicate/object`) in Sect. 3. Naturally, they are versioned as well, so that links and their components carry an independent revision number. 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. Moreover, in Sect. 4 we argue along a DITA use case that versioned links are closer to current authoring practices with "islands of consistency" that allow for a more flexible change management in actual workflows. Sect. 5 concludes the paper.

2. CHANGE MANAGEMENT AND CONSISTENCY

Change management (CM) comprises all the knowledge management tasks that are concerned with the consequences of modifications, particularly with respect to the consistency of the underlying knowledge base. For the purposes of this paper we assume that it is organized in three recursively applicable (and possibly empty) phases:

Change Impact Analysis (CIA) For a given change of an artifact A, a change impact analysis draws on dependency information for artifacts and results in a set of affected items. A CIA may be used for multiple purposes. For example, it can be consulted in order to assess the acceptability of a change by predicting the size of its impact (e.g. with a change propagation index as in [8]). This becomes relevant e.g. in software production or migration. Other purposes consist in acting on found impacts via impact management processes.

Automatic Impact Management builds on propagation rules and change patterns afforded by the underlying ontology and can be automatically executed by the system. Note that distinct underlying ontologies supply different relationships between objects and thus distinct notions of dependency.

Manual Impact Management Here, the user decides for each potentially affected object found by the CIA whether they need to be modified or not. This process needs to be supported by the system and user interfaces to be feasible in practice, e.g. with services that present the relationships leading to this point as in [6].

Typically, a change management process performs a CIA for an artifact first, then an automatic and thereafter manual impact management are applied iteratively on the CIA data (see for example [9, 175ff.]). Note that links are made heavy use of, but it is not clear what happens if a link itself was changed: Is the CIA based on the old or the new link or even both links? We see directly that change management for links as versioned objects is not sufficient, they rather need to be treated as first-class objects. The process by which a change of an artifact yields sequential changes (as well as its product) is often called **propagation of change** in the literature (e.g. [9, 8, 3]).

To get a feeling for the consistency issues involved consider a document collection $\mathcal{K}$ of DITA files, which formalize the scenario "Getting Ready for the Day" as in Fig. 1. In particular, there are concepts `coffeeMachine`, and `coffee`, which are used in a task `morningProcedure`, that describes the suggested procedure for getting ready for the day.

In Fig. 2 we consider three changes with distinct propagation consequences:

- **"id" changed:** Here, we consider the case where an author of $\mathcal{K}$ changed the value of the `id` attribute of the concept `coffeeMachine` e.g. to "coffeeMaker". Then the change results via an automatic impact management in automatic updates from all references to `coffeeMachine` to `coffeeMaker`. This process of **automatic propagation** yields a consistent document collection.
- **"link" changed:** Now, let us look at the consequences if the related `link` (in the concept `coffeeMachine`) to the Wikipedia entry for "Coffeemaker" `http://en.wikipedia.org/wiki/Coffeemaker` is changed to "Coffeepreparation". The automatic impact management can determine that changes in `related-links` do not have consequences for tasks using the changed concept, but it cannot decide automatically whether the corresponding `linktext` element needs to be updated or not. Here, we speak of **semi-automatic propagation**, which leaves $\mathcal{K}$ in a possibly inconsistent state.
- **"xref" changed:** Finally, we envision a modification of the reference to the concept `coffee` within the concept `coffeeMachine` to a concept `decaf`. This means that the coffee machine is modified into a decaf machine, which might have consequences for the task `morningProcedure`. Moreover, the informal descriptions within `conbody` might also need to be adapted. Here, the author herself has to decide on necessary updates, that is a **manual propagation** of change is asked for. As the state of $\mathcal{K}$ is unclear, it may or it may not be consistent.

Note that impact management can easily become more and more complex as new change events are triggered by propagated changes. In general, there are the following concerns about change management of collections:

- Most changes are rather informal (i.e., concerning informal text "tweaking a bit here and there") and must be treated manually (automation would be an AI-hard problem). As a consequence the triggered (manual) impact management cascade can easily outgrow the user's cognitive limits: Giffin et al. reported for example a maximum of 2579 linked changes [8, p. 16] in a Software Engineering study over an 8-year course. Fortunately though, they found "*the majority [. . . to be . . .] comprised of less than 10 changes*".
- Even in formal systems rules for automatic modification after certain types of change are hard to conceive.
- Automatic impact management might be explicitly objectionable in many workflows ; for instance, in cooperative mathematical processes different (e.g. evolving) assumptions in theorems occur frequently and have distinct consequences. Thus a global consistency may be cumbersome and even harmful in the creative process. An author might willingly commit an inconsistent document collection, as long as the document in question or some part of it stays consistent, and global consistency can be achieved for publication.
- Storage with version management alone does not make consistency management tractable if commit policies assume semantic validity requirements. In [13] for example we resorted to a relaxed formalization tool to

HotDrinks.dita

```
<concept id="coffee"><title>coffee</title>
    <shortdesc>A hot drink based on brewing coffee beans</shortdesc>
</concept>
<concept id="coffeeMachine">
    <conbody> A coffee machine produces <xref href="#coffee">coffee</xref>.</conbody>
    <related-links>
        <link format="html" href="http://en.wikipedia.org/wiki/Coffeemaker" scope="external">
            <linktext>Coffeemaker</linktext>
        </link>
    </related-links>
</concept>
```

morning.ditamap:

```
<map>
    <topicref keys="HotDrinks" href="HotDrinks.dita/>
    <keydef keys="coffee-machine" href="HotDrinks#coffeeMachine"/>
    <keydef keys="coffee" href="HotDrinks#coffee"/>
    <topicref href="morningProcedure.dita" type="task"/>
</map>
```

morningProcedure.dita:

```
<task id="morningProcedure">
    <title>Get ready for the day</title>
    <taskbody>
    <prereq>Alarm is on and you wake up.</prereq>
    <steps>
        <step><cmd>Set alarm off.</cmd></step>
        <step><cmd>Get out of bed.</cmd></step>
        <step><cmd>Start <keyword keyref="coffee-machine"/>.</cmd></step>
        <step><cmd>Take shower.</cmd></step>
        <step><cmd>Enjoy the sniff of <term keyref="coffee">coffee</term>.</cmd></step>
        <step><cmd>Enjoy the cup of <conref href="HotDrinks#coffee"/>.</cmd></step>
```

Figure 1: Our CIA Scenario in DITA Format: "Getting ready for the day"

overcome consistency requirements in early development phases.

This leaves us with the questions where consistency management has its rightful place in collection workflows, and what a sensible scope of it would be. We recommend that this should be left to the user; the next section provides us with the necessary tools.

3. CHANGE MANAGEMENT WITH VERSIONED LINKS

We have seen that inconsistency issues often arise because changes in one place in a knowledge base $\mathcal{K}$ affect linked objects in other places, which if adapted to the original change trigger affects at yet another set of linked objects in yet other places in $\mathcal{K}$ and so on. The idea for improved handling of changes is that changes need not be propagated to linked objects if the original link itself continues to exist. Instead of resolving links late, i.e., resolving links according to current objects, one can resolve links early, i.e., resolving links according to the objects one started out with.

Termed this way, one can see easily that the technique of "late binding" (also called dynamic/name binding) in programming languages in Software Engineering, where the method being called upon an object is looked up by its name at runtime, is used in a similar situation. Indeed, it comes with similar problems of inconsistencies, which often lead directly to compiler errors. There, late binding problems are treated e.g. 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. The introduction of versioned links to a document collection $\mathcal{K}$ can be seen as a carry-over of a late-binding-problem solution developed in Software Engineering extended by a finer granularity.

Concretely, 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.[1] *ii)* The advent of versioned query interfaces like [21, 7] 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 [15], 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 correspond to text files

[1] In this paper we assume a concept of global revisions as employed e.g. in the Subversion system [17]. There, any commit to the repository increments the revision number.

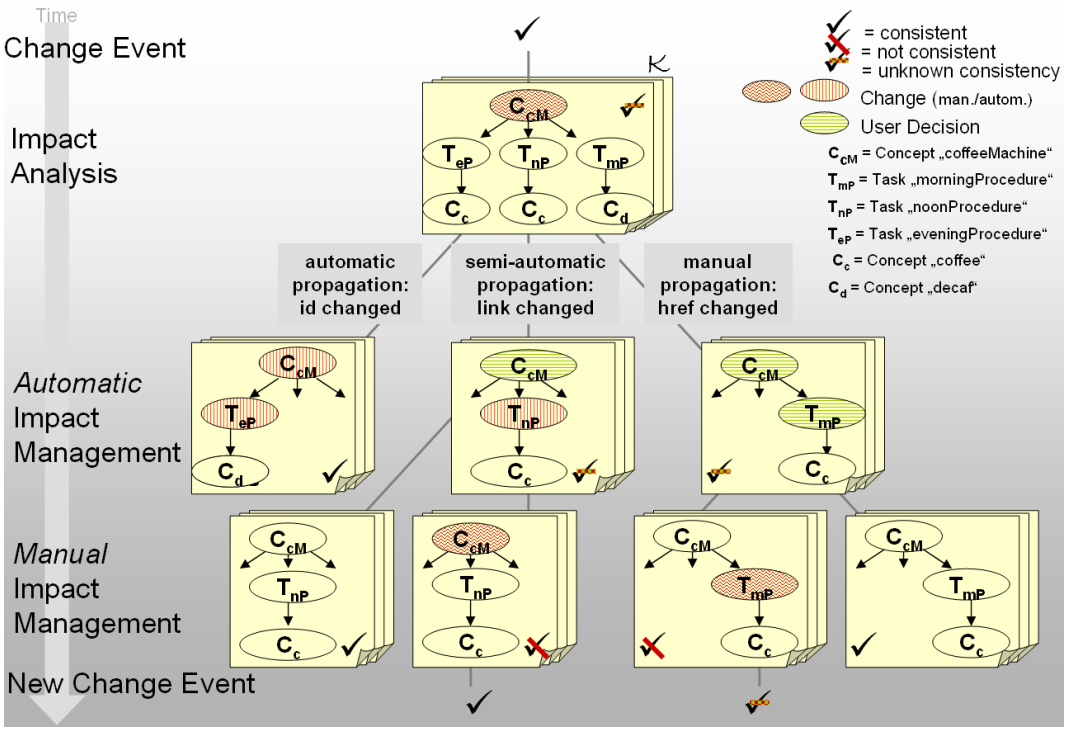

Figure 2: Change Management and Consistency

or text nodes in XML)[2]. 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 a **fragment identifier** (i.e., a name path, where all names are XML element names).

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 knowledge management in the large can be seen as in Fig. 3. We employ repositories to model the development of document collections over time, where the collection $\mathcal{K}$ 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. 3 modulo the correspondence relation given by the dotted lines.

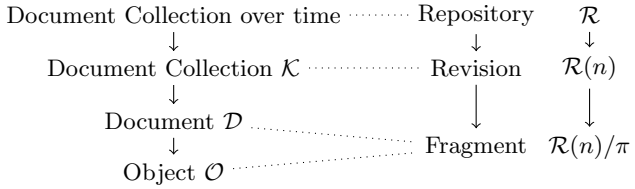

Figure 3: A Realization of Document Collections over Time

3.2 Versioned Links

Before we can define the concept “versioned link”, we need to think about the status of links in semantic 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 [14]) where `subject`, `predicate`, and `object` are name paths in $\mathcal{T}$. 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 $\mathcal{K} = \mathcal{R}(n)$ 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.

[2]The original *fs*-trees had the notion of symbolic links and repository externals which we do not need here.

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

i) `conref` links in DITA files induce document links for a `predicate` "input", which tells the formatting engine to replace the value of its `href` attribute (which resolves into a URI to a specific DITA object) with the title of its `object`.

ii) Similarly, `\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⟫.

iii) `<a href=''`⟪URI⟫`''`⟪attribs⟫`>`⟪label⟫`</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 ⟪label⟫ (details specified by ⟪attribs⟫).

Note that all of these links rely on name paths in $\mathcal{K}$ (realized as URIs) for identification of resources (nodes in $\mathcal{K}$). 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 a format $\mathcal{F}$ an **$\mathcal{F}$-reference**. Even though $\mathcal{F}$-references dominate in semantic formats, we will also cover proper links represented in any RDF representation format; they sometimes exist as standoff markup in KM systems.

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 $\mathcal{R}$. 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 2 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. We distinguish versioned links into **inter-revision** links/references, iff they involve at least two different revisions, and **intra-revision** links/references otherwise.

The purpose of versioned links and references 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. Intuitively, a versioned link to the definition insulates the theorem against the change.

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.

Versioning systems usually reserve a special, intensional "revision identifier" for the respective youngest revision (called the **head revision** and denoted with $\uparrow$). Therefore, we define a versioned name path as **head path**, iff it is of the form $\langle \pi, \uparrow \rangle$ for some name path π. To formally differentiate between versioned and unversioned links, we call a versioned link a **head link**, iff all of its three versioned name paths are head paths.

Wikipedia, for example, uses an underlying versioning system. It strongly advises using versioned references for citing articles (see [20]), on the other hand all links between articles are head links. To understand the advantages and disadvantages of using versioned links, we discuss and exemplify workflows with and without versioned links in the next section.

4. ISLANDS OF CONSISTENCY WITH VERSIONED LINKS

Consider the following situation: Author Mike of the "Getting Ready for the Day" DITA scenario (Fig. 1) wants to update `morningProcedure.dita` to reflect his personal habit change from brewing only real coffee in the morning to also brewing decaf for his girl-friend. He realizes that the concepts in file `HotDrinks.dita` do not yet distinguish between coffee and decaf. He modifies the concept `coffee` to include only 'real' coffee and adds a concept `coffee-generic` to hither relink all present `coffee` references (see Fig. 4 in the propagation phase). The transformation of concerned links into versioned links means, that these links are frozen to point at the object versioned last. Without the use of versioned links, the whole set-up is now inconsistent as the relinking changes are not yet propagated to `morningProcedure.dita`. Note that consistency restraints (with respect to reality reflection wrt. Mike's changed habit) were fulfilled before the refinement. If Mike were in a hurry now and if he thus checked the file into a version management system, then this inconsistency were manifested which might yield consequences for collaborators. But let's say, Mike still has time to follow up on consequential changes.

The change impact analysis (if present and enabled) returns a list of potential conflict locations. In this small example the CIA already contains the concept `coffeeMachine` in `HotDrinks.dita`, the `keydef` element `coffee` in `morning.ditamap`, and a `conref` element in `morningProcedure.dita`. For use of DITA Priestley suggests to "*factor out any context-specific elements*" [18], which is practically done by using a DITA map for defining `keydef` and the (`keys`/`href` attributes in the) `topicref` elements to contain context-specific information and to reference hither from everywhere else. To achieve broadest possible consistency Mike thus starts propagating the change with modifying the `keydef` element in `morning.ditamap`. Note that he probably oversees that `term` elements are only interpretation help for a locally set string (here: "coffee") and therefore need further attention to keep consistency. If this new change triggers a new CIA yielding a list of all parents of affected `term` nodes, then transforming them into versioned links would allow Mike to work the remnant propagation requirements first. Again, without versioned links Mike has to hope that he can finish the entire propagation before check-in into the version management system, otherwise he starts to manifest rather complex (i.e., more and more dependent and interrelated) consistency issues. We also like to point out, that without versioned links the original CIA needs to include all potential subsequent changes as it is supposed to show the

```
HotDrinks.dita
<concept id="coffee"><title>real coffee</title>
    <shortdesc>A caffeinated, hot drink based on brewing coffee beans</shortdesc>
</concept>
<concept id="decaf"/>
<concept id="coffee-generic"><title>coffee</title></concept>
<concept id="coffeeMachine">
    <conbody> A coffee machine produces <xref href="#coffee">coffee</xref></conbody> …
</concept>
<concept id="decafMachine">
    <conbody> A decaf machine produces <xref href="#decaf">coffee</xref></conbody>
</concept>
```

```
morning.ditamap:
<map>
    <topicref keys="HotDrinks" href="HotDrinks.dita/>
    <keydef keys="coffee-machine" href="HotDrinks#coffeeMachine"/>
    <keydef keys="coffee" href="HotDrinks#coffee-generic"/>
    <topicref href="morningProcedure.dita" type="task"/>
</map>
```

```
morningProcedure.dita:
<task id="morningProcedure">
    <title>Get ready for the day</title>
    <taskbody>
    <prereq>Alarm is on and you wake up.</prereq>
    <steps> …
        <step><cmd>Start <keyword keyref="coffee-machine"/>.</cmd></step> …
        <step><cmd>Enjoy the sniff of <term keyref="coffee">coffee</term>.</cmd></step>
        <step><cmd>Enjoy the cup of <conref href="HotDrinks#coffee-generic"/>.</cmd></step>
```

Figure 4: Our CIA Scenario 1 with Propagation of Change (marked grey)

impact of a change. Therefore, the use of versioned links reduces the overall-complexity of changes to a step-by-step complexity. This propagation would have been easier for Mike, if he had consistently used `conref` elements for referencing as the resp. string is generated from the title of the link target. Here, he will realize only after controlling all the items found by the second CIA, that no further change is needed. Generally, we envision that using CIA as a tool will drastically change DITA best practices.

Before Mike started the whole process, he talked about his plans to Andi by which Andi was motivated to create her own `morningProcedureAndi`. As the communicated idea was to refine the old generic concept `coffee` to a new specific concept `coffee` and otherwise to relink `coffee` references to `coffee-generic` ones, she assumed the concept `coffeeMachine` to become generic as well. Therefore, in order to write her own task, she added the concept `decaf` to `HotDrinks.dita` and copied and modified the original task `morningProcedure` accordingly into `morningProcedureAndi`. Unfortunately, Mike thought it obvious that with the refinement of `coffee` comes along the refinement of `coffeeMachine`. If he has not transformed the links of his first CIA into versioned links (updating *her* task document by that), this misunderstanding now hinders collaboration on the document collection as the concept `coffeeMachine` is used inconsistently in Andi's task. Unfortunately, if Andi is authoring her reference after he refined `coffee`, then the misconception might live on anyway.

To sum up, versioned links allow new, much more flexible workflows. In particular, the complex recursive propagation of change becomes time- and order-independent.

4.1 Islands of Consistency

This example exhibits many typical aspects commonly found in the human, informal development and change process of technical documents. Conceptually, we can divide this into two phases:

- **Knowledge Exploration** In this phase knowledge items are created, In particular, concepts are formed and put down in definitions, conjectures about the concepts are proven, and local narrative structures that motivate and connect all of these are established. Note that the consistency of inter- and intra-document links is not the foremost concern here.

- **Codification** Published knowledge, however, certainly should aim for consistency. The exploration phase thus is followed by a codification phase, when the practice of reorganizing knowledge in documents to streamline the elegance and beauty of arguments happens. In particular, in the codification phase documents are brought into the form of a coherent document (collection) addressed at a particular audience.

Actually, in real life, document development and change usually proceed in multiple phases, interleaving the exploration and codification phases, gradually shifting emphasis from the former to the latter, and from local codification to global coherence concerns, but the conceptual division of these two phases remains useful. Even in the knowledge exploration phase, a (possibly temporary) document form is usually chosen to write down the knowledge items. Note that in these "development documents", global consistency is usually not a primary concern in the early exploration-heavy stages, since this is deferred to the codification phase. In the middle phase of the document development and modification process, when concepts, definitions, and links start stabilizing and attention shifts to detail and codification, we

observe that authors start establishing document fragments that are internally consistent, but may be inconsistent with other parts of the document collection under development. We call such document fragments **islands of consistency** and observe that much of the codification proceeds by enlarging these islands and merging them until a globally consistent document (collection) is reached.

We claim that humans implicitly use versioned links in the development and change of technical documents, since they *allow* us to organize consistency in mutable document collections by fixing revisions and thus insulating against the effects of changes in linked objects. In our elaboration of the "Getting Ready for the Day" scenario, this was also evidential. Explicit versioned links (as presented in Sect. 3) enable the process of maintaining and developing islands of consistency without time or order constraints. Additionally, Haramundanis reports a new modularization paradigm for the information engineer in [10]. This means in particular, that the amount of links is dramatically rising and with it the number of potential inconsistency issues. The underlying reason for this modularization is a growing need for collaboration on document collections. We have outlined in our use case how islands of consistency come in here especially handy.

But maintaing islands of consistency has hidden costs: It introduces inter-revision links – which finally have to disappear – in a document collection and thus weakens its coherence.[3] Therefore, in a sense all what the introduction of versioned links does is that it allows to move parts of the problem of *consistency management* in the exploration phase into one of *coherence management* in the codification phase. In this respect versioned links are closer to current scientific publication behavior where we almost only reference archival papers which never change. In particular, new versions of publications are considered as distinct entities with different "URIs"[4].

5. CONCLUSION

We have presented the concept and practices of using versioned links as a tool for managing change in document collections including knowledge repositories, particularly as a tool for maintaining "islands of consistency". Essentially the introduction of versioned links allows to move parts of the problem of consistency management in the exploration phase into one of coherence management in the codification phase. However, we contend that this allows for much more flexible and natural workflows, and is thus well worth the (minimal) effort in extending the representation formats to versioned links.

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 "islands of consistency" practices versioned links are designed for (to get an overview of their differences see e.g. [16]). Indeed, one can see and use each local repository in a distributed network as such an island, 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 course-granular for technical 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 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 [21], 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.

In this paper we have defined the concepts and looked into the workflows afforded by versioned links. The eventual practicability of the extension will of course only become apparent when versioned links are supported in editing and knowledge management environments. For instance, editors could have a configuration option that allows to specify the default behavior when no revision is given, that is either we assume head revision or the current revision. Similarly, the underlying revision control system could be extended to automatically introduce versioned links; Consider a situation, where an object b is referenced by an object c. In this case, the RCS can change the head link in c to a versioned link to b at the last revision before the change. It is easy to see that versioning all such references to b conserves consistency. This behavior is essential in multi-author situations, for example, if the author of b does not have write access to c and only its authors can propagate the changes to upgrade the reference on c to b to a head link again. So in this case, the RCS would probably notify the authors of c of a coherence enhancement opportunity. For the codification phase we wish for a coherence management module in editors. It could step through inter-revision links and jointly present difference-marked up versions of `subject`, `predicate`, or `object`, for instance. Then it might offer the author the choice between upgrading the link revision or copying the object. For collective authoring situations in larger document collections we could imagine a notification system for revision updates.

[3]We could define/measure the "revision coherence" of a document collection as the number of intra-revision links.

[4]In situations, where this practice is lax (e.g. in editions of books), this has been known to lead to problems, see [4] for an example, where the loci of branch cuts in definitions of special functions vary between editions of [1].

6. REFERENCES

[1] M. Abramowitz and I. A. Stegun, editors. *Handbook of Mathematical Functions (with Formulas, Graphs, and Mathematical Tables)*. Applied Mathematics Series. National Bureau of Standards, 1964.

[2] ACM Special Interest Group for Design of Communication. *Proceedings of the 28th ACM International Conference on Design of Communication*, SIGDOC '10, New York, NY, USA, 2010. ACM Press.

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

[4] R. Corless, J. Davenport, D. Jeffrey, and S. Watt. According to Abramowitz and Stegun. *SIGSAM Bulletin 2*, 34:58–65, 2000.

[5] OASIS Darwin Information Typing Architecture (DITA).

[6] DocTIP.

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

[8] M. Giffin, O. de Weck, G. Bounova, R. Keller, C. Eckert, and P. J. Clarkson. Change propagation analysis in complex technical systems. *Journal of Mechanical Design*, 131(8):081001, 2009.

[9] J. Han. Supporting impact analysis and change propagation in software engineering environments. In *Proceedings of the Eighth IEEE International Workshop on Software Technology and Engineering Practice*, pages 172–182. IEEE Computer Society, 1996.

[10] K. Haramundanis. Experience report: modularization - the new paradigm for the information engineer. In B. Mehlenbacher, A. Protopsaltis, A. Williams, and S. Slatterey, editors, *Proceedings of the 27th annual ACM international conference on Design of communication (SIGDOC)*, pages 151–154, New York, NY, USA, 2009. ACM Special Interest Group for Design of Communication, ACM Press.

[11] K. Haramundanis. Modularizing in glossaries: an experience report. In *Proceedings of the 28th ACM International Conference on Design of Communication* [2], pages 131–134.

[12] D. 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.

[13] A. Kohlhase, M. Kohlhase, and C. Lange. sTeX – a system for flexible formalization of linked data. In A. Paschke, N. Henze, T. Pellegrini, and H. Weigand, editors, *Proceedings of the 6th International Conference on Semantic Systems (I-Semantics) and the 5th International Conference on Pragmatic Web*. ACM, 2010.

[14] F. Manola and E. Miller. RDF Primer. W3C Recommendation, World Wide Web Consortium (W3C), Feb. 2004.

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

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

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

[18] M. Priestley. Scenario-based and model-driven information development with xml dita. In *SIGDOC '03: Proceedings of the 21st annual international conference on Documentation*, pages 45–51, New York, NY, USA, 2003. ACM.

[19] F. Sousa, M. Aparicio, and C. J. Costa. Organizational wiki as a knowledge management tool. In *Proceedings of the 28th ACM International Conference on Design of Communication* [2], pages 33–39.

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

[21] V. Zholudev and M. 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`.