On Product Versioning for Hypertexts

Tien N. Nguyen, Cheng Thao, and Ethan V. Munson

University of Wisconsin-Milwaukee
{tien,chengt,munson}@cs.uwm.edu

Abstract. Versioned hypermedia has shown its success in promoting better understanding and management of evolving document collections in many domains. However, providing versioning capability for a hypermedia system raises several important structural and cognitive issues. Our research has produced *Molhado*, the first hypermedia infrastructure that applies the *product versioning* model to versioned hypermedia. Molhado not only supports configuration management for hypermedia structures in a fine-grained manner, but also provides version control for individual hyperlinks and document nodes. This paper explains how the product versioning model in Molhado addresses serious issues identified by earlier research on versioned hypermedia. We will also discuss the new issues raised by using this versioning model.

1 Introduction

Versioned hypermedia (or *hypertext versioning*) is concerned with storing, retrieving, and navigating prior states of a hypertext, and with allowing groups of collaborating authors to develop new states over time [1]. Hypertext versioning has shown its usefulness in several domains. In software engineering, the ability to capture the evolution of relationships between development artifacts makes versioned hypermedia more powerful than traditional versioning systems that only capture the evolution of the artifacts themselves. Relationships between requirements, design, and implementation documents can be represented directly so that traceability analysis and cost estimation become more tractable.

In legal systems, laws, regulations, and tax codes are a set of complex information artifacts with many relationships. It is important to store and retrieve previous document revisions because, in legal systems that prevent ex-post-facto laws, the version of a law that affects a case is the one in effect at the time of an infraction [1]. Hypertext support can make it easy to navigate to related laws, precedents, regulations, and codes within the set of applicable laws at the time in question. Thus legal systems can benefit from versioned hypermedia capability.

In the World Wide Web, the most popular and successful hypertext system, documents are interrelated to each other via HTML hyperlinks. Versioned hypermedia could allow users to roll a Web site back to a specific time and navigate it as if they were interacting with the Web sites as of that day. Using this facility, an E-commerce site could determine the validity of customer complaints about pricing errors or a news site could allow users to view the site for a particular day and time.

SCM 2005, September 5-6, 2005, Lisbon, Portugal.

However, providing versioning capability for a hypermedia system raises some important structural and cognitive issues, described in detail by Østerbye [2]. The structural issues are immutability of versions, version control for links, and versioning for hypermedia structure. The cognitive issues are version creation and hypermedia element version selection.

Our research has produced Molhado [3], an infrastructure for hypertext versioning and software configuration management that is well-suited for managing logical relationships among software documents. Molhado uses a single versioning mechanism for all software components and for the hypertext structures (including anchors and links) that connect them. Molhado can track changes at a very fine-grained level, allowing users to return to a consistent previous state not only of a hypertext network but also of a single node or link. Molhado supports both embedded (as in HTML) and first-class hyperlinks (as in XLink [4]). Molhado is, to the best of our knowledge, the first system that applies the *product versioning* model to versioned hypermedia. Molhado's product versioning addresses all of the issues mentioned by Østerbye. Of course, it raises other issues.

This paper is divided into seven sections, the first being this introduction. The next section presents related work and describes the most important versioning models used for hypertexts. Section 3 summarizes structural and cognitive issues with versioned hypermedia raised by Østerbye. Section 4 describes Molhado's versioned hypermedia model and explains how it addresses those issues. Section 5 presents distinguished functionality of the Molhado versioned hypermedia tool. Section 6 discusses new issues raised by the use of the product versioning model and the final section presents conclusions.

2 Related work

There are three important approaches to version control for hypertext structure and related software artifacts: the *composition model*, the *total versioning model*, and the *product versioning model*.

In the *composition model*, versions of atomic components are maintained and assembled into versions of composite components. Each atomic component (usually a file) has a version history. Versions of composite components or of the entire system are defined by revision selection rules (RSRs) that specify which component versions are part of a version of the larger artifact.

Composition-model systems that provide versioning only for data include Xanadu [5], KMS [6], DIF [7], and Hyperform [8]. The HyperWeb [9] and HyperCASE [10] systems are both based on RCS [11], in which the smallest versionable information unit is a file.

The use of RSRs allows versions of complex objects to be constructed by composition of versions of simpler objects, but it creates some problems [12]. For example, the indirect representation of a hypertext structure makes it difficult to analyze the structure, unless the rules are actually executed. So, to avoid the use of RSRs, some systems maintain a notion of *current context* [13–15]. A context has been defined as a coherent structure, such as a document, a document

collection [2], or a partition of a hypertext [13]. However, the complexity and overhead of maintaining the current context becomes significant if composite components have many nested levels and hyperlinks among them.

While the previous systems focused on versioning the linked objects, other research has explored how to maintain version histories for links as well while still using composition versioning. In Neptune [16], the composite is used to represent the isolated work area of each collaborator. HyperPro [2] records changes for links by placing links inside a "version group" (i.e., a composite) that is versioned. It supports both links to a specific version of a node and links to a node without regard to its version. Like HyperPro, HyperProp [17] does not record the history of links individually. In both HyperPro and HyperProp, the RSRs are stored on the structure, affecting all link endpoints, and provides for selection of specific document revisions from a versioned document. With this RSR approach, it is inefficient to evaluate rules across the revisions of a specific link [18].

The research addressing versioning in open hypermedia systems has also applied the composition versioning approach. Microcosm [19] has a context-like structure called the "application", which is versioned. In the versioning proposal for Chimera [20], a "configuration" is a named set of versions of Chimera hypermedia elements, representing the subset of a hypertext structure that might be affected by modifications to an externally stored object. In the hypermedia version control framework (HURL) [21], a hypertext structure is represented by an "association", which is a collection of link identifiers, anchor identifiers, and the documents that define a connection. The IAM group has investigated the Fundamental Open Hypermedia Model (FOHM)'s contextual model [22] to see if it could replace the selection engine in a hypertext versioning server.

In the *total versioning model*, all items are versioned, including composite and atomic components. Each item still has its own version space. The relationships among the items' version spaces are defined by RSRs or by versions of composites referring to versions of other components. In PCTE [23], a new version is created by recursively copying the whole composition hierarchy and establishing successor relationships between all components. In CoVer [14] and VerSE [15], the RSRs are stored on the containment arc between a container and its contents, providing selection of link revisions and document revisions from versioned links and versioned documents, respectively. This structure versioning increases the work that must be performed to ensure that a structure container holds a consistent hypertext [18]. The SEPIA [24] system has the notion of "composite node", which contains a partially ordered set of nodes and links and represents subgraphs of the hypermedia network. The total versioning model, in general, suffers from the version proliferation problem [25], where each small change in a leaf of the hierarchy creates versions of multiple objects higher in the compositional hierarchy or hyperlinked to that leaf node.

In contrast to total versioning, *product versioning* establishes a single view of a software product, or even an entire database. This is done by arranging versions of all items in a uniform, global version space. This approach is popular in software configuration management (SCM) and versioning systems such as

in change-based systems (COV [26], Aide-de-camp [27], PIE [28], etc), or state-based systems (Voodoo [29]). Wagner's product versioning infrastructure [30] integrated version management with incremental program analyses. Our system, Molhado, is currently the only versioned hypermedia system that applies this product versioning model.

3 Issues with versioning for hypertexts

This section summarizes structural and cognitive issues with providing the versioning capability for a hypermedia system mentioned by Østerbye [2].

3.1 Immutability of versions

The first structural issue is called *immutability of versions*. It is obviously that the contents of a frozen version of a resource should be immutable, that is, they cannot be changed without creating a new version of the resource; it is less clear how links pointing to it and its attributes should be treated. Yet it may be useful to allow frozen versions to have new links (for instance, annotations or comments) coming from and going to them without necessarily creating a new version of the resource. At the same time, some links are carrying semantics or substantial parts of the resource itself, and thus their modification should definitely require the creation of a new version of the whole resource.

3.2 Version control for links

The second structural issue is associated with version control for links. Versioning for links is very important. The fact that existing hypermedia systems for software development do not version individual links is a significant factor preventing their wider use in the software engineering domain [1]. However, providing version control for individual links raises some serious problems.

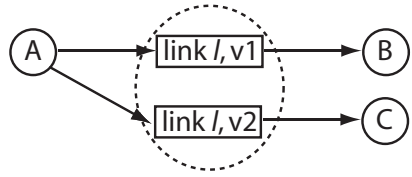

Fig. 1. First issue with link versioning

The first one that Østerbye mentioned in his paper is the navigation problem if a hyperlink has several versions. For example, in Figure 1, the version *v1* of a link connects two nodes *A* and *B*, and the version *v2* of that link connects *A* and *C*. The question is that when users navigate through the link, which destination

should be used. Human intervention would create cognitive overhead for users and would make the user interfaces for versioned hypermedia complex.

Another problem with the version control for links is shown in Figure 2. Suppose that we have "$A \rightarrow$ the link $\rightarrow B$". If B now is modified into B' (a new version *v2* of B), a new version of the link l must also be created connecting A to B'. The reason for this phenomenon is that links and resources have their own version histories. This is due to the fact most of existing versioned hypermedia systems followed either composition versioning or total versioning models.

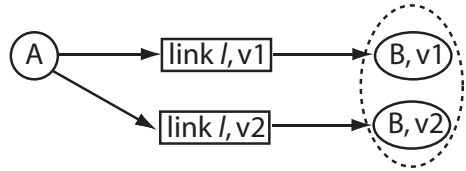

Fig. 2. Second issue with link versioning

The third problem with link versioning is whether a system should treat a hyperlink as a first-class object or a relation of anchor values. If the former one is chosen, the question is how the system determines when a new version of a link must be created [2]. If the latter choice is made, it is very hard for a system to distinguish two versions of a link that contains the exact set of member anchors or member resources.

3.3 Version control for hypermedia structure

The third structural issue that Østerbye mentioned is with version control for hypermedia structure. Hypermedia structure often refers to the network consisting of document nodes, anchors, and connecting hyperlinks. Østerbye suggested that the hypertext versioning system must allow users to return to a consistent previous state of entire network, rather than just a previous state of a single document node or link. Providing version control for both hypermedia structure and individual nodes/links is not easy to achieve [2].

3.4 Hypermedia element selection

A serious cognitive issue as providing versioning capability for hypermedia systems is associated with hypermedia element version selection. A prominent problem, when introducing versions of nodes, is to determine which element in the versioned group (i.e. a group of all object versions) the link points to [2]. In some systems (e.g. HAM, HyperPro), a link can point to either a specific element (i.e. a specific object version) or entire group. The link could also point to the current element, meaning the newest element in the versioned group. In a situation where the elements in the versioned group are organized in a version tree, it

is not clear which is the newest element. Some systems were based on a query language to do element selection. This increases the cognitive overhead for users, who have to specify a selection criteria each time a link is created. If a selection query must be specified each time a link is created, the users are discouraged from making links. Special care must therefore be taken at the interface level to simplify the use of link selection.

Another related issue is the place to store element version selection rules. There are two prominent choices. The first one is that element selection rules are stored on the structure, affecting all link endpoints, and provides selection of specific document revision from a versioned document (e.g. in HyperPro and HyperProp). With this approach, it is inefficient to evaluate rules across the revisions of a specific link [18]. The other popular choice is that the rules are stored on the containment arc between a container and its containees, providing selection of link revisions and document revisions from versioned links and versioned documents respectively. This structure versioning increases the work that must be performed to ensure that a structure container holds a consistent hypertext [18]. CoVer's [14] and VerSE's [15] followed this scheme.

3.5 Version creation

According to Østerbye, cognitive overhead can also increase with the version creation operation. If a user is forced to come up with names for each and every document node that is created, it will distract his/her attention from the actual subject. Similarly, if the user must explicitly create new versions of nodes or links all the time, and maintain some consistency among their versions, it will further distract his/her attention from the real work to be done [2].

4 Versioned hypermedia in Molhado

This section describes the Molhado versioned hypermedia model. More details can be found in another document [3].

4.1 Data model

The data model used in Molhado is the Fluid Internal Representation (Fluid IR). The main concepts in this data model are *node*, *slot*, *attribute*, and *sequence*. A node is the basic unit of *identity* and is used to represent any abstraction. A slot is a location that can store a value in any data type, possibly a reference to a node. A slot can exist in isolation but typically slots are attached to nodes, using an attribute. An attribute is a mapping from nodes to slots. An attribute may have particular slots for some nodes and map all other nodes to a default slot. The data model can thus be regarded as an attribute table whose rows correspond to nodes and columns correspond to attributes. The cells of the attribute table are slots (see Figure 3). Slots in an attribute table can belong to three types: constant, simple, or versioned slots.

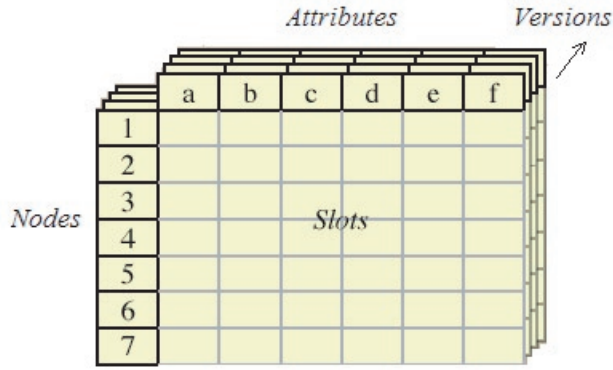

Fig. 3. Data model

A *constant slot* is immutable; such a slot can only be given a value once, when it is defined. A *simple slot* may be assigned even after it has been defined. The third kind of slot is the *versioned slot*, which may have different values in different versions (*slot revisions*). A *sequence* is a container with slots of the same data type. It has a unique identifier. Sequences may be fixed or variable in size and share common slots together. Once we add versioning, the attribute table gets a third dimension: the version.

4.2 Product versioning

The version dimension follows the product versioning model. Instead of focusing on individual components, Molhado versions a software project as a whole (see Figure 4). All system objects including (atomic and composite) components and hypertext structures are versioned in a *uniform, global version space*. Object properties can be defined as versioned or un-versioned. A *version* is global across the whole project and is a point in a *tree-structured discrete time* abstraction, rather than being a *particular state* of an object as in total and composition versioning models. That is, the third dimension in the attribute table in Figure 3 is tree-structured and versions move discretely from one point to another.

The state of the whole software system is captured at certain discrete time points and only these captured versions can be retrieved in later sessions (for example, the versions *v1.0*, *v2.0*, *v3.0*, etc in Figure 4). The *current version* is the version designating the current state of the project. When the current version is set to a captured version, the state of the whole project is set back to that version (for example, *v2.0* in Figure 4). Changes made to versioned objects of the project at the current version create a temporary version, *branching off* the current version. That temporary version will only be recorded if a user explicitly requests that it be captured. To record the history of an individual object, the whole project is captured. Capturing the whole project is quite efficient because the versioning system only records changes and works at a very small granularity [3].

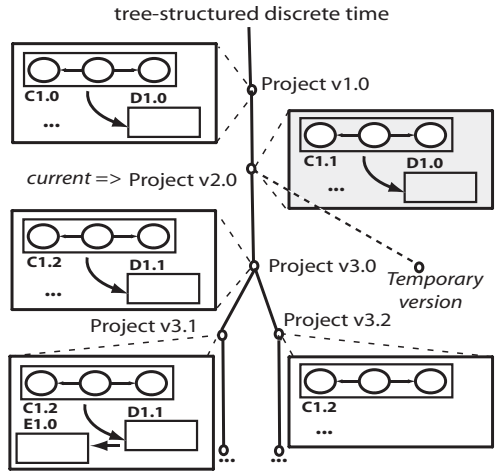

Fig. 4. Product versioning in Molhado

4.3 Structure-oriented representation

To be able to control versions of software documents at a fine granularity, Molhado follows a *structure-oriented* approach where each document is considered to be logically structured into fine units, called *structural units* or *logical units*. This approach is often taken in *structured document* research, e.g. SGML and XML. In this approach, each software document is represented by a *document tree* or a *document graph* in which each node encodes a logical unit of the document. Since XML has become the standard structured document format and very successful in representing many different data types, it is very natural to use XML for representing non-program artifacts. Syntactical rules for a document and its structural units are defined by users in a specification such as a *Document Type Definition* (DTD) or *XSchema* specification. For a program, an abstract syntax tree (AST) perfectly represents its logical structure.

To represent documents and hypermedia structure in this structure-oriented approach, tree and directed graph data structures are built from the nodes, slots, and attributes in the data model. A directed graph is defined with an attribute (the "children" attribute) that maps each node to a sequence holding its children. Trees additionally have the "parent" attribute, which maps each node to its parent. We have developed a fine-grained versioning scheme for tree and directed graph data structures in which fine-grained changes to the structure of a tree or a graph and to the contents of associated slots are recorded [3].

Figure 5 shows an example of our representation for an XML document. Each node in the XML document tree has an additional slot that stores its *operator* via "operator" attribute. An operator, which can be shared among nodes, identifies the syntactical type of its node and determines the number and syntactical types of the node's children. Nodes in an XML tree have operators drawn from two categories: *intermediate* and *text* operators. The unique *text operator* is used to

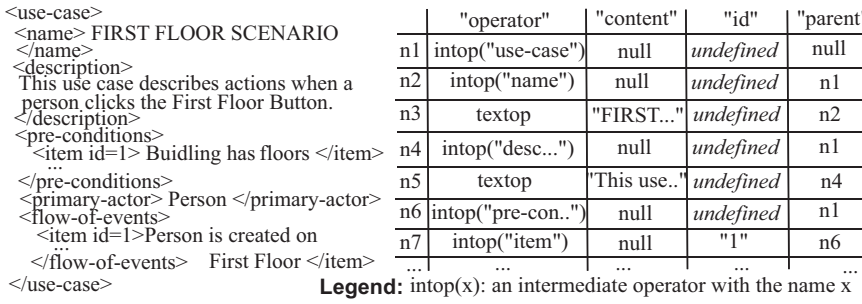

Fig. 5. Structure-oriented representation

represent XML's character data (CDATA) construct. Each node associated with the text operator has an additional slot (defined by the "content" attribute) that holds the CDATA string. Each element node in an XML document is associated with an *intermediate operator*, whose name is the element's name. Each node associated with an intermediate operator has one additional slot for each XML element-level attribute that is defined for that element (e.g. the "id" attribute). This document tree is versioned according to our tree-based versioning scheme.

4.4 Hypermedia entities

To enable HTML-style hyperlinks among these documents, an "href" attribute is defined for each node in a document's tree or graph. A "href" attribute contains a URL referring to a document node. Therefore, embedded hyperlinks can be attached to and can point to any document node.

The Molhado versioned hypermedia model is based on the following concepts: *linkbase*, *hypertext network*, *link*, and *anchor*. A linkbase is a container for hypertext networks and/or other linkbases. The relation between a linkbase and a hypertext network is the same as the relation between a directory and a file in a file system. A hypertext network can belong to only one linkbase. A hypertext network contains links and anchors. A link is a first-class entity and is an association among a set of anchors. An anchor can belong to multiple links. A link or an anchor can also belong to multiple hypertext networks. An anchor is used to denote the region of interest within a document, and it refers to either a document or a document node. This separation between anchors and document nodes allows for the separation between hypertext networks and document contents. Links and anchors can be associated with any attribute-value pairs.

4.5 Version control for hypermedia structure

To handle compositional relations among components or artifacts, Molhado also provides a structure versioning mechanism at both coarse-grained and fine-grained levels [3]. At the coarse-grained level, a composite component can contain atomic components and/or other composite components. At the fine-grained

level, an atomic component is allowed to have internal structure that can contain logical units.

A linkbase is implemented as a composite component, whose internal structure is a tree structure composing of other linkbases and/or hypertext networks. A linkbase is versioned according to the fine-grained versioning scheme for the tree data structure [3]. A hypertext network is implemented as a type of atomic component, whose internal structure is a directed graph. Each link or anchor is represented by a node in that graph. A directed edge connects an anchor's node to a link's node if the link contains the anchor. An additional attribute, attribute "ref", is defined for each anchor's node in the graph. The value of a "ref" slot is a *reference* to either a component or a logical unit within a component (but *not* to a hypertext network or to a linkbase). In general, via our versioning scheme for directed graphs, the history of a hypertext network is recorded [3]:

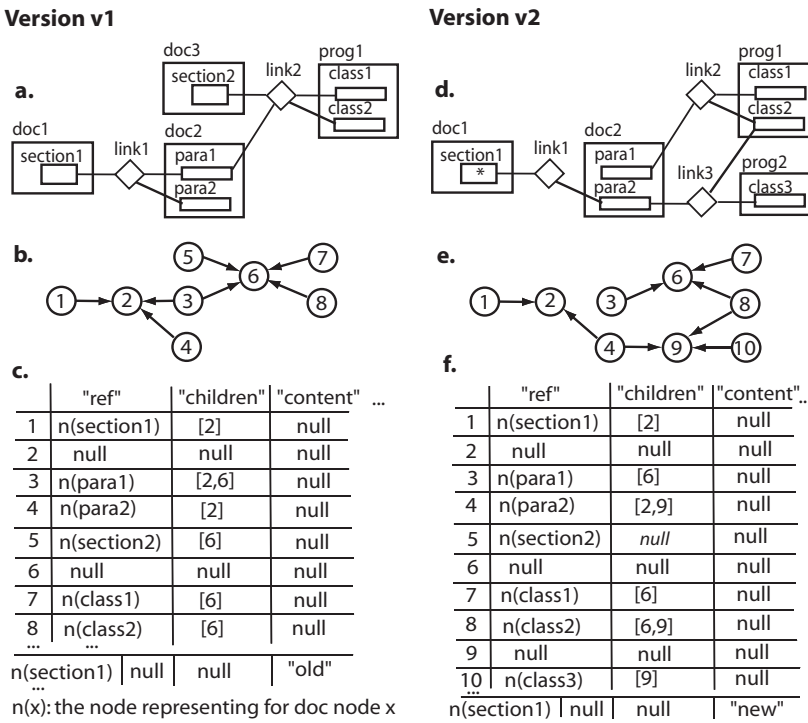

Fig. 6. Versioning for hypertext structure

Figure 6 shows an example of hypertext network versioning. Figure 6a) and d) display the network at two versions *v1* and *v2*. The directed graphs representing for structures of the network at these two versions are in Figure 6b) and e). Links' nodes (e.g. nodes 2 and 6) have edges coming into them and do not refer to anything. The attribute table is updated to reflect the changes to the hypertext

network. Figure 6c) shows part of the attribute table for the network at version *v1* (document nodes in the attribute table are not shown except n(section1)). In this example, attribute "content" defines for each document node a slot that contains a string value. The "ref" cell for an anchor node (e.g. node 1) contains a reference to the corresponding document node (e.g. n(section1)). Figure 6f) shows the attribute table at version *v2*. Node 5 was deleted since *doc3* was removed. Node 3 now has only one child. Node 9 (representing *link3*) and node 10 (representing *class3*) are just created. The "ref" cell for node 10 points to document node representing *class3*. The "content" slot of n(section1) has changed from "old" to "new" to reflect the change in the content of (section 1, doc 1) in the network.

In Molhado's product versioning model, a version is a point in tree-structured discrete time line. Therefore, returning to previous state of entire project is a basic versioning operation. When users set the current version to a recorded version, the state of entire software project will be set back to that version. Thus, documents as well as linkbases and hypertext networks will regain their contents and structures at that version.

4.6 Version control for links

Via the example in Figure 6, we can see that the directed graph versioning scheme also takes care of recording the history of individual hyperlinks (represented as a graph node). This is because the connection structure of a graph node and associated slots are captured over time.

Let us revisit the first issue with version control for hyperlinks (see Figure 1). In this case, Molhado avoids this problem. The nature of product versioning in Molhado allows developers to navigate to the right destination at the current version. In Molhado (see Figure 7), two global versions *v1* and *v2* would be created. At the version *v1*, the connection would be "$A \rightarrow$ the link $\rightarrow B$", and at the version *v2*, the connection would be "$A \rightarrow$ the link $\rightarrow C$". When the current version is selected explicitly or implicitly, the destination of the link will be automatically chosen at the current version.

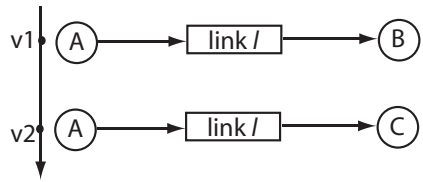

Fig. 7. No problem with link versioning in Molhado

Let us consider the second issue with versioning for hyperlinks (see Figure 2). This scenario can be handled by putting all hypermedia entities including links and resources in one global product versioning space such as in Molhado. Thus, when users select *v2* the current version, the destination of the link *l* would

be the new version of the resource B (see Figure 8). The key idea is that, in Molhado, link traversal occurs only among nodes at the *current* version.

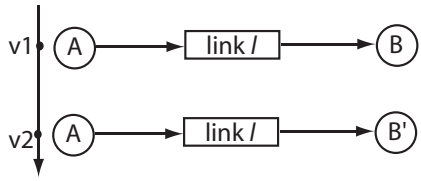

Fig. 8. No problem with link versioning in Molhado (2)

The third issue with link versioning is whether a system should treat a hyperlink as a first-class object or a relation of anchor values. In Molhado, a hyperlink is a first-class entity. It is represented as a node in a directed graph (representing a hypertext network). Each graph node might be associated with multiple attributes (representing properties of hypermedia entities). As explained, via the fine-grained versioning scheme for attributed directed graphs, a hypertext network as well as an individual link is versioned (see the example in Figure 6). In general, when a versioned attribute value associated with the hyperlink changes, or the connection of member anchors changes, or a document node on which a member anchor rests changes, a new version must be created. In this case, a new global version point is created in the tree-structured discrete time line, rather than a new version of resources or a new version of links.

In brief, Molhado provides configuration management for hypermedia structure as well as version control for individual links and document nodes without having problems that occurred in existing versioned hypermedia systems. Now, let us revisit other issues mentioned by Østerbye and relate them to Molhado.

4.7 Revisiting other issues

Immutability of versions: There are two characteristics in Molhado's product versioning model that suggested us a simple solution for this issue. The first one is that Molhado can control which aspects of a software project are versioned or un-versioned. The other one is that resource versions and hyperlink structure versions are maintained in the same global version space. To address this issue, in Molhado, users are allowed to specify the set of "substantial" links (i.e. the ones that if modified would create a new version), while all other links would be considered as annotation or comment links and would not require a new version if changed. Substantial links are modeled as *versioned slots* defined by "href" attribute, while other links as *simple slots* defined by an additional attribute. Therefore, when a substantial link is created and linked to a resource at the current version, a temporary version will be created and this change is hold at this temporary version. But when a non-substantial link is created, the change is hold at the current version and no temporary version will be created.

For hypertext networks, when a first-class hyperlink and its anchors are created and pointed to document nodes at the current version, a new version is created. This choice is reasonable since a first-class hyperlink and its anchors are considered as *annotations* on top of document nodes and are not parts of the document contents. In this case, the change between these two versions (the old and new ones) is the addition of the new first-class hyperlink and anchors. But the components themselves are not changed between these two versions.

Hypermedia element selection: With product versioning in Molhado, versioning for individual entities is subsidiary to versioning for entire project. That is, an individual entity do not have its own version numbers. The current version of entire project is selected implicitly (via user operations such as mouse actions) or explicitly (via user commands). Molhado knows how to relate objects and hypermedia structures at the current version together. Therefore, there is no version selection rules or selection queries involving in user operations on links, anchors, hypertext networks, and linkbases. For example, users' traversal can be done via HTML-style hyperlinks or via a hypertext network without involving users' selection of revisions of individual hypertext entities since the traversal occurs among nodes at the current version. When creating a link, users also do not have to make any selection about versions of targets. This characteristic of Molhado has facilitated the construction of a graphical user interfaces (GUI) for its versioned hypermedia services in the Software Concordance environment [31], which will be described later.

Version creation: In Molhado, individual document nodes or links share the same version space. To keep their version histories, a user has to commit changes to entire project. However, it does not require the user to check in or check out document nodes or hypertext elements individually. The user modifies documents and hypertext structures. When the user is ready to record the changes, he/she can issue a commit command and a new version will be created. Intermediate versions during an editing session can be used for undoing tasks but may not be saved in the SCM repository. In the total versioning model, when a hypermedia entity gets a new version, all other connected hypermedia entities (links, nodes, etc) and its ancestor entities in any compositional hierarchy need to be checked in individually. Although Molhado does not have the type of version creation problem that Østerbye described, a single change to a *versioned* entity might potentially create a new version of entire project.

5 The versioned hypermedia tool

5.1 Library functions

The Molhado's versioned hypermedia infrastructure is implemented in terms of a set of library functions. Versioned hypermedia functionality can be divided into two groups: 1) linkbase and hypertext network services, 2) link and anchor services. The first group includes functions to create a linkbase or a hypertext network, to delete existing hypertext networks or linkbases, to re-structure linkbases and relocate networks among linkbases, to open an existing hypertext

network, select a hypertext network to be *active*, to import and export a hypertext network from and to XLink format at any version. Services for links include link creation, deletion, renaming, attribute's value viewing, and link history viewing. Services for anchors include deleting an anchor, adding an anchor into the active link, removing an anchor off some link, renaming an anchor, and displaying the structural unit that the anchor refers to. More details on this set of library functions can be found in another document [32].

5.2 User operations

This set of library functions can be used in any editing or development environment. Automatic link generation algorithms could be employed to create links and our infrastructure could be used to store versions of documents and links. Molhado's hypertext versioning services were integrated into the Software Concordance (SC) development environment [31] (see Figure 9). Configuration management tasks are handled by Molhado's configuration management infrastructure [33]. The rest of this section describes the distinguished versioned hypermedia functionality in the SC environment.

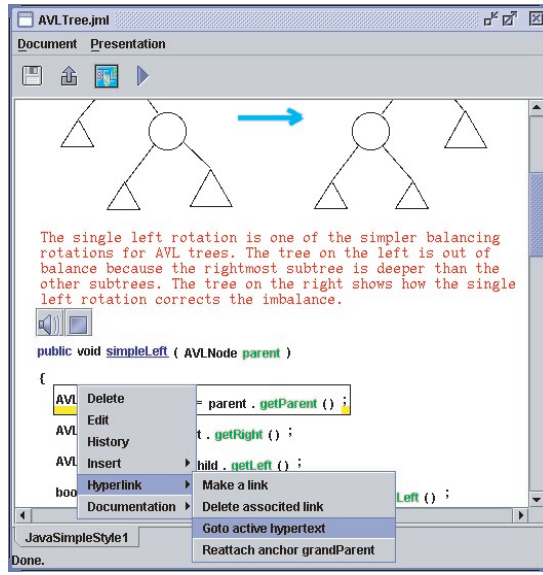

Fig. 9. Structured document editor

Molhado and the SC user interfaces support a variety of transactions. First of all, a user can open an existing project. After selecting the *current (working) version* from a project history window, the system displays the project's directory structure and documents in a project structure window. From this

window, the user can choose to edit, delete, import, or export any document and hypertext network. Also, via this window, the user can graphically modify the project's directory structure. If any modification is made to the versioned components at the current version, a new version would be temporarily created, branching off the current version. Subsequent modifications will not change the temporary status of the version until a *capture* or a *commit* command is issued. If the user does not want to keep the temporary version, he/she can *discard* it. Otherwise, the user can *capture* the state of the project at a version. The *capture* command changes a temporary version into a captured one. A unique name as well as date, authors, and descriptions can be attached to the newly captured version for later retrieval. A captured version plays the role of a checkpoint that the user can retrieve and refer to.

The user can *branch off* a version by just *switching* the current version and then starting the modifications. While working on one version, the user can always *switch* to work on (view or modify) any other version. This switching feature allows the user to work on many versions at the same time during one session. This capability is called *multi-version editing*. The user may *commit* changes at any time. Upon issuing this command, the user is asked which *uncaptured, temporary versions* should be saved and the chosen versions are then saved along with any already captured versions. Only the differences are stored. Saving complete version snapshots can improve version access time.

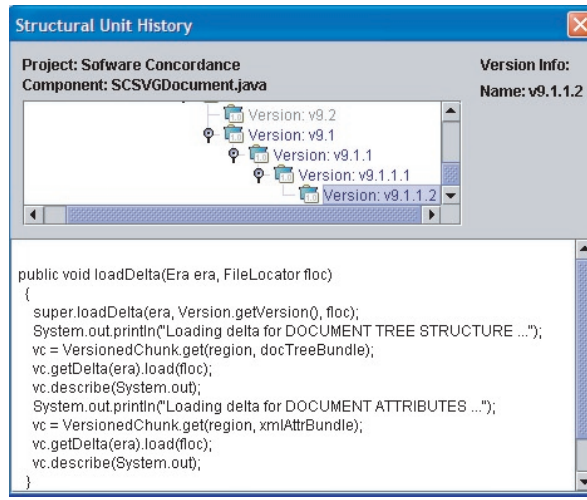

Fig. 10. History of a document node

The user can open a software document at the current version (e.g. XML, HTML, Java program, SVG graphic, UML diagram). An appropriate editor will be invoked. The editors are enhanced by versioned hypermedia services. Figure 9 shows the structured editor for a Java program. When the user right-clicks

on a document node, a popup menu is displayed to allow the user to create an anchor at the node and add it to the active hypertext network, to open the active hypertext network of an anchor defined at the document node, or to create or delete a HTML-style hyperlink at the document node.

In addition, the user is able to view a document node's history (see Figure 10). Note that the method "loadDelta" was not created until *v9.1*. Therefore, the earlier versions on the top window are "disabled". If the selected logical unit is the root of a document, the history of entire document will be displayed.

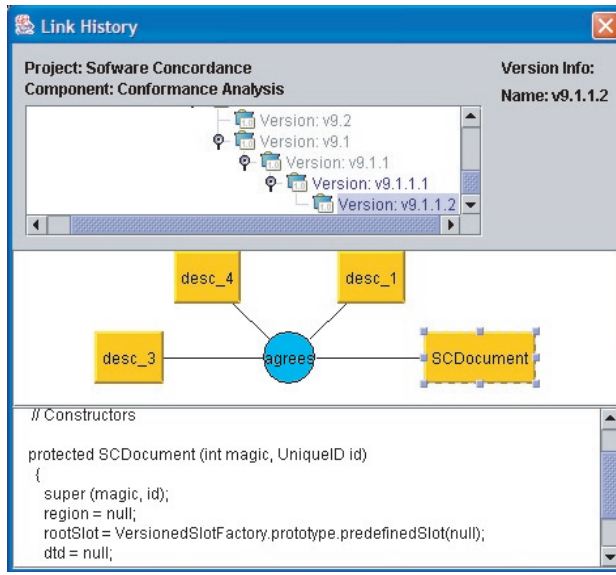

Fig. 11. History of a hyperlink

At any time, the user can create a hyperlink and make it active in a hypertext network. Then, he/she can add anchors into that hyperlink or any other hyperlinks currently managed in the system. Members of the hyperlink and its properties can be modified and changes are recorded. Figure 11 shows the history of the link "agrees" between several requirement/design items and the class "SCDocument". The constructor of the class was displayed in the bottom window since the user clicked on the corresponding anchor.

The user can also display, modify, or navigate through a hypertext network via a simple editor. In Figure 12, there are two types of links: causal and non-causal. There are directed edges coming from source anchors to a causal link, and from a causal link to its target anchors. There are only non-directed edges for non-causal links (e.g. link "n1"). While non-causal links are directly represented by our hyperlinks, causal links are extended from them with additional attributes. Via anchors, the user can traverse to corresponding document nodes.

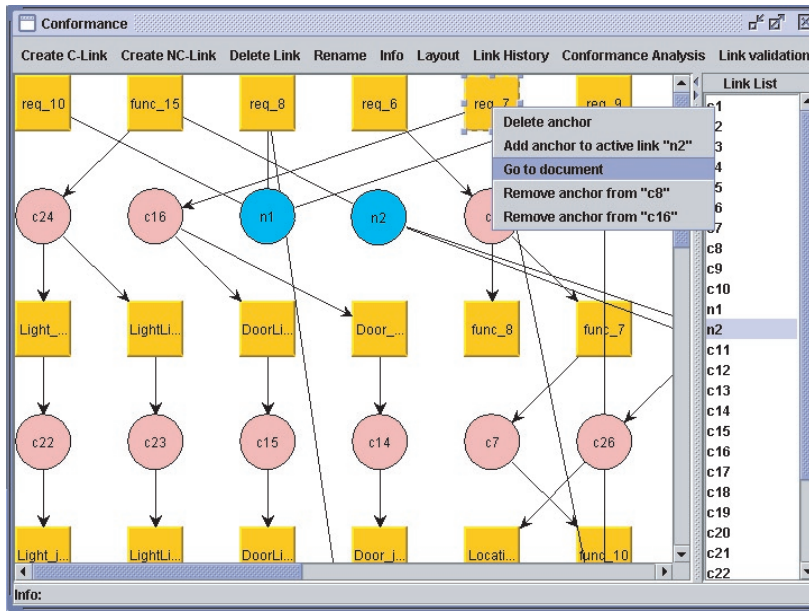

Fig. 12. A hypertext network

In this environment, multiple hypertext networks can also be defined during a software development process. With our tool, document contents will not be changed when a new network is defined since hypertext structure is separated from document content.

6 Issues for future research

While using product versioning, we have encountered three problems that have not been described for composition or total versioning systems. We plan to explore these issues in the near future.

6.1 Hyperlinks across different versions

Molhado's current interfaces are not sufficient to allow users to easily create hyperlinks between different versions. This is because links only exist within a single version. So, there is no way to say "make a link to document D at version v_k," because the linking interfaces have no parameter for the version. While this may not be the most common use case for versioned hypermedia, it is certainly necessary to support it and we have identified two approaches that should suffice.

The first solution is to allow links to other versions to be made as *computational links*. The anchors of a computational link are essentially queries against a database of links. Using the example mentioned above, the anchor's query would

be "get document D for version v_k." The second solution is to change Molhado's interfaces to include or even require a version identifier for all accesses. While this is logically possible, we have hesitated to support this choice because it is inconvenient in the most common cases.

6.2 Hyperlinks across different projects

Another similar issue concerns about how to create a hyperlink among components of different projects. Since a hyperlink is an entity that lives within a version space of a single project, with the current interfaces in Molhado, there is no way to do hyperlinking across different projects. However, the *computational link* approach should suffice to handle this. In this case, the query would ask for the opening of a different project, and loading a version and then a document.

6.3 The scope of the product versioning space

Molhado is built specifically for software engineering environments, where it makes sense for software developers to handle versions of entire software projects. Molhado's notion of a "project" is also applicable to Web-based applications or legal document management systems, representing an entire Web-based application, a Web site, or a collection of legal documents and tax codes.

However, we suspect that there will be hypertext collections that are ill-suited to being controlled in one version space. It is certainly the case that linking material maintained under Molhado's product versioning model with material managed using other versioning models will be a challenge. While this is clearly a topic for future research, we reject the idea that Molhado's apparent "incompatibility" with composition or total versioning systems is a reason to reject the product versioning model. Rather, we think that versioning needs a generalized access and maintenance model that covers all approaches.

7 Conclusions

The Molhado hypertext versioning infrastructure is well-suited for managing logical relationships among software documents. It is the first system that applies the product versioning model to hypertexts. The structural and cognitive issues that Østerbye [2] described has been well-addressed in Molhado by using product versioning. This version control model facilitates the development of a GUI for versioned hypermedia services. It reduces cognitive overhead for users in version creation and version selection of hypermedia elements in user operations. Software components and hypertext structures are uniformly versioned in a fine-grained manner. Users are allowed to return to a consistent previous state not only of a hypertext network but also of a single node and of a hyperlink. The use of product versioning for hypertexts also raises new issues that have not been described in existing versioned hypermedia systems such as creating hyperlinks across different versions or different projects. Our future plan includes exploring those issues and carrying out experimental and usability studies for the tool.

References

1. Whitehead, Jr., E.J.: An Analysis of the Hypertext Versioning Domain. PhD thesis, University of California – Irvine (2000)
2. Østerbye, K.: Structural and cognitive problems in providing version control for hypertext. In: Proceedings of the ACM conference on Hypertext and Hypermedia. (1992) 33–42
3. Nguyen, T.N., Munson, E.V., Boyland, J.T., Thao, C.: The Molhado Hypertext Versioning System. In: Proceedings of the Fifteenth Conference on Hypertext and Hypermedia, ACM Press (2004)
4. W3C: W3C XML Linking. http://www.w3c.org/XML/Linking (2005)
5. Nelson, T.H.: Xanalogical structure, needed now more than ever: parallel documents, deep links to content, deep versioning, and deep re-use. ACM Computing Surveys (CSUR) **31** (1999) 33
6. Akscyn, R.M., McCracken, D.L., Yoder, E.A.: KMS: a distributed hypermedia system for managing knowledge in organizations. Communications of the ACM **31** (1988) 820–835
7. Garg, P.K., Scacchi, W.: A hypertext system to manage software documents. IEEE Software **7** (1990) 90–98
8. Wiil, U.K., Leggett, J.J.: Hyperform: using extensibility to develop dynamic, open, and distributed hypertext systems. In: Proceedings of the ACM conference on Hypertext and Hypermedia, ACM Press (1992) 251–261
9. Ferrans, J.C., Hurst, D.W., Sennett, M.A., Covnot, B.M., Ji, W., Kajka, P., Ouyang, W.: HyperWeb: a framework for hypermedia-based environments. In: Proceedings of the Symposium on Software Development Environments, ACM Press (1992) 1–10
10. Cybulski, Reed: A Hypertext Based Software Engineering Environment. IEEE Software **9** (1992) 62–68
11. Tichy, W.F.: Design, implementation, and evaluation of a revision control system. In: Proceedings of the 6th International Conference on Software engineering, IEEE Computer Society Press (1982) 58–67
12. Asklund, U., Bendix, L., Christensen, H., Magnusson, B.: The unified extensional versioning model. In: Proceedings of the 9th Software Configuration Management Workshop, Springer (1999)
13. Delisle, N.M., Schwartz, M.D.: Contexts: partitioning concept for hypertext. ACM Trans. Inf. Syst. **5** (1987) 168–186
14. Haake, A.: CoVer: a contextual version server for hypertext applications. In: Proceedings of the ACM conference on Hypertext and Hypermedia, ACM Press (1992) 43–52
15. Haake, A., Hicks, D.: VerSE: towards hypertext versioning styles. In: Proceedings of the 7th ACM conference on Hypertext and Hypermedia, ACM Press (1996) 224–234
16. Delisle, Schwartz: Neptune: A hypertext system for CAD applications. In: Proceedings of ACM SIGMOD '86, ACM Press (1986) 132–142
17. Soares, L., Filho, G.S., Rodrigues, R., Muchaluat, D.: Versioning support in HyperProp system. Multimedia Tools and Applications **8** (1999) 325–339
18. Whitehead, Jr., E.J.: Design spaces for link and structure versioning. In: Proceedings of the conference on Hypertext and Hypermedia, ACM Press (2001) 195–204
19. Melly, Hall, W.: Version control in Microcosm. In: Proceedings of the Workshop on the Role of Version Control in CSCW. (1995)

20. Whitehead, Jr., E.J.: A proposal for versioning support for the Chimera system. In: Proceeedings of the Workshop on Versioning in Hypertext Systems, ACM Press (1994)
21. Hicks, D.L., Leggett, J.J., Nurnberg, P.J., Schnase, J.L.: A hypermedia version control framework. ACM Transactions on Information Systems (TOIS) **16** (1998) 127–160
22. Millard, D.E., Moreau, L., Davis, H.C., Reich, S.: FOHM: a fundamental open hypertext model for investigating interoperability between hypertext domains. In: Proceedings of the ACM Conference on Hypertext and Hypermedia, ACM Press (2000) 93–102
23. Wakeman, L., Lowett, J.: PCTE: the standard for open repositories. Prentice Hall (1993)
24. Streitz, N., Haake, J., Hannemann, J., Lemke, A., Schuler, W., Schutt, H., Thuring, M.: SEPIA: a cooperative hypermedia authoring environment. In: Proceedings of the ACM conference on Hypertext and Hypermedia, ACM Press (1992) 11–22
25. Conradi, R., Westfechtel, B.: Version models for software configuration management. ACM Computing Surveys (CSUR) **30** (1998) 232–282
26. Lie, A., Conradi, R., Didriksen, T., Karlsson, E., Hallsteinsen, S., Holager, P.: Change oriented versioning. In: Proceedings of the 2nd European Conference on Software Engineering. (1989)
27. Cronk, R.: Tributaries and deltas. BYTE (1992) 177–186
28. Goldstein, Bobrow: A Layer Approach to Software Design. Interactive Programming Environments. McGraw-Hill (1984)
29. Reichenberger, C.: VOODOO: A Tool for Orthogonal Version Management. In: Proceedings of the Software Configuration Management Workshop, SCM-5, Springer (1995) 61–79
30. Wagner, T.A., Graham, S.L.: Incremental analysis of real programming languages. In: Proceedings of the 1997 ACM SIGPLAN conference on Programming language design and implementation, ACM Press (1997) 31–43
31. Nguyen, T.N., Munson, E.V.: The Software Concordance: A New Software Document Management Environment. In: Proceedings of the ACM Conference on Computer Documentation, ACM Press (2003)
32. Nguyen, T.N.: Object-oriented Software Configuration Management. PhD thesis, University of Wisconsin – Milwaukee (2005)
33. Nguyen, T.N., Munson, E.V., Boyland, J.T., Thao, C.: An Infrastructure for Development of Object-Oriented, Multi-level Configuration Management Services. In: Proceedings of 27th International Conference on Software Engineering (ICSE 2005), ACM Press (2005)