Design Spaces for Link and Structure Versioning

E. James Whitehead, Jr.

University of California, Santa Cruz
E-mail: ejw@soe.ucsc.edu

ABSTRACT

This paper reflects upon existing composite-based hypertext versioning systems, and presents two high-level design spaces that capture the range of potential choices in system data models for versioning links, and versioning hypertext structure. These two design spaces rest upon a foundation consisting of a containment model, describing choices for containment in hypertext systems, and the design space for persistently recording an object's revision history, with applicability to all versioning systems. Two example points in the structure versioning design space are presented, corresponding to most existing composite-based hypertext versioning systems. Using the presented design spaces allows the data models of existing hypertext versioning systems to be decomposed and compared in a principled way, and provides new system designers significant insight into the design tradeoffs between various link and structure versioning approaches.

KEYWORDS: Hypertext versioning, link and structure versioning, containment, configuration management

INTRODUCTION

Consider software engineering. A large software project consists of many thousands of files, comprising requirements and design documents, source code, test cases, build files, bug reports, memos, email, and Web pages. There are many relationships between these files, such as a source file that satisfies a requirement stated in another document, or a test case that examines whether the code does indeed meet that requirement. Chimera [2] and DHM [16] are two examples of hypertext systems whose goal is to capture the relationships between software project files as hypertext links. Once these relationships are in the hypertext system, they allow for rapid navigation to related files, as well as visualization and analysis of the relationship network. The act of instantiating the relationships makes concrete the effect that changing a single software file can have on its network of relationships, since modifying a file can create new relationships, and can alter or destroy existing ones.

Software engineering is a domain where best common practice involves maintaining and composing previous states of the project, a problem addressed by the discipline of software configuration management [7]. Hence, the introduction of hypertext into a version controlled software project necessarily entails accommodating versioned files, and project configurations. As a consequence, the fact that existing hypertext systems for software development do not version links is a significant factor preventing their wider use in this domain.

Other domains have the same characteristics of large amounts of content, with enormous numbers of inter-relationships, where prior states of the content must be preserved. Document management is one, since collections of documents in many contexts, from aerospace engineering documents [25] to interoperability standards, have a need to be under configuration control, and possess a wide range of relationships that could usefully be captured as hypertext links. Legal documents, comprising laws, regulations, and tax codes, are an important class of documents, chock full of interrelationships. 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. This is especially relevant for tax codes, which change frequently. Hypertext support can make it easy to navigate to related laws, precedents, regulations, and codes. Audit papers, inter-related information gathered about a company creating a network of content substantiation to develop an independent opinion concerning the accuracy of its financial statements, also can benefit from the application of hypertext [10]. Due to the collaborative nature of the task, the need to freeze a state of the company's documents for analysis, along with the emergent understanding of the financial statements made by the company and the change this implies to inter-document linkages, the audit working papers, and the final audit report, hypertext versioning is necessary for the introduction of hypertext into financial audits.

Across the domains of software engineering, document management, legal, and audit, the requirement to preserve the prior states of individual documents necessarily entails that systems which capture inter-document relationships as hypertext links must also provide hypertext versioning support. Over the past fifteen years, there has been a slow but steady stream of research on hypertext versioning. *Composite-based versioning systems* comprise an important class of hypertext versioning systems, and are characterized by the use of a container object (a composite) to contain documents and the hypertext network. Prominent

composite-based systems include Neptune [8,9], CoVer [17,19,18], HyperPro [28], VerSE [20], HyperProp [33,32] and Melly's versioning support for Microcosm [26]. The Hypermedia Version Control Framework [21], though a policy-neutral toolkit, also contains composite objects and thus could be used like other composite-based systems. These systems can be viewed as having explored single points in a complex, multi-faceted space of possible design tradeoffs in accomplishing their goal of versioning hypertext networks. However, the knowledge generated has been specific to the data model and system on which the research was performed, and is difficult to apply to different systems or problem areas. As a consequence, *the application of hypertext to software engineering, document management, legal, audit, and other similar domains, is limited by the absence of systematically organized knowledge concerning hypertext versioning.*

The concerns addressed by hypertext versioning research include such issues as collaboration support, visualizing versioned spaces, merging hypertext networks, representing variants, and navigating through versioned spaces. Though these issues are important, and must be addressed in any hypertext versioning system, this paper focuses instead on the essence of hypertext versioning: how to version hypertext links, and hypertext structures. The goal of this paper is to provide a comprehensive description of the key decisions and tradeoffs involved in the design spaces for versioning links and structure. This information is conveyed in a system-independent manner that makes data modeling decisions explicit. The design spaces for link versioning, and structure versioning both depend on two additional design spaces, those of containment, and persistent storage of revision histories. Hence, these two design spaces will be presented first, and the link versioning and structure versioning design spaces will follow, building upon them.

The main contribution of this work is its systematic overview of the possible choices involved in containment, revision history versioning, versioning links and versioning structure. These design spaces allow existing hypertext versioning systems functionality to be better understood, and compared in a principled way using the design spaces as a comparison framework. The design spaces also identify approaches that have not yet been tried, and hence provide several interesting research directions. The improved understanding of hypertext versioning provided by these design spaces is expected to lead to greater use of hypertext versioning capability in domains that require version-aware hypertext capability.

The remainder of the paper is organized as follows. Definitions of common terms are given in the next section. This is followed by, in order, descriptions of the design spaces for containment, persistent storage of revision histories, link versioning, and structure versioning. A brief related work section completes the paper.

DEFINITIONS

This section contains definitions of terms that will be used in the description of the four design spaces explained in this paper. We begin with definitions of the basic hypertext abstractions, the work, anchor, link, and link structure.

Work: An artifact intended to create a communicative experience, such as a document, image, or song.

Anchor: A handle for a specific set of symbols within a work.

Link: An association among a set of works, a set of anchors, or their combination.

Link structure: A set of links. *Structure* is used in an evocative sense, to describe the graph created by this link set.

Objects persistently represent the basic hypertext abstractions.

Object: A single or aggregate data item that represents an abstraction. Objects represent abstractions such as works, anchors, links, and link structures.

The following terms are used to characterize the versioning state of an object, and the object used to capture its revision history.

Unversioned object: An object that has only one state, the current state, and modifications overwrite it.

Revision: A snapshot of an instant in the evolution of an object.

Versioned-object: An object that signifies a specific abstraction, independent of any specific revision, or instant in time. A versioned-object contains revisions of the object it signifies.

Revision selection rule: An expression that is evaluated over the members of a revision history to select one (and sometimes multiple) members of that history.

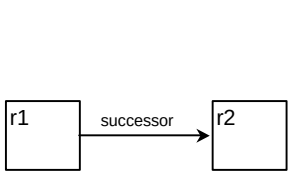
CONTAINMENT DESIGN SPACE

Containment is one of the most common relationships found in the data models of hypertext versioning systems, since static links, versioned objects, workspaces, compound documents, and user-defined collections can all be viewed as types of containers. As a result, understanding the relationships between entities in hypertext versioning systems requires an understanding of the different permutations of containment.

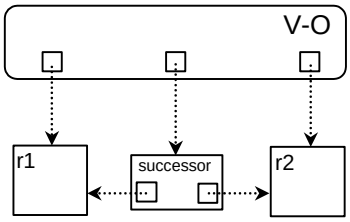
In its basic form, a container models a set where each element is an entity (an abstraction). The container is an entity that holds the set.

There are two main aspects to the containment design space:

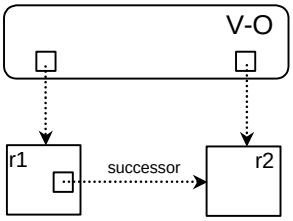
- **Abstract properties of the container:** Qualities of the container that are mathematic set properties, rather than properties of a specific computer representation, these being:
  - *Containment*: For a given entity, the number of containers that can hold it. Choices are: (a) single containment, an entity belongs to just one containment set, or (b) multiple containment, an entity belongs to multiple containment sets,
  - *Membership*: For a given container, the number of times can it contain a given entity. Choices are: (a) single membership, an entity can belong to a



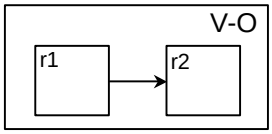
a) A linear revision history containing two revisions, r1, and r2. Revision r2 is the successor of r1.



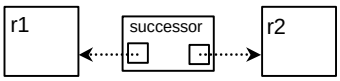
b) A versioned object, **V-O**, referentially contains two revisions and a first-class successor relationship.



c) A versioned object, **V-O**, referentially contains two revisions. Revision r1 referentially contains its successor, r2.



d) A versioned object, **V-O**, inclusively contains revisions r1 and r2, and the successor relationship.



e) Revisions r1 and r2 are independent objects, connected by a first-class link that represents the successor relationship.

Figure 1 – State based design choices for representing an object's revision history.

containment set only once, or (b) multiple membership, an entity can belong to a containment set multiple times, in which case the containment set is a bag, or multiset,

- *Ordering*: The persistent ordering of a container. Choices are: (a) ordered, the entities within the containment set have a fixed successive arrangement, or (b) unordered, the entities have no prescribed arrangement, (c) indexed, the arrangement is determined by a specification based on entity values or metadata, (d) grouped, subsets of members are ordered, but between subsets there is no ordering.

- **Containment type:** How containment relationships are represented: (a) inclusion, or (b) referential (both described below).

Broadly, there are two ways to represent that a container contains a particular entity. The container can physically include the contained item, or it can use an identifier as a reference to its members. The former case is known as inclusion containment; the latter, referential. Whenever two entities have a containment relationship between them, this relationship can be represented using references, following the permutations of identifier storage. The identifier can be stored on the container, on the containee, or on both. Additionally, identifier storage can be delegated to a separate entity, a first-class relationship. However, since the first-class relationship is itself a container, the same permutations of identifier storage apply between the container and one endpoint of the relationship, and between the containee and the other endpoint. Typically the first-class relationship holds identifiers for both the container and containee.

PERSISTENT STORAGE OF REVISION HISTORIES

Objects are used to represent such abstractions as works, anchors, links, containers, workspaces, etc., within the computer. This section presents the *revision history design space*, describing several different techniques for recording the revision history of objects. The revision history design space is foundational, since it describes the basic techniques for recording the evolution over time of works, anchors, links, containers, and workspaces. These approaches create an archive of past revisions, and add the dimension of time where there was previously only the current time. The design space of structure versioning directly builds on this design space, since it depends on exactly how works, anchors, and links are versioned.

Limiting the inquiry just to state-based versioning approaches (i.e., not considering change-based systems such as PIE [15], where changes are the primary objects instead of revisions), there are several approaches for recording the revision history of an object: these are the versioned-object, within-object, and predecessor/successor relationship only approaches, described below.

Versioned objects:

In this approach, each revision is a separate object that is *referentially* contained within a versioned object, a container that holds all revisions of the object (shown in Figures 1b and 1c). Links or relationships record the predecessors and successors of revision objects. The containment relationship between the revision objects and the versioned object must be a reference type, typically a containment relationship on the container object, i.e., where the container holds the containee's identifier. An advantage of this approach is the ability to record metadata both on the versioned object and on the individual revisions.

There is also a range of choices for how to represent the predecessor/successor relationships. Since each revision can be viewed as having a set of predecessors, and a set of successors, the containment design space is applicable. Since the versioned object approach requires each revision to be a separate object, an individual revision cannot use inclusion containment for its predecessors and successors. However, all of the referential containment types could be

employed. It is possible for each revision to store the predecessor/successor relationships on the revision. It is also possible for revisions to hold only the predecessor relationships, wherein the predecessor relationships on child revisions do double duty as the successor relationships for its parent. First class relationship objects, or hypertext links, can also be used (see Figure 1b), and have the advantage that the revision objects do not need to store predecessor/successor information, and could potentially participate in multiple version histories. First class relationship objects must themselves be contained within the versioned object, and any reference containment type can be used for this.

When revisions are contained using referential containment, the revisions can belong to containers other than the versioned object, such as user-created containers (e.g., folders or directories), workspaces, and configurations. When the container contents themselves are versioned, referential containment allows revisions to be reused across revisions of these versioned containers, thus resulting in fewer objects than would occur if this reuse wasn't possible, and the revisions needed to be copied to belong to each container revision.

Systems that contain revisions inside versioned objects using reference containment include HyperPro [28], CoVer [18], HyperProp [32], HyperForm [37], HyperDisco [38] and the Hypermedia Version Control Framework [21].

Within-object versioning

In this scheme, the versioned object uses *inclusion* containment to hold revisions, and hence all revisions are *within* the versioned object. Thus, *within-object versioning differs from the versioned object approach in the type of containment used, within-object versioning employing inclusion, and versioned object using referential types.* Within-object versioning is shown in Figure 1d. Examples include the ",v" files of RCS [34] and the ".s" files of SCCS [30], the versioning capability of some word processors (e.g., Microsoft Word), along with Palimpsest [11], VTML [36,4], EH [13], P-Edit [23], MVPE [31], Historian [1], VE [3], Timewarp [12], and Delta [6]. While these systems all share the quality of inclusively containing all revisions, their concrete representations vary significantly.

The advantage of this technique is that all revisions are stored within a single object, and it is possible to guarantee the stability of references within these objects, since the current location of an endpoint can always be computed. When changes are recorded down to the keystroke level, within-object versioning can support remote collaborative authoring where all collaborators simultaneously work on the document, since all operations by all collaborators are recorded. Recording all revisions can be a drawback, since a publicly accessible document might not want to reveal all of its prior revisions.

Within-object versioning has the drawback that revisions cannot participate in other containment structures, unless a replica of a specific revision is made and then placed into the container. Alternately, it is possible to use referential containment to hold the entire versioned object, and its included revisions, within multiple containers.

Two significant design choices for within-object revision are the size of the minimum length content chunk, and the range of attributes that can be set on each chunk. In contrast to the attributes settable on whole objects, attributes set on content chunks have much finer granularity, since they are applied to subparts of the entire object. Chunk size varies, with the largest minimum chunk size being a programming language function [1], but with other choices being a single line (e.g., C preprocessor), a programming language token [6], all the way on down to a single character [11,36]. Settable attributes always include the person who made a change and the time when the change was made. Other common attributes include the revision number of the change, and a rationale/comment field. Most systems limit settable attributes to those that are predefined by the system, however some provide the ability to set arbitrary attributes, and retrieve them using predicates. Arbitrarily settable attributes allow within-object versioning systems to also handle within-object variant representation tasks.

Predecessor/successor relationships only

Each revision is stored in a separate object, but no container object represents a particular versioned object (see Figure 1e). Predecessor and successor relationships exist between revisions in a version history. Typically a repository, or super-container holds all revisions of all objects, as well as all relationships between them, in a large pool of objects. The only information available to determine that a subset of the object pool comprises a version history is the predecessor and successor relationships between object/revisions.

The advantage of this approach is that it can support versioning without using container objects. In conjunction with a decentralized name or addressing scheme, it can model revision histories that span organizational and machine boundaries, since it avoids the issue of which machine hosts the collection representing a versioned object. However, when revision histories span organizational boundaries, referential integrity is a potential problem, as communication and coordination between the machines storing the individual revisions cannot be guaranteed. Disadvantages of the approach include inefficient revision selection, hence inefficient creation of arbitrary configurations, and inefficient setting of metadata that must be unique across the version history of an object, such as labels. Examples of this approach include Xanadu [27], and the NTT Labs. versioning proposal [29].

LINK VERSIONING DESIGN SPACE

Using the design space for persistent storage of revision histories, it is now possible to concisely characterize the design space for link versioning. The representation of links typically takes one of two forms: either the link is an independent object, or the link is contained within an object representing a work. Independent links have the entire object versioning design space available for representing the revision history of a link. When the link is contained within an object representing a work, it has greater constraints on how it can be versioned; typically link versioning is a side effect of work versioning.

Independent links

As an independent system object, a link's history can be recorded using any of the techniques for recording the revision history of objects. That is, the versioned object, within-object versioning, and predecessor/successor relationship approaches (shown in Figure 1) could potentially be used.

For links, the versioned object approach is typically used, wherein a container object referentially contains all revisions of the link. The versioned object approach has the advantage that it permits the creation of composites containing a consistent set of documents and links, such as the most recent revision as of a specific time, or a specific snapshot in the development of the composite. By-reference containment also allows the creation of containers that model a link structure by containing a single revision of multiple links, in this way capturing a link structure [21].

Within-object versioning has the advantage of only needing one object to record all revisions of a single link. It has the drawback of making composite creation more difficult, since either the entire versioned object would need to be contained, or the individual revisions would need to be copied out of the versioned object and placed into the container. An important design choice for within-object versioning is the size of the minimum length content chunk. This is less important for link versioning. Fine-grain change tracking, as provided by VTML [36] and Palimpsest [11], typically does not provide much value for links, since they have minimal content beyond the link endpoints, and hence do not justify the added complexity of such change tracking. However, if a link has significant chunks of metadata, such as an annotation, fine-grain change tracking of these textual metadata items could be valuable.

Using only predecessor and successor relationships to capture the revision history of links is also possible. This would eliminate the need for a container object representing all revisions of the link, and would permit link revisions to more easily span control boundaries. It has the drawback that it is difficult to efficiently evaluate revision selection rules. No existing hypertext system versions its links in this way.

Links as a dependent part of works

In some hypertext systems, links are contained within, and hence dependent upon, the objects representing linked works. Links can be embedded within that portion of the work object representing the content of the work, as is the case with HTML links on the Web. Alternately, links can be contained as metadata about the work object content, as is the case with the source link in WebDAV [14].

When links are contained within a work object's content, their history is the same as the content, and hence when the work object content has a new revision made, so too do the links in the content. The versioning of links is completely subsidiary to the versioning of the content. When links are contained as metadata, there are two choices. First, if the metadata is versioned along with the rest of the object, then link versioning is again subsidiary to versioning of the object. However, it is conceivable that metadata could be versioned separately from the main object, with each item of metadata possessing its own revision history. This has the advantage that metadata items, like links, can conceptually be part of the object, but still have independent version histories. The drawback is that this makes the object substantially more complex, since each item of metadata can contain multiple revisions. No existing hypertext system provides metadata versioning services.

STRUCTURE VERSIONING DESIGN SPACE

Abstractly, structure versioning is the act of maintaining the revision history of a link set. Since a set is represented within the computer by a container, the essence of structure versioning is placing a set of links into a container, termed the *structure container*, and then versioning the structure container. This premise underlies the structure versioning design space. The existence of the structure container means the containment design space will be brought to bear, and the need to version the structure container brings in the versioning design space as well. The structure versioning design space thus depends on the existence of these other two design spaces for the terms used to describe its own design choices.

Two criteria determine whether a particular structure versioning design choice is complete. The *symbolic rendition criterion* asserts that there must be sufficient information to create a symbolic rendition (i.e., a view, such as a screen display of a document and its link endpoints) of each work, including rendition of anchors or link endpoints. So, if the structure container does not include works, then the links, anchors, collections, and revision selection rules held by the structure container must possess enough information to connect links to the works. If works are part of the structure container, then among the works, links, anchors, and revision selection rules, there must be enough information to connect a specific link revision to a specific work and/or anchor revision.

The *link traversal criterion* asserts there must be sufficient information to perform a link traversal from an anchor. If anchors are not part of the system's data model, then there must be sufficient information to traverse a link from the symbolic depiction of a link endpoint (e.g., some symbol that represents the endpoint of a link that connects entire works).

The primary elements of the structure design space are:

What does the structure container hold?

While the structure container must, at minimum, contain links, it is by no means limited to them. The structure container may also hold works, anchors, and other container objects. If the structure container only contains links, it is capable only of representing a link structure. If the structure container also holds works and anchors, it can represent not only the link structure, but also a consistent slice through a hypertext, holding not just the links, but also the linked works, along with the anchor points within those works. It is also possible that the structure container will only hold works, if links are a dependent part of works. While this approach similarly permits versioning of a

consistent slice through a hypertext, it has the significant drawback of not versioning the structure independent of the content. This, in turn, makes it substantially more difficult to maintain multiple versioned structures over a set of works.

If the goal is just to version structure, then the structure container need only contain sufficient information to satisfy both completeness criteria. So, if a link endpoint specifies an anchor revision, and an anchor revision specifies a work revision, then the structure container need only contain links, since it is possible, given a link revision, to determine the information needed to create a symbolic rendition, and to perform a link traversal. If, however, a link endpoint only specifies a versioned object, either the structure container or the containment relationship between the link and its containees (works or anchors) must hold a revision selection rule (discussed further below) that selects a specific revision, and hence satisfies the completeness criteria.

Since a link in conjunction with a revision selection rule contains sufficient information to satisfy the two completeness criteria, these criteria alone do not provide any motivation for adding works, anchors, or other container objects (e.g., collections, composites) into the structure container. However, most composite-based hypertext versioning systems do include works and links within composites, which act as structure containers. Here the motivation is to make the composite do dual duty, as both the structure container and as a workspace. Workspaces provide the benefit of maintaining an internally consistent subset of the entire object space.

This point in the structure versioning design space is fully specified by giving a complete list of the entities contained by the structure container.

Versioning design space choice for structure container and its containees.

For the structure container, and all of its containees, one of the choices of the versioning design space—versioned object, within-object versioning, predecessor and successor relationships—must be made. While it is mandatory for the structure container to be versioned in order to provide structure versioning, versioning for containees is optional. For example, both HyperPro [28] and HyperProp [32,33] do not version links individually, and version structure by placing the links inside containers that are versioned, and therefore each revision of the container records a specific revision of the link structure.

This point in the structure versioning design space is fully specified by listing, for the structure container and each of its contained entities, the choice of versioning mechanism employed to record the revision history of the entity. If the entity is not versioned at all, that is noted instead.

Containment design space choice for all pairs of containers and containees.

For each container/containee pair involved in structure versioning, their containment relationship needs to be specified by choosing a point in the containment design space. This applies not just to every object contained by the structure container, but also to the endpoints of links (since links are modeled as containers), and to the containment relationship between anchors and their work objects. Furthermore, for each container/containee pair, if the containee is versioned using either the versioned object or within-object approach, it is necessary to determine whether the versioned object, or an individual revision, is contained.

Of the many choices inherent in each containment relationship, whether the containment type is inclusion or referential has the greatest impact on structure versioning, since inclusion containment implies that versioning of the contained item is dependent on versioning of the container. Referential containment leaves the containee free to have a revision history that is independent of the revision history of its container.

If the structure container only allows single containment of its objects, this leads to significant object duplication during the evolution of the structure, since every revision of the structure container constitutes a separate container, and singly contained objects can only belong to one. As a result, new structure container revisions that employ single containment must replicate all contained objects when a new revision, or working copy is made. For example, this is the case with Neptune [9].

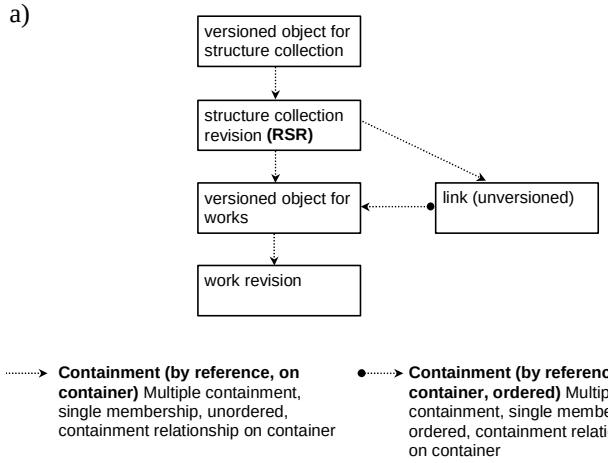
When the structure container holds the link and a single revision of its endpoint objects (anchor, work, or both), and the revision selection rule is located on the structure collection, an additional dynamic containment choice becomes available. Typically, a link referentially contains its endpoints, selecting a specific revision of endpoint objects using the revision selection rule. The structure container also evaluates the revision selection rule to select an individual revision of each endpoint object (anchor or work). Thus, the revision selection step is duplicated by the link and the structure collection. To avoid this duplication, the link could employ *indirect referential containment*, where the link endpoint is the revision selected by the structure container. That is, the link endpoint is a binding point that is filled-in by the revision selection rule evaluation of the structure collection.

Location and scope of revision selection rule.

Dynamic containment using a revision selection rule is often used in structure versioning, providing several benefits. Revision selection rules may be located either in the structure container, or on a specific containment relationship. When located on the structure container, the scope of the rule is all containment relationships where the containee is a versioned object (i.e., the container used in either the versioned object or within-object versioning approaches). When located on a single containment relationship, its scope is that relationship. The advantage of having the revision selection rule on the structure container is evaluation efficiency, and ease of maintenance. The advantage of having revision selection rules on each containment arc is flexibility, with each containment arc permitting a separate revision selection rule.

Revision selection rules bring several benefits. They allow more expressive selection of revisions than just explicit

a)



b)

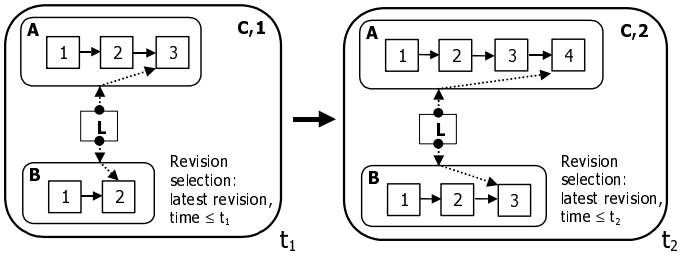


Figure 2(a) shows a containment diagram for example 1, versioned structure with unversioned links. The structure collection revision contains a revision selection rule (RSR). Figure 2(b) shows an unversioned link, L, between two versioned works, A, and B, in a versioned structure collection C.

selection by revision identifier, permitting selection such as "most recent revision," or "most recent as of a specific time," or, in combination with a human-readable label, "the revision with label *Beta_Release_2*".

Revision selection rules also allow a single unversioned link to refer to different revisions over time. This trick is accomplished by having the link endpoint be a versioned object, and then using the revision selection rule to select a specific revision. Since the rule is stored separate from the link, the rule, and hence the selected revision, can change without modifying the link. If the holder of the revision selection rule is versioned, the link then has the appearance of being versioned, since the selected revisions change over time with the revision selection rule. Example 1, below, describes this approach in more detail.

Finally, revision selection rules on the structure container provide a single modification point for changing its contained revisions, a useful trait when performing time-based revision selection or label-based revision selection. Label-based revision selection has the additional benefit of creating internally consistent hypertext structures, assuming the hypertext was consistent when the labels were applied.

This point in the structure versioning design space is complete when, for each container/containee pair (including the structure container and its containees, as well as links and their contained endpoints), where the ultimate containee is a revision, a decision is made whether a revision selection rule on the structure collection, or on the particular containment relationship, determines the revision endpoint of the containment arc. Alternately, if no revision selection rule is employed, meaning a specific revision is explicitly selected, this is noted as well.

STRUCTURE VERSIONING EXAMPLES

The following sections provide two examples that highlight use of the structure versioning design space.

Example 1: Versioned Structure with Unversioned Links

In this example, the following choices were made within the structure versioning design space:

- Abstractions present: structure collection, works, links. No anchors are present, links join whole works.
- Structure container contains: links, work versioned objects
- Versioning design space choices:
  - Structure containers are versioned, using versioned object approach
  - Links are *unversioned*
  - Works are versioned, using versioned object approach
- Containment design choices:
  - Structure container → link, work versioned object: *referential*, multiple containment, single membership, unordered, containment relationship on structure container
  - Link → work versioned object: *referential*, multiple containment, single membership, *successively ordered*, containment relationship on link (container)
- Revision selection rule: stored on collection, affects all link endpoints, provides selection of specific work revision from work versioned object.

A containment diagram showing these design choices is shown in Figure 2a. An instance of this containment structure is shown in Figure 2b. In the figure, the structure container, C, referentially contains two versioned work objects, A and B. By using the versioned work object as the link endpoint, and then letting the revision selection rule choose the specific revision, the need to duplicate link L to point to newly created revisions is eliminated. Including time in the revision selection rules allows prior link endpoints to be recovered when reverting to a prior container revision. This approach achieves structure versioning using unversioned links. Essentially, this approach stores separate revisions of links without explicitly recording their predecessor and successor relationships. Instead, the predecessor and successor relationships for the links are implicitly recorded by the structure container's revision history, since the unversioned

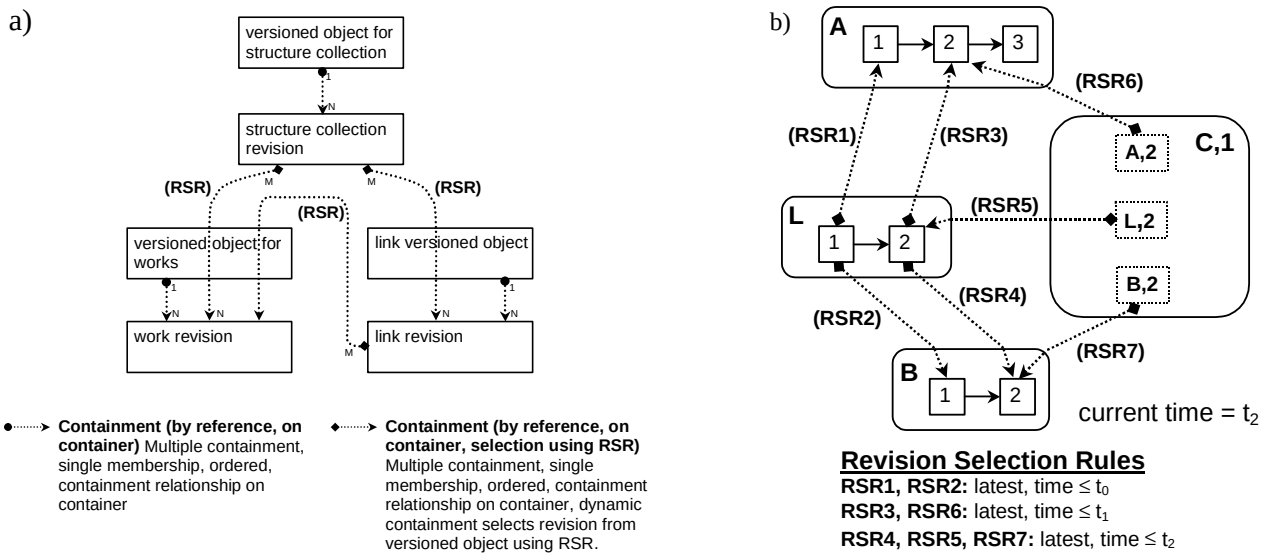


Figure 3a shows the containment diagram for example 2, versioned structure with versioned links. Figure 3b is an example of this containment structure, with a structure container C holding revision 2 of link L, and works A and B.

links are contained by the structure container. As a result, each revision of the structure container records a specific revision of the link structure. This approach has the drawback that it is not possible to efficiently evaluate revision selection rules across the revisions of a specific link. This approach is used by both HyperPro [28] and HyperProp [32,33], because they focus on recording the history of link structure, not individual links.

The symbolic rendition completion criteria is met, since each revision of the structure container holds a versioned object for each work, a revision selection rule that selects the revision to display, and all links. Thus, for each work, all information is present that is needed to create a symbolic rendition. The link traversal criteria is also met, since the link endpoints, work versioned objects, are also present in each revision of the structure collection, and the revision selection rule chooses the specific work revision for each endpoint.

Variants of this unversioned link approach are possible, but have not been explored in the existing literature. For works, it is possible that within-object versioning could be used, since the structure container holds the entire versioned object, and it permits efficient evaluation of revision selection rules. Use of the predecessor and successor relationships approach for versioning works is not compatible, since links require efficient evaluation of revision selection rules.

There is really only one place where the revision selection rule can be stored, and that is the structure container. The link cannot store the rule, since that would entail creating a new link for every revision of the structure container in order to preserve its selected revision at the time the container was frozen. If the work versioned objects store the rule, it is impossible to freeze the rule when a new structure container revision is made, since a new revision of the structure container does not imply a new revision of its contained versioned objects, since they are unversioned.

Example 2: Versioned Structure with Versioned Links

In this example, the following choices were made within the structure versioning design space:

- Abstractions present: structure collection, works, links. No anchors are present, links join whole works.
- Structure container contains: link revisions, work revisions
- Versioning design space choices:
  - Structure containers, links, and works are versioned, using versioned object approach
- Containment design choices:
  - Structure container → link revision, work revision Link → work revision: *referential*, multiple containment, single membership, *successively ordered*, containment relationship on link (container), dynamic containment via revision selection rule over work versioned object.
- Revision selection rule: stored on containment arc between structure container and its containees, providing selection of link revisions from link versioned objects and selection of specific work revisions from work versioned objects.

Figure 3a shows the containment diagram for these structure versioning design choices. The distinguishing element of this example is its use of versioned links, and the use of revision selection rules on all containment arcs of the structure collection revisions, and link revisions. An example instance of this structure versioning approach is shown in Figure 3b. In essence, this is the structure versioning approach used by CoVer [17,18], and VerSE [20].

While the placement of revision selection rules on containment arcs yields excellent revision selection flexibility, it also has two drawbacks. First, evaluation of individual revision selection rules is less efficient than

evaluation of one rule for all containment arcs. Second, it increases the work that must be performed to ensure the structure container holds a consistent hypertext. In Figure 3b, consider if RSR3 were changed to be "latest, time ≤ t_2" and hence L,2 selected A,3 instead of A,2. A link traversal across L,2 starting from its other endpoint, B,2, would result in the display of A,3 even though the structure container currently holds A,2. A user might perceive this as inconsistent.

RELATED WORK

Within hypertext versioning Fabio Vitali has written the only published survey, a short paper that belongs to a special issue of ACM Computing Surveys on hypertext [35]. This paper provides a good overview of hypertext versioning, presenting its advantages for history recording, work accountability, collaboration, and reference permanence. Despite being a good introduction to the hypertext versioning literature, this paper does not contain a detailed survey of versioning data models or design spaces. Of course, that was not its objective.

The Hypermedia Version Control Framework by Hicks et al. presents the HURL data model, and a conceptual architecture for hypertext versioning in open hypertext systems [21]. HURL comprehensively describes the data modeling issues inherent in hypertext versioning, and, like this paper, is based on an analysis of existing hypertext versioning systems. HURL extends the SP3/HB3 [24] data model with versioning capabilities. With judicious specialization (via subtyping) of HURL concepts, it has equivalent expressive power to the model presented in this paper. While the current paper emphasizes design choices as a means to characterize design spaces, [21] presents this information as a series of issues. An additional difference is the HURL model has been validated by implementation in a running system.

Outside of hypertext versioning, two survey articles are relevant to hypertext versioning: the Conradi and Westfechtel survey of versioning data models [5] in versioning and configuration management systems, and the Katz survey of versioning in engineering database systems [22]. In addition to the predominantly state-based versioning, where each revision has distinct, persistent identity, which Conradi and Westfechtel term *extensional* versioning, they also discuss *intensional* versioning, where revisions are constructed from property-based descriptions. They also provide a more detailed discussion of change-based versioning, and a taxonomy of versioning data models. However, their taxonomy is not based on a strong containment model, and does not clearly differentiate between versioned-object, within-object, and predecessor/ successor relationships only approaches.

Many of the versioning issues encountered in hypertext versioning and configuration management are also found in engineering database systems that support the development of integrated circuits. Randy Katz performed a substantive survey of the version data models in engineering database systems [22]. Similar to this work, Katz's survey used as basic modeling primitives derivation (predecessor/ successor), composition (containment), and variant (is-kind-of) relationships. These were then used to show how engineering database systems model various versioning and work scenarios. The survey provides a set of unified terminology, a unified data model, and a high-level conceptual architecture. However, containment in the Katz survey is just a simple "is-part-of" relationship, and does not involve a distinction between inclusion and reference containment. Additionally, all relationships in [22] are predefined, and there is no abstraction that corresponds to the hypertext link, or link structure.

CONCLUSIONS

Building on the design spaces for containment, persistent storage of revision histories, and link versioning, the structure versioning design space concisely describes a range of techniques for recording the history of hypertext link structures. The major aspects of this space are a determination of the objects contained by the structure container, the versioning design space choice for the structure container and its containees, the containment design space choice for all container/containee pairs, and the location and scope of revision selection rules. Where previously composite-based hypertext versioning systems have explored only individual points, the structure versioning design space teases out their commonality, providing a map of design possibilities and a coherent model for describing the structure versioning capabilities of these systems. The two examples of the structure design space encompass the design choices of most existing composite-based hypertext versioning systems, and serve as validation of the structure design space.

This work's primary motivation is to enhance the state of hypertext versioning knowledge so that future engineers creating systems for use in software engineering, document management, audits, law, and archives, would be able to quickly learn what is known about link structure versioning. This knowledge would allow them to add hypertext capabilities to systems that otherwise would not, due to the lack of understanding concerning the interactions of links and the versioned objects that populate these systems. Since the difficulty of versioning link structures has limited the application of hypertext in many domains, it is hoped this work will extend the utility of hypertext systems.

ACKNOWLEDGMENTS

I would like to thank my dissertation committee, Richard N. Taylor (Chair), David S. Rosenblum, and Mark S. Ackerman, for their feedback on those portions of this paper that derive from my PhD dissertation. Discussion and comments from David Hicks dramatically increased my understanding of the HURL model and helped refine the paper.

REFERENCES

[1] M. Abu-Shakra and G. L. Fisher, "Multi-Grain Version Control in the Historian System," *Proc. SCM-8*, Brussels, Belgium, July 20-21, 1998, pp. 46-56.

[2] K. M. Anderson, R. N. Taylor, and E. J. Whitehead, Jr., "Chimera: Hypertext for Heterogeneous Software

Environments," *Proc. ECHT'94*, Edinburgh, Scotland, Sept. 18-23, 1994, pp. 94-107.

[3] D. L. Atkins, "Version Sensitive Editing: Change History as a Programming Tool," *Proc. SCM-8*, Brussels, Belgium, July 20-21, 1998, pp. 146-157.

[4] L. Bendix and F. Vitali, "VTML for Fine-Grained Change Tracking in Editing Structured Documents," *Proc. SCM-9*, Toulouse, France, Sept. 5-7, 1999, pp. 139-156.

[5] R. Conradi and B. Westfechtel, "Version Models for Software Configuration Management," *ACM Computing Surveys*, vol. 30, no. 2 (1998), pp. 232-282.

[6] J. O. Coplien, D. L. DeBruler, and M. B. Thompson, "The Delta System: A Nontraditional Approach to Software Version Management," *Proc. International Switching Symposium*, Phoenix, Arizona, March, 1987, pp. 181-197.

[7] S. Dart, "Concepts in Configuration Management Systems," *Proc. Third Int'l Workshop on Software Configuration Management*, Trondheim, Norway, June, 1991, pp. 1-18.

[8] N. Delisle and M. Schwartz, "Neptune: A Hypertext System for CAD Applications," *Proc. Int'l Conference on the Management of Data (SIGMOD'86)*, Washington, DC, May 28-30, 1986, pp. 132-143.

[9] N. M. Delisle and M. D. Schwartz, "Contexts-A Partitioning Concept for Hypertext," *ACM Transactions on Office Information Systems*, vol. 5, no. 2 (1987), pp. 168-186.

[10] L. DeYoung, "Hypertext Challenges in the Auditing Domain," *Proc. Hypertext'89*, Pittsburgh, PA, Nov. 5-8, 1989, pp. 169-180.

[11] D. G. Durand, "Palimpsest: Change-Oriented Concurrency Control for the Support of Collaborative Applications," Ph.D. Dissertation. Boston University, Boston, MA, 1999.

[12] W. K. Edwards and E. D. Mynatt, "Timewarp: Techniques for Autonomous Collaboration," *Proc. CHI'97*, Atlanta, GA, March 22-27, 1997, pp. 218 - 225.

[13] C. W. Fraser and E. W. Myers, "An Editor for Revision Control," *ACM Transactions on Programming Languages and Systems*, vol. 9, no. 2 (1987), pp. 277-295.

[14] Y. Goland, E. J. Whitehead, Jr., A. Faizi, S. Carter, and D. Jensen, "HTTP Extensions for Distributed Authoring -- WEBDAV," Microsoft, U.C. Irvine, Netscape, Novell. Internet Proposed Standard Request for Comments (RFC) 2518, Feburary, 1999.

[15] I. P. Goldstein and D. P. Bobrow, "A Layered Approach to Software Design," in *Interactive Programming Environments*, D. R. Barstow, H. E. Shrobe, and E. Sandewall, Eds. New York, NY: McGraw-Hill, 1984, pp. 387-413.

[16] K. Grønbæk, "Composites in a Dexter-Based Hypermedia Framework," *Proc. ECHT'94*, Edinburgh, Scotland, Sept. 18-23, 1994, pp. 59-69.

[17] A. Haake, "CoVer: A Contextual Version Server for Hypertext Applications," *Proc. ECHT'92*, Milano, Italy, Nov. 30-Dec. 4, 1992, pp. 43-52.

[18] A. Haake, "Under CoVer: The Implementation of a Contextual Version Server for Hypertext Applications," *Proc. ECHT'94*, Edinburgh, Scotland, Sept. 18-23, 1994, pp. 81-93.

[19] A. Haake and J. Haake, "Take CoVer: Exploiting Version Support in Cooperative Systems," *Proc. InterCHI'93 - Human Factors in Computer Systems*, Amsterdam, Netherlands, April, 1993, pp. 406-413.

[20] A. Haake and D. Hicks, "VerSE: Towards Hypertext Versioning Styles," *Proc. Hypertext '96*, Washington, DC, March 16-20, 1996, pp. 224-234.

[21] D. L. Hicks, J. J. Leggett, P. J. Nürnberg, and J. L. Schnase, "A Hypermedia Version Control Framework," *ACM Transactions on Information Systems*, vol. 16, no. 2 (1998), pp. 127-160.

[22] R. H. Katz, "Toward a Unified Framework for Version Modeling in Engineering Databases," *ACM Computing Surveys*, vol. 22, no. 4 (1990), pp. 375-408.

[23] V. Kruskal, "Managing Multi-Version Programs with an Editor," *IBM Journal of Research and Development*, vol. 28, no. 1 (1984), pp. 74-81.

[24] J. J. Leggett and J. L. Schnase, "Viewing Dexter with Open Eyes," *Communications of the ACM*, vol. 37, no. 2 (1994), pp. 76-86.

[25] K. C. Malcolm, S. E. Poltrock, and D. Schuler, "Industrial Strength Hypermedia: Requirements for a Large Engineering Enterprise," *Proc. Hypertext'91*, San Antonio, Texas, Dec. 15-18, 1991, pp. 13-24.

[26] M. Melly and W. Hall, "Version Control in Microcosm," *Proc. Workshop on the Role of Version Control in CSCW (held with ECSCW'95)*, Stockholm, Sweden, September, 1995.

[27] T. H. Nelson, *Literary Machines*, 93.1 ed. Sausalito, CA: Mindful Press, 1981.

[28] K. Østerbye, "Structural and Cognitive Problems in Providing Version Control for Hypertext," *Proc. ECHT'92*, Milano, Italy, Nov. 30-Dec. 4, 1992, pp. 33-42.

[29] K. Ota, K. Takahashi, and K. Sekiya, "Version management with meta-level links via HTTP/1.1," (1996). Internet-Draft (expired), accessed Nov., 1999, www.ics.uci.edu/pub/ietf/webdav/draft-ota-http-version-00.txt.

[30] M. J. Rochkind, "The Source Code Control System," *IEEE Transactions on Software Engineering*, vol. 1, no. 4 (1975), pp. 364-370.

[31] N. Sarnak, R. Bernstein, and V. Kruskal, "Creation and Maintenance of Multiple Versions," *Proc. International Workshop on Software Version and Configuration Control*, Grassau, Germany, 1988, pp. 264-275.

[32] L. F. G. Soares, G. L. d. S. Filho, R. F. Rodrigues, and D. Muchaluat, "Versioning Support in the HyperProp System," *Multimedia Tools and Applications*, vol. 8, no. 3 (1999), pp. 325-339.

[33] L. F. G. Soares, N. L. R. Rodriguez, and M. A. Casanova, "Nested Composite Nodes and Version Control in an Open Hypermedia System," *Int'l Journal on Information Systems*, vol. 20, no. 6 (1995), pp. 501-520.

[34] W. F. Tichy, "RCS - A System for Version Control," *Software-Practice and Experience*, vol. 15, no. 7 (1985), pp. 637-654.

[35] F. Vitali, "Versioning Hypermedia," *ACM Computing Surveys* vol. 31, no. 4 (1999).

[36] F. Vitali and D. G. Durand, "Using Versioning to Support Collaboration on the WWW," *Proc. WWW4*, Boston, MA, November, 1995, pp. 37-50.

[37] U. K. Wiil and J. J. Leggett, "Hyperform: Using Extensibility to Develop Dynamic, Open and Distributed Hypertext Systems," *Proc. ECHT'92*, Milano, Italy, Nov. 30-Dec. 4, 1992, pp. 251-261.

[38] U. K. Wiil and J. J. Leggett, "The HyperDisco Approach to Open Hypermedia Systems," *Proc. Hypertext '96*, Washington, DC, March 16-20, 1996, pp. 140-148.