Reconciling Versioning and Context in Hypermedia Structure Servers

Jon Griffiths, David E. Millard, Hugh Davis, Danius T. Michaelides, Mark J. Weal

Intelligence, Agents, Multimedia Group
Dept. of Electronics and Computer Science, University of Southampton, U. K.
{jpg96r, dem, hcd, dtm, mjw}@ecs.soton.ac.uk

Abstract. Contextual structure servers and versioning servers share a similar goal in allowing different views on a stored structure according to the viewer's perspective. In this paper we argue that a generic contextual model can be used to facilitate versioning. In order to prove our hypothesis we have drawn on our experiences with OHP-Version to extend FOHM's contextual model.

1 INTRODUCTION

There is a long standing tradition of versioning structure within hypermedia systems [16, 7, 14]. Versioning provides a failsafe baseline for experimentation. Hypertext authors can rollback the hypertext network to its previous state after changes have been made to it. It is also useful for supporting hypermedia collaborative work when needing to freeze versions where many users are working on a shared resource. Versioning is particularly useful in application areas where inter-document relationships cannot afford to be lost, such as in legal and audit documents.

More recently there has been work focusing on contextual structure servers [10, 1]. These dynamically generate different views of complex structure (such as a hypermedia linkbase) according to the context of the viewer.

A contextual server allows for many different views on a given structure, depending on the viewer's position in *n* dimensional context space (examples of contextual dimensions include: age, expertise, goals, interaction history, etc.). Since time can be viewed as just one of these contextual dimensions it seems intuitive that any complete contextual model should be powerful enough to drive the selection mechanism of a versioning server.

This is not just functional re-use. It recognises that versioning is a specific kind of contextualisation and that there is an equivalence between different versions of an object and different contextual views of an object.

It is our opinion that reconciling versioning and contextualisation mechanisms would result in a more natural and consistent approach to both problems.

1.1 Objectives

At Southampton we have been experimenting with a contextual structure server known as Auld Linky [9]. It is a simple stand-alone server that stores hypermedia structure ex-

pressed in the Fundamental Open Hypermedia Model (FOHM) [12], a generic contextual hypermedia format that evolved from the OHP suite of interoperability protocols.

Separate to this we have also been developing a proposal for OHP-Version, a versioning framework and protocol to control the versioning of OHP-Nav links.

In this paper we use FOHM and Auld Linky to ground a discussion of how a contextual structure server could be used to provide the kind of versioning supported by OHP-Version.

During this process we answer several questions:

1. Can a contextual model be powerful enough to provide versioning support (i.e. can a contextual structure server act as the selection engine behind a versioning service)?
2. We will answer this by looking at whether FOHM's contextual model is powerful enough to support versioning, and if not, how can it be extended so that it is?
3. What are the consequences of such a contextual model for the non-versioning applications of context?

2 Background

Frank Halasz discussed the issues of hypermedia versioning at the Hypertext'87 conference [5]. He highlighted the need to provide version support for both hypermedia documents and structure.

One of the earliest systems to consider versioning was Xanadu [13]. Versioning is used as a device to prevent unnecessary duplication of content between document versions. For example if a document contains the same content as a predecessor version then instead of containing a duplicate copy of that content it will contain content links that point to the same content within the older document.

Notable systems that directly tackle the problem of versioning hypermedia structure include CoVer [3], VerSE [4] and HB3 [6]. CoVer and VerSE implement policies that employ versioning as a tool for aiding users when performing a task (such as writing a research paper). These systems are also known as contextual versioning systems, but they have a very different notion of context compared to that described within this paper. Their meaning of context is a description of the task being performed when the objects (involved in carrying out that task) are versioned. Such context information is stored in order to assist with the version identification process.

The HB3 system, on the other hand, focuses on versioning support at the hyperbase level. It enables versioning of navigational hypermedia structure: nodes, links, composites and whole hypertext networks. HB3 can support a wide variety of versioning paradigms as versioning applications can build on this low-level versioning capability to construct more complex versioning styles.

The success of the World Wide Web has also led to an interest in web-based versioning. But the embedded nature of the Web means that hypermedia structure can only be versioned implicitly by versioning those web documents that contain the embedded structure. The most well known system is WebDAV (Web Distributed, Authoring and

Versioning) [8]. It extends HTTP to provide complete version control over Web documents, including the ability to retrieve, delete, copy, move and lock both document revisions and their associated revision properties.

2.1 OHP-Version

The Open Hypermedia Systems Working Group (OHSWG) has spent much effort on addressing the problem of interoperability between Open Hypermedia Systems (OHS). Their work culminated in the creation of OHP-Nav, a standardised linking protocol that allows components of an OHS to discuss navigational hypermedia [2, 11, 15].

The IAM group at Southampton has been investigating how to supplement OHP-Nav with versioning functionality, leading to the creation of the OHP-Version protocol. This protocol has four goals:

1. To store and select different revisions of OHP hypermedia objects (nodes, anchors, endpoints and link objects).
2. To preserve the state of composite structures (such as the collection of OHP-Nav objects, and connections between them, that together form a link structure).
3. To capture the state of the entire hypertext network at a given moment in time by versioning a collection of hypertext structures.
4. To maintain the version history of OHP hypermedia objects, composite structures and the hypertext network.

The approach taken when developing OHP-Version was to store all versioning data in a separate version server rather than in the OHP-Nav linkserver itself. This was to ensure that minimal disruption was caused to non-versioning clients when introducing versioning to the OHP environment. However some minor changes still had to be made. For example to ensure that OHP-Nav clients can still retrieve the latest revisions of hypermedia objects it was necessary to append a new property, called the ‘default status’ attribute, to all OHP objects. In addition, the OHP-Nav linkserver had to be re-programmed so that it permits non-versioning clients to manipulate latest revisions only.

3 Alternative Versioning Strategies

Auld Linky [9] is a stand-alone structure server that serves FOHM structures expressed in XML. Queries are sent to Linky in the form of a FOHM XML pattern which is matched against Linky’s loaded structures. Those that match are returned.

Linky also supports FOHM’s context model. Queries can be sent with an associated context description, in this case Linky matches the pattern, but then culls away any parts of the structure whose context objects do not match that of the query. It then checks that the culled structure still matches the original pattern before returning it.

Figure 1 shows how parts of a Navigational Link are culled away. In this case the query has requested all link structures given a context with keyword *age* and the value *Child*. The first link destination has a context where age is set to *Adult* and thus the

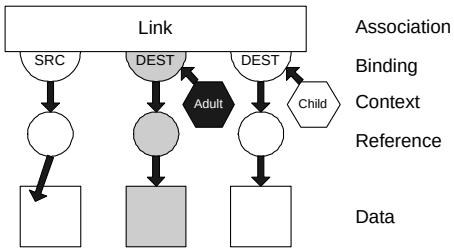


Fig. 1. A Contextual Link in FOHM

context match fails (represented in black) this results in that entire branch of the link (represented in grey) being removed before the link is returned.

Our purpose is to reconcile this kind of contextual model with versioning. In this section we compare several strategies for versioning given the facilities of a contextual structure server.

3.1 Naming Schemes

Perhaps the most pragmatic versioning approach would be to employ a naming scheme to capture the version history of individual hypermedia objects, similar to that of the Concurrent Version System (CVS). The nature of a naming scheme is to make the id of each object a meaningful name that imparts information about the version history of the objects in terms of how it is related to other versions. Thus the format for a naming scheme for FOHM could be: {`object_id:revision_id`}. The object id is a unique name that is shared amongst objects that belong to the same versioned object. It indicates which objects are related to one another. The revision id is a name that is unique amongst the objects. It describes the object's location within the version tree.

Because complex FOHM structures involve so many named objects (for example, there are six named objects in a basic two-sided navigational link), Auld Linky allows linkbase authors to create anonymous objects. These have no first class status of their own and instead are considered as part of their parents. This is a problem because to correctly version all the objects in a linkbase (without massive duplication) they must all be named.

The easiest way to achieve this would be for a structure server to automatically name all anonymous objects at the time of their creation within the server (for example, when a static linkbase was loaded). This would solve the naming problem within the server while still allowing linkbase authors the luxury of anonymous objects.

From the revision id the identity of an object's parent, children and siblings can be deduced. The revision id would also enable the order of the objects, in terms of eldest to youngest, to be calculated. Such a naming scheme could easily be implemented to capture the version history of hypermedia objects. However, this versioning solution does not help to reconcile the role of context in versioning structure.

3.2 Concepts

One strategy that we have used successfully in the past to model different views of an object is to explicitly model the relationship between them using a *concept* object [1]. These are Associations of type ‘concept’ that relate several objects that represent the same conceptual entity.

Context objects attached to the bindings of the Association implement conditional membership of the concept. Thus the same concept can be seen differently according to the context of the viewer. This could even transform the object into totally different structures, for example a biographic description might be a short piece of text in one context (a single Data object) but become a larger composite document (made up of a tree of Associations) in another.

The concept structure that we have previously used does not contain enough information to capture the evolution of its members (i.e. it contains no ordering information). This is necessary with versioning as it is important to record the order in which versions were created. In addition, version structures may branch (i.e. any given version of a structure might spawn many new versions). To model this requires a tree structure rather than an ordered set.

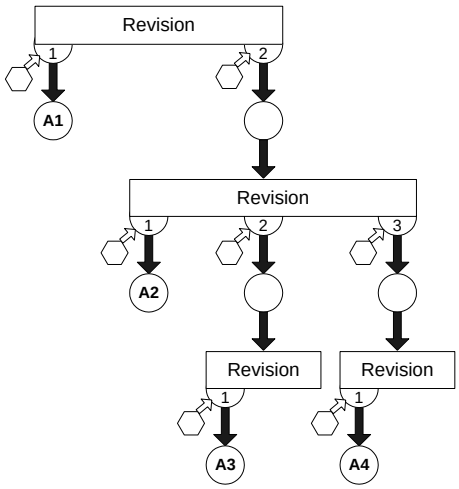


Fig. 2. A Revision Tree in FOHM

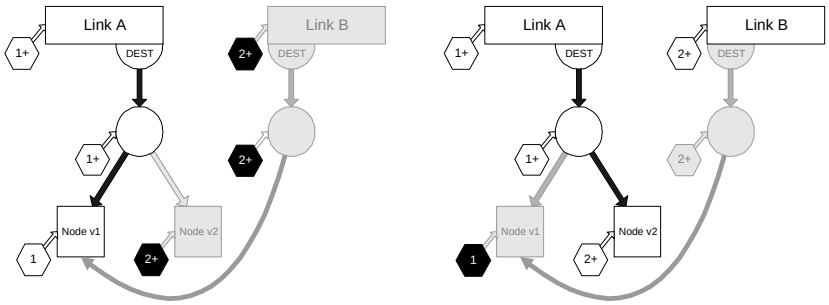
Figure 2 shows a tree of *revision* structures that manages to capture the version history of an object (in this case a Navigational Anchor, represented in FOHM by a Reference object). Each revision structure stores an ordered list of revisions, where position 1 represents the original.

Unfortunately a tree structure is cumbersome to manipulate and difficult to query, especially one that is *n* levels deep. This makes concept structures an awkward strategy for versioning. Ideally the tree would be transparent to a query, which indicates

that concept structures should be treated as more fundamental, perhaps as part of the contextual model itself.

3.3 Multiple Connections

Amending the FOHM data model to permit many-to-many relationships between all its objects presents an opportunity for another approach to versioning. In this scenario each FOHM object operates as its own concept structure with context determining which objects are connected at run-time.



(a) Query in Context 1 (b) Query in Context 2

Fig. 3. Contextual Queries with Multiple Connections. The black contexts are the ones that have failed, this results in the grey structure being trimmed away.

Figure 3(a) shows two Link structures where the context objects describe in which time slice the object is valid. The number within the context object refers to the time slice that the object was created in, a plus sign denotes that the object is visible in all subsequent time slices.

Link A has been created in time slice 1, a second version of Node v1 has then been created in time slice 2 (Node v2). When the structures are queried in the Context of time slice 1, Node v2 is removed and the viewer sees the older version of the link structure.

Link B has also been authored in time slice 2. The author explicitly wants to refer to the older version of the Node (perhaps they are discussing a particular draft of a document) and so creates a reference to Node v1. When the query is sent in the context of time slice 1, Link B is totally invisible.

Figure 3(b) shows the same structure queried in the context of time slice 2. In this instance the latest version of the Link A structure is returned (including Node v2) and the Link B object is visible. The problem is that since Node v1 is invisible in this context it does not appear as part of the Link B structure, even though this was the author's intention.

The inability to refer to older versions of objects is a fundamental problem with this strategy. The source of the problem is that FOHM's contextual model only allows

objects to be contextual, whereas we require the *connections* between objects to be contextual. In the next section we explore what a contextual model that includes contextual connections would look like.

4 Contextual Connections and Pattern Propagation

Having explored the various strategies we have discovered that FOHM's contextual model is not powerful enough. To support versioning a contextual model requires contextual connections as well as objects that describe in which context they are visible. In this section we look at how connections are formed within FOHM and how we might make them contextual.

4.1 Patterns

FOHM Objects may be referenced by a unique id, i.e. pattern match for a given type of object with a particular id. Because the ids are guaranteed to be unique, Linky can circumvent the matching process and use more efficient lookup algorithms (such as a hash table).

If we want this look-up to become contextual we need to allow non-unique ids and full patterns to be used (rather than just a direct id lookup). This would force the structure server to use the same pattern matching process internally as clients of its external services. We have called this approach *pattern propagation*. It allows a pattern to be attached to a static structure to be resolved at run-time during the query process.

We can generalise this internal pattern match further by allowing query contexts to be attached to the stored patterns. This extension allows structures to be connected together *in context*. For example, a link could be authored to a node in the context of a particular time slice. When the link is retrieved it will include the revision of the node from that time slice, even if the original query was made in the context of the current time slice.

4.2 Storing the Patterns

In the contextual structure server that we are using, Auld Linky, the id lookup objects are currently stored with a simple flag to show that they are to be used as an id lookup as opposed to being first class objects themselves.

Replacing this with a pattern to be further matched is not as simple as adding additional elements, as the context for the pattern propagation must also be stored. Furthermore we need to allow queries to be made independently of type (i.e. a reference should be able to store a pattern that resolves into *either* a Data object or an Association).

This would require two extensions to the current Linky FOHM definition. Firstly, it must become possible for authors to create a *ReferencableObject* as well as an Association or Data object. This would match with either of the other two. Secondly all FOHM's first class objects need to be able to contain *QueryRules* when written in their pattern state. This would include the context in which to make the query.

Once these patterns have been stored in the linkbase we are then faced with the problem of getting them out again (since a query would cause them to be automatically resolved). One solution would be to place a flag in the query rules that specified the *depth* to which patterns should be resolved. A depth of *n* would return all patterns, while adjusting the depth to other numbers would allow an application to retrieve the upper-most parts of large structures without invoking potentially expensive pattern matching processes on the lower parts (for example, to return a tour structure without processing the members of that tour).

4.3 Merging Contexts

When a contextual query is sent to the structure server it already has a Context object attached to it. With the current context model this single context is used throughout the matching process, but with pattern propagation it may change as the matching process works its way down different branches of the structure.

This means that there is an issue of how the query contexts and the pattern contexts should be merged. There are four choices:

1. *Replace the Context*. The pattern context could entirely replace the query one.
2. *Merge Contexts (do not replace keys)*. The two contexts could be merged with the original keys' values taking precedence over the new values.
3. *Merge Contexts (replace keys)*. The two contexts could be merged with new keys' values taking precedence over the original values.
4. *Merge Contexts (conjoin keys)*. The two contexts could be merged with the values of like keys being AND'ed together.

Without merging the original context information would be discarded even if it has no bearing on the new context information. Conjoining contexts may prove to be very difficult, particularly as Linky allows values to be *constrained* (matched via included code) and AND'ing arbitrary code is non-trivial. However, one might imagine situations where both the remaining merging scenarios are required.

For example, when using context to control children's access to adult material it seems sensible to maintain the existing keys (i.e. the age is always set to child, thus a child cannot 'move' into an adult context). However, when going the other way it makes sense to allow the context to be overridden (allowing adults to view material in the context of a child).

To allow this fine control a flag would be needed that specifies how each context value should be merged with a matching one on a stored pattern's context. This provides a fine grained way for query behaviour to be specified.

4.4 Multiple Connections and First Class Bindings

The current implementation of Auld Linky permits an Association object to contain multiple members (represented by Binding objects), but in all other cases it only allows for single connections between objects. Pattern propagation makes no guarantees about the number of matches it returns, therefore FOHM would need to be extended to allow

multiple connections at all points. Section 3.3 discussed a similar situation, where *n* ids could be attached at every point of the structure. Here we can only attach one id, or pattern, but it may resolve into many objects.

Multiple connections would have a negative impact on the pattern matching process as the server needs to attempt to match all possible permutations at any point where there is a collection of objects.

A further requirement is that structure servers should explicitly name all objects that undergo frequent revision. This would allow them to be versioned separately. FOHM currently defines Bindings as implicit objects wholly contained within Association objects. This presents some difficulties for FOHM versioning. Firstly, Bindings cannot be versioned directly. If clients want to version a specific Binding, then they have to version the whole Association object within which the Binding is contained. Secondly, the act of versioning an Association causes the creation of new revisions of all of the Bindings contained within it.

4.5 Maintaining the Version History

The problem with using pattern propagation to support versioning is that it does not record the version history of individual objects.

One solution is to store the version history implicitly within Context objects attached to each object. They will record the object's parent, children and its immediate siblings. This is a far from ideal solution. It is difficult to maintain and the role of Context objects is not to store meta-data about a structure, but to store meta-data that describes in which contexts that structure is visible.

A second option is to record the version history as a additional structure. This could even be stored in the same structure server as the evolving structure it is being used to record (this could take a similar form to the revision tree structure described in Section 3.2).

5 How Would Versioning Work in Practice?

In this section we describe how a contextual structure server implementing pattern propagation would be used to support the two core versioning operations of revision retrieval and revision creation.

5.1 Retrieving the Latest Version

One of the fundamental capabilities of a versioning system is to allow users to easily retrieve the latest version of a structure. It is also important to support naïve (non-versioning aware) clients by giving access to the latest version of a structure by default.

When using a contextual structure server to support versioning there is a problem in that naïve clients will query the server with a blank context object and thus match all versions (when what they really require is the latest version).

In Linky this is caused by the model of context matching. If a key is present in both contexts then it must have the same value to match but if it is only present in one context then it is assumed to match by default.

The way that Linky avoids this problem is to allows context values to be flagged as *required*, this reverses the normal matching behaviour and ensures that if only one of the context objects specifies a key then those context objects fail to match.

We could use this required flag within the linkbase on all object revisions save the latest one, this means that a user must specify a version key explicitly to retrieve anything other than the latest version.

It is also possible that with branching version structures there is more than one version available in a particular time slice and therefore more than one 'latest' version. It is up to the authors of such branches whether or not they use a similar mechanism to hide their versions from the view of naïve clients.

5.2 Retrieving an Earlier Version

Several approaches can be adopted for retrieving an earlier version of an object.

The first is to send a contextualised query that will match against the patterns stored in the Auld Linky structure server. This query will contain a context object that has a key that describes the versioning dimension that we are interested in, for example time. The Merge value will specify that we only want to match against stored patterns that are equal to the context value, such as FOHM objects that were created between 10:00am and 11:00am.

The second approach is to search for a specific object version by traversing the version history structure (such as the one shown in Figure 2). This is only possible if the version histories of FOHM objects have been explicitly recorded as described in section 4.5.

Both of these empower the *user* to select a particular revision. With pattern propagation it becomes possible for the *author* of the structure to choose a revision by connecting an object to their structure in the context of an earlier version.

5.3 Creating a New Version

To create a new revision a new object must be created in the structure server with the same id as the old revision. It is not necessary to change any existing connections. However, the structure server will have a way of denoting the latest revisions of an object and this must be updated to the new revision (for example, in Auld Linky the old revision must have its required flag set to true).

If a version history of the structure is being explicitly recorded it would also need to be updated.

6 CONCLUSION

In this paper our aim was to find out if a contextual model could be powerful enough to replace the selection engine of a versioning server. We approached this question by examining FOHM's contextual model as implemented within Auld Linky.

We found that FOHM does not have a powerful enough context model in its present form. This is because FOHM allows objects to be contextual but not the connections between them. By extending FOHM to include multiple connections and context on those connections (via pattern propagation) FOHM could support versioning with context.

More generally this means that it is possible to reconcile context and versioning. General contextual models can be devised that are powerful enough to cope with the requirements of a versioning selection engine.

The extended contextual model is also more expressive when considering the non-versioning applications of context. It allows structures to be connected together in context. For example a link could be authored to a tour in the context of a child. When the link is retrieved it will include the children's version of that tour, even if the original query was made in the context of an adult.

By reconciling context and versioning we view a contextual structure server as a component that may slot into a versioning framework to take over the role of the selection engine. This results in an integrated approach to context and versioning where object revisions are manipulated in the same manner as other contextual views.

Other aspects of a versioning service (such as policy management and enforcement, object locking, run-time consistency etc.) can be provided by other components in the system framework, an architecture advocated by Component Based Open Hypermedia Systems (CB-OHSs) such as Construct [18, 17]. If a version history is required then it could be explicitly maintained by a separate knowledgeable component (that could still use the structure server for storage).

In summary, we have shown that versioning is a specific kind of contextualisation and that it can be supported by a generic contextual model. We have presented an extension to FOHM that describes how this might be done in practice. It is our opinion that versioning is better seen as a specific use of context and that reconciling the two results in a more natural and consistent approach to both.

7 ACKNOWLEDGEMENTS

This research is funded in part by EPSRC IRC project "EQUATOR" GR/N15986/01 and by EPSRC DIM project "LinkMe" GR/M25919.

References

1. Christopher Bailey, Wendy Hall, David E. Millard, and Mark J. Weal. A Structural Approach to Adaptive Hypermedia. In *2nd International Conference on Adaptive Hypermedia and Adaptive Web Based Systems, Malaga, Spain*, 2002.
2. Hugh Davis, Siegfried Reich, and David Millard. A proposal for a common navigational hypertext protocol. Technical report, Dept. of Electronics and Computer Science, 1997. Presented at 3.5 Open Hypermedia System Working Group Meeting. Aarhus University, Denmark. September 8-11.
3. Anja Haake. Under CoVer: the Implementation of a Contextual Version Server for Hypertext Applications. In *ECHT '94. Proceedings of the ACM European conference on Hypermedia technology, Sept. 18-23, 1994, Edinburgh, Scotland, UK*, pages 81–93, 1994.

4. Anja Haake and David Hicks. Verse: Towards hypertext versioning styles. In *Proceedings of the '96 ACM Conference on Hypertext, March 16-20, 1996, Washington, D.C.*, pages 224–234, 1996.
5. Frank Halasz. Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems. *Communications of the ACM*, 31(7):836–852, 1988.
6. D.L. Hicks, J.J. Leggett, P.J. Nrnberg, and J.L. Schnase. A Hypermedia Version Control Framework. *ACM Transactions on Information Systems*, 16(2):127–160, 1998.
7. E. James Whitehead Jr. Design Spaces for Link and Structure Versioning. In *Proceedings of the '01 ACM Conference on Hypertext, August 14 - 18, Arhus, Denmark*, pages 195–204, 2001.
8. E. James Whitehead Jr. WebDAV and DeltaV: Collaborative Authoring, Versioning, and Configuration Management for the Web. In *Proceedings of the '01 ACM Conference on Hypertext, August 14 - 18, Arhus, Denmark*, pages 259–260, 2001.
9. D.T. Michaelides, D.E. Millard, M.J. Weal, and D. DeRoure. Auld leaky: A contextual open hypermedia link server. In *Hypermedia:Openness, Structural Awareness, and Adaptivity (Proceedings of OHS-7, SC-3 and AH-3), Published in Lecture Notes in Computer Science, (LNCS 2266), Springer Verlag, Heidelberg (ISSN 0302-9743)*, pages 59–70, 2001.
10. David Millard and Hugh Davis. Navigating Spaces: The Semantics of Cross Domain Interoperability. In Siegfried Reich and Kenneth M. Anderson, editors, *Proceedings of OHS 6 and SC2, Published in Lecture Notes in Computer Science, (LNCS 1903), Springer Verlag, Heidelberg (ISSN 0302-9743)*, pages 129–139, 2000.
11. David Millard, Hugh Davis, and Luc Moreau. Standardizing Hypertext: Where Next for OHP? In Siegfried Reich and Kenneth M. Anderson, editors, *Proceedings of OHS 6 and SC2, Published in Lecture Notes in Computer Science, (LNCS 1903), Springer Verlag, Heidelberg (ISSN 0302-9743)*, pages 3–12, 2000.
12. David E. Millard, Luc Moreau, Hugh C. Davis, and Siegfried Reich. FOHM: A Fundamental Open Hypertext Model for Investigating Interoperability Between Hypertext Domains. In *Proceedings of the '00 ACM Conference on Hypertext, May 30 - June 3, San Antonio, TX*, pages 93–102, 2000.
13. Theodor Holm Nelson. *Literary Machines*. Published by the author. Mindful Press, 1987.
14. Kasper Østerbye. Structural and Cognitive Problems in Providing Version Control for Hypertext. In *ECHT '92. Proceedings of the ACM conference on Hypertext, November 30-December 4, 1992, Milan, Italy*, 1992.
15. Siegfried Reich, Uffe K. Wiil, Peter J.Nürnberg, Hugh C. Davis, Kaj Grønbæk, Kenneth M. Anderson, David E. Millard, and Jörg M. Haake. Addressing Interoperability in Open Hypermedia: The Design of the Open Hypermedia Protocol. *New Review of Hypermedia and Multimedia*, pages 207–243, 2000.
16. F. Vitali. Versioning Hypermedia. *ACM Computing Surveys*, 31(4), 1999.
17. Uffe K. Wiil, David L. Hicks, and Peter J. Nürnberg. Multiple Open Services: A New Approach to Service Provision in Open Hypermedia Systems. In *Proceedings of the '01 ACM Conference on Hypertext, August 14 - 18, Arhus, Denmark*, pages 83–92, 2001.
18. Uffe Kock Wiil and Peter J. Nürnberg. Evolving Hypermedia Middleware Services: Lessons and Observations. In *Proceedings of of the 1999 ACM Symposium on Applied Computing (SAC '99), San Antonio, TX*, pages 427–436, February 1999.