

**Version Control in Hypermedia:
An Open Systems Perspective**

David L. Hicks
John J. Leggett
John L. Schnase

Hypermedia Research Lab

JANUARY 1993
TAMU-HRL 93-001

Table of Contents

1. INTRODUCTION	1
2. AN OPEN SYSTEMS PERSPECTIVE	2
3. VERSION CONTROL DATA MODELS	4
4. VERSION CONTROL AT THE HYPERBASE LEVEL	6
4.1 Selecting the Version Control Data Model	7
4.2 Composition Mechanisms	8
4.3 Internode Referencing	10
4.4 Storage Considerations	11
5. VERSION CONTROL AT THE APPLICATION LEVEL	11
5.1 Selecting the Version Control Data Model	12
5.2 User Interface Considerations	13
5.3 Intranode Specification	13
6. RELATED WORK	14
7. SUMMARY	15
REFERENCES	17

Version Control in Hypermedia: An Open Systems Perspective

DAVID L. HICKS
JOHN J. LEGGETT
JOHN L. SCHNASE

1. INTRODUCTION

The need for version control arises naturally in application domains that support the process of evolutionary development [Zdonik 1986]. Version control systems help manage the intermediate revisions characteristic of such environments and assist in maintaining the relationships between revisions. Version control is particularly important when the number of intermediate revisions grows large, or the relationships among revisions become complex. Historically, much of the research and development related to versioning has been in the software engineering field [Katz 1990; Tichy 1988]. Recently, researchers from other disciplines have begun to examine the benefits version control can offer their application domains. In many cases, researchers are finding that techniques used in existing version control systems can be customized for their environment [Chou and Kim 1986].

Hypermedia systems are already being used for evolutionary development tasks such as authoring and idea processing, and version control has been identified as one of the critical research areas in the hypermedia field [Halasz 1988]. In spite of its importance, version control has thus far been neglected by most hypermedia researchers, and current hypermedia systems have little or no facilities for version control. However, future hypermedia systems will be employed to support tasks that have even more extensive version control requirements, such as software configuration management and collaborative work. The absence of a version control facility will limit the potential applications of hypermedia systems. It is therefore critically important that existing version control mechanisms be reviewed, extended, and tailored for the hypermedia domain.

This paper examines the major issues of version control in hypermedia from the perspective of an

open systems architecture. This perspective provides a context for the detailed discussion of version control issues throughout the paper. We have attempted to maintain the discussion at a level that is independent of any particular model of hypermedia.

2. AN OPEN SYSTEMS PERSPECTIVE

The majority of present generation hypermedia systems have monolithic architectures [Pearl 1989]. Monolithic hypermedia systems are self-contained, closed systems in which the manipulation and access of data for authoring and browsing purposes can only be done using the tools provided by the hypermedia system developer. Most monolithic systems implement backend functionality by building directly upon an existing file or database management system [Kacmar and Leggett 1991]. Node and link data are generally stored in formats that prevent external applications from directly accessing content or structural information.

Recently, however, several research efforts have begun to focus on the design and implementation of open, distributed hypermedia architectures that utilize a distinct hyperbase management system (HBMS) [Schnase et al. 1991, 1992; Schnase 1992; Schütt and Streitz 1990; Shackelford 1992; Smith and Smith 1991; Wiil 1992; Wiil and Leggett 1992]. An HBMS is a server that provides the backend functionality required to support hypermedia operations. A hyperbase management system provides facilities for object management to accommodate the data contained in hypermedia nodes, along with facilities to manage the structural connections among nodes. Hyperbase management systems also provide additional features such as transaction management, concurrency control, and distribution. Since HBMSs generally do not impose any format on the objects they manage, they may be used by a variety of applications for object management services. Likewise, the structural facilities of an HBMS may be used for link management services to support linking within and among diverse applications.

Figure 1 illustrates a typical open hypermedia systems environment. It depicts the relationship between a hyperbase server and its client applications. Applications interact with the HBMS through the application interface to store and retrieve objects or perform hypermedia operations.

The HBMS channels requests from applications to object and structure management subsystems which, in turn, interact with a physical storage manager to perform the desired operations.

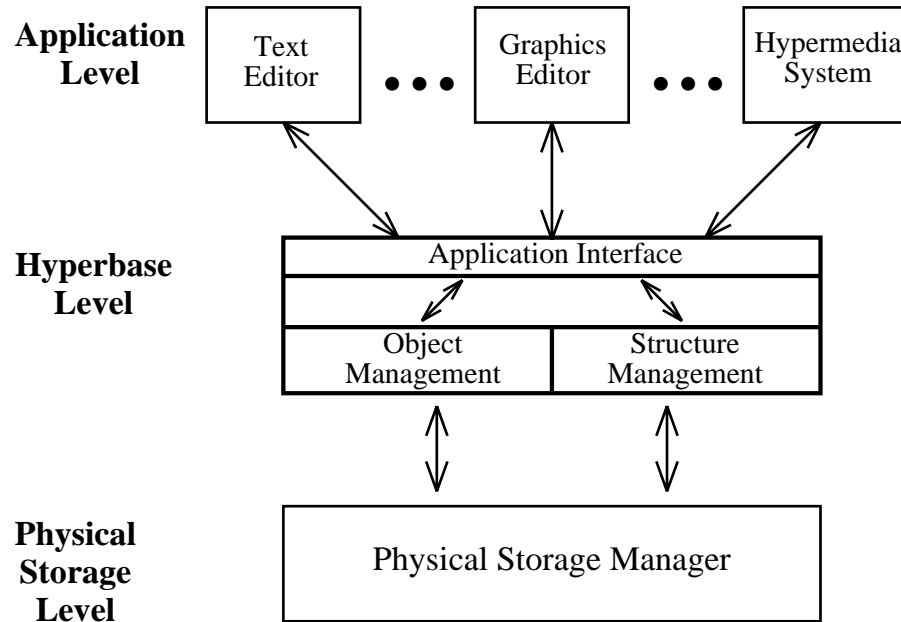


Figure 1. A typical hypermedia open systems environment.

Most of the research related to version control in hypermedia has concentrated on monolithic hypermedia systems [Akscyn et al. 1988; Delisle and Schwartz 1987; Prevelakis 1990; Yankelovich et al. 1988]. Other research efforts have focused on version control for specific application domains such as CAD [Chou and Kim 1986; Katz 1987] and CASE [Cohen et al. 1988; Demleitner 1988; Leblang and Chase 1984]. This paper examines version control in an open hypermedia system architecture containing a hyperbase management system. We refer to this architectural environment simply as *the hyperbase environment*. The features that characterize hyperbases have a major impact on the design and implementation of version control services. As indicated in Figure 1, a variety of application types can utilize HBMS services, including monolithic hypermedia systems. The version control process must be carefully analyzed to determine how it can best be integrated into the hyperbase environment to support this diversity of applications.

An important characteristic of the hyperbase environment is that two distinct levels exist at which versioning functionality can reside: (1) the hyperbase level and (2) the application level (see Figure 1). At the hyperbase level, basic operations are provided for use by the developers of HBMS client applications. Application-specific services are implemented at the application level. As version control is integrated into the hyperbase environment, each aspect must be carefully considered to determine the proper level for implementation. This partitioning of functionality between the hyperbase and application levels is one of the distinguishing characteristics of the open systems perspective.

3. VERSION CONTROL DATA MODELS

The data model is an important aspect of a version control system. It describes the objects allowed in the system, relationships that can exist among these objects, and allowable paths of object evolution. The linear model and tree model are two of the most popular version control data models. As illustrated in Figure 2, these basic models vary in the complexity of evolution paths they support. The linear data model restricts object development to a single linear evolution path. This model was used by some of the earliest software configuration management systems [Rochkind 1975]. Later systems adopted the tree data model [Belkhatir and Estublier 1986; Mahler and Lampen 1988; Tichy 1985]. The tree model augments the linear model to allow branch points to occur in the paths of evolution. This expands the system's support for object evolution to include variants (objects that develop in parallel from a common ancestor) [Tichy 1988].

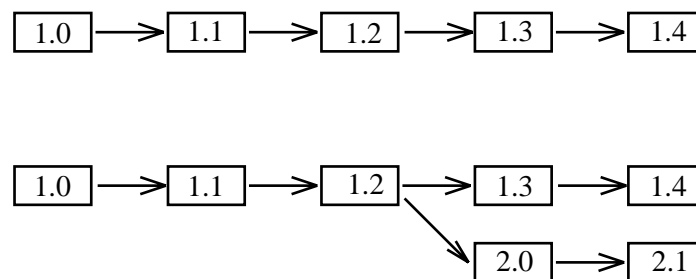


Figure 2. Linear and Tree Data Models

An interesting alternative to the tree data model is the layered network data model of the Personal Information Environment System [Goldstein and Bobrow 1984]. In this model, objects correspond

to nodes in a network, and arcs between nodes represent relationships between objects. A set of related arcs can be grouped together to form a layer, and related layers can be grouped together to form contexts. Figure 3 illustrates a software module and its associated documentation as represented by a layered network. Release 1.1 of module A uses the quicksort algorithm and its documentation consists of three text nodes. These objects and the links between them comprise layer A, the single layer in this layered network. Figure 4 illustrates a context that contains layers A and B. Layer A serves as the base of this context. Layer B changes the context by adding new nodes and arcs and modifying existing ones. In this example, the addition of layer B changes the sort routine and associated documentation from a quicksort to a heapsort. The representational capabilities of the network data model encompass and extend those of the tree data model [Hicks et al. 1991]. Advanced data models such as the layered network model provide customized support for object evolution in specific application domains.

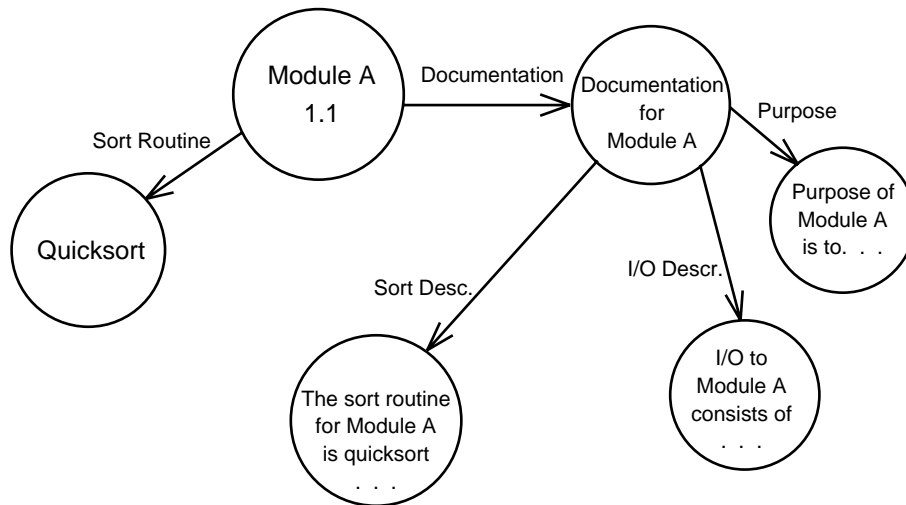


Figure 3. The Layered Network Data Model – Layer A.

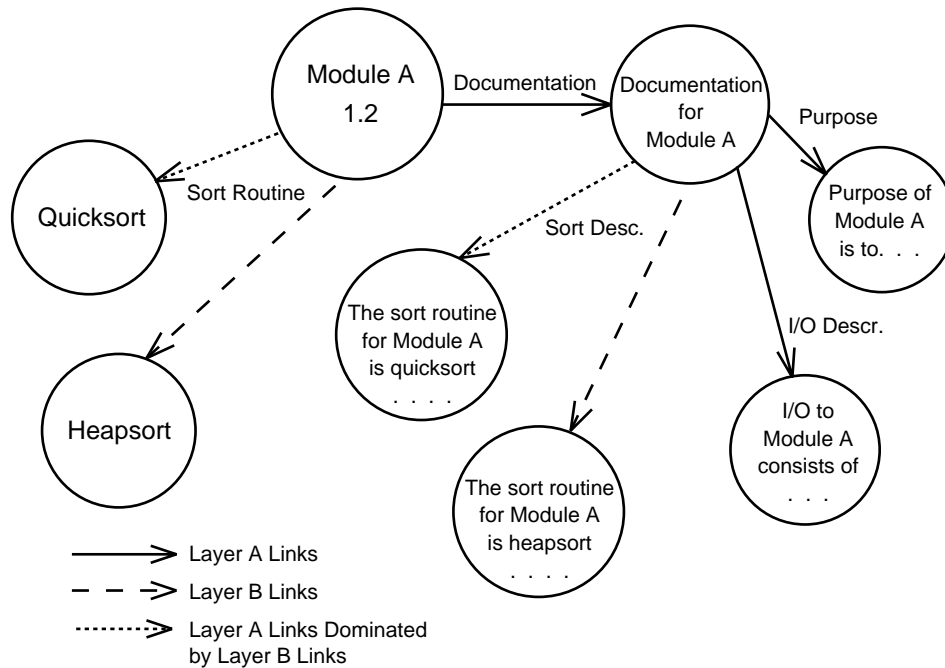


Figure 4. The Layered Network Data Model – Layers A and B.

4. VERSION CONTROL AT THE HYPERBASE LEVEL

Version control in hypermedia consists of managing the evolution of: (1) data contained in hypermedia nodes and (2) structural connections among nodes. This process directly involves basic hypermedia objects and operations. In a hyperbase environment, hypermedia object management services are provided at the hyperbase level. Aspects of the version control process that involve basic hypermedia objects should also be implemented at this level. These basic version control services may also be used by application developers to provide version control in other client applications. A desirable and important characteristic for hyperbase-level version control is that it should not reflect application-level version control policies. The remainder of this section examines four specific aspects of the version control process that involve services provided at the hyperbase level: selecting the version control data model, composition mechanisms, internode referencing, and storage considerations.

4.1 Selecting the Version Control Data Model

The selection of a version control data model is more complex in a hyperbase environment than in a monolithic hypermedia system. In the monolithic case, a version control data model can be selected solely on its suitability for a specific application, the hypermedia system. In selecting a version control data model for a hyperbase environment, the variety of client applications must be considered. Object evolution can vary significantly among various application domains. For example, it is unlikely that a graphics editor and software configuration management tool will have identical version control requirements. *The hyperbase version control data model must be able to accommodate a diversity of object evolution patterns.*

An analysis of existing version control data models is helpful in determining the characteristics of an appropriate hyperbase-level data model. The basic data models used by early version control systems, such as the linear or tree model, were patterned after the object evolution process as it occurs at the physical storage level. In systems that use early version control data models, there is a direct mapping from user interface level operations to physical storage level operations. At the user interface level, the concept of an object directly corresponds to a physical storage level data object, and an object's evolution history, as perceived by the user, directly reflects its physical derivation.

The elaborate data models used by more recent version control systems support evolution at a more abstract level. This creates a disparity between user interface level operations and the physical storage level operations that support them. The user interface level operations of object storage and retrieval often have no direct counterparts at the physical storage level. An object at the user interface level may actually consist of a combination of several physical storage level objects. Similarly, there is no direct correlation between an object's perceived evolution path at the user interface level and the actual derivation of physical storage level entities.

The capabilities provided at the user interface level by both basic and elaborate data models must, at some point, be mapped to physical storage operations. In systems that use a basic data model, this mapping is direct. With elaborate data models, the mapping is more complex. The underlying physical storage level operations comprise the only similarities common to all version control data models. In a hyperbase environment the HBMS provides physical storage operations for client

applications. This suggests that the most appropriate version control data model for the hyperbase level is one which can support the evolutionary process as it occurs at the physical storage level. A more elaborate data model would be too application specific and would not provide the flexibility required at the hyperbase level to support a variety of application-level capabilities.

Basic data models are particularly capable of supporting development at the physical storage level as shown by the widespread use of these models in many version control systems [Tichy 1988]. Such data models could conceivably be used to support version control of hyperbase-level data objects. It is important to emphasize that a hyperbase-level version control data model should in no way limit the development of elaborate application-specific data models in client applications. The hyperbase-level model should simply provide a foundation of basic version control operations upon which application-specific models can be constructed. This strategy would allow developers to concentrate on the details of constructing application-specific data models and leave the concerns of managing physical storage level object evolution to the HBMS. Application-level data models are considered further in Section 5.

4.2 Composition Mechanisms

Tools are provided in many hypermedia systems to assist node and link management. In general, these systems allow users to partition a set of nodes and links to facilitate working with specific parts of a hypermedia network. Node and link management is also an important consideration in a hyperbase environment. A typical hyperbase will support a variety of applications each potentially having several users. This type of multi-application and multi-user environment can quickly generate a large number of nodes and links. With the addition of version control services, the number of hyperbase objects will grow even faster.

A composition mechanism at the hyperbase level is crucial to the development of application-level tools for node and link management. The ability to group nodes and links is fundamental to many important hypermedia operations. It provides direct support for the development of several important application-level operations such as grouping related changes to a hypermedia network, defining multiple versions of structure over a set of hypermedia nodes, and the customization of user workspaces. This capability also allows specific versions of nodes and links to be selected

and grouped into a baseline configuration [Cohen et al. 1988]. In addition to supporting these application-level operations, a composition mechanism is also critical to the development of facilities that support collaboration, concurrency control, and other advanced hypermedia operations. Experiences from systems without a composition mechanism substantiate the need for this functionality [Wiil and Østerbye 1990].

Various mechanisms have been suggested to support composition in hypermedia. Intermedia groups links into webs, allowing multiple versions of structure to be defined over a set of hypermedia nodes [Yankelovich et al. 1988]. Halasz has suggested the use of composites to group related sets of nodes and links [Halasz 1988]. The Neptune/HAM system was extended to include the notion of contexts that allow a network of nodes and links to be partitioned [Delisle and Schwartz 1987]. Prevelakis [1990] has proposed an extension to the layered network model that uses overlays and perspectives to group hypermedia objects. HyperBase uses a composite object as a grouping mechanism for nodes and links [Schütt and Streit 1990].

The implementation of a grouping mechanism for hypermedia objects raises several important issues including: which hypermedia objects are eligible for group membership, whether hypermedia objects can belong to multiple groups, and the semantics of group membership. The grouping mechanisms described above vary in the way they have addressed these issues. Many of the differences can be attributed to the different hypermedia data models inherent in these systems. Most of the mechanisms allow both nodes and links to be included in a group. Object membership in multiple groups is influenced heavily by the semantics of group membership. Composition mechanisms like the context used in the extension to the HAM/Neptune system effectively partition the network of hyperbase objects when groups are created. This approach does not allow object membership in multiple groups. In systems like HyperBase [Schütt and Streit 1990], where objects are included by reference, object membership in multiple groups is possible.

A composition mechanism for the hyperbase environment should allow all hyperbase objects to be included in groups. This will enable nodes, links, and even groups themselves to be members of a group. Objects should be included in groups by reference, so they can easily belong to multiple groups. A composition mechanism with these characteristics would readily support each of the application-level operations described above and provide a good basis for the development of

facilities for collaboration and other hypermedia system operations.

An important advantage associated with a composition mechanism is that it enables versioning to occur at multiple levels [Hicks et al. 1991]. Version histories can be maintained for individual hyperbase objects and groups as they are defined and evolve over time. Versioning at both levels allows the complete derivation history of an evolving system to be captured, and it will accommodate HBMS client applications that need to capture the derivation information of system configurations as well as individual objects.

4.3 Internode Referencing

In most hypermedia systems, it is possible to construct links between specific regions within nodes. Two levels of specification are required to completely identify the endpoints of this type of link – the internode level and the intranode level. The internode level resolves a link endpoint to the node level of granularity. At the intranode level, a location or region within the target node is designated as the precise endpoint for the link. Together these two levels completely define the link endpoint.

Versioning complicates link specification in hypermedia by introducing complexities at both the internode and intranode levels of specification. In a hyperbase environment, internode specification consists of identifying a particular object in the hyperbase that contains the target of a link. Since this involves hyperbase-level operations, this issue should be addressed at the hyperbase level. Intranode references specify locations or regions within hyperbase objects that serve as link endpoints. This level of specification requires application-specific knowledge (such as the format in which data objects are stored). The issues associated with intranode referencing are more appropriately addressed at the application level [Schütt and Streitz 1990; Schnase et al. 1991]. Intranode referencing will be discussed further in Section 5.

Internode references must identify a specific object in a hyperbase as the target of a link [Khoshafian and Copeland 1986]. In a hyperbase with version control facilities, several versions of an object might exist. *The object identification process must be augmented to include the specification of version information in order to uniquely identify each object.* As version

information is incorporated into the object identification process, the types of version specifications likely to be requested by developers should be considered. For example, an application developer might require support for links that always reference the oldest version, newest version or a specific version of a node. The augmented object identification process should support these types of references. One potential mechanism to address this issue is the use of object attributes [Delisle and Schwartz 1986; Schütt and Streitz 1990; Wiil and Østerbye 1990]. Object version identification is then accomplished by specifying predicates over the attribute values.

4.4 Storage Considerations

A storage management strategy that balances the conflicting goals of reduced space and rapid access to hypermedia objects is needed. Fortunately, the version control process provides opportunities that can be exploited to significantly reduce storage space. *Hyperbases can use physical storage level object derivation information to reduce storage requirements.* This process has been well researched in the software configuration management area [Katz 1990; Tichy 1988]. Developers have used a variety of techniques such as conditional compilation and delta storage strategies to isolate and store only the differences between successive object versions. In several cases, significant reductions in storage requirements have been observed [Leblang and Chase 1987]. Many of these same techniques could be tuned and applied to hyperbase objects.

It is important to emphasize that these storage reduction techniques have been specifically designed for the management of textual data. Hyperbases must also manage a variety of additional non-textual data types, some having large object sizes. Appropriate delta calculation algorithms and other strategies will be required to control the storage requirements for these objects [Scacchi 1988]. It is also important for hyperbases to facilitate the definition of new data types along with appropriate storage management strategies.

5. VERSION CONTROL AT THE APPLICATION LEVEL

The development process differs significantly among application domains, creating the need for different types of version control facilities. Hyperbase management systems provide a clear separation between application-independent and application-dependent functionality. When

constructing a hyperbase client application, the application-independent facilities of the hyperbase level are utilized to implement application-dependent functionality at the application level. The operations of the hyperbase level can be selectively combined to build a version control component that is tailored for a specific application. Hyperbase-level functionality eases the construction of version control services by relieving developers from many of the low level physical storage tasks associated with version control and allowing them to concentrate on application-specific details. The remainder of this section examines three specific aspects of the version control process that involve services provided at the application level: selecting the version control data model, user interface considerations, and intranode referencing.

5.1 Selecting the Version Control Data Model

The data model at the application level should be designed to support the development process in a specific application domain. In some application domains developing objects are large and change infrequently, while others are characterized by smaller objects that change continuously. It is critical to maintain the temporal relationships between evolving objects in some application domains, while in others, this information is unimportant. When designing an application-level data model, the developer should be more concerned with these types of application-specific characteristics than with physical storage considerations. After an appropriate application-level model has been developed, its capabilities can be mapped to physical storage operations. In a hyperbase environment, the functionality of the hyperbase-level data model facilitates this process by providing physical storage level version control operations. Specialized application-level version control data models have been developed for areas such as software configuration management [Katz 1990; Tichy 1988], CAD [Chou and Kim 1986], and hypermedia publishing [Weber 1991]. The layered network model is a good example of such a model for the software configuration management domain.

A common concern among many application-level data models is the maintenance of valid configurations. The flexibility of abstract data models can sometimes allow objects to be combined in incorrect ways. For example, the user of a software configuration management system might mistakenly attempt to combine incompatible object modules. The overlay-based data model developed by Prevelakis [1990] addresses this issue by using signatures to represent object

dependencies and constraints. Signatures are attached to individual objects and provide information about various aspects of the object. The system can check for valid configurations by examining object signatures. *Application-level version control systems require mechanisms to ensure the correctness and consistency of configurations in application-level data models.*

5.2 User Interface Considerations

User interface considerations represent an important category of application-dependent version control issues. Collectively, issues in this category determine how end users will interact with a version control facility. Version creation is one of the most basic user interface considerations. This process can range from being completely transparent to the user (automatic), to the requirement of an explicit command to create a new version. If an automatic process is to be used, the frequency of version creation must be determined. In some application areas it is important to capture all intermediate revisions of evolving objects, while in others less frequent versioning is sufficient. The granularity of version creation must also be determined. Versions can be created with every individual edit, edit session, or at some higher level of granularity.

The provision of defaults is also an important user interface consideration. Version control facilities increase the complexity of an application by requiring the specification of version information in order to perform hypermedia operations. *Version control facilities should not impede the normal use of an application by requiring excessive user interaction.* To reduce the overhead created by version control facilities, appropriate defaults are needed to support commonly performed operations. Defaults will enable version control services to remain transparent to the user when direct interaction is not required.

5.3 Intranode Specification

Intranode referencing identifies a specific location or region within a node as the endpoint of a link. The process of intranode referencing is highly application-dependent and requires detailed knowledge of object structure. Various strategies have been used for intranode specification, such as specifying target objects in object-based systems or character offsets in text-based systems. Versioning complicates the process of intranode specification. When objects are modified and new

versions created, it is possible for the semantics of intranode references to become corrupt. In an object-based system, this can occur when an object serving as a link endpoint is deleted, moved, or otherwise modified, such that it no longer designates the appropriate location. Modifications made to a text-based object can cause character offsets to identify incorrect text regions. *Application-level version control systems require mechanisms to ensure the semantics of intranode references across object versions.*

6. RELATED WORK

The largest body of research pertaining to version control is found in the software engineering field [Katz 1990; Tichy 1988]. Research results from this area are particularly applicable to hyperbase-level version control issues. The data model and storage management facilities of software configuration management systems provide a solid basis for development of hyperbase-level version control. Additionally, some of the features which characterize more recent software configuration management systems pertain to application-level version control issues. In particular, the layered network data model of the PIE [Goldstein and Bobrow 1984] system has been suggested as a starting point for the development of an application-level version control data model for hypermedia systems [Halasz 1988].

Most research on version control in hypermedia has concentrated on monolithic hypermedia systems. The Neptune system supports linear version histories for hypermedia nodes [Delisle and Schwartz 1987]. Links can be made to either a specific or the latest version of a node, and the hypermedia network can be partitioned into private workspaces called contexts. The KMS system provides time-based, linear version histories for individual frames, trees of frames, and framesets [Pearl 1989]. The Intermedia system separates the storage of data and structure [Yankelovich et al. 1988], enabling multiple versions of structure to exist over a common set of hypermedia nodes. More recently, Prevelakis has proposed a hypermedia version control data model based on the extensions of overlays and perspectives [Prevelakis 1990] to the layered network model.

In general, version control has not been addressed in the open systems environment. A few recent research efforts have focused on version control for specific hyperbase client applications. HyperBase is an HBMS under development at the Integrated Publication and Information Systems

Institute (IPSI) [Schütt and Streitz 1990]. Researchers at IPSI are developing a hypermedia version server that supports hypermedia publishing applications [Weber 1991]. The Fenris HBMS is being developed at the University of Aalborg [Østerbye et al. 1992]. Research is underway to extend one of the Fenris client applications, the HyperPro programming environment, to include version control facilities. Version control in the open systems environment and more specifically at the hyperbase level, is currently represented by research efforts at Texas A&M University on HB2 [Schnase 1992], the University of North Carolina on DGS [Shackelford 1992], and the University of Aalborg on Hyperform [Wiil and Leggett 1992].

7. SUMMARY

Version control is a critically important area of hypermedia research. While the majority of current research efforts in this area have concentrated on monolithic systems, we have examined version control in an open hypermedia system architecture containing a hyperbase management system. This broader perspective has provided insight on the nature of version control issues. It is clear that a subset of version control operations are fundamental to application areas supporting evolutionary development. An architectural layer providing these services can serve as a foundation for the development of version control facilities. The higher level aspects of the version control process are application specific and must be considered on an individual basis.

We have partitioned hypermedia version control functionality and associated issues into application-independent and application-dependent categories. When examined from this perspective certain issues, such as the version control data model, are found to consist of component issues that occur in both categories. The open systems architecture readily accommodates the separation of functionality into these categories. Application independent version control operations can be implemented at the hyperbase level for easy access by client applications. Utilizing these services, developers can address application dependent issues on an individual basis at the application level.

The issues discussed in this paper should provide a basis for future research. Hyperbase-level issues that are particularly important are those involving the version control data model and the composition mechanism. Application-level issues that are particularly important are those

involving the version control data model and user interface considerations. As research on version control in open system environments continues, the separation of functionality between the hyperbase and application levels should be re-examined for optimal partitioning.

REFERENCES

- Akscyn, R., McCracken D., and Yoder, E. 1988. KMS: A distributed hypermedia system for managing knowledge in organizations. *Commun. ACM*, 31, 7, (July), 820-835.
- Belkhatir, N., and Estublier, J. 1986. Protection and cooperation in a software engineering environment. *Proceedings: Advanced Programming Environments*, (Trondheim, Norway, June), *Lecture Notes in Computer Science*, 244, Springer-Verlag, pp. 221-229.
- Chou, Hong-Tai, and Kim, Won. 1986. A unifying framework for version control in a CAD environment. *Proceedings of the 12th International Conference on Very Large Data Bases*, (Kyoto, Japan, August). pp. 336-344.
- Cohen, Ellis S., Soni, D., Gluecker, R., Hasling, W., Schwanke, R., and Wagner, M. 1988. Version management in Gypsy. *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, (Boston, Mass., November), pp. 201-215.
- Delisle, Norman, and Schwartz, Mayer. 1986. Neptune: A hypertext system for CAD applications. *Proceedings of the SIGMOD '86 Conference*, (Washington, D.C., May), pp. 132-143.
- Delisle, N., and Schwartz, M. 1987. Contexts – A partitioning concept for hypertext. *ACM Trans. Off. Inf. Syst.*, 5, 2, (April), 168-186.
- Demleitner, Norbert. 1988. PAPICS: A practical approach to configuration management. *Proceedings of the International Workshop on Software Version and Configuration Control*, (Grassau, FRG, January), pp. 381-390.
- Goldstein, Ira P., and Bobrow, Daniel G. 1984. A layered approach to software design. In *Interactive Programming Environments*, D. R. Barstow, H. E. Shrobe, and E. Sandewall, Eds., McGraw-Hill, New York, pp. 387-413.
- Halasz, F. 1988. Reflections on NoteCards: Seven issues for the next generation of hypermedia systems. *Commun. ACM*, 31, 7, (July), 836-852.
- Hicks, David L., Leggett, John J., and Schnase, John L. 1991. Version control in hypertext systems. Department of Computer Science Technical Report No. TAMU 91-004, Texas A&M University, College Station, Texas.

Kacmar, C. J., and Leggett, J. J. 1991. PROXHY: A Process-Oriented Extensible Hypertext Architecture. *ACM Trans. on Inf. Syst.*, 9, 4, (October), 399-419.

Katz, Randy H., Bhateja, R., Chang, E., Gedye, D., and Trijanto, V. 1987. Design version management. *IEEE Design & Test*, 4, 1, (February), 12-22.

Katz, Randy H. 1990. Toward a unified framework for version modeling in engineering databases. *ACM Computing Surveys*, 22, 4, (December), 375-408.

Khoshafian, S., and Copeland, G. 1986. Object identity. *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, (September), pp. 406-416.

Leblang, David B., and Chase, Robert P. 1984. Computer-aided software engineering in a distributed workstation environment. *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, (Pittsburgh, Penn., April), pp. 104-112.

Leblang, David B., and Chase, Robert P. 1987. Parallel software configuration management in a network environment. *IEEE Software*, 4, 6, (November), 28-35.

Mahler, Axel, and Lampen, Andreas. 1988. Shape – A software configuration management tool. *Proceedings of the International Workshop on Software Version and Configuration Control*, (Grassau, FRG, January), pp. 228-243.

Pearl, Amy. 1989. Sun's link service: A protocol for open linking. *Proceedings of the Second ACM Conference on Hypertext (Hypertext '89)*, (Pittsburgh, PA, November), pp. 137-146.

Prevelakis, Vassilis. 1990. Versioning issues for hypertext systems. In *Object Management*, D. Tschritzis, Ed., pp. 89-105.

Rochkind, Marc J. 1975. The source code control system. *IEEE Trans. on Softw. Eng.*, SE-1, 4, (December), 364-370.

Scacchi, Walt. 1988. Summary of plenary discussion "CM for non-textual representation." *Proceedings of the International Workshop on Software Version and Configuration Control*, (Grassau, FRG, January), pp. 363-368.

Schnase, J. L. 1992. HB2: A hyperbase management system for open, distributed hypermedia system architectures. Dissertation, Department of Computer Science, Texas A&M University, College Station, Texas 77843.

Schnase, J. L., Leggett, J. J., and Hicks, D. L. 1991. HB1: Initial design and implementation of a hyperbase management system. Department of Computer Science Technical Report No. TAMU 91-003, Texas A&M University, College Station, Texas.

Schnase, J., Leggett, J., Hicks, D., and Szabo, R. 1993. Semantic data modeling of hypermedia associations. *ACM Trans. Inf. Syst.*, 11, 1, (January), 27-50.

Schütt, Helge A., and Streitz, N. 1990. *HyperBase*: A hypermedia engine based on a relational database management system. In *Hypertext: Concepts, Systems and Applications, Proceedings of the European Conference on Hypertext*, (INRIA, France, November), A. Rizk, N. Streitz, and J. Andre, Eds., Cambridge University Press, pp. 95-108.

Shackelford, D. E. 1992. Implementation design document for the UNC distributed graph storage system. Department of Computer Science Technical Report TR92-015, The University of North Carolina at Chapel Hill, March.

Smith, J. B., and Smith, F. D. 1991. ABC: A hypermedia system for artifact-based collaboration. *Proceedings of the Third ACM Conference on Hypertext (Hypertext '91)*, (San Antonio, Texas, December), pp. 179-192.

Tichy, Walter F. 1985. RCS – A system for version control. *Software -- Practice and Experience*, 15, 7, (July), 637-654.

Tichy, Walter F. 1988. Tools for software configuration management. *Proceedings of the International Workshop on Software Version and Configuration Control*, (Grassau, FRG, January), pp. 1-20.

Weber, Anja. 1991. Versioning issues in hypermedia publishing environments. Arbeitspapiere 542, Gesellschaft für mathematik und dataverarbeitung MBH, Schloss Birlinghoven, Postfach 12 40, D-5205 Sankt Augustin 1, June 1991.

Wiil, Uffe K., and Østerbye, Kasper. 1990. Experiences with HyperBase – A multi-user back-end for hypertext applications with emphasis on collaboration support. The University of Aalborg Technical Report R-90-38, Institute for Electronic Systems, Department of Mathematics and Computer Science, Aalborg, Denmark.

Wiil, Uffe K. 1992. Issues in the design of EHTS: A multiuser hypertext system for collaboration. *Proceedings of the Hawaii International Conference on System Sciences (HICSS-25)*, (Kauai, Hawaii, January), pp. 629-639.

Wiil, Uffe, K., and Leggett, J. L. 1992. Hyperform: An extensible hyperbase management system. Department of Computer Science Technical Report No. TAMU-HRL 92-003, Texas A&M University, College Station, Texas.

Yankelovich, N., Haan, B., Meyrowitz, N., and Drucker, S. 1988. Intermedia: The concept and the construction of a seamless information environment. *IEEE Computer*, 21, 1, (January), 81-96.

Zdonik, Stanley B. 1986. Version management in an object-oriented database. *Proceedings: Advanced Programming Environments*, (Trondheim, Norway, June), *Lecture Notes in Computer Science*, 244, Springer-Verlag, pp. 405-421.

Østerbye, K., Nørmark, K., and Mejdahl Jeppesen, H. 1992. Hyperstructure programming environments. Presented at the Fifth Nordic Workshop on Programming Environment Research. (Finland, January). Obtainable through the author.