CHAPTER 6

INTEGRATION ISSUES

1. Introduction

Hypertext systems have not yet been accepted as fundamental tools for the augmentation of human intellect since most of them are "insular, monolithic packages that demand the user disown his or her present computing environment to use the functions of hypertext and hypermedia." [Meyrowitz, 1989]. Similar to the "cut and paste" paradigm in windowing environments, linking functionality must become an integral part of the computing environment. Application developers must be provided with tools to enable all applications to "link up" in a standard manner.

Most current hypertext systems are closed systems – material created in one system cannot be transferred or integrated with material created in another system because of proprietary document formats and storage mechanisms. Conversion programs are difficult to write since the formats are not disclosed by organizations [Fountain et al., 1990].

In order to make systems open and also integrate hypertext functionality into the desktop, researchers have been working on various hypertext models and interchange standards. This chapter explores these models and standards.

2. Models and Frameworks

2.1 Hypertext Abstract Machine (HAM)

One of the first approaches to a generic hypertext implementation model was the Hypertext Abstract Machine (HAM), "a general purpose, transaction–based, multi–user server for a hypertext storage system." [Campbell & Goodman, 1988]. HAM's emphasis was on developing an appropriate storage model. It provided a general and flexible model that could be used in several, different hypertext applications. The Hypertext System Architecture based on HAM contains the following layers (See Figure 6.1):

User Interface: A window–based interactive environment for applications to communicate with users.

Application: The actual application which may or may not run on the same machine as the HAM.

Hypertext Abstract Machine: An engine which manages all information about the hypertext and communicates with the application through a byte stream protocol.

Host file system or storage system: A repository to store all the hypertext graphs or databases.

Click here for Picture

Figure 6.1 Hypertext Abstract Machine [Campbell & Goodman, 1988]

The HAM storage model consists of five major objects: graphs (networks of nodes and links containing one or more contexts), contexts (partitions of data within a graph), nodes, links, and attributes carrying semantics. The following operations could be performed on HAM objects: create, delete, destroy, change, get, filter, and special. The HAM architecture provided version control, filtering and data security. The HAM storage model has been successfully tested against systems such as Guide, Intermedia, and NoteCards.

2.2 Link Engine/Hypermedia Engine/Link Service

Developers of Intermedia were interested in developing a multiuser hypermedia framework whereby hypertext functionality could be handled at the system level − linking would be available for all participating applications [Haan et al., 1992]. They proposed "The IRIS Hypermedia Services" to provide an integrated desktop environment for hypermedia applications such as InterWord, InterDraw, InterVal, InterVideo, and InterPlay. These services contain the following components: Intermedia Layer, Link Client, and Link Server. These components are independent of both operating system and Graphical User Interface.

The documents themselves are stored as Unix files while the link and anchor data are stored in a DBMS. The Link Client, the Link Server, and the DBMS together form the Link Engine. The Intermedia Layer is responsible for all live data manipulation while the Link Engine is responsible for the storage and retrieval of link data. With such an approach Intermedia documents could be interchanged with KMS using the Dexter Interchange Format (described in Section 3.?).

According to Intermedia researchers, following are the requirements to make hypermedia an integrated part of the computing environment:

1. Integration of hypermedia into the desktop. The Link Engine must be integrated into the computing environment just as the file system is today. A higher level toolkit or an application programmer interface (API) must be provided for application developers to issue calls for hypermedia support.

2. Hypermedia systems must provide multiple contexts or multiple webs in order to fully exploit hypertext linking across all applications.

3. Hypermedia applications must support filtering and incremental query construction.

4. Wide Area Hypermedia − Hypermedia functionality must be extended to support Wide Area Networks in addition to LANs.

5. Building an integrated hypermedia environment is made easier with object−oriented techniques. Also, the most logical DBMS to use for storing link and anchor data would be an Object−Oriented Data Base Management System.

In order to provide an integrated desktop with full hypermedia functionality, Bieber has proposed a system−wide hypermedia engine based on the notion of a generalized hypermedia using bridge laws (See Chapter 2, Section 5 − Dynamic Hypertext) [Bieber, 1993]. This engine would bind independent back−end applications such as Decision Support Systems, Expert Systems, Databases and front−ends (interface−oriented applications such as word processors, graphics packages) through message−passing mechanisms. Bridge laws map the objects defined in the back−end such as models, variables, calculations to objects in the front−end such as nodes, links, and link markers.

Bieber has suggested the following front−end and back−end requirements for system−level approaches to hypermedia integration or client/engine cooperation.

Front−end Requirements:

In order for the hypermedia engine to provide functionality such as management of link markers, comments, trails, overviews, filters, forward navigation and backtracking, the front−end should provide the following:

Tracking the location of objects such as link markers and providing their identifiers to the engine when a link marker is selected.

Front−end must request from the engine editing permissions for insertions, deletions, and modifications.

User interface must provide hypermedia prompts.

When the front–end saves a document with embedded hypermedia objects, the objects should also be saved.

Back–end Requirements:

The hypermedia engine would provide functionality such as linking, annotation, backtracking, filtering, and overviews on behalf of the back–end. The back–end should provide the following functionality:

Provide specific information about its structure and its applications' documents.

Bridge laws must be written by developers. This could be done through a bridge law editor instead of writing predicate logic.

Back–ends should provide control information and interpretive mechanisms along with the objects that are sent through messages. For example, objects that have to serve as link markers could be tagged.

Back–ends should support hypermedia engine commands same as the front–end (command lists and context sensitive information).

Back–end should incorporate a standard document interchange standard such as ODA or SGML.

Similar to Intermedia's Link Engine and Bieber's Hypermedia Engine, Sun's Link Service offers an extensible protocol to create and maintain relationships between autonomous front–end applications [Pearl, 1989]. Similar to the approaches seen earlier, editing and storing of data objects is managed by independent applications which also provide some amount of front–end operations on links. The Link Service stores only the representations of the nodes rather than the nodes themselves. Thus, the definition and granularity of nodes are left to the individual applications. Also, the storage of node data is independent of the storage of link data.

The Link Service makes it easier for applications to add hypertext functionality by providing a simple protocol, a shared back–end or link server, a library, and utilities to manage the link database (See Figure 6.2). Applications communicate with the link server through the Link Service protocol. This service allows independent applications to integrate linking mechanisms into their standard functionality and become part of an extensible and open hypertext system. Existing text and graphics editors can be integrated into such a framework without any modifications. Due to the separation of node and link data, the Link Service does not provide version control, node content editors, concurrent multi–user access, or other forms of data integration.

<u>Click here for Picture</u>

Figure 6.2 Link Service – An Architecture for Open Hypertext [Pearl, 1989].

Some of the issues involved in developing such an open hypertext system include the following [Pearl, 1989]:

The User Interface for the creation and management of links should be consistent with the editors provided by the individual applications.

Since the Link Service and the applications are separate processes decisions must be made about sharing/dividing the responsibility for exception handling and user dialogs.

The Link Service should detect and remove dangling links, either implicitly or explicitly. Implicit removal can happen when a user tries to follow a link from its valid end to its invalid end by suggesting to the user to remove the link. Explicit removal can happen, through a link garbage collection mechanism, by tracking links, validating nodes and removing invalid links.

While versioning of data objects can be left to the individual applications, the Link Service must still handle the versioning of links. However, the consistency of a versioned hypertext cannot be guaranteed if nodes are versioned separately from links.

Unstructured documents such as ASCII files cannot be handled elegantly since they are not uniquely indexed nor do they carry semantics.

The issue of traversing links across networks and locating objects located at remote sites is very important due to performance and cost factors. Also, decisions have to be made about where on the network should the Link Service process be located. Another related issue is the invocation of an application which may not be currently running although the user is following a link to a node managed by that application.

2.3 Hypermedia Toolkit

The toolkit approach mentioned by Haan et al., has been attempted by Puttress and Guimaraes. They have proposed a toolkit that could be used by application developers to add hypermedia functionality to their existing toolkit, independent of specific applications or environment [Puttress & Guimaraes, 1990]. The hypermedia toolkit architecture is similar to other multi−tiered architectures (See Figure 6.3). The layers are: Application Software, Hypermedia Toolkit Layer, Storage System, and Representation System. The hypermedia toolkit consists of the following three components:

a) *Storage System Interface* (also called Eggs): This interface consists of a set of C++ classes, providing a hypermedia structure to the stored application data. It provides the mapping between the application above and the storage system below. Thus, the storage system can be modified or changed without modifying the application. Similar to the HAM approach, the data model is made of graphs, contexts, nodes, links, attributes, and symbols. This interface does not interpret node data − it is just considered as a stream of bytes with no structure or meaning. It provides version control and concurrency control mechanisms. There is finer transaction management under the control of the application.

b) *Application Interface:* This interface is composed of data objects that communicate with the application above.

c) *Representation System Interface:* This interface is responsible for the presentation of views using user interface toolkits, independent of the display platform. The Application Interface and the Representation Interface are made of a set of C++ classes, together called Hypermedia Object−oriented Toolkit (HOT). HOT provides the abstractions required for hypermedia applications while encapsulating the details of the storage and representation systems. HOT consists of Data classes that include: HGraph, HContext, HNode, and HLink. It also consists of View classes for each of the Data classes: HGraphView, HContextView, HNodeView, HLinkView and HFrame.

Click here for Picture

Figure 6.3 Hypermedia Toolkit Architecture [Puttress & Guimaraes, 1990].

Puttress and Guimaraes report that this architecture will be extended to support multi−user environments, to provide effective means of sharing and communication between users of hypermedia applications, and exploring means to the development of collaborative hypermedia [Puttress & Guimaraes, 1990].

2.4 HDM

The Hypermedia Design Model (HDM) is a hypertext model being developed as part of the HYTEA project by an European Consortium [Garzotto et al., 1991]. The basic features of HDM include the representation of hypertext applications through primitives: types entities composed of hierarchies of components; different perspectives for each component; units corresponding to component−perspective pairs; bodies representing the actual contents of units; structural links relating components belonging to the same entity; application links relating components belonging to different entities; and browsing semantics determining the visualization and dynamic properties of the application. These primitives are similar to objects defined in HAM.

The HDM is concerned with authoring−in−the−large or the definition of the topology of the hypertext network. It does not deal with authoring−in−the−small or filling in the contents of nodes and their presentation. This is because, Garzotto et al., believe that systematic and rational structural decisions about the hypertext should be made before the actual hypertext is written so that a coherent and expressive application can be developed from the very beginning instead of being added later. These HDM design specifications can be translated automatically into a lower−level node and link specification resulting in the actual implementation of the topology. HDM is still evolving and requires experimentation with a large number of applications. It is a step to induce a methodology for hypertext design based on a top−down, model−based approach.

2.5 Dexter Hypertext Reference Model

The Dexter Hypertext Reference Model captures the important abstractions found in a wide range of existing and future hypertext systems [Halasz & Schwartz, 1990]. The goal of the model is to provide a systematic basis for comparing systems and to develop interchange and interoperability standards. The Dexter model divides a hypertext system into three layers (See Figure 6.4):

a. Runtime Layer

This layer deals with the presentation of hypertext and the dynamics of user interaction. Since it is too broad and diverse to be developed into a generic model, the Dexter model does not go into the details of the presentation mechanism. However, presentation mechanisms can be specified containing information about how a component/network is to be presented to the user. These presentation specifications provide an interface between the runtime layer and the storage layer.

b. Storage Layer

This is the main focus of the Dexter model. It models a database that is composed of a hierarchy of data−containing components which are interconnected by relational links. Components have unique identifiers and links can be identified by a set of two or more component identifiers. Components correspond to the general notion of nodes and can contain text, graphics, images, audio, video etc. The components are treated as generic containers of data and the model does not specify any structure within the containers. Thus, the storage layer does not differentiate between text components and graphics components. It focuses mainly on the mechanism by which components and links are tied together to form hypertext networks.

c. Within Component Layer

This layer is concerned with the contents and structure within components of the hypertext network. Since the range of possible content/structure that can be included in a component is open−ended, the Dexter model treats this layer as being outside its scope. The assumption is that document structure models such as ODA, SGML, IGES etc., will be used in conjunction with this model to capture content/structure. However, a critical interface between the storage layer and the within−component layer called anchoring discusses the mechanism of addressing locations or items within the content of an individual component. Anchors can identified by a unique anchor identifier.

Click here for Picture

Figure 6.4 Dexter Hypertext Reference Model [Halasz & Schwartz, 1990].

Halasz and Schwartz claimed that no existing systems support all the mechanisms discussed in the Dexter model. However, existing systems are yet to be fully compared to this model. The model has been used in developing the Dexter Interchange Format, a hypertext interchange standard.

2.6 Trellis Hypertext Reference Model

The Trellis Hypertext Reference Model or "r–model" looks at hypertext as different levels of abstraction [Furuta & Stotts, 1990]. In general, hypertext can be divided into:

a) Abstract Level: This layer is made of abstractly defined independent components that are connected together in some fashion. It does not describe the details of presentation.

b) Concrete Level: Concrete representations in which the characteristics of the hypertext's physical display have been established. That is, the contents of each of the windows is specified but not laid out.

c) Visible Level: This layer is responsible for the layout and presentation of the hypertext network on a physical display.

The representations in the abstract level are at the greatest level of abstraction while those in the visible level are the lowest level. Whereas the abstract level can be standardized using document exchange protocols such as SGML, the visible level can be standardized using X Windows.

While some researchers believe that hypertext can be standardized, there are others who contend that the current notion of hypertext is not mature enough to be standardized. There is the third group which believes that currently identified and well–established features of hypertext may be standardized [ECHT '90 Panel Discussion, 1990]. Hardt–Kornacki et al., feel that "the components of hypermedia that are ready for standardization are not necessarily hypermedia–specific and the hypermedia aspects of these systems are not yet ready for standardization." [Hardt–Kornacki et al., 1990]. They contend that objects which have data and structure need rich and flexible standard representations in matters of exchange and authoring. However, since the same objects might be involved in non–hypermedia applications, they question the prudence of hypermedia–based object presentation standards. However, it is not clear if they mean that models such as Dexter and Trellis are not yet required. Also, standardizing the user interface for hypermedia systems is premature and naive. They add that it may be beneficial to standardize a generic set of tools that can be used to build various components of hypertext. They contend that none of the supporting infrastructure in a hypermedia environment such as display devices, user interfaces, file systems, broadband communications etc., are standardized.

However, such layered approaches (back–end, engine, and front–end) require significant efforts from software developers in writing exchange protocols between the layers. If these protocols are application dependent, they cannot serve as standard approaches. Hence, Isakowitz feels that alternatives to the layered architecture approach should also be explored [Isakowitz, 1993]. Mechanisms have to be devised to automate this layer–to–layer communication so that they are independent of applications.

2.7 General Hypertext Framework

Whereas most models have focused on design metaphors and implementation abstractions, very little work has been in the area of a general framework for hypertext functionality. Rao and Turoff observed that "Hypertext should be treated as a general purpose tool with approaches to handling nodes, links, and retrieval, that fits within the context of any application and conveys common meanings to users. To accomplish this, we need a comprehensive framework for hypertext based on a cognitive model that allows for the representation of the complete range of human intellectual abilities." [Rao and Turoff, 1990]. They proposed such a framework based on Guilford's Structure of the Intellect Model (See Figure 6.5). They contend that hypertext systems tend to suffer from a lack of coherence due to ambiguity in meanings assigned to nodes and links. This framework classified nodes into six different semantic types – detail, collection, proposition, summary, issue, and observation. Links can be categorized into major types – Convergent links and Divergent Links. Convergent links can be classified into specification, membership, association, path, alternative and inference links. These links help in focusing or narrowing the pattern of relationships between ideas. Divergent links are classified into elaboration, opposition, tentative, branch, lateral, and extrapolation links. These links expand or broaden relationships between ideas.

Click here for Picture

Figure 6.5 General Framework for Hypertext Functionality [Rao & Turoff, 1990].

Rao and Turoff believe that such a comprehensive framework would help designers develop better interface metaphors and implementation models for hypertext systems. They concluded that sixteen different hypertext systems which they reviewed fall under this framework (in their own limited ways) and that their semantic morphology could be extended to all future systems. Such a taxonomy would also help collaborative hypertext where members of a group could contribute adequately and understand each other's judgements in carrying out the group objective. A first step towards the implementation of a hypertext system based on such a framework is to develop an appropriate design metaphor/user interface that would reduce functional opacity (mismatch between the framework and the metaphor) and system opacity (mismatch between the metaphor and the implementation model). That is, the region called design metaphor in Figure 6.5 could be expanded on all four sides thus "squeezing out" the shaded regions named functional opacity and system opacity.

3. Interchange Standards

Unlike linear documents which are static, generic, and structured, hypertext documents are unstructured and can be dynamic. Hence, current structured document standards are not sufficient to represent hypertext networks. A tree based hierarchy is relevant but NOT sufficient for hypertext. There should be a hierarchical framework with a system of typed links to cover the cross–references of structured documents and the links of hypertext. The current forms of ODA and SGML are not sufficient enough for the representation and exchange) of hypertext. These need extensions to provide a proper typed–link mechanism. SGML does not specify layout or presentation information (which is important for hypertext) or how to handle images and graphics. ODA does address these issues but it is not sufficient.

At the same time, a single standard may not be enough due to the diversity of usage of hypertext applications – large volume hypertext systems are different from highly interactive systems. While the former require highly efficient search capabilities (and standards), the latter require better individualized responses and navigational tools. Hence, these two may require different standards.

In this section, we discuss specific limitations of SGML and ODA and research efforts to extend them to handle hypertext [Newcomb et al., 1991]. Some researchers are also interested in combining the best of ODA and SGML and extending them to form a comprehensive hypermedia standard.

3.1 Limitations of SGML

1. SGML allows cross–referencing within the same document. This can be done by assigning unique identifiers to elements that need to be referred to elsewhere. However, the uniqueness of the identifier (and hence, the element) is applicable only within the current local document. Hence, only elements within the

same document (and only those having unique identifiers) can be linked. Therefore, this mechanism can only be used in a hypertext document to refer to elements within the same document and not other documents.

2. SGML cannot support time dependent data such as audio and video and also graphics and images. Rendering of events is not possible in SGML, that is, displaying a map of NY and a link that zooms into Manhattan.

3.2 Limitations of ODA and possible modifications

ODA, a standard for the storage and interchange of multimedia documents, deals with both logical structure and layout structure or presentation (unlike SGML). ODA currently includes graphics and images and extensions are being considered to handle audio, video, and hypertext [Cole & Brown,1990].

a. Separation of logical structure and layout structure

Though ODA supports both logical structure and layout structure, they are not completely separated. In order to change the style of a document the logical structure must be edited since the layout process uses the logical structure, the generic structures and the content architectures to create the specific layout. This limitation can be eliminated by carrying over the SGML mechanism of applying different set of layout and presentation styles (or style sheets) for different views of the same logical document.

b. Comprehensive attribute inheritance

The ODA mechanism for inheriting layout attributes (such as placement of blocks of contents within pages and rectangular areas called frames) and presentation attributes (such as character sets and the placement of items within blocks) is not sufficient. If an attribute value is not specified for the object or its class, then the value can only be inherited according to the object's position in the tree and not according to its class (chapter, list etc.).

Attribute inheritance can be achieved by adding a facility called "style tables" which will enable the style inherited by an object (and hence its format) to depend both on its class and is position in the document. This will be very valuable for hypertext in order to distinguish between objects of the same type that have different status (such as open and close buttons). It can also be extended to specify changes of state (for example, when selecting a hotspot) by changing the style table.

c. Links

ODA does not have the ability to specify the purpose of a link and also how the layout process can express that purpose. This can be accomplished by having classes for links (just as there are classes for logical objects). The class of the link will determine how and where in the document the link can be used. Thus, the representation of the link will depend on both the class and its position in the document.

d. Selective and multiple presentation

ODA does not have the ability to suppress the appearance of a logical object (or contents) during the layout process nor the ability to present the object many times. Such a feature will be of great help in a hypertext document where a reviewer's comments can be suppressed from appearing in a printout or different versions of the same basic document can be produced for various purposes. This can also be accomplished by the usage of style tables suggested earlier.

e. Complete interactivity

The ODA layout process is sequential and page based and hence does not provide complete interactivity. It does not support online editing capabilities such as the ability to scroll through a document, the ability to display selected items (outlining facility), the ability to popup additional information on demand (such as footnotes, glossaries etc.), the ability to "fold" documents revealing hidden sections only on request, the ability to follow links automatically.

Complete interactivity would require extensions. Outlining can be done by having style tables that select objects by class and required level. Popup displays can be arranged by changing to a different style table and returning to the original table after the popup information has been displayed. Similarly, folding can be achieved indirectly through popups and popdowns. Link traversal can be done by replacing the current object with the target object or displaying the target object as a temporary popup item. A style table can be used to specify whether or not to display the linked object.

3.3 HyTime

The Hypermedia/Time−based Structuring Language or HyTime is and International Standard for representing hypertext links, and synchronization of static and time−based information contained in multiple conventional and multimedia documents and information objects [SIGLINK, 1992]. It addresses the limitations of SGML [Newcomb et al., 1991]. HyTime supports cross−referencing facilities to uniquely identified elements in external documents. It also extends SGML's reference capability to accommodate elements with no unique identifiers in the same document. It provides pointers or location addressing schemes that contain the necessary information in order to locate cross−referenced data. It is independent of data content notations, link types, processing, presentation, and semantics. HyTime supports addressing by name, by position in the document, and by semantic construct. Links can be established to documents that conform to HyTime as well as those that do not.

HyTime allows all kinds of multimedia and hypertext technologies (whether proprietary or not) to be combined in any information product. It addresses only the issue of interchange of hypermedia information and not the standardization of presentation (same as SGML), user interfaces, query languages etc. Objects in a HyTime hypertext document can include formatted and unformatted documents, audio and video segments, still images, animations, and graphics.

HyTime is an SGML application conforming to ISO 8879. It provides the notion of "Architectural Form" to SGML. An architectural form is a syntax template around which a document author can build semantic constructs for linking and coordinate space addressing. It is highly flexible and extensible. The interchange format can be defined in Abstract Syntax Notation 1 (ISO 8824) and can be encoded according to the basic encoding rules of ISO 8825 for interchange using protocols conforming to the OSI model. The full set of HyTime functionality supports "integrated open hypermedia", the "bibliographic model" of hyperlinking that allows links to anything, anywhere, anytime.

HyTime is intended for use as the infrastructure of platform−independent information exchange for hypermedia and synchronized and non−synchronized multimedia applications. Application developers will use HyTime constructs to design their information structures and objects and the HyTime language to represent them for interchange [SIGLINK, 1992].

3.4 MHEG

CCITT has proposed the future international standard for multimedia and hypermedia information objects, also known as the MHEG Standard. "The scope of the MHEG standard is to define the representation and encoding of multimedia and hypermedia information objects that will be interchanged as a whole within or across applications or services, by any means of interchange including storage devices, telecommunications or broadcast networks." [CCITT, 1992]. The initial objectives of the MHEG standard include meeting the following requirements:

Provide abstractions for real−time presentation including multimedia synchronization and interactivity.

Provide abstractions for real−time interchange with minimal buffering using normal speed data communications.

Provide abstractions for direct manipulation of information without any additional processing.

Provide linking facilities between elements of composite multimedia objects.

The main MHEG classes include: Content Class, Selection Class and Modification Class, Link Class, Script Class, Composite Class, and Description Class. The objects play a federating role, enabling different applications to share the basic information resource. These objects can be encoded using ASN.1 or SGML and will provide a common base for other CCITT recommendations, ISO and other standards, user defined architectures and applications.

4. Summary

Hypermedia systems have been closed systems with proprietary storage mechanisms and very little or no interoperability. A number of layered architectures, models or engines, and frameworks have been proposed and developed by researchers in an effort to make hypertext systems more generic and integrated into the desktop environment. Application development toolkits have been developed to assist programmers in adding hypertext functionality to existing systems. In order to make hypertext systems fully open and integrated, the following issues must be addressed: interoperability, programmability, node and link typing, distributed linking, concurrency control for multi−user access in a shared environment, maintaining public and private links, operating systems support, networking, bridge laws, linking protocols, multimedia support, operating systems support, user interface consistency, and version control [Malcolm et al., 1991]. Most of these requirements can be addressed using object−oriented techniques [Lange, 1993].

In order to make hypertext systems fully portable, existing document standards such as ODA and SGML must be extended to support unstructured documents and linking. International standards such as HyTime and MHEG are emerging to support hypertext functionality and multimedia information in applications. Only when hypertext functionality becomes an integral part of our computing environment will knowledge workers accept and incorporate hypertext into their daily work process.

References

[Bieber, 1993]. Bieber, Michael. Providing Information Systems with Full Hypermedia Functionality, Proceedings of the Twenty−Sixth Hawaii International Conference on System Sciences, 1993.

[CCITT, 1992]. CCITT Press Release on MHEG, December 1992.

[Campbell & Goodman, 1988]. Campbell, Brad, and Goodman, Joseph M. HAM: A General Purpose Hypertext Abstract Machine, Communications of the ACM, July 1988.

[Cole and Brown, 1990]. Cole, Fred and Brown, Heather. Standards : What can Hypertext learn from paper documents ?, Proceedings of the Hypertext Standardization Workshop by NIST, Jan 1990.

[ECHT '90 Panel Discussion, 1990]. Panel Discussion. Hypertext and Electronic Publishing, Proceedings of European Conference on Hypertext, November 1990.

[Fountain et al., 1990]. Fountain, Andrew M., Hall, Wendy, Heath, Ian and Davis, Hugh C. MICROCOSM: An Open Model For Hypermedia With Dynamic Linking, Proceedings of ECHT '90, 1990.

[Furuta & Stotts, 1990]. Furuta, Richard and Stotts, P. David. The Trellis Hypertext Reference Model, Proceedings of the Hypertext Standardization Workshop by National Institute of Science and Technology (NIST), January 1990.

[Garzotto et al., 1991]. Garzotto, Franca, Paolini, Paolo, and, Schwabe, Daniel. HDM – A Model for the Design of Hypertext Applications, Proceedings of Hypertext '91, ACM Press,December 1991.

[Haan et al., 1992]. Haan, Bernard J., Kahn, Paul, Riley, Victor A., Coombs, James H., and Meyrowitz, Norman K. IRIS Hypermedia Services, Communications of the ACM, January 1992.

[Halasz & Schwartz, 1990]. Halasz, Frank and Schwartz, Mayer. The Dexter Hypertext Reference Model, Proceedings of the Hypertext Standardization Workshop by National Institute of Science and Technology (NIST), January 1990.

[Isakowitz, 1993]. Isakowitz, Tomas. Hypermedia, Information Systems, and Organizations: A Research Agenda, Proceedings of the Twenty–Sixth Hawaii International Conference on System Sciences, January 1993.

[Lange, 1993]. Lange, Danny. Object–Oriented Hypermodeling of Hypertext Supported Information Systems, Proceedings of the Twenty–Sixth Hawaii International Conference on System Sciences, January 1993.

[Malcolm et al., 1991]. Malcolm, Kathryn C., Poltrock, Steven E., and Schuler, Douglas. Industrial Strength Hypermedia: Requirements for a Large Engineering Enterprise, Proceedings of Hypertext '91, ACM Press, December 1991.

[Meyrowitz, 1989]. Meyrowitz, Norman. The Missing Link: Why We're All Doing Hypertext Wrong, The Society of Text, MIT Press, 1989.

[Newcomb et al., 1991]. Newcomb, Steven, ????, and Newcomb, Victoria. The HyTime Hypermedia/Time–based Document Structuring Language, Communications of the ACM, November 1991.

[Pearl, 1989]. Pearl, Amy. Sun's Link Service: A Protocol for Open Linking, Proceedings of Hypertext '89, ACM Press, 1989.

[Puttress & Guimaraes, 1990]. Puttress, J.J., and Guimaraes, N.M. The Toolkit Approach to Hypermedia, Proceedings of the European Conference on Hypertext '90, 1990.

[Rao & Turoff, 1990]. Rao, Usha & Turoff, Murray. Hypertext Functionality: A Theoretical Framework, International Journal of Human–Computer Interaction, 1990.

[SIGLINK, 1992]. SIGLINK Newsletter. Excerpts on the HyTime International Standard, March 1992.