Sun's Link Service: A Protocol for Open Linking

Amy Pearl

Sun Microsystems
2550 Garcia Avenue
Mountain View, CA 94043

ABSTRACT

Sun's Link Service, a product shipped with Sun's programming in the large software development environment, the Network Software Environment, allows users to make and maintain explicit and persistent bidirectional relationships between autonomous frontend applications. The Link Service defines a protocol for an extensible and loosely coupled, or *open*, hypertext system. An interesting instance of this is the ability to link to objects in *closed* hypertext systems if they integrate with the Link Service. The Link Service addresses link maintenance and automated versioning. Link endpoints, or nodes, are defined by the integrating applications, and are not restricted to points, whole documents, or *cards*.

INTRODUCTION

The majority of the current generation hypertext systems are monolithic: they manage the storage of data as well as linking information. They are designed to be *turnkey*, or complete, authoring tools, supporting some range of media and offering users the ability to make connections between nodes in the system. The author edits data, of whatever media, in the hypertext system, and organizes the data into nodes, using the system's hypertext user interface. The system provides all available editors for manipulating the data as well as mechanisms for manipulating the links. The node data representation in these hypertext systems is structured and proprietary. All these frontend applications are closely coupled with the underlying hypertext substrate. As a result, these systems are *closed* because authors are dependent on the node editing capabilities that the hypertext system provides. If the system does not provide the particular media type editor that the user requires, only the hypertext system developers can make that functionality available. [Aksc87] [Camp87] [Engl84] [Trig86] Some other problems that result from such closed systems are the inability to link to pre-existing objects, and the inability to access the objects in the hypertext from outside the hypertext system.

Sun's Link Service assumes a layering of editing functions, where the management of data and the management of links is loosely coupled. The link and node data are stored separately. Editing and storing of objects is managed by independent editing applications, which also provide a portion of the frontend operations for operating on links. The Link Service stores only *representations* of the nodes, rather than the nodes themselves. This permits the nodes to be represented in any data format, and any existing application to be added as a front end. The majority of the linking functionality, including storage of the link information, is provided by a separate, shared backend. The Link Service provides both this backend as well as a library for the applications that defines a protocol for integrating with the Link Service. In such an *open* hypertext system, independent applications can integrate linking mechanisms into their standard functionality, and become part of an extensible, loosely coupled, frontend interface to the hypertext system (see figure 1). Incorporating existing editors aids users, since they don't have to adapt to

new editors, and the range of media available is independent of that explicitly provided by the Link Service. One interesting feature is that the frontend application the user may wish to have incorporated with this service may be another, presumably closed, hypertext system. For example, if a closed hypertext system such as KMS were integrated with the Link Service, users could make links between any KMS object and any object managed by another application integrated with the Link Service.

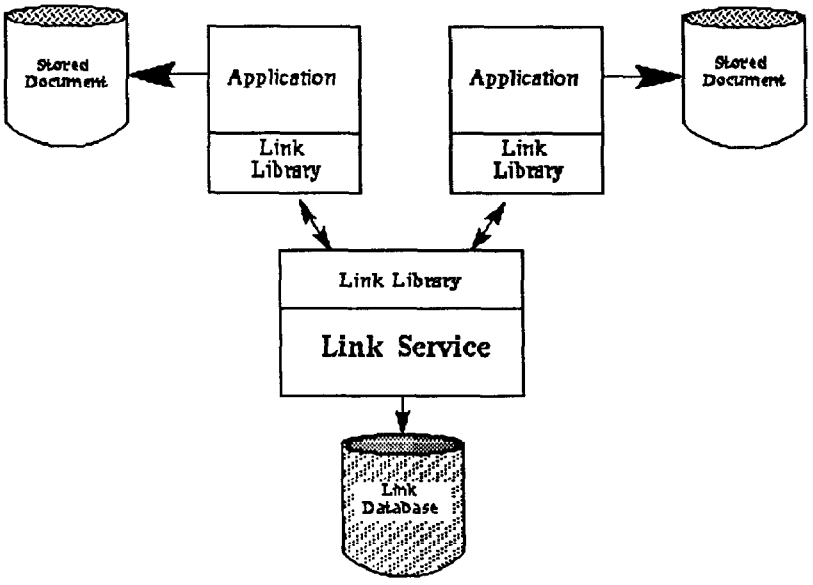

Figure 1. An Architecture for Open Hypertext

The Link Service is a product shipped with Sun's programming in the large software development environment, the Network Software Environment, NSE. We would like to see integration with the Link Service become a standard part of each application running on Sun workstations. In this paper I describe some of the issues involved in open hypertext, the features and implementation of our system, and, finally, some future goals.

ISSUES IN OPEN HYPERTEXT

An open hypertext system emphasizes autonomy and extensibility. This frees the frontends from the constraints of homogeneity, which is useful when considering application-specific details. In the case of editing objects, application-specific refers to such things as how to create and manipulate the application's objects. In the case of manipulating links the definition of a node, or linkable object, is application-specific. The primary cost of this flexibility is a potential loss of consistency, notably in application controlled areas such as the user interface and versioning. The Link Service specifies only the linking behavior for the frontend applications. Because our layers separate control of object and link data management, the Link Service has limited control over versioning, and has no control over the user interface for editing objects, the applications' portion of the user interface for handling links (link indication and object selection), concurrent multi-user features, data integration, or other interactions between frontend applications. Maintaining link consistency between a set of changing documents not under central management is also a difficult issue in open hypertext. In this section, I describe these issues, and how the Link Service addresses them. I also describe a particular

problem that face applications whose data has no structure, like flat ASCII text editors, and some of the issues in the SunOS environment, which is multi-tasking and distributed.

Linkable objects

Since most linking systems consider the data objects to be part of the information that they manage, they must go to some lengths constraining what comprises a linkable object, or *node*. For simplicity, many of these systems require that one or both ends be a whole unit, like a card, or document [Conk87]. The Link Service has no control over the objects, and therefore the definitions and granularity of nodes are left to the applications themselves. Nodes may be whole documents, spans of text, points (either insertion or geometric), whatever the managing application can uniquely identify and reconstitute from that unique identity. The Link Service requires that applications mark their objects that have links with a link indicator, such as an icon, or glyph. For consistency, we provide a particular bitmap (see figure 4).

User Interface

One of the design goals for the Link Service was to have minimal impact on the look and feel of the integrating applications. Our goal was to be minimally intrusive both for application integrators as well as for application *users* who may be unconcerned with linking. We accomplished this by segregating the user interface for *linking* as much as possible from the application's pre-existing user interface for editing. Thus, the application integrator has minimal work to do to their user interface to integrate with the Link Service. And users who previously used the application see minimal change to its user interface. The majority of the linking interface is contained in the separate linking command panel, that, like the applications, resides on the desktop. This appears the same across applications (see figure 2). Only two aspects of the user interface for linking affects these applications, and here is where inconsistencies can arise:

- they must provide some mechanism for selecting or indicating their objects when the user wants to do link operations on them
- they must provide some visual indication (such as an icon or glyph) for their objects that have links.

Notably, several of the applications that have already integrated with the Link Service have different *selection* paradigms. Preserving the user interface of existing CASE applications effectively eliminates general interface consistency among these applications. The implementation of the functions for editing the data are left to the application, and the linking interface doesn't affect these operations at all.

Another user interface issue raised by applications integrating with the Link Service is the distribution of function. Because the Link Service and the applications are separate processes, decisions must be made about which of them is responsible for exception handling or dialogues with the user. In general, these issues are not faced by closed hypertext, as the system is designed as an integrated application. The Link Service can only achieve some of this consistency of user interface if user interface standards, such as OpenLook [Hoeb89], are applied to all the applications.

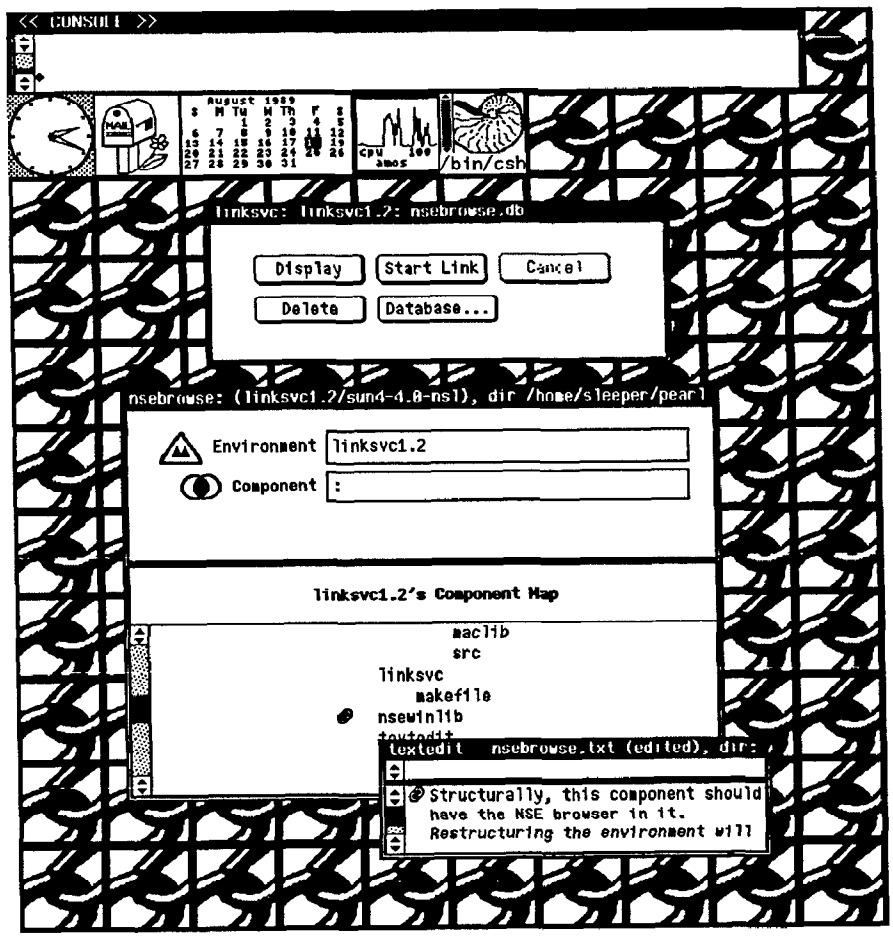

Figure 2. The NSE browser, *textedit*, and the Link Service Command Panel

Link maintenance

When one end of a link is deleted (e.g. a file containing the reference to the end of the link is deleted), the link becomes invalid. In an open hypertext system, it is difficult for the system managing the links, and consequently the application managing the node at the other end of the link, to know that the link is no longer valid. In the Link Service there are two mechanisms for discovering dangling links: implicit and explicit. The implicit discovery occurs when a user attempts to follow from the valid end of the link to the invalid node. The Link Service informs the user that the link is no longer valid and suggests deletion of the link. However, if both ends of the link become invalid before a user attempts to traverse the link, the invalid link cannot be discovered in this way. The Link Service provides an explicit link garbage collection mechanism, called *Confirm All* that goes through and checks whether each object in each link is still valid by querying that object's managing application. A closed hypertext system can more easily detect such changes that disturb link consistency. It is possible to do so in an open hypertext system, but imposes additional burdens on the frontend applications.

Versioning support

Versioning in hypertext is generally a complex issue [Hala87]. Versioning of nodes may or may not be independent of specific versions of links to those nodes. While the Link Service leaves the issue of versioning of the data objects to the individual applications, it still must handle versioning of the *sets* of links, as Intermedia [Meyr86] does. Unfortunately, the consistency of a versioned hypertext body cannot be guaranteed if the

nodes are not versioned along with the links between them. Since the Link Service is unable to enforce any connection between the versioning of the data objects and the link objects, it cannot guarantee the consistency of the hypertext *corpus*. In a separate but related issue, users may want a link to connect to a specific version of a node. Applications *can* embed version information in the key for the object that they store with the link database. Targeting the right granularity of versioning for a hypertext system is a difficult issue, whether the system is open or closed. PIE supports versioning for both a set of links as well as the data objects [Gold87]. Controlling versioning policies, and maintaining consistency, is possible, and easier in such a closed hypertext system, though still not trivial.

Handling structureless documents

Applications that integrate with the Link Service store in the link database unique keys for their objects. This is easy for applications that maintain unique internal identifiers for their internal objects. Unfortunately, text editors and other applications that operate on unstructured documents, such as ASCII text files, assume little, if any, structure or semantics about these files. In the current implementation of our link-aware text editors, we use an admittedly too simple method for constructing the key. We assume that the object eligible for linking is a single full line of text. The key is the composition of the text file name and the contents of the line of text. When the editor subsequently has to display this object, it loads the file named in the key, and searches for the first line in the file that matches the line of text in the key. In Intermedia [Evet87] links are defined by position and extent, along with a timestamp, which are tracked as the document changes. This allows the linking of *arbitrary spans* of text. One problem with this method is the overhead of computing a link position after the file has been heavily edited. An additional problem in the non-object model world of Unix is that there is no protection of the file from modification by non-link integrated applications, which can disturb any accounting.

Issues in the distributed workstation world

A Sun workstation environment is multi-tasking and file systems on remote file servers may be transparently available to users. Several issues arise in this kind of an environment, not all of which are particular to open hypertext systems:

Remote nodes Because a user has access to remote file systems, links may be made between nodes that reside on different machines. There must be some addressing mechanism for the Link Service to locate the object on the network using information stored in the link. When, if ever, should it be apparent to a user that linked objects reside remotely? Following a link may be costly, especially if the destination node resides remotely. It may be desirable to indicate, or allow queries about, the *cost* of following a link. Hyperties [Shne86] allows users to preview nodes. It may be a good idea to allow users to at least preview the cost of following a link, if that cost may be high.

Link server location When users interact with the Link Service, where on the network should that process be located? Should there be one link server for each user, one for each physical or logical machine, or one for each notion of a 'working environment'? Must these servers communicate with each other?

Application invocation This issue is of particular concern in an open hypertext system, where the application frontends are separate processes from the hypertext server. At any time, many separate application frontends may or may not be executing. The user can follow a link to a node managed by an application that is not currently running. Should the link server invoke that application, and if so, by what mechanism? And, in a networked environment, on what machine?

IMPLEMENTATION

Whenever practical, it is good design practice to create systems that are extensible by published protocols. There is a tension between support for heterogeneity that is a goal of such open systems and the notion of integration, one of our major design goals, which is aided by homogeneity. An open system implies that the system can be extended to mediate different heterogeneous implementations; integration implies much greater cooperation and conformity. The Link Service modulates this tension by defining as simple and unrestrictive a protocol as possible. In this way we preserve the autonomy of individual tools, yet provide some measure of integration. Integrating with the Link Service requires minor modification of the applications. They implement a simple protocol, described below, requiring little change of them. The Link Service includes the protocol specification, a link *server* program, a library for integrating new applications, and utilities for managing the link databases. An integrated, link-aware, simple flat text editor (Sun's *textedit*) accompanies the Link Service. This section describes briefly what the end user actually sees, and the underlying architecture and protocol implementation.

What the User Sees

To start the Link Service, the user invokes it, resulting in the appearance of a main command panel.

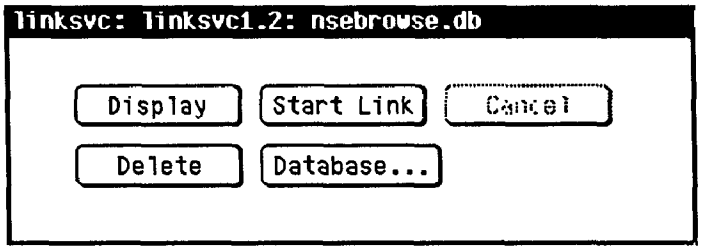

Figure 3. The Link Service Command Panel

Users explicitly invoke the integrated applications whose objects they want to view. In order to make a link, the user selects an object in one application, presses the *Start Link* button (which then changes to *End Link*) in the command panel to indicate the start of a link, selects the second object in an application, and presses *End Link* button to indicate the end of the link. The linked objects now have link indicators marking them.

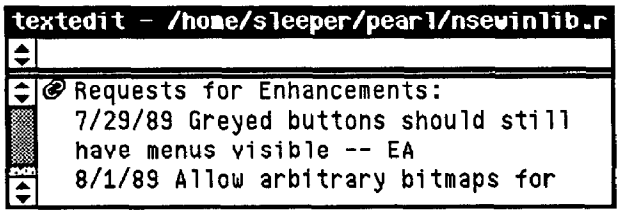

Figure 4. Link indicators in *textedit*

Subsequently, users can select either object, press the *Display* button in the command panel to indicate that they want to follow the link. Linkable objects can have multiple objects to which they are linked, as well as be the destination of multiple links. If there is only one link for the selected object, the object on the other end of the link is displayed by the application that manages it. If there is more than one link emanating from that object, the Link Service provides a dialogue box that lists the objects that can be

displayed. The user selects one of the objects, and the application of the object at the other end displays the object.

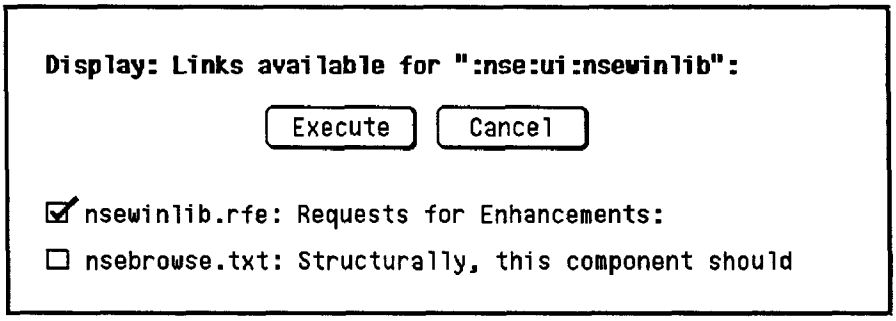

Figure 5. The Display Dialog Box

Since the Link Service is shipped as part of the NSE, its versioning facility supports the NSE's versioning paradigm. The NSE handles versioning for integrating between private and public workspaces. Versioning of NSE objects, such as link databases, follows a copy-modify merge paradigm [Adam89]. The merging mechanism for link databases is called *linkdb_resolve*. When a developer has completed some significant phase of development and testing, and wants to merge changes of a link database into a public workspace, they ask the NSE to merge the database back to the public workspace. The NSE automatically detects whether some other user has made changes in the public workspace since the user acquired the database into their private workspace. If so, the NSE invokes *linkdb_resolve*, which presents a window-based merging tool. Users may compare the version of the objects in the link database in their private workspace to those in the public one. In the merging subwindow, *linkdb_resolve* presents a best-guess merge of those link objects that appear to be able to coexist between the two workspaces. After running *linkdb_resolve*, users must run the explicit link maintenance function to detect any dangling links before again attempting to copy back their link database to the public workspace.

The protocol

The Link Service design separates the link data from the object data. The object data is managed by the individual application frontends. In order to make links between objects managed by different applications, these applications communicate using the Link Service. The Link Service command panel is part of a separate server process that runs on the user's machine. This process mediates communication between the integrated, link-aware applications. The Link Service provides a link library that applications include in order to communicate with the link server. This library is also part of the link server process. The applications communicate with the central link server process, which controls access to the link database (see figure 1).

This communication implements the Link Service protocol. The manager of the object *data* provides an interface for a user to create and modify those objects, and provides the application-specific functions to do what the user asks (e.g. edit text or graphics). Similarly, the Link Service provides the interface for the user to create and modify the *links* between the data objects, and provides the functionality for these requests. The link server manages access to a link database which stores the *link* data. When an application begins to execute, it registers with the Link Service, announcing its availability for displaying links, and telling the service what class of object, or data representation, it handles. Later it provides keys for its objects, and includes this class with those keys. This enables the link server to identify what registered application can handle requests about each object. The application also registers a set of functions that the Link Service

can call in order to communicate with that application. The Link Service sends three kinds of messages to the applications. These messages ask the application to:

- provide an application specific key (ASK) for the applications's currently selected object
- display the existence of an object with a specific ASK.
- validate the existence of an object with a specific ASK.

The Link Service also relays state changes to registered applications, e.g. if a link has been made to one of the application's objects. Link Service links are pairs of pointers to the linked objects. These connections are stored in link databases, named by the user, which leaves the applications' object storage undisturbed.

Integrated standalone applications

At present, we have integrated four Sun applications with the Link Service. They are: *textedit*, Sun's window-based simple text editor; *vi*, Unix's terminal-based simple text editor; the NSE browser; and *SunTraq*, Sun's project scheduling tool. Outside of Sun, third party vendors have integrated products such as document processing packages and CASE frontend tools such as structured analysis and design tools. All of these applications are window based, are event driven, and have some notion of selection within the application. They all existed before the Link Service, and were not originally developed with the Link Service in mind.

The primary issues that arise when integrating applications are: what is an identifiably linkable object in the application's object space, how can the application represent that object by a unique identifier, and how does the user select or identify the object on which they want to perform link operations? Intermedia [Meyr86] faced similar problems, but developed customized frontend applications (InterText, InterDraw, InterPix, InterSpect, and InterVal) themselves. This gave them greater control over their design, implementation, integration and consistency.

EXTENSIONS

The Link Service today is an initial attempt at a protocol that encourages independent applications to communicate about their objects in a fairly restrictive way. It has provided us with valuable feedback from application integrators about changes and improvements they would like to see. In this section I discuss the need to generalize the programmatic interface; the move at Sun toward a more consistent look and feel in all applications, and how that impacts our hypertext interface; and expanding the types of links that can be permitted between objects.

In the current implementation, the messages that applications are expected to support are few (display or verify this object), and specified by the protocol. We would like to permit applications to extend the set of possible messages arbitrarily. We expect the set of messages to expand to at least support notification and data transfer. This means that applications can agree on their own protocols for increased application integration.

As discussed earlier, imposing little control over the applications permits an inconsistent user interface across application frontends. However, Sun is currently addressing the general issue of a consistent look and feel among all applications that run on Sun workstations. OpenLook is the specification for such a consistent application user interface [Hoeb89]. OpenLook currently makes recommendations for selection (solving the above inconsistency), as well as for cut, paste, and property sheets, but not for other hypertext-specific user interface, such as link indication. Once applications' user interfaces

become more consistent with each other, by conforming to the OpenLook guidelines, we will be able to embed more of the linking user interface into the applications.

Currently, Link Service links are untyped, have no attributes, and no directionality. Types and attributes allow contextual behavior in link operations, and variation in the kind of relationship between endpoints. Types and attributes have been discussed in the hypertext literature [Trig83] [Conk87]. It is difficult to ascertain *a priori* the set of link types that integration programmers and end users will require. We prefer to allow the dynamic creation of types, rather than having a pre-defined fixed set, with a fixed set of associated semantics.

An example of the kind of hypertext application that the Link Service can enable is a frontend authoring application that, like Notecards or Hypercard, uses the card metaphor. The frontend would manage the cards, or the linkable objects they contained, and would register the unique identifiers for endpoints of links with the Link Service. All the customizing of display would be managed by the card authoring application. Another example of a hypertext-based application that could be developed using the Link Service as a platform is a tool to document the relationship between objects that are part of a software development cycle. This is, for example, one of the goals of DynamicDesign, from Tektronix [Bige87]. It should be possible to identify, using the software developer's existing applications, what requirements have not yet been fulfilled, or what bug reports have been satisfied by the current product. With an application that connects the relevant software development objects using the Link Service, the user could query the application about the state of these relationships.

CONCLUSION

With an open protocol, the power of each element of a system expands as it interoperates with others. Open linking can make the power of hypertext available to the world of software. We hope to see linking, and attendant hypertext capabilities, as much a standard part of the computer desktop as the cutting and pasting of text are today.

ACKNOWLEDGMENTS

For their contributions to the design of Link Service 1.0, I would like to acknowledge Dan Walsh, Russel Sandberg, Dave Goldberg, Evan Adams, Azad Bolour, Masahiro Honda, Scott Ritchie, and Tom Lyon. Sami Shaio and Judith Parkes were major contributors to our thinking about linking to unstructured documents. Azad Bolour provided the interface for *dbm*, a Unix database; Thomas Maslen integrated *textedit*, our first application, and provided a link database interface for the NSE's merge program.

My thanks to Dennis Abbe, who said of the design "this sounds a lot like hypertext," and handed me articles on Notecards. And my thanks to Carolyn Foss, Dwight Hare, Sami Shaio, and Nancy Yavne for reviewing this paper.

REFERENCES

[Adam89] E. Adams, M. Honda and T. Miller. "Object Management in a CASE Environment", Proceedings of the 11th International Conference on Software Engineering, May 15-18, 1989, Pittsburgh, pp. 154-163.

[Aksc87] R. Akscyn, D. L. McCracken, and E. Yoder. "KMS: A Distributed Hypermedia System for Sharing Knowledge in Organizations." *Hypertext '87 Papers*, Chapel Hill, NC, November 13-15, 1987.

[Bige88] J. Bigelow. Hypertext and CASE, IEEE Software,V5 N2, March, 1988.

[Bige87] J. Bigelow and V. Riley. Manipulating Source Code in DynamicDesign, *Hypertext '87 Papers*, Chapel Hill, NC, November 13-15, 1987.

[Camp87] B. Campbell and J. M. Goodman. "HAM: A General-Purpose Hypertext Abstract Machine," *Hypertext '87 Papers*, Chapel Hill, NC, November 13-15, 1987.

[Conk87] J. Conklin. "Hypertext: An Introduction and Survey," IEEE Computer, September, 1987, pp. 17-41.

[Engl84] D. C. Englebart. "Collaboration Support Provisions in AUGMENT," Proceedings of the AFIPS Office Automation Conference, Los Angeles, California, February 20-22, 1984, pp. 51-58.

[Evet87] C. Evett. "Tracking the End-Points of Persistent Links in Text Documents," *Hypertext '87 Position Paper*, Chapel Hill, NC, November 13- 15, 1987.

[Gold87] I. Goldstein and D. Bobrow. A layered approach to software design., In D. Barstow, H. Shrobe, and E. Sandewall (Eds.) *Interactive Programming Environments*. McGraw-Hill: 1987, pp. 387-413.

[Hala87] F. G. Halasz. "Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems," *Hypertext '87 Papers*, Chapel Hill, NC, November 13-15, 1987.

[Hoeb89] T. Hoeber. The OPEN LOOK Graphical User Interface Style Guide, Sun Microsystems, May 1989

[Meyr86] N. Meyrowitz, et. al. *Intermedia: The Architecture and Construction of an Object-Oriented Hypermedia System and Applications Framework*, Proceedings of OOPSLA '86, Portland, OR, September, 1986.

[Shne86] B. Shneiderman and J. Morariu. "The Interactive Encyclopedia System (TIES)," Department of Computer Science, University of Maryland, College Park, MD 20742, June 1986.

[Trig83] R. H. Trigg. *A Network-based Approach to Text Handling for the Online Scientific Community*, Ph.D.. Thesis, University of Maryland, 1983.

[Trig86] R. H. Trigg, L. Suchman, and F. G. Halasz. "Supporting Colaboration in NoteCards." *Computer-Supported Cooperative Work (CSCW '86) Proceedings*, Austin, TX, December 3-5, 1986.