ABSTRACT

COMPUTER SCIENCE

West, Tracey W.   B.S. Clark Atlanta University, 1993
M.S. Clark Atlanta University, 1995

PASSWORD BASED SECURITY IN HYPERMEDIA SYSTEMS

Advisor: Dr. David Kerven

Thesis dated May, 1995

This study has been performed to investigate the problem of lack of security in hypermedia systems. This study was based on the architectural design of past and existing hypermedia systems. The Dexter Reference model was used as the basis for this research. As an extension of the model, a password-based security mechanism was designed.

The design approach used was to examine various hypermedia systems comparing and contrasting their similarities and differences. Issues of security and design methods were discussed as a way to increase the security in hypermedia environments.

The results of this research demonstrate that private information can be stored on hypermedia systems and made only accessible to authorized users. This refinement of the Dexter Reference model may have an impact upon the design and evaluation of current and future hypermedia systems.

CLARK ATLANTA UNIVERSITY

PASSWORD BASED SECURITY IN
HYPERMEDIA SYSTEMS

A THESIS SUBMITTED TO
THE FACULTY OF CLARK ATLANTA UNIVERSITY
IN CANDIDACY FOR THE DEGREE OF
MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BY

TRACEY W. WEST

ATLANTA, GEORGIA

MAY 1995

ACKNOWLEDGEMENTS

I would like to give thanks to God and to my family and friends for giving me the strength and encouragement to make it to this day. Thanks to my advisor, Dr. David Kerven, my committee, Dr. Erika Rogers and Dr. Radhakrishnan Srikanth and also to Dr. Kenneth Perry.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

GLOSSARY

Augment. One of the oldest operational hypertext systems originally known as NLS.

Vannevar Bush. The first person who describe the idea of hypertext.

Hypermedia. A database consisting of different types of information such as text, graphics, sound, speech, photographs and video.

Hypertext. Nonlinear organization of information.

Intermedia. A hypermedia system developed at Brown University.

Interoperability. The ability of a system to incorporate information objects that were not created in the original hypermedia environment.

Instantiation. A presentation of the component to the user.

KMS. The Knowledge Management System that is a hypertext system primarily designed for multi-user problem solving.

Links. Connection between one document and another document.

Memex System. The first hypertext system.

Navigation. The act of using hypertext to move around the database without getting lost or confused.

Neptune. A multilevel hypertext system developed by Neptune. Tektronix.

NoteCards. A hypertext system developed for Xerox LISP workstations.

UNIX. An operating system developed by Bell Laboratories.

Xanadu System. The hypertext system developed by Ted Nelson.

CHAPTER 1

INTRODUCTION

The advancement of computer technology and the rapid interest in on-line systems have sparked the recent burst of awareness in hypermedia systems. **What is a hypermedia environment?** A hypermedia environment is a collection of information objects and a set of associations among those objects. The information objects, nodes, contain data of a variety of media types: text, audio, video, graphics, and/or images. The associations, links, serve to connect related information objects. Together, these two sets are often referred to as a hyperbase. The final component of a hypermedia environment is a set of applications which provide access to the hyperbase.

A node consists of an information object and a representation of associations between this information object and other related nodes in the hyperbase (Kerven 1993), and it represents the fundamental unit of hypertext (Campbell 1988). Hypermedia provides the ability to interactively navigate among information objects through links. The relationships between associated objects represented as pointers, are often referred to as links or Hyperlinks (Martin 1990).

Hypermedia applications have many uses today. Current systems allow you to find everything from up-to-minute news and financial data to information about music, literature, free software packages, pets, medicine, and government issues (Bowen 1995). Many companies use hypermedia applications for training purposes: e.g., AT&T's Concept Presentation System (CPS) (Benimoff 1993), and schools use it for educational purposes: e.g., the Interactive NOVA hypertext system that allows students to browse large amounts of biology information relative to their current assignments in school (Nielsen 1993). This technology can also be used to provide efficient, interactive access to data such as The Crompton's Encyclopedia and Webster's Dictionary (Nielsen 1993). Some other uses of hypermedia systems have been for law practices and medical uses. Hypertext systems developed for commercial and experimental uses have been applied to a variety of domains: AUGMENT for computer supported collaborative work; Intermedia for Educational Hypermedia; and KMS and NoteCards for information management and organization.

1.1 Hypermedia Environment Security Problem

In current hypermedia systems, lack of security is an issue that is not often addressed. This lack of security hinders the user from storing private data on the system. Hypermedia systems provide the user with access to large amounts of information available at the click of a mouse.

With all this information easily accessible, some questions and concerns about protecting certain data arise. Various hypermedia systems have been designed with little or no security mechanisms in place, which can be an important aspect to users who are interested in protecting critical data. For network based systems, mechanisms to protect private data from being viewed by the common user must be available.

This research examines several hypermedia environment models: Dexter, HAM, and an Object-Oriented model, and will proceed to build upon the most widely accepted one, the Dexter model. A password based security system model will be developed as an extension of the Dexter model to address the lack of security problem. The design approach, is to develop a security mechanism based on hypertext links used to restrict access to private information. The nodes associated with these links will only be accessible to those users with valid user names and passwords. This research was inspired by the Dexter model's conspicuous lack of any type of security mechanism. The developed extended model represents a refinement of the Dexter model that may have an impact upon the design and implementation of future hypertext systems as well as the evaluation of current systems. In the model, private data is viewable by only authorized users. A prototype environment has been built to evaluate the model. The prototype environment was developed in the C programming language and implemented upon the world wide web environment.

1.2 Research Overview

The remainder of this research is outlined as follows: Chapter 2 describes the background research, the architecture of hypermedia systems, and the Dexter Reference model. After discussing the model, security methods and the issue of security in operating systems are introduced.

Chapter 3 covers the conceptual framework and results of the proposed model. The model for security is presented as an extension of the Dexter Reference model, and is followed by the results from this research.

Chapter 4 concludes this research and presents future work for the proposed model. Suggestions regarding future security developments in hypermedia systems are discussed.

CHAPTER 2

BACKGROUND RESEARCH

The hypertext concept was developed in 1945 when Vannevar Bush suggested the *memex system* (Bush 1945). In an article in the *Atlantic Monthly,* Bush described a nonlinear system that would be used for organizing scientific information in order to facilitate browsing and searching through data that was connected by links. Other systems built upon Bush's idea: NLS/AUGMENT along with the *Xanadu* system (Nelson 1990), were developed which addressed the problems of accessing, addressing and organizing large amounts of data. However, these and other hypertext systems were developed with restrictions on hyperbase placement and platforms.

Hypertext is a powerful way to retrieve information that could not be accessed as efficiently through the traditional database or information retrieval techniques. In (Nielsen 1990), Nielsen stated that:

> Hypertext is non-sequential writing: a directed graph, where each node contains some amount of text or other information. The nodes are connected by directed links. In most hypertext systems, a node may have several out-going links, each of which is then associated with some smaller part of the node called the anchor. When users activate an anchor, they follow the associated link to its destination node, thus navigating the hypertext network. Users backtrack by following the links they have used in navigation in the reverse direction. (Nielsen, 1990, 298).

Hypertext is represented as a collection of components related through a series of links, containing a source and destination. The components, as well as the network itself, are meant to be "visited" through the use of an interactive browser for an undetermined amount of time, in which the "visit" is terminated by the end of the application or by traversing the links to other components.

Hypermedia is one way of combining hypertext and multimedia together, by having each component of the hypertext be a self-contained component of the multimedia presentation (Hardman 1994).

2.1 Reference Models

As mentioned in chapter 1, a hypermedia environment is a collection of information objects, a set of associations among those objects, and a suite of applications to explore the data network. Several approaches have been used to define the general architecture for hypermedia environments: the Dexter model (Halasz 1989), the Neptune/HAM model (Campbell 1988), and an object-oriented model (Kerven 1993). The following figure shows an overview of the architectures of these three hypermedia reference models. In each model, the basic components of a hypermedia environment are defined but with different terminologies.

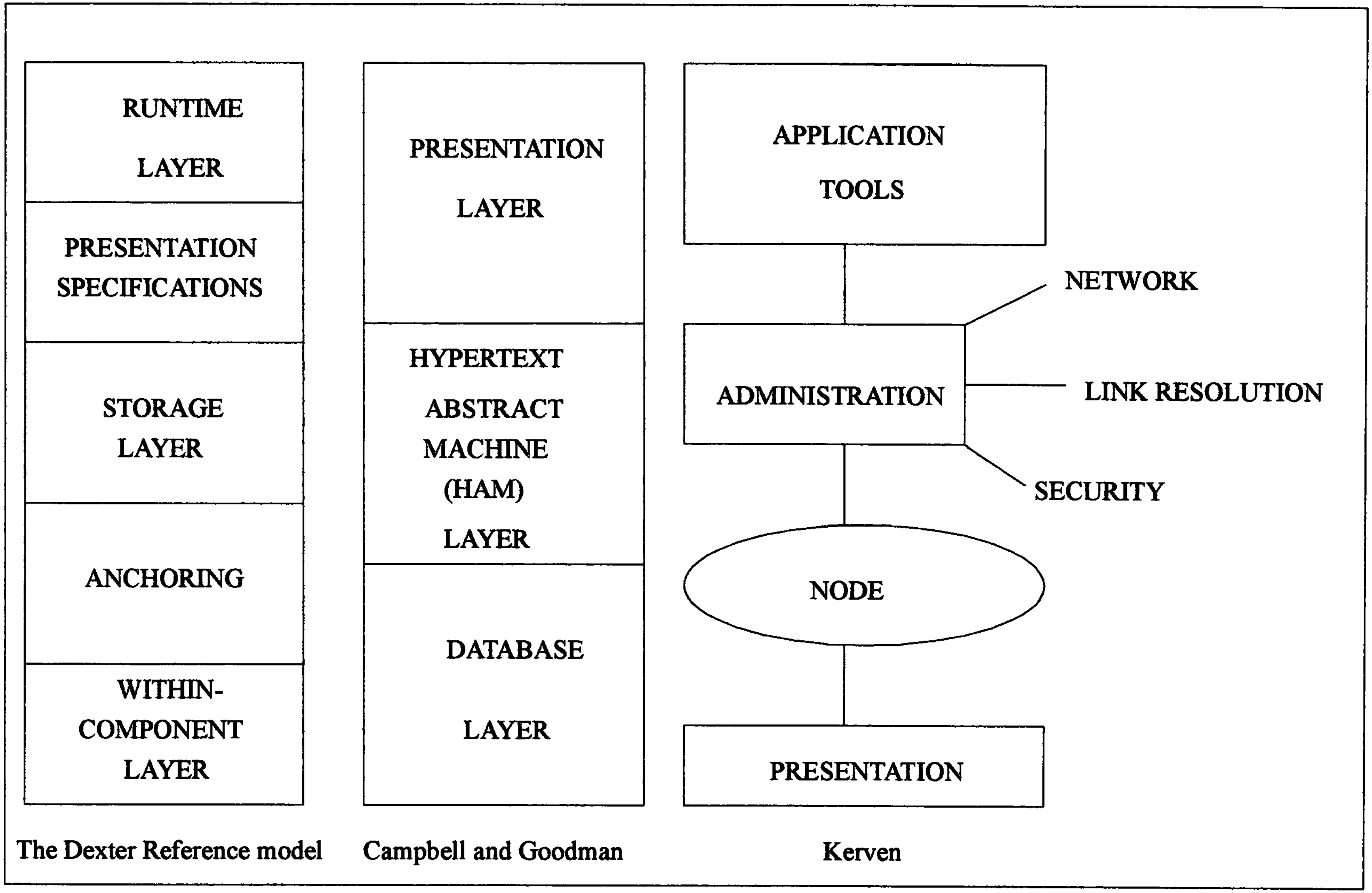

Figure 1. Hypermedia Environment Model Architectures

The Dexter Reference model (Halasz 1989) was designed as an attempt to have a model capable of representing the important abstractions found in a number of existing and future hypertext systems. The goal of the model is to provide a principled basis for comparing systems as well as for developing interchange and interoperability standards (Halasz 1989). This model came into existence from a series of workshops on hypertext organized by Jan Walker and John Legget. Extensions of this model to incorporate temporal information have been proposed in the Amsterdam model (Hardman 1994). The model consists of three layers: runtime layer, storage layer and the within-component layer. An overview of the model is given in section 2.2.

According to (Campbell 1988), the architecture of hypertext systems can be divided into three levels: Presentation Level - user interface, Hypertext Abstract Machine (HAM) Level - nodes and links, and the Database Level - storage, shared data and network access. No current hypertext systems follow this model exactly, but are composed of a mixture of these and other features using different terminology. In the HAM model, the lowest level of abstraction, the database level, contains the database of information, the basic node/link network structure, for the system. In the Dexter model (Halasz 1989), the database level is encompassed by the within-component layer and in (Kerven 1993) by the presentation level as well as a portion of the administrative level. In both (Kerven 1993) and HAM (Campbell 1988), in addition to the node/link structure, other issues such as security and multi-user access, are considered. In (Kerven 1993) these areas are addressed in the administration level. Data security is provided by an Access Control list mechanism, which is used to describe a set of users and their access privileges. In the present form of the Dexter model (Halasz 1989), **no security mechanism is included.**

The content of a node is specified in the within-component layer in (Halasz 1989), the database layer in (Campbell 1988) and the node layer in (Kerven 1993). In the Dexter model (Halasz 1989), anchoring, is a mechanism used for addressing locations or items within a node or "component". This mechanism is an interface between the storage layer and the within-component layer in model.

In order to communicate with the system, user interface applications (Campbell 1988) are used. This level of abstraction is referred to as the runtime layer in the Dexter model (Halasz 1989) and application tools in (Kerven 1993). At this level, applications such as browsing and editing are supported in (Kerven 1993).

In addition to the user interface application, the Dexter model contains an interface between the runtime layer and the storage layer referred to as presentation specifications (Halasz 1989).

Presentation specifications are mechanisms used for encoding information about a component to be presented to the user.

Generally, hypertext models use different terminologies but have the same functionality. In each system, the three basic hypermedia environment components are supported: the node, the links, and the access applications. These three components are crucial in the development of any hypermedia environment.

2.2 The Dexter Reference Model

The Dexter Reference model was designed as an attempt to capture both past and existing features found in a number of hypertext systems. The goal of the model is to provide a principled basis for comparing systems as well as for developing interchange and interoperability standards (Halasz 1989).

Often different terminologies are used to refer to the same information. Since the Dexter model represents a standard among hypertext systems, common terminology is to be used. The term "component" is used to refer to a unit of information to which a link is made. Components are the fundamental units of hypertext (Campbell 1988) which can contain graphics, text, animation, audio and/or images. Composite components are components created from the combination of other components.

The model is divided into three layers: the within-component layer, the storage layer and the runtime layer. The within-component layer is concerned with the structure and the contents stored within each node. This layer makes distinctions among media types. It is only highlighted in the model because the Dexter model represents a generic hypermedia system and this layer can store various data types. The storage layer, the main concern of the model, describes the structure of the nodes and links. At this level, components are generic and no distinction among media types is made. The runtime layer represents the interface between the user and the other layers of the model. In the Dexter model, presentation specifications allow components to be tailored in *a priori* manner. This mechanism is an interface between the runtime and storage layers which allows a component to have different ways of being presented when following different nodes to access the same data.

Navigating through links to access data in a hypermedia environment is possible through the anchoring mechanism. In the Dexter model, anchoring is used for addressing locations within a component. This mechanism is an interface between the storage and within-component layers.

Figure 2 provides a visual mapping of a component in the three layers of the model. The runtime layer is the user interface to the system and displays what the user sees while running the system. The storage layer, which is hidden from the user, contains the node/link information for the system. The within-component layer shows how the components are physically stored within the environment.

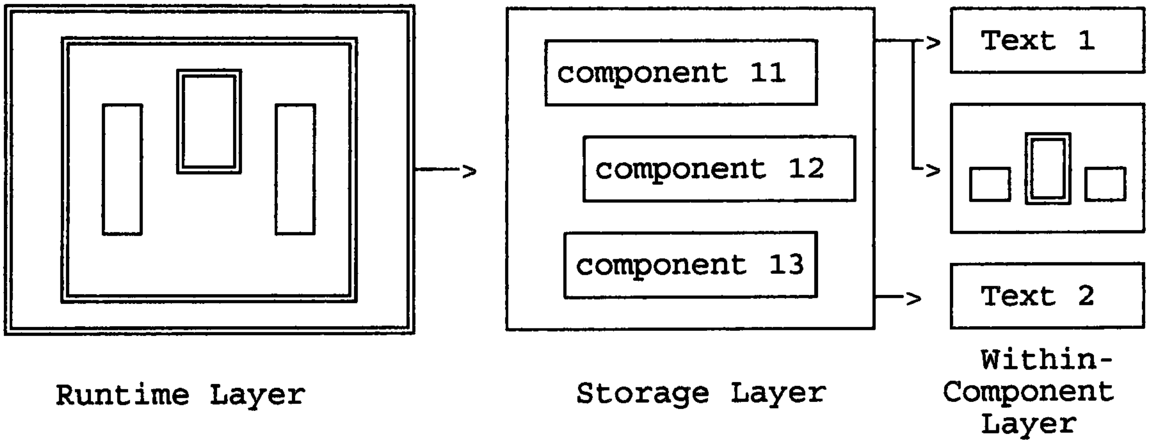

Figure 2. Layers of the Dexter model

2.3 Security Requirements

Several general principles have been identified by (Saltzer 1975) that can be used as a guide for designing a secure system, and these principles should be considered to protect against illegal entry:

- The security system should be made public but should not disclose information to the intruder about how the system works.
- Intruders can often gain access to systems through default measures, so the default on a system should have restricted access.

- To gain access into certain files, permission should be authorized. A file system should always have a strategy in place to check current authorization.
- The least privileges possible should be made for each process. If a user is not required to write (w) to a file, the access permission should not be granted.
- Protection mechanisms should be made in a simple, uniform way and at the lowest level of the system.
- Security should be done in a way that will not require much work from the user. It should not be a time consuming effort that takes away from the productivity of the user.

2.4 Security in Operating Systems

The most important fundamental of all the system programs is the operating system (Tanenbaum 1987). An operating system is a collection of programs that controls the operation of hardware and software and provides the base upon which the application programs can be written. The most visible part of any operating system is the file system (Tanenbaum 1987). In a file system, protecting information is very important to users of the system. It is essential for files to be protected from intruders who might edit, copy, or delete files from the system.

A strong security mechanism should be in place to prevent any unauthorized accesses.

Security issues can be divided into two different areas: loss of data and intruders. There are many causes of data loss which can be handled through proper backup maintenance. When dealing with the issue of handling unauthorized users, it can be of two types: passive intruders and active intruders. A passive intruder does not cause any harm to the system. Their major concern is to view *secure* data. An active intruder is one who wants to cause harm to the system and whose goal is to destroy or alter files.

Besides dealing with unauthorized access, mechanisms are required for specifying authorized access. Access privileges should be implemented in a way that is transparent to the users. The most widely used form of authentication is requiring a user to type in a password. A password's physical characteristics include its size and its makeup (the alphabets) (Wood 1977). Access to UNIX systems, requires a user name and a password. Once the two have been entered correctly, the user gains access to appropriate files. The UNIX system contains a password file that is used to store information such as the user name and the encrypted password for each user on the system. This file can be viewed by anyone but can only be changed or altered by privileged users and selected programs.

UNIX file security deals with who can access a file and what privileges they have over that file. Associated with each file are the categories of owner, group and other. The mode or permission of a file is in the form of three patterns of *rwx*; the file permission determines who can read (r), write (w) and execute (x) a file. The first pattern gives the permission for the owner. The second pattern gives the permission for the group, and the last pattern is for any other user.

Files can be encrypted by using the *crypt* command. The *crypt* command uses a key which scrambles the standard input into an unreadable code which is sent to standard output. Encrypted files can be decrypted by using the same *crypt* command with the appropriate key. File encryption is a method of hiding information from other users, intruders as well as the system administrator. With time and patience, encrypted files can be converted into standard text; however, this process is a non-trivial task.

CHAPTER 3

CONCEPTUAL FRAMEWORK AND METHODOLOGY

In this chapter, the framework of the proposed model is described. This thesis research extends the Dexter Reference model to include a password based security model. The proposed security component will be incorporated into the Dexter model to guard against passive intruders. The Dexter model was chosen for this research for several reasons:

- It was designed as a comparative model.
- It was designed for developing interchange and interoperability standards.
- It was the most widely accepted model.

Some users will have privileges that will not be granted to other users. The security mechanism will be based on hypertext links that will restrict access to private information. The nodes associated with these links will only be accessible to those users with valid user names and passwords. The extensions will be placed primarily in the Dexter model's storage layer. The following section gives a description of the proposed model. The model has been evaluated through the design, development, and testing of a prototype. In the conclusion of this chapter, the results of designing and testing the developed prototype are given.

3.1 An Overview of the Storage Layer

The driving force behind the Dexter model is the storage layer, which describes the hypertext's structure. The primary entities in the storage layer are components: atomic, composite, and links. Components are connected by links. Composite components are the result of combining other components. The structure of composite components is in the form of a direct acyclic graph (DAG). The NLS/AUGMENT system is one of the few extant hypermedia environment which supports document compositions; however, these composites are restricted to hierarchical organizations.

The storage layer supports two functions used to retrieve components: the accessor and the resolver. A unique identifier (UID) is associated with each component. If given a UID, the accessor function maps the UID to the particular component. Many times components are not addressable in this manner due to components that have been altered or deleted over time. In order for these components to be accessed, component specifications and the resolver function are used.

The resolver function is used to map a component specification into a UID, which in turn allows the accessor function to retrieve the component. The resolver is only a partial function since some component specification may not lead to existing components. In the case where the UID is the component specification the resolver function acts as the identity function.

To achieve span-to-span links, the Dexter model uses an anchor, as an indirect addressing entity. An anchor is designed with two parts: an anchor id and an anchor value. The purpose of these two anchor attributes is to serve as the interface between the storage layer and within-component layer. The anchor id is the unique identifier that describes the anchor within its component's scope. The anchor value represents a spatial location within the context of the particular component.

A component specification, an anchor id, a direction, and a presentation specification, is referred to as a specifier. In the model, the anchor ID along with the component specification mechanism can be combined to designate the endpoints of a link. The direction specifies whether the endpoint is to be considered as source, destination, both or neither; the model uses codes: FROM, TO, BIDIRECT, and NONE, respectively.

The Dexter model is formalized using the Z Notation, a language based on set theory. This formalism is used to define the necessary abstractions and their uses in the model and will, therefore, be used to define the model extensions.

The following types described in Z notation are the classes used to modify the model to include the password security extension. This formalization is used in both the storage and runtime layers.

The primary entities of the storage layer are a recursive type BASE_COMPONENT which contains atoms, links composite components and passwords. The following formally describes the extended BASE_COMPONENT in the extended model.

```
BASE_COMPONENT::=  atom<<ATOM>>
                 | link<<LINK>>
                 | composite<<seq BASE_COMPONENT>>
                 | password<<seq PASSWORD_ENTRY>>
```

An atom is modeled as an individual media element of a hypertext system and defined by the type ATOM.

[ATOM]

Links are defined as mechanisms used to associate information objects. The following figure shows the link schema.

```
==LINK===========================================================
        specifiers : seq SPECIFIER
-----------------------------------------------------------------
        # specifiers >= 2
        Is : ran specifiers ● s.direction = TO v BIDIRECT
=================================================================
```

Figure 3. Z Notation Specification of a Link

A link is a sequence of specifiers, which are identified as containing a presentation specification, a component specification, an anchor id and a direction. The following figure illustrates how a specifier is defined.

SPECIFIER
componentSpec: COMPONENT_SPEC
anchorSpec: ANCHOR_ID
presentSpec: PRESENT_SPEC
direction: DIRECTION

Figure 4. Z Notation Specification of a Specifier

In the storage layer, the specifier identifies a component specifications which comes from a given set COMPONENT_SPEC. Anchors are composed of two parts: anchor id and an anchor value. An anchor id is identified by a unique id and comes from the given set ANCHOR_ID. Anchor values, identified as variable fields, comes from the given set ANCHOR_VALUE. They are formalized in the model as:

ANCHOR
anchor_id : ANCHOR_ID
anchor_val : ANCHOR_VAL

Figure 5. Z Notation Specification of an Anchor

The presentation specifications represent the visual display of a component, and come from the set PRESENT_SPEC. Direction was introduced to describe the end point of a link as a source, destination, both or neither. It is described in the model as the enumeration:

DIRECTION:= FROM | TO | BYDIRECT | NONE

A composite is defined as a recursive sequence of base components. In addition to the sequence of atoms, links and composite components, the proposed model includes a password_entry field. The password field is defined as:

PASSWORD	
logicaldocumentSpec:	DOCUMENT_SPEC
usernameSpec:	USER_NAME
passwordSpec:	ENCRYPT_PASSWORD

Figure 6. Password Entry Type

In the proposed model, logical document specifications come from the given set DOCUMENT_SPEC. This is a given set of documents that can only be accessed by authorized users. User name specifications are identified as the login names for the users and come from the given set USER_NAME. Password specifications are the passwords for the users and come from a given set ENCRYPT_PASSWORD.

A password entry is a sequence of password entries. Figure 7 illustrates the password file.

```
PASSWORD_ENTRY
    password_entry : seq PASSWORD_ENTRYS
    ------------------------------------
    # password_entry >= 1
```

Figure 7. Password File

In the model, predicates are introduced to show whether a component is an atom, link or composite. The following shows the predicates for the extended model.

```
PREDICATE
  isAtom_ : P COMPONENT
  ---------------------
  isLink_: P COMPONENT
  ---------------------
  isComposite_: P COMPONENT
  ---------------------
  isPassword_: P COMPONENT
  ---------------------
   Vc : COMPONENT ●
       isAtom c ↔ base(c) ε̂ ran atom ^
       isLink c ↔ base(c) ε̂ ran link ^
       isComposite c ↔ base(c) ε̂ ran composite ^
       isPassword c ↔ base(c) ε̂ ran password
```

Figure 8. Predicate Schema

A "type" consistency is defined for components to be "type consistent" if they are both of the same type. In the proposed model, the following figure illustrates type consistency.

```
=PREDICATE_CONSISTENCY==========================================

     _typeConsistent_: COMPONENT ↔ COMPONENT
───────────────────────────
     V c1, c2 : COMPONENT ●
           C1 typeConsistent c2 ↔
               (isAtom c1 ^ isAtom c2) v
               (isLink c1 ^ isLink c2) v
               (isComposite c1 ^ isComposite c2) v
               (isPassword c1 ^ isPassword  c2)
================================================================
```

Figure 9. Type Consistent Components

These are the necessary extensions needed to modify the storage layer in the Dexter model. The following section gives an overview of the runtime layer and the formal specifications needed to incorporate the password based security mechanism into the runtime layer.

3.2 An Overview of the Runtime Layer

The fundamental concept in the runtime layer is the instantiation of a component, or the presentation of the component to the user (Halasz 1989). Once a hypertext has been accessed by the user, a copy of that component is returned to the user's view. If changes are made, the revised components may be returned to the storage layer. Each presentation of a component is given a unique instantiation identifier (IID); this allows multiple simultaneous instantiations for any component.

The instantiation affects both components and anchors. In the model, a link marker is a presentation of an anchor. In order for the link marker to be accessed properly, the instantiated entities contain a base instantiation, a sequence of link markers, and a function used to map link markers to the anchors they instantiate.

In the runtime layer, a session entity tracks mapping between components and their instantiations. When a user accesses a hypertext, a session is opened. This allows the user to edit the component, modify the component (realizing edits), and destroy the component (unpresenting the component). The entire presentation of the component is removed whenever the user deletes a component through one of its instantiations. The session only remains open until the user has completed interacting with the hypertext.

The session entity contains: the hypertext that is being accessed, a mapping from the IID's of current instantiations to their components in the hypertext, a resolver function, a history of actions, a realizer function and an instantiation function. In addition to these entities, the proposed model will include a user name and password field.

The purpose of the resolvers in the runtime and storage layers are the same: they attempt to map a specification to a UID. The resolver in the runtime layer can also use additional information concerning the current session such as the session history; this type of information is not accessible to the storage layer resolver function.

One of the most important functions in the runtime layer is the session's instantiator function. This function's input consists of a component (UID) and a presentation specification. In return, the instantiator returns an instantiation of the component as a part of the current session. The presentation specification is primitive in the model; it specifies how the instantiated component is to be presented to the user. The instantiator function determines how discrepancies between the presentation specification passed to the instantiator and the presentation specification attached to the component are resolved. Currently, this determination is not explicitly specified in the model.

In the model, in the runtime layer, an instantiation is represented as:

[BASE_INSTANTIATION, LINK_MARKER]

The following figure defines the contents of an instantiated component.

```
==INSTANTIATION=====================================
        base: BASE_INSTANTIATION
----------------------------------------------------
        links: seq LINK_MARKER
----------------------------------------------------
        linkAnchor: LINK_MARKER → ANCHOR_ID
----------------------------------------------------
        dom linkAnchor = ran links
====================================================
```

Figure 10. Instantiation of a Component

During a session, the following operations can be performed on a hypertext:

```
OPERATION:= OPEN | CLOSE | PRESENT | UNPRESENT |
            CREATE | EDIT | SAVE | DELETE
```

A session, in the extended model, is represented in the following figure.

```
==SESSION==================================================
  H : HYPERTEXT
-----------------------------------------------------------
  history: seq OPERATION
-----------------------------------------------------------
  instants : IID ↠ (INSTANTIATION x UID)
-----------------------------------------------------------
  instantiator: UID x PRESENT_SPEC → INSTANTIATION
-----------------------------------------------------------
  realizer : INSTANTIATION → COMPONENT
-----------------------------------------------------------
  runTimeResolver : COMPONENT_SPEC ↠ UID
-----------------------------------------------------------
  usernameSpec : USER_NAME
-----------------------------------------------------------
  passwordSpec : ENCRYPT_PASSWORD
-----------------------------------------------------------
  head(history) = OPEN
-----------------------------------------------------------
  V uid : UID; ps : PRESENT_SPEC |
  uid ∈ dom H.accessor
  realizer(instantiator(uid,ps)) = H.accessor(uid) ^
  H.resolver ≤ runtimeResolver
===========================================================
```

Figure 11. Contents of a Session

The runtime resolver is responsible for mapping a component specification into a UID. In addition to the information that the runtime resolver can access in a session: information on the session and the history of actions, it will also have access to the user names and passwords. Once a session is open for a user to access a hypertext, the runtime layer is responsible for supplying the user name and password. This information is available to the resolver function which maps the component specifications (usernameSpec and passwordSpec) into a UID. The user name and password are compared to the password entity containing valid user names and passwords. If a match is found, the accessor function returns the UID associated with the component requested by the user. If no match is found, the accessor function returns the UID of the component which denies the user access to the component. Figure 12 shows the mapping specifications for the proposed model.

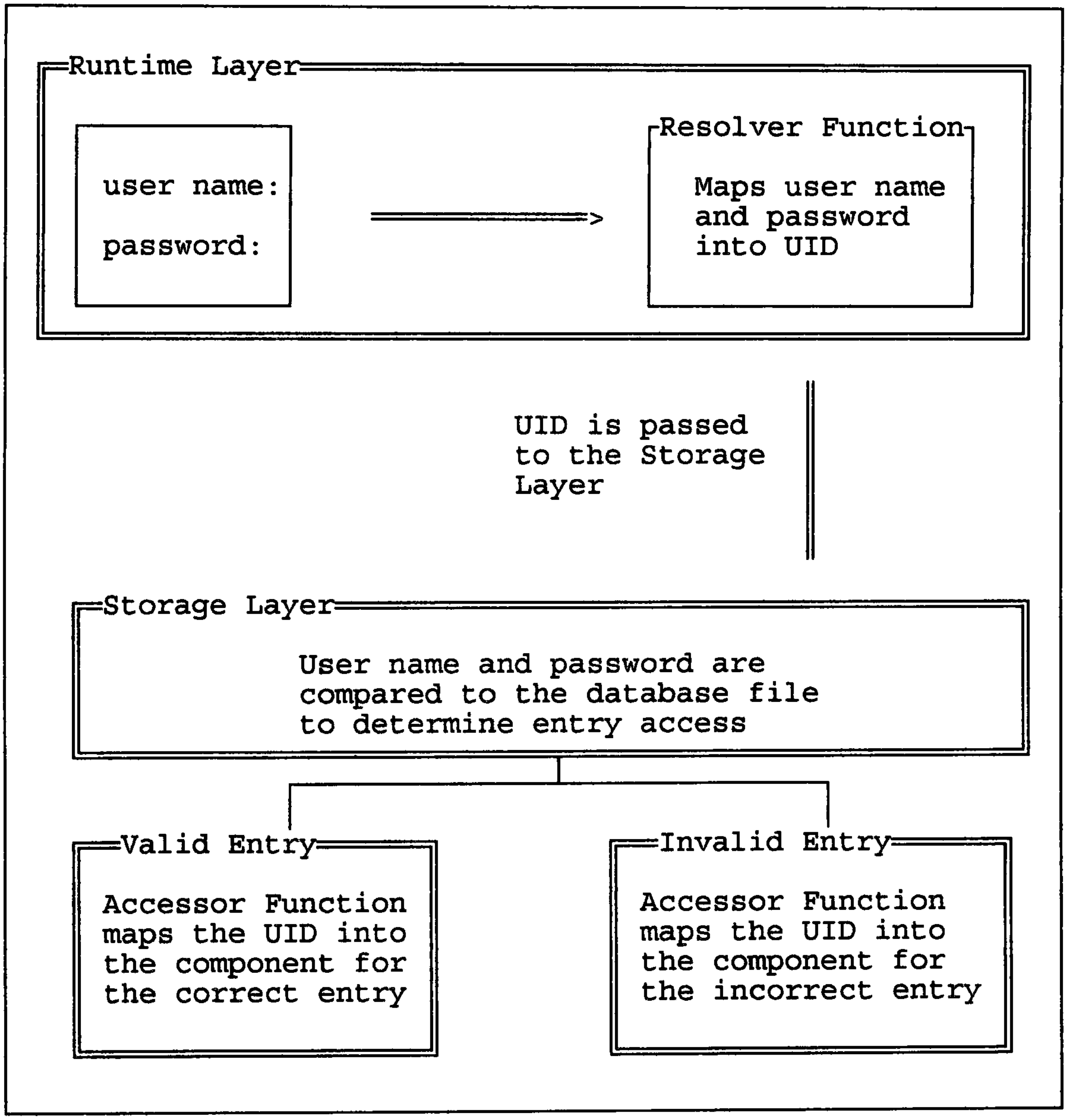
Runtime Layer
user name:
password:
Resolver Function
Maps user name and password into UID
UID is passed to the Storage Layer
Storage Layer
User name and password are compared to the database file to determine entry access
Valid Entry
Accessor Function maps the UID into the component for the correct entry
Invalid Entry
Accessor Function maps the UID into the component for the incorrect entry

Figure 12. Security Implementation

3.3 Testing and Evaluations

The model of a prototype was evaluated through the development of a prototype in the world wide web environment. In the model, the security page was generated when the user clicked on the hypertext symbol representing a link to a private information object. Once the hypertext was accessed, a page was generated for the user to enter a user name and password. The user name and password were sent to the server where they were compared against a database containing the list of authorized users. If a correct entry was entered, the requested page was presented. If an incorrect response was entered, a document was generated requesting the user to try again. This mechanism represents the prototype's ability to supply the user name and password in the model's session entity.

In the model, the user name and password are incorporated into the runtime layer and passed to the storage layer; however, for testing purposes in the world wide web environment the user name and encrypted password are incorporated during runtime and passed to the storage layer where encryption is performed.

In the prototype model, observational testing was done to evaluate the model. The test were generated in the world wide web environment. A survey was given to fifteen users containing a file of user names and passwords. The users were asked to enter selections from the file and record the results.

The results from the proposed model demonstrated that a password based security mechanism can be implemented into an existing hypermedia system. The prototype developed as an extension of the Dexter Reference model, shows in the simplest form how a security mechanism can be incorporated into a system. With this mechanism in place, private information can be stored on hypermedia systems.

In evaluating this approach there were some strengths and weaknesses. The strengths were: no user was able to gain access to the private information on the system with an incorrect user name and password; however, those users with valid entries were able to gain access.

A weakness of this approach was the limitation of having a user name and password determine the access privileges for the information. This was a very short test because either the entries were correct, which provided access to the requested information or incorrect, which terminated the request.

The implementation for the prototype model used to generate the password security mechanism is given in Appendix B. The sample of the test data file is also given.

CHAPTER 4

CONCLUSION AND FUTURE WORK

In this chapter the summary and future work are presented. In conclusion, key points that are essential for the future developments for hypermedia systems are given.

4.1 Summary of Research

This research addressed the lack of password security in current hypermedia models. Various hypermedia systems are designed with little or no security mechanisms. This is a concern to users interested in protecting crucial data from the common user. Mechanisms used to protect private data from being viewed by passive intruders should be provided for network based systems.

A password based security mechanism was developed as an extension to the Dexter Reference model to address this problem. After comparing and contrasting among several hypertext models, the Dexter model was chosen because of its advantages. The Dexter model was designed for comparing and contrasting the interchange and interoperability standards found in various hypermedia systems, and is also the most widely accepted hypertext model.

The research approach was to develop a security mechanism based on hypertext links. A password entity containing valid user names and passwords, was associated with the hypertext links which provided access to the private data. A prototype environment was developed for testing and evaluating the results. The following gives a brief summary of this research.

Chapter 1 provided an introduction to the concept of hypertext and the various uses of hypermedia applications. This is followed by the problem statement, design approach and motivation for this research.

Chapter 2 contained relevant hypermedia models which lead into a discussion of the Dexter Reference model. Security in operating system and security methods were presented as the basis for the proposed security model.

Chapter 3 described the development of the proposed model. A description of the layers in the model were discussed followed by the model prototype. The results evaluating this research were given.

4.2 Future Work

In conclusion, this research examined the minimum security features for a hypermedia system and answered the question regarding lack of security in hypermedia systems. The prototype developed addressed a password based approach to solving the problem.

There are a few suggestions for future work for this research that can impact upon the future design of hypermedia systems. These suggestions are outlined as follows:

1. Incorporation of additional features such as performing security on a global access. In the proposed model, the password based mechanism only supports local access to information. The security model is placed on nodes associated with links to private information.

2. Modify the runtime layer of the prototype to avoid the unencrypted passwords to be transmitted across the network.

3. Design additional security features to guard against the active intruder. The prototype model is designed to guard against the passive intruder only.

In addition to the model prototype, these suggestions can be made to alleviate the lack of security problem in the development of future hypermedia systems.

APPENDIX A

TRADEMARKS USED

Interactive NOVA is a trademark of Apple Computers, Inc. WGBH Educational Foundation, and Peace River Films, Inc.

UNIX is a trademark of AT&T Bell Laboratories.

Xanadu is a trademark of the Xanadu Operating Company.

APPENDIX B

PROTOTYPE ENVIRONMENT

The link/node security mechanism will be integrated into the storage layer in the Dexter model (Halasz 1989). The user name and password will be the mechanism for the security check. Once the user has entered the required information, the user name and password are compared against the database containing the valid user names and passwords. If a match is detected, permission is granted to the user and the requested information is presented. If there is no match, then the unauthorized user is asked to backtrack to the previous node.

User names and passwords which are not present in the data file, can only access the public information present on the system. When the user name and password are entered correctly, the authorized user is presented with the requested information. This process continues until the user exits the system.

The following programs generate the password security mechanism.

```
/* HTML Document which generates a Home Page */

<html>
<head>
<TITLE>Tracey W. West</TITLE>
<link rev="made" href="mailto:twest@diamond.cau.auc.edu">
</head>
<body>
<hr>
<center>
<h1>Tracey W. West</h1>
</center>
Tracey W. West, Graduate Student
<br>
Department of Computer and Information Sciences
<br>
Clark Atlanta University
<br>
240 James P. Brawley Drive at Fair Street
<br>
Atlanta, Georgia 30314
<br>
email:  twest@diamond.cau.auc.edu
<hr>
<h2>School</h2>
<hr><address><a href=http://www.cau.auc.edu>Clark Atlanta
University</a></address></hr>

<h2>Advisors</h2><hr>
<menu>
<li><a href=http://jasper.cau.auc.edu/Faculty/dsk/dave.htm>Dr.
David Kerven</a>
<li><a href=http://jasper.cau.auc.edu/Faculty/srikanth.htm>Dr.
Radhakrishnan Srikanth</a>
<li>Dr. Erika Rogers
<HR>
We are working on setting up password based security for
hypermedia
<a href="/tmp_mnt/home/twest/security/secure">documents</a>.
</body>
</html>
```

```
/*  Password Access file where user names and passwords are
     checked for validation of private data files.
     This model is still in the development stages
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "review.h"

main()
{
  char password[30], name[30];
  char *e;
  char salt[] = { 'o', 'p'};
  int pw, doc, user_name;
  int len, i, cap, access;
  FILE *ptr;
  Pair inputs[500];


  printf("Content-type: text/html%c%c",10,10);

  printf("<html>\n<head>\n<title>Security Access Page
   Test</title>\n");
  printf("<link
   rev=\"made\"href=\"mailto:twest@diamond.cau.auc.edu\">\n");
  printf("</head>\n<body>\n");

  if(strcmp(getenv("REQUEST_METHOD"),"POST")) {
    printf("This script should be referenced with a METHOD of
     POST.\n");
    exit(1);
  }

if(strcmp(getenv("CONTENT_TYPE"),"application/x-www-form-url
encoded")) {
    printf("This script can only be used to decode form
 results.\n");
    exit(1);
  }
  len = atoi (getenv ("CONTENT_LENGTH"));

  for (i = 0; len && (!feof(stdin)); i++)
  {
    inputs[i].attrib = build (256,'=',&len);
    inputs[i].value = build (1024,'&',&len);
    if (!strcmp (inputs[i].attrib, "password"))
      pw=i;
    if (!strcmp (inputs[i].attrib, "document"))
      doc=i;
/* page 2  - Password Access File*/

    if (!strcmp (inputs[i].attrib, "name"))
      user_name=i;
  }
  cap=i;

  e = (char *) crypt(inputs[pw].value,salt);

  if ((ptr = fopen ("/usr/local/httpd_exp/cgi-bin/data1",
    "r")) == NULL)
  {
    printf("Error opening password file.\n");
    printf("Contact security administrator.\n");
    printf("</body></html>");
  }

  access = 0;

  while (fscanf (ptr, "%s%s", name, password)!= EOF)
  {
    if (strcmp (name, inputs[user_name].value) == 0)
      if (strcmp (password, e) == 0)
      {
        access = 1;
     printf ("Click <a href> here</a> to view the requested
            page.</pre></hr></h1>\n");
      }
   }
   if (access != 1)
    {
     printf ("<pre>\n<h2><hr>LOGIN INCORRECT, TRY AGAIN.</h2>
            \n\n");
     printf ("<h2><hr>Click <a
            href=\"http://jasper.cau.auc.edu:8080/access\"
         enctype=\"application/x-www-form-urlencoded\">
         here</a> to return to secure page.
         </pre></hr></h2>\n");
     rewind (ptr);
     }

  fclose (ptr);
}
```

```
/* Security Access Page where the user names and passwords are
    entered
/*


#include <stdio.h>

main()
{
  char *document;

  document=(char *)getenv("PATH_INFO");

  printf("Content-type: text/html%c%c",10,10);
  printf("<html>\n<head>\n<title>Security Access
Page</title>\n");
    printf("<link rev=\"made\"
href=\"mailto:twest@diamond.cau.auc.edu\">\n");
  printf("</head>\n<body>\n");

  if((!document) || (!document[0])){
    printf("Error finding document.\n");
    printf("</body>\n</html>\n");
  }

  printf("<hr>\n<h1>Access Page for %s</h1>",document+1);
   printf("<hr>\n<form method=\"POST\"
action=\"http://jasper.cau.auc.edu:8080/cgi-bin/access\"
enctype=\"application/x-www-form-urlencoded\">\n");
  printf("<input type=\"hidden\" name=\"document\"
value=\"%s\">\n",
      document+1);

  printf ("Please enter you user name and password.\n");
  printf ("When you are finished, click submit.\n<p>\n");

  printf("User Name :<input type=\"text\" size=25
name=\"name\">\n");
  printf("<br>\n Password:<input type=\"password\"size=20
name=\"password\">\n");
  printf("<br>\n<input type=\"reset\" value=\"Clear
Entries\">\n");
  printf("<input type=\"submit\" value=\"Access
Document\">\n");
  printf("</form>\n");
  printf("<hr>\n<h6>Page Generated Automatically, contact
Tracey W. West ");
  printf("(twest@diamond.cau.auc.edu) with any
questions.</h6>");
  printf("</body>\n</html>\n");
}
```

```
/* Password Data file containing the secure document, valid
      user names and encrypted passwords.
   This is the sample test data file.
/*


first_secure twest opakS9olxszH.
first_secure bjones opZWC1k03tc5M
first_secure cmcclain opboM78FNnQm6
first_secure tbeavers ope/YqU9OoudM
first_secure ssmith opXlvYK2YIVTg
first_secure tluv opQnFqTv5cx32
first_secure dsk opZzSTLpgVyzw
first_secure erogers opjs/bBz5Hkv.
first_secure srikanth mac-ixz
```

```
/* Mapfile entry containing the secure document and path. */


first_secure /usr/local/httpd_exp/cgi-bin/first_secure
```

REFERENCE LIST

Benimoff, N., and M. Burns. 1993. Multimedia User Interfaces for Telecommunications Products and Services. At&T Technical Journal, (May/June): 42-9.

Bowen, C. 1993. Open Sesame!. Home PC Magazine 2(3) (March): 117-24.

Bush, V. As We May Think. 1945. Atlantic Monthly 7 (July): 101-8.

Campbell, B., and J. Goodman. Ham: A General Purpose Hypertext Abstract Machine. Communications of the ACM 31(7) (July): 856-61.

Gray, S. H. 1993. Hypertext and the Technology of Conversation Orderly Situational Choice. Greenwood Press.

Hardman L., D. Bulterman, and G. Rossum. 1994. The Amsterdam Hypermedia Model. Communications of the ACM (February): 50-62.

Halasz, F., and M. Schwartz. 1987. The Dexter Hypertext Reference Model. Paper submitted to the NIST Hypertext Standardization Workshop. NIST, 500-178.

Kahn, Paul and Normal Meyrowitz. 1988. Guide, HyperCard and Intermedia: A Comparison of Hypertext/Hypermedia Systems. IRIS Technical Report (May): 99-107.

Kerven, David. 1993. An Abstract Architecture for Distributed, Object-Oriented Hypermedia Systems. Ph.D. diss., The University of Southwestern Louisiana.

Kochan, S., and P. Wood. 1984. Exploring the UNIX System. Hayden Books.

Martin, J. 1990. Hyperdocuments & How to Create Them. Englewood Cliffs, NJ: Prentice Hall.

Nelson, T. H. 1990. Literary Machines. Mindfull Press.

Nielsen, J. 1990. The Art of Navigating Through Hypertext. Communications of the ACM 33(3) (March): 296-310.

Nielsen, J. 1993. Hypertext & Hypermedia. Harcourt Brace & Company, Publishers.

Shneiderman, B. 1989. Hypertext Hands-On!. Reading, Mass.: Addison-Wesley Publishing Co.

Tanenbaum, A. S. 1987. Operating Systems: Design and Implementation. Prentice Hall Publishers.

Wood, H. M. 1977. The Uses of Passwords for Controlled Access to Computer Resources. National Bureau of Standards (NBS Publication).