ELECTRONIC PUBLISHING, VOL. 6(2), 93–113 (JUNE 1993)

Canto: a hypertext data model

CHARLES K. NICHOLAS
Computer Science Department
University of Maryland Baltimore County
Catonsville, Maryland 21128, USA

LINDA H. ROSENBERG[1]
UNISYS Government Services
Lanham, Maryland 20706, USA

SUMMARY

The Canto hypertext data model is characterized by a hierarchical schema mechanism that allows a predetermined, open-ended schema to be embedded in the hyperdocument. Canto uses two types of nodes: concept nodes, which provide organizational structure, and information nodes, which contain text and other data. The operations provided by the Canto Data Model are defined formally using the Z specification language. The Canto Schema Language gives the hypertext designer access to these operations. We show that the operations defined in Canto preserve properties of accessibility and data integrity. We evaluated Canto with a controlled experiment involving human users. These tests showed that a hypertext system developed with Canto was easier to use than an otherwise similar system that did not employ a schema mechanism. Several applications have been developed using Canto. One such application, which we describe in some detail, involves the use of Canto to teach students the skill of program reading.

KEY WORDS Hypertext data model Hypertext schema Hypertext testing Z specification

INTRODUCTION

Hypertext is a methodology for reading or writing in a non-linear environment. In general, all hypertext systems have two primary components, nodes and links. Nodes are receptacles for information in the form of text, video, etc. Links represent connections between nodes, creating paths the user travels to access the desired information. Various commercial hypertext systems are in use, such as Guide, KMS, and HyperCard [1].

Four problems plague existing hypertext systems:

- users can feel disoriented, or "lost in space", in the sense of not knowing where they are in the network, or how to get from where they are to some other place [1].
- users may suffer from information myopia, in which the large amount of available data makes it difficult to decide which information is important [2].
- it is often difficult to customize hypertext systems to meet the needs of specific applications [3].
- The non-linear representation of hypertext information can present problems for the system designer and user. When designing a hypertext system, for example, one such difficulty comes from the fact that it is unnatural to break thoughts into discrete units [2].

[1] The work described in this article was conducted while this author was affiliated with Goucher College, Towson, Maryland 21204, USA.

CCC 0894–3982/93/020093–21

Received 20 August 1991
Revised 10 December 1992

A data model for hypertext could serve as the basis for making the overall structure of the document clear to the user. Without a data model, the mental effort required to separate information into discrete thoughts, to identify different types of information, and to create and label links between nodes can be prohibitive. The mental structure of the information in the user's head may be very different from the node and link structure provided by the system. Frequently the two primitive constructs of nodes and links are insufficient: this simplest hypertext model lacks an aggregation mechanism that would allow designers to treat groups of nodes and links as unique entities [3]. The hypertext structure also causes problems for the reader, who must try to locate the needed information, and then relate that information to other facts in the network without the aid of traditional structural clues, such as sentences and paragraphs [4].

The problems of hypertext are inter-related. When users have to take time to think about path selection, they may become disoriented. Therefore, decreasing the number of paths or the amount of material a user must read to determine the correct path will decrease the user's disorientation. This also will decrease the amount of information myopia the user experiences, since there is less information to overwhelm the user. People often get lost in an unfamiliar environment. In hypertext, this is partially due to the use of a non-linear structure, which usually does not contain the "clues" for reader orientation, such as paragraphs, pages, and chapters, that a linear system has [2].

The Canto hypertext data model was designed to serve as a basis for a hypertext system into which data can easily be placed, located or removed without the user becoming disoriented. The key idea in Canto is an integrated hypertext schema. The schema is a description of the data, and a specification of the attributes and the relationships among the data types. The schema fits the information in the hypertext into a specific framework [5]. The schema may be external, as in some database management systems, or a separate part of the hypertext, as in relational DBMS, or embedded in the data. We contend that schema design should be an integral part of the entire hypertext system development, and that the schema should be embedded within the data. The hypertext schema constrains the structure in a positive way, as a context-free grammar (in a document type definition) constrains an SGML document [6]. The hypertext schema and the data itself should be considered part of the document as a whole, just as the document type definition and an instance of that document type definition make up a complete SGML document. Although some hypertext designers recognize the necessity of some type of structure other than one implied by the data and express the need for a schema mechanism, there currently is no data model for expressing such a mechanism [3,7].

There are two types of nodes in Canto: concept nodes to contain the ideas and sub-ideas and provide organizational structure, and information nodes to contain the data. In Canto's hierarchical structure, users access information much like they would in an outline, progressing through the main ideas until the desired concept is found. This is a familiar process to most people and translates well into a hypertext system. The hierarchical structure is maintained through a system of implicit links between concept nodes.

In this paper we present the Canto data model, and describe how Canto reduces user disorientation. The use of this schema minimizes the user's feeling of being lost within the data [8]. We first describe the important features of the Canto hypertext data model. In the sections that follow, we present a formal specification of Canto's functionality, and we discuss the schema definition language that a hypertext system designer uses to establish a hypertext schema. We then present the schema manipulation language, which allows for

alterations to the schema and data. We then describe the testing we conducted to determine the usability of hypertext systems built using Canto. We conclude with a discussion of related work.

THE CANTO HYPERTEXT DATA MODEL

In Canto, we say that a hypertext application consists of three entities

$$HT =< N_c, N_i, L >$$

in which N_c is a set of concept nodes representing the structure, N_i is a set of information nodes containing information for that application and L is a set of implicit and explicit links.

Concept nodes contain a key phrase that identifies the information stored in the corresponding portion of the structure. A subtree of related ideas can be aggregated under a single concept node. Concept nodes also may contain explicit links to other nodes. Data is contained in linear linked lists of information nodes, where each list is owned by a specific concept node. Information nodes hold the data in an application specific format. A concept node may or may not have information nodes associated with it. Information nodes associated with concept nodes are expected to contain data relating to the topic implied by the concept node's key phrase.

In Canto, links may be either implicit or explicit. Implicit links are derived from the structure and order of the nodes. Implicit links connect concept nodes to other concept nodes, or to information nodes. Explicit links may connect any two nodes, and are used to represent non-hierarchical relationships [8]. The general form of a Canto hyperdocument is shown in Figure 1. A specific example is shown in Figure 4.

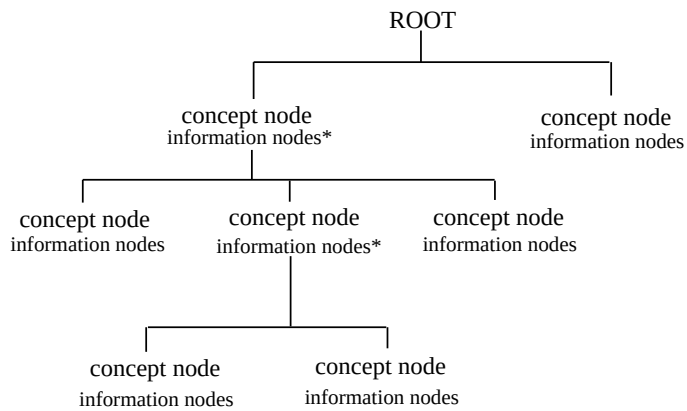

Figure 1. The general form of a Canto hyperdocument. The asterisks represent optional nodes

Concept nodes can be embedded in a Canto hyperdocument in two ways:

1. they can be installed in the most basic hypertext model, i.e. nodes and links, as shown in the following examples. Additional hypertext features, such as node aggregates, can be simulated with concept nodes, or added to the model as a new type of object.

2. they can also be installed in more elaborate models like Dexter or Trellis, applying the schema on top of the existing model.

A FORMAL SPECIFICATION OF CANTO

Using a formal specification language assists in the development, understanding and debugging of a system by adding precision and resolving ambiguity. The components and operations of Canto are formally specified using the Z specification language [9]. These specifications describe what can be done using Canto, but not how it is accomplished. The specifications are not meant to describe any application of the Canto model in its entirety, only to describe the essential features. Application details, such as the user interface, are not addressed by these specifications.

Formal specifications are used here for three reasons: (1) to precisely describe the functionality of the Canto data model; (2) to guide in the implementation of the Canto schema mechanism; (3) to show that the resulting hypertext satisfies the desirable properties of validity and accessibility. Validity relates to the structural integrity of the concept nodes. Validity means that any subtree preserves the meaning of its root concept node. This principle implies that the key phrase of a child will be a sub-idea of the key phrase of the parent, at least in the sense that concept nodes do not contradict their ancestors. Accessibility implies that any node can be reached by sequentially traversing some set of links. Accessibility and validity are complementary properties since validity determines the structure needed to access the nodes.

The Z specification language was chosen because of its mathematical basis and modeling capabilities. Z uses unique symbolism to represent concepts or components. (In this paper, the term *spec* will be used for a Z schema since Canto also uses the term schema, but with a different meaning.) Spec definitions are enclosed in a three-sided box—omitting the right side of the box. A horizontal line divides the box into two segments. The word heading the box is the name of the spec. Above the dividing line are the declarations including variables and showing state changes. Below the line preconditions and relationships between the variables are stated. The preconditions must be true for the remainder of the spec to be executed. These relationships are true in every state of the specification and are maintained by all operations. The Appendix summarizes the Z notation used in this paper.

Z specifications are created using a pyramid approach—each description or definition is built using prior definitions. The specifications in the following sections start with the components of concept nodes (identification number and key phrase), and build to the state space HYP that describes the Canto hypertext data model. Since the schema is built in and imposed on the hyperdocument, the same sort of operations that manipulate the data also manipulate the schema. In this paper, operations on the hypertext system such as movement, addition, and deletion of nodes and links, are described only for concept nodes. These operations can also be described for information nodes. (Some specifications used are omitted from this paper, but appear in detail in the first author's dissertation [8].)

Basic components

Canto comprises two types of nodes, concept nodes (cnodes) and information nodes (inodes). Cnodes contain an identification number (CID) and a key phrase. The CID is a sequence of positive integers, separated by hyphens. For example, the node 1–3–2 would

be a third level node, reached via the root node's first child (1), to the third child (1–3), to the second child (1–3–2). The root node has a CID of zero. CIDs for the root node's children consist of single positive integers. Other CIDs are created by adding ($\frown$) a dash and an integer to an existing CID.

$$CID{:}\mathbb{N}_1 \times \mathbb{N}_1$$
$$\forall n{:}\mathbb{N}_1 \bullet CID = prev(CID) \frown '-' \frown n$$

The other component of a cnode is the key phrase, indicating the theme of the following nodes. This is introduced in the specification using a basic type definition.

$$[KEY\_PHRASE]$$

Now a cnode can be defined ($\widehat{=}$) as a function ($\rightarrow$) from an identifying number (CID) to a key phrase. Canto does not require each key phrase to be unique.

$$CNODE \mathrel{\widehat{=}} CID \rightarrow KEY\_PHRASE$$

In the following specifications, each concept node contains a CID with an associated key phrase.

The information node's identification number, NID, consists of the associated cnode's number followed by a suffix. This suffix is defined as an element in an ordered set of letter(s). The prior definition of CID is expanded to define NID. The NID indicates this mapping of information nodes to a specific concept node.

$$suffix \in A,B,C,\ldots,Z,AA,AB,\ldots$$
$$NID = CID \cup suffix$$

The data contained in the information node (inode) is defined using a basic type definition, similar to the KEY_PHRASE definition.

$$[Data]$$

The inode can now be defined as a function mapping the identification number (NID) to the data.

$$INODE \mathrel{\widehat{=}} NID \rightarrow Data$$

196z Explicit linking is done by indicating the identification numbers of the two nodes to be connected. An explicit link can connect nodes of either type. The order in which the node's identification number is given indicates the direction of the link.

$$LINK == \{CID \times CID\} \cup \{CID \times NID\} \cup \{NID \times NID\} \cup \{NID \times CID\}$$

Enough basic components have now been defined to construct the specifications of Canto. The first aspect of the system must describe the state space called HYP corresponding to the hypertext data model. No constraint is placed on the organizational structure of the

nodes through this specification. Three variables, *cused*, *nused*, and *lused*, are declared as a power set ($\mathbb{P}$) to contain the concept nodes, information nodes, and explicit links used within the hypertext system, namely, dom CNODE, dom INODE, and dom LINK. These variables must be in the form previously defined by CNODE, INODE, and LINK. These will be used later for node traversal, insertion, deletion and retrieval.

HYP

$$cused : \mathbb{P}\, CID$$
$$nused : \mathbb{P}\, NID$$
$$lused : \mathbb{P}\, LINK$$

$$cused = \text{dom}\ CNODE$$
$$nused = \text{dom}\ INODE$$
$$lused = \text{dom}\ LINK$$

The state HYP is initially empty. Nodes and links can then be added to the new hyperdocument.

$$HYP_{init} \mathrel{\widehat{=}} \{\}$$

Just as the hypertext system must be initialized, the user's starting location within the system must be stated. The spec for the user's starting location must assume that the hyperdocument already contains at least one concept node. The starting position is the root concept node, which has a *cid* of 0. The symbol Ξ indicates that the state space HYP will not be altered by this specification. The symbol ? indicates input of a concept node id of the type defined by CID.

START

$$\Xi HYP$$
$$cid? : CID$$

$$HYP \neq \{\}$$
$$cid? = 0$$

The specifications for operations of insertion, deletion and retrieval of nodes in Canto can now be described. An additional construct would be useful to indicate whether or not an operation has been successful. Using a free type definition, RESULT is defined to have two values.

$$RESULT ::= ok \mid not\_ok$$

Movement

Movement through the hypertext structure involves movement through the concept nodes. HYP is not altered in this specification, and the concept nodes in the hypertext are contained in *cused*. The node being retrieved is *cnode*?. As the user moves from one concept node

to another, the change in *cnode* is indicated by *cnode*'. Vertical movement is defined in terms of the *succ* and *prev* functions, which refer to the next lower and higher levels in the hierarchy, respectively. Horizontal movement within the level of the concept nodes is done by increasing or decreasing the *cid* of the current cnode. In the last two clauses, the symbols $+$ and $-$ indicate addition and subtraction within the last syllable of the identification number.

Hop_Cnode

$$\Xi HYP$$
$$cused = \text{dom } CNODE$$
$$cnode?:cused$$

$$cnode' = succ(cnode) \lor$$
$$cnode' = prev(cnode) \lor$$
$$cnode' = cnode(cid) + 1 \lor$$
$$cnode' = cnode(cid) - 1$$

For example, to move vertically from the concept node with the identification number 1–2–3, the successor would be the concept node with the identification number 1–2–3–1. The previous concept node would have the identification number 1–2. To move horizontally from the concept node, the last syllable of the identification number 1–2–3 is reduced by 1 to move to the left (1–2–2) or increased by 1 to move to the right (1–2–4).

Insertion

One function this system must be able to do is add concept nodes, allowing the schema HYP to grow (ΔHYP indicates change). This specification, Add_Cnode, expects a *cid* and a key phrase as input, and produces one of the previously defined responses as output. Movement to where the node is to be added is accomplished through Hop_Cnode. If the cnode already exists, the set *cused* is not changed and *result* is set to not_ok. If the cnode does not exist, it is added to the existing set *cused*, resulting in a new set *cused*'. Since specifications state how operations are performed but not how they are implemented, it must be assumed that the designer of the hypertext system correctly places the concept nodes in the structure.

Add_Cnode

$$\Delta HYP$$
$$cid?:CID$$
$$key?:KEY$$
$$result!:RESULT$$

$$Hop\_Cnode$$
$$(cid? \in cused \land$$
$$\quad cused' = cused \land$$
$$\quad result! = not\_ok) \lor$$
$$(cid? \notin cused \land$$
$$\quad cused' = cused \cup \{cid? \mapsto key?\} \land$$
$$\quad result! = ok)$$

Deletion

The deletion of a concept node involves re-linking the hypertext and determining what should be done with any information nodes associated with the concept node being deleted, and deleting and storing any explicit links. In the specification Del_Cnode, the *cid* of the node to be deleted is input (?). If *cid*? exists and is not the root, the key phrase associated with the it is output (!). The hypertext is re-linked by linking the concept node following the deleted concept node to the previous concept node. The information nodes of the deleted concept node are output as the set *poss*!. The deleted concept node is then placed in a history file (as described by the spec called Store_Cnode) to prevent loss of information and allow versioning. Existing explicit links must also be located and removed or re-linked using the spec Del_Explink.

There is also the possibility that a concept node will be deleted but the associated information nodes will be attached to the previous concept node. In this case, different specifications are used to delete the concept nodes and re-attach the information nodes, storing the concept node in the history file and checking for explicit links.

Del_Cnode

ΔHYP
$cid?:CID$
$key!:KEY$
$poss!:INODE$
$result!:RESULT$

Hop_Cnode
$(cid? \notin cused \land$
$\quad key! = not\_found \land$
$\quad result! = not\_ok) \lor$
$(cid? \in cused \land$
$\quad cid? \neq 0 \land$
$\quad key! = cused(cid?) \land$
$\quad cused(prev(cnode)) = cused(succ(cnode)) \land$
$\quad \forall a{:}suffix \mid poss! = \{n{:}cid? \mid nused(n) = data\} \land$
$\quad Store\_Cnode \land$
$\quad Del\_Explink \land$
$\quad result! = ok)$

Retrieval

Concept nodes may be retrieved either through the identification number or through the key phrase. Retrieval using the id number of the cnode (namely *cid*?) will result in a key phrase output (*key*!) with no change to the hypertext.

Read_Cnode

ΞHYP
$cid?:CID$
$key!:KEY$
$result!:RESULT$

$(cid? \notin cused \land$
$\quad key! = not\_found \land$
$\quad result! = not\_ok) \lor$
$(cid? \in cused \land$
$\quad key! = cused(cid?) \land$
$\quad result! = ok)$

Canto schema

The objects of nodes and links have now been defined. The operations and rules of the Canto Data Model have been specified. To complete the data model, a schema mechanism is needed. The schema mechanism is a language for specifying the initial structure and how the structure can be altered. A single sentence in this language is a hypertext schema. We now need a specification to specify what the schema is composed of, and the operations that alter it. This spec, "Schema", is a subset of the functionality of the schema mechanism.

Schema

ΔHYP

$Add\_Cnode \lor Del\_Cnode$

A hypertext system based on Canto implements the schema using concept nodes as described above. In addition, the system supports information nodes that the user may visit, but which are not explicitly a part of the schema.

Using these specifications, validity can be defined as the property that the concept nodes do not contradict their ancestors. That is, for any two concept nodes C_1 and C_2, if C_1 is an ancestor of C_2 then C_2's key phrase does not contradict C_1's key phrase.

Accessibility implies that any node can be reached by sequentially traversing some set of links. That is, for any information node $I \in$ nused there exists a sequence of concept nodes, beginning with the root and continuing C_{i_1}, C_{i_2}, ..., C_{i_n} such that the root is the parent of C_{i_1}, C_{i_1} is the parent of C_{i_2}, etc., and C_{i_n} is the parent of information node tmitI.

Integrity implies that the information nodes do not contradict their ancestors. Data contained within an information node will relate to the key phrase of the concept node owner. That is, for any information node I, the data contained in node I does not contradict the key phrase in the concept node that is I's parent.

It can be shown that the standard operations for a hypertext system (insert, delete, traverse, and read) can be accomplished without adversely affecting the validity of the structure or the accessibility and integrity of the data. It can also be shown that any hypertext system can be structured using the Canto schema [8].

The question now arises, how can we enforce the properties of accessibility and validity? Accessibility is enforced by Add_Inode, which states that an information node may only be added to a concept node with an appropriate key phrase. Validity of the concept node structure is enforced by restricting Add_Cnode and Del_Cnode so that it's the designer who establishes the set of concept nodes in an appropriate hypertext structure. Users cannot disturb the structure without permission. The structure may evolve over time, with inodes becoming cnodes, etc. Some concepts may fade in importance, others may need further elaboration. To allow for these alterations, a schema manipulation language is provided. Responsibility for integrity lies with the designer or anyone with modification authority.

CANTO SCHEMA DEFINITION LANGUAGE (CSDL)

These Z specifications formalize the relationships between the data types in Canto. The schema is a hierarchy of concept nodes. A concept node (cnode) contains a descriptive key phrase. The terms "Hatch" and "Parent" are syntactic shorthand used at the beginning of the schema definition. A Hatch indicates the aggregation of cnodes for a specific parent. The term parent is used in the definition as a first-level grouping of hatches.

An instance of the schema, coupled with the data contained in the information nodes (inodes) makes up a complete hyperdocument for an application. The first part of the schema description for a specific application contains the parents and hatches as indicated. The second section contains the concept nodes and the associated information nodes (a concept node may have zero or more associated information nodes). The hypertext may also contain external links.

Figure 2 indicates how the components of the CSDL are used. An information node may contain data or an external link to another node. As indicated in the specifications, the identification number of a concept node is composed of numeric syllables. The identification number of an associated information node is the same as its parent but concludes with a letter. Tokens are set in bold type.

```
parent id
hatch :: id(key) --> id(key) ... id(key)
cnode id :: key
inode id :: {text}
inode id :: link ==> id
link ==> id == id
```

Figure 2. The components of the Canto Schema Definition Language

A general form of the CSDL schema is shown in Figure 3. This general form is used to indicate the usage and organization of the components, and is application-independent.

CANTO SCHEMA MANIPULATION LANGUAGE (CSML)

The Canto Schema Manipulation Language (CSML) is used to alter an existing structure or change the data within that structure. Some of the Z specification operations that are evolutionary in nature are invoked in CSML. The tokens are capitalized and set in bold type. The information the user must enter is set in italics.

```
hatch :: 0(name) --> 1(name) 2(name) ... i(name)

Parent 1
hatch :: 1(name) --> 1-1(name) 1-2(name) ... 1-j(name)
hatch :: 1-1(name) --> 1-1-1(name) 1-1-2(name) ... 1-1-k(name)
hatch :: 1-2(name) --> 1-2-1(name) 1-2-2(name) ... 1-2-k(name)
hatch :: . . .
hatch :: 1-j(name) --> 1-j-1(name) 1-j-2(name) ... 1-j-k(name)
hatch :: 1-1-1(name) --> 1-1-1-1(name) 1-1-1-2(name) ...
...

Parent i
hatch :: i(name) --> i-1(name) i-2(name) ... i-j(name)
hatch :: i-1(name) --> i-1-1(name) i-1-2(name) ... i-1-k(name)
hatch :: i-2(name) --> i-2-1(name) i-2-2(name) ... i-2-k(name)
hatch :: ...
hatch :: i-j(name) --> i-j-1(name) i-j-2(name) ... i-j-k(name)
hatch :: i-1-1(name) --> i-1-1-1(name) i-1-1-2(name) ...

cnode 0 :: key phrase
inode 0a :: {text}
cnode 1 :: key phrase
inode 1a :: {text}
cnode 1-1 :: key phrase
inode 1-1a :: {text}
cnode 1- ...
inode 1-...
cnode ...
cnode i :: key phrase
inode ia :: {text}
cnode i-1 :: key phrase
inode i-1a :: {text}
cnode i-...
inode i-...a
```

Figure 3. General form of a Canto schema. "Name" indicates the content of a concept node. The letters i, j, and k are positive integers, and the letter 'a' refers to any (upper-case) letter

As indicated in the specifications, the schema for the hypertext is derived through the concept nodes. The specification SCHEMA indicates the primary manipulations CSML must invoke. The first manipulation is to add a concept node to the hypertext. This allows the structure to grow. The designer of the hypertext system must create the identification numbers for the nodes to correspond to the organization of the nodes and implicit linking.

To add a concept node, the designer enters the identification number and the key phrase:

ADD CNODE *id* **::** *key phrase*

The command to delete a concept node is:

DROP CNODE *id*

CSML provides the ability to create or remove explicit links between a source node id_s and a target node id_t:

CREATE LINK `==>` id_s `==` id_t

DELETE LINK `==>` id_s `==` id_t

Although the information nodes are not a part of the schema, they are an integral part of the hypertext. Operations for addition and deletion are also needed for information nodes but are not shown here. These operations adjust the identification numbers of the nodes in the hypertext system as needed. CSML furnishes the ability to reorganize the document using a different hierarchy.

TESTING

Testing the effect and usage of the Canto hypertext data model was done in two separate studies. The first study focused on the effect of a schema mechanism on hypertext use, the second study focused on the use of Canto in an educational setting.

Schema testing methodology

The Goal/Question/Metric paradigm was used to develop the testing. To our knowledge, this paradigm has not previously been used to evaluate a hypertext system. The G/Q/M paradigm is a framework for defining goals, then refining them into specific, quantifiable questions about the software process and product [10].

The goals in this experiment are formulated in terms of the purpose and perspective of the study. These goals relate to the problems currently found in hypertext systems. The last purpose-related goal (G6) evaluates the schema mechanism through the user interface. As noted by Nielsen, the user interface can strongly influence the test results, and must be evaluated along with the system as a whole [11].

The primary purpose-related goal is:

- to determine if the use of a schema in a hypertext system improves the use of that system.

The secondary purpose-related goals are:

- to determine the user's orientation within a hypertext system using Canto in order to assess orientation capabilities.
- to evaluate how a hypertext system built with a schema mechanism helps assist users in maintaining their orientation within the system in order to prevent disorientation.
- to evaluate how a hypertext system built with a schema mechanism assists users in determining where the information they are seeking is located in relation to their current location.
- to characterize the use of a hypertext system with a schema in assisting the user in choosing the correct path to access the desired information.
- to evaluate ease of use (learning, efficiency) in order to improve the user interface.

The perspective-related goal is:

- to examine the acceptability, effectiveness and reliability of a hypertext schema from the viewpoint of the user.

This set of goals generates questions designed to define and quantify the goals. The aim is to satisfy the intuitive notion of the goal as completely and consistently as possible. There are several areas or subgoals (shown in italics) related to the process and product to be addressed by the questions. In the following section the term Brand X refers to the hypertext system that uses a schema mechanism, and the term Brand Y refers to a hypertext system that is similar to Brand X in every way, except that it does not use a hypertext schema.

The process questions, which relate to the retrieval of information, include:

- How does Brand X's use in retrieving specific information compare to Brand Y?
- Does the user's perception of the data organization resemble or match the actual data organization?
- Would the user prefer to access information using a hypertext system as opposed to a linear document?
- Can a user determine where the desired information is located?
- Can a user easily and directly access information?
- At a given time can users determine what steps they have taken to get to that location?
- Did the user get the correct information?

The product questions related to the hypertext system include:

- Does the arrangement of the information have an effect on how the information is accessed?
- Is data within a structure or organization easier or more difficult to access than data without a structure?
- Do the users of Brand X locate the information faster than the users of Brand Y?
- Do the users of Brand X require fewer keystrokes than the users of Brand Y?
- Is the Brand X system easy to use?
- Are the keystrokes easy to remember?
- Are the meanings of the symbols used easy to remember?
- Are the screens easy to read and locate information on?
- Does a user's expertise (in computers or hypertext) influence system use?
- Do individual characteristics of the user influence their use of the hypertext systems?

Test environment

In this test environment, the product is the hypertext system built using the Canto schema mechanism, and the process is the use of Canto on the specified data set. In this test, 41 faculty and student volunteers were randomly assigned to one of the two systems. The object was for the users to locate a specific word and return the related fact. Each system contained the same 120 words and facts. All users searched for the same ten words.

The first hypertext system, Brand X, used the Canto schema mechanism. Navigation used the implicit linking of the schema. No explicit linking was used. The words were organized into a hierarchical structure with four levels. The system contained approximately 375 concept nodes and approximately 120 information nodes. The words were grouped using primarily intuitive concepts, but unusual enough that the user could not rely on prior experience with word searching. Each level was grouped according to the following:

Level 1: function (animal, food, sport)
Level 2: a grouping associated with the topic on Level 1:
    1(animal): 0 legs, 1 leg, 2 legs
    2(food): fruit, vegetable, dairy
    3(sport): ball, water, other
Level 3: first letter of the desired word (a, b, c, . . . , z)
Level 4: listing of words

The nodes on level 1 and level 2 have an associated information node explaining the concept. The words in level 4 have an associated information node that contains the fact the user is searching for.

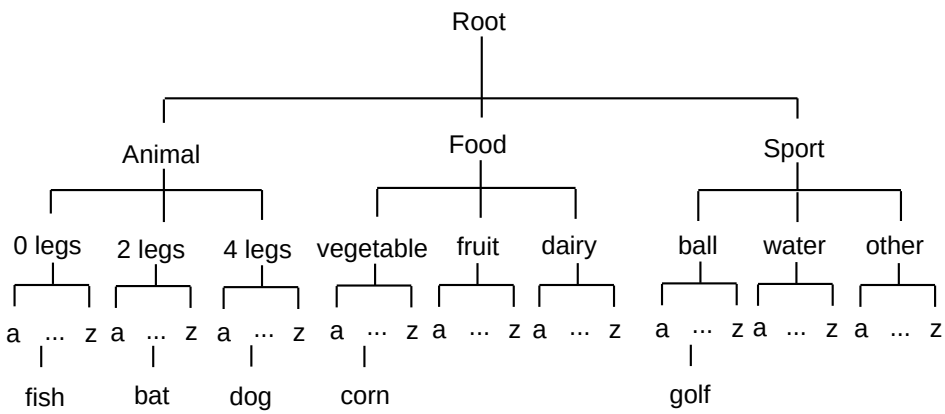

Figure 4. Concept nodes in the test system

The Brand Y hypertext system, built specifically for this test, uses no schema mechanism, and relies entirely on explicit linking to retrieve information. The desired information was accessed by reading the displayed sentence or text, then keying on a word (anchor) to move to another sentence. Brand Y used over 500 nodes (sentences). There were no suggested paths; the user had to determine where to go to access the information. The text was structured so that the user had to move through at least four sentences before the desired fact could be located. This minimized the differences in access times and keystrokes between the two test situations since Brand X users must move through four levels before accessing the desired fact.

The 26 metrics used for the evaluation of the hypertext systems came from several sources. The first set of metrics (1–22) was from a questionnaire answered by the user. Four additional metrics were also used for evaluation. The accuracy of the information

retrieved by the user was metric 23. The questions asked by the user were evaluated as metric 24. The total time of session was metric 25. The logged data of the user's keystrokes was metric 26. This metric was later broken into five individual components. This log of keystrokes was created automatically by the system while the users were working, without their knowledge.

Test results

The results are summarized in Tables 1, 2, and 3. The first two groups of metrics analyzed are from the user's responses on the questionnaire. After users found the five answers, they were to rate the system on its ease of use, the ease of determining where the information was located, and the ease of getting the information. The scale of possible responses was easy(1) to difficult(5).

The second group of metrics, shown in Table 2, is also from the questionnaire answered by the users. They were to determine how often they felt lost within the system, and how often they felt frustrated. The responses were later given numerical values for evaluation. The choices were: never(1), rarely(2), sometimes(3), often(4).

Table 1. Users' ratings of the Brand X and Brand Y systems

	Easy (1–2)	Medium (3)	Difficult (4–5)	Mean	
System use	83%	13%	4%	1.7	Brand X
	78%	22%	0%	2.0	Brand Y
Where is the	96%	4%	0%	1.3	Brand X
Information	44%	39%	17%	2.7	Brand Y
Getting the	87%	13%	0%	1.5	Brand X
information	72%	11%	17%	2.2	Brand Y

Table 2. Users' evaluation of the process of locating information

	Never (1)	Rarely (2)	Sometimes (3)	Often (4)	
Lost	43%	48%	8%	0%	Brand X
	6%	33%	61%	0%	Brand Y
Frustrated	78%	22%	0%	0%	Brand X
	0%	39%	22%	39%	Brand Y

Table 3. Summary of users' keystroke counts and times

	Mean	Std dev.	25%–75%	
Keystrokes	36	9.48	31–42	Brand X
	141	51.81	93–193	Brand Y
Time	565	162.7	427–688	Brand X
	1080	328.1	819–1365	Brand Y

Another important group of metrics is summarized in Table 3. These counts were accumulated by the computer. The time shown is the total seconds it took the user to locate the five words and the associated fact. The counts are the total number of keystrokes used to locate the five facts. This count includes both valid and invalid keystrokes. (A keystroke was considered invalid if the selection did not exist or if the user tried to go "up" through the previous selections past the beginning of the file.) The statistics in Table 3 are the mean, standard deviation, and the range for the middle two quartiles (25%–75%). It was felt that these statistics were sufficient to indicate the general trend of the data.

The results of this test indicate that Brand X, using the Canto model, was an easier system to use. The users rarely, if ever, got lost within the system, and could quickly and efficiently locate the desired information with very little effort needed to determine the correct path. The users of Brand X indicated that they would rather use a hypertext system than a printed list. Because of this test comparing a structured and a non-structured hypertext system, we conclude that a hypertext system created using a schema mechanism and built with structured data is usable [8].

The purpose of this testing of Canto was to determine if the use of a schema mechanism in a hypertext system alleviates some of the user's difficulties. This test was constructed to eliminate or reduce all bias and system differences between a system with a structure, Canto, and a hypertext system without structure, Brand Y. The two systems had almost identical user interfaces. The questions asked of users of both systems were the same, and the users were randomly assigned to a system. The only factor allowed to vary in the two test environments was the organization of the data.

Educational testing

The ability to read programs methodically and accurately is a critical skill in programming. Program reading is the basis for modifying and validating programs written by others, for selecting and adapting program designs, and for verifying the correctness of one's own programs. Programmers can become more effective through critical study of programs written by others. Given a well-documented program, the reading process can generally proceed top-down, from overall design to successively lower levels of detail. In reading the best-documented programs, one needs an occasional foray into details, if only to understand the context of documentation that is intended to precede the details. In reading a totally mysterious program, it is useful to back out of details periodically to form hypotheses or guesses that can help fit details together more easily. Because reading programs is valuable to programmers, there is a need for a convenient tool (other than ordinary paper listings, which can be unwieldy) that makes teaching this skill viable. Since hypertext is well equipped to locate portions of a document or program, and since Canto's hierarchical structure is well suited to Pascal, it is appropriate to use Canto to teach the skill of reading Pascal programs.

Hyper-Pascal was designed to help students learn to read Pascal programs. Hyper-Pascal was built using Canto, and is currently running at Goucher College. Teaching programming to beginning students involves many concepts, such as logic design, determining the correct syntax, and variable usage. Learning and understanding the syntax and correct statement usage is a major problem for students. Studies have shown that reading program segments and modules encourages proper syntax and statement usage [12].

The first step in the design of Hyper-Pascal was to configure the general schema to

Pascal. Since Pascal has very specific component ordering, it was relatively easy to break the components into a concept node structure and indicate information node content. The first level of concept nodes consists of three sections: program heading, declarations, and modules. Each of these concepts can then logically be broken into smaller components or concepts. This breakdown of Pascal into a hierarchy of concepts forms the schema. Some decisions had to be made when determining how the concepts should be separated. The granularity of the program components could not be too small when using the system to read programs, so the text of the modules was placed in one information node.

Figure 5 shows the first screen for a 56-line program called “Sample”. On the top line, the name of the program appears, followed by the selections. In the lower portion of the screen, the symbols used are explained, and possible keystrokes and their meaning are listed. On this screen, no information nodes are associated with any of the concept nodes listed, so the symbol (+) does not appear. Each of these choices has additional related concepts, so the symbol (*) does not appear either.

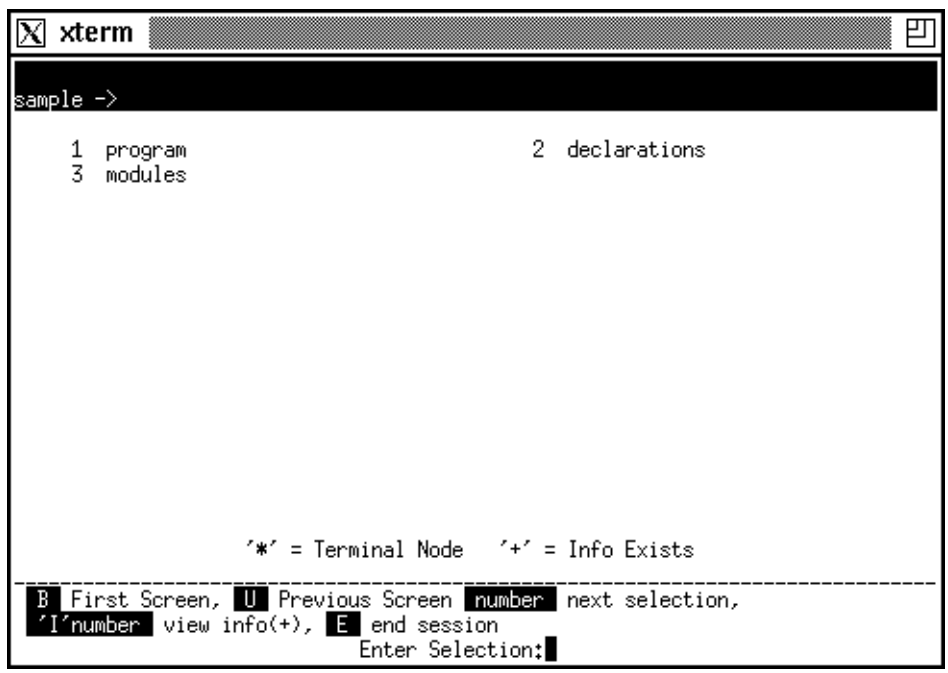

Figure 5. The first screen generated by Hyper-Pascal when run on the sample program

If the user chooses selection 3 (modules) from the first screen and then chooses the option to view the information in the main body (I3), the screen shown in Figure 6 appears. The top of this screen indicates the selections the user has made to arrive here: sample –> modules –> main body. The next section of the screen lists the information associated with the main body of the program.

We tested Hyper-Pascal using volunteers who were students, or teachers who had taught Pascal programming. Three different-sized programs were used. For each program, the user was asked specific questions about the program, i.e. to identify a module type and purpose.

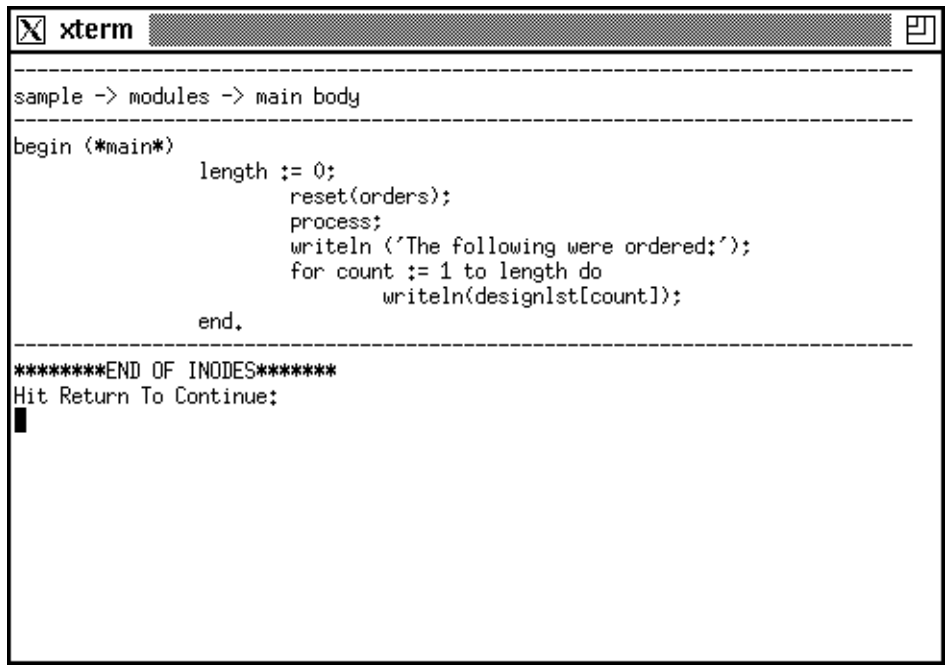

Figure 6. The Hyper-Pascal screen showing the main body of the sample program

We quickly determined that the benefits of using Hyper-Pascal to read programs were best derived with longer programs of multiple modules. Apparently, in shorter programs a linear listing is still easier to use. With longer programs, it became difficult to locate the desired variable or module in the linear listing, but with Hyper-Pascal the information was quickly retrieved.

Our findings indicate that using Hyper-Pascal to teach the skill of reading programs provides a valuable asset to students. The users felt the system was easy to use and was helpful in determining program design and function. They could easily locate the desired information and did not feel lost or overwhelmed within the information [13].

RELATED WORK

The notion of hierarchical structure in hypertext is not new. DIF (Document Integration Facility) is a hypertext system designed to help integrate and manage documents produced and used throughout the software project's life cycle. DIF defined the structure and forms using a tree structure. These forms, defined only once and inherited by all projects, define the software process to be followed, thereby defining the nature of the information [14]. In Canto, the hierarchical structure is used to assist readers in locating themselves within the data, and locating the desired information.

Various structures have been applied to hypertext models. A three-layered structure was used by the Dexter Reference Model. This model was designed to be a formal model of important hypertext abstractions, and to serve as a source for interchange and interoperability

standards, specifically to standardize hypertext terminology. The three layers that make up Dexter are: the storage layer, presentation of the hypertext, user interaction; the runtime layer, a 'database' containing a network of nodes and links; and the within-component layer, the content/structure inside the nodes [15]. The Trellis Hypertext Reference Model identifies five different levels of abstraction in hypertext: abstract component, abstract hypertext, concrete context, concrete hypertext, visible hypertext. The highest levels are abstract, and specify hypertext and the components, but not the details of user presentation. In the lower levels these abstractions are transformed into more concrete representations, first displaying content, then mapping it to windows, but not tying down how the windows will be displayed. The lowest level in Trellis details the concrete hypertext mappings to the visible form that the reader sees [12].

Several other authors have touched on the idea of schema in hypertext. Lange's object-oriented hypertext model separates presentation and browsing semantics from the model, and places them with the application design. In this model the interior of a node has a schema, called a substructure, that becomes a template for the node contents. The interior schema represents a collection of named slots within the node, and has anchors and destinations identified for the node [7]. Aquanet is a hypertext system designed to support knowledge-structuring tasks. It is a browser-based tool that allows users to graphically represent information in order to explore its structure. The knowledge structures merge hypertext constructs with frame-based representations using basic objects (nodes) and relations (links). In Aquanet, schemas are used to define the appearance of the screen in order to improve readability, not to structure the data internally [16]. The Hypermedia Design Model (HDM) is a design model for hypertext applications. In HDM, the term "schema" is used to describe a hierarchical structural pattern of entity types allowing for different hyperviews as required for a particular situation. At this time, however, an application-independent model does not exist [17]. In Canto, the schema assures the validity, integrity and accessibility of the data. The formal specifications of the data model and schema mechanism define the schema explicitly, making it easier to use a hypertext system built on the Canto data model.

CONCLUSION

The salient characteristic of the Canto data model is a comprehensive, application-independent, formally specified schema mechanism. Every Canto schema consists of concept nodes that organize the document, and information nodes containing text and other material. The schema for a specific application of Canto is prepared using the Canto Schema Description Language. The Canto Schema Manipulation Language allows the structure of the schema to be modified. We used the G/Q/M paradigm to develop a statistical testing methodology for evaluating Canto. This testing showed that a hypertext system designed using the Canto schema mechanism was easier to use, and allowed users to find the desired information faster, than would have otherwise been the case. These results indicate that the Canto hypertext data model is a viable and valuable tool for hypertext system design.

ACKNOWLEDGEMENTS

We thank the editor and anonymous referees for their helpful comments on earlier versions of this paper.

REFERENCES

1. Jeff Conklin, 'A survey of hypertext', Technical Report MCC Technical Report Number STP-356-86, Rev. 2, Software Technology Program, (December 3, 1987).
2. Christopher Dede, 'Role of hypertext in transforming information into knowledge', in *National Educational Computing Conference Proceedings*, 1988, pp. 95–101.
3. Frank G. Halasz, 'Reflections on Notecards: Seven issues for the next generation of hypermedia systems', *Communications of the ACM*, **31**(7), 836–852, (1988).
4. William O. Beeman, Kenneth T. Anderson, Gail Bader, James Larkin, Anne P. McClard, Patrick Mcquillan, and Mark Shields, 'Hypertext and pluralism: from linear to non-linear thinking', in *Hypertext'87 Proceedings*, 1987, pp. 67–88.
5. James Martin, *Principles of Database Design*, Prentice-Hall, 1976.
6. Charles F. Goldfarb, *The SGML Handbook*, Oxford University Press, 1990.
7. Danny Lange, 'A formal model of hypertext', in *Hypertext Standardization Workshop*. National Institute of Standards and Technology, (January 16–18, 1990).
8. Linda H. Rosenberg, *Canto—A Hypertext Data Model*, Ph.D. dissertation, University of Maryland Baltimore County, June 1991.
9. J.M. Spivey, *The Z Notation—A Reference Manual*, Prentice-Hall, 1989.
10. H. Dieter Rombach and Victor Basili, 'Quantitative assessment of maintenance: An industrial case study', in *Proceedings from the Conference on Software Maintenance*, 1987.
11. Jakob Nielsen, *Hypertext & Hypermedia*, Academic Press, 1990.
12. Richard Furuta and P. David Stotts, 'A functional meta-structure for hypertext models and systems', *Electronic Publishing: Origination, Dissemination, and Design*, **3**(4), 179–205, (1990).
13. Charles Nicholas and Linda Rosenberg, 'Using hypertext methodologies in teaching Pascal', in *National Educational Computing Conference Proceedings*, 219–224, 1990.
14. Pankaj K. Garg and Walt Scacchi, 'A hypertext system to manage software life cycle documents', in *Proceedings of the Twenty-First Annual Hawaii International Conference on System Sciences*, 1988, Vol. II, pp. 337–345.
15. Frank Halasz and Mayer Schwartz, 'The Dexter hypertext reference model', in *Hypertext Standardization Workshop*. National Institute of Standards and Technology, (January 16–18, 1990).
16. Catherine C. Marshall, Frank G. Halasz, Russell A. Rogers, and William C. Janssen, 'Aquanet: A hypertext tool to hold your knowledge in place', in *Hypertext '91 Proceedings*, December, 1991.
17. Franca Garzotto, Paolo Paolini, and Daniel Schwabe, 'HDM—a model for the design of hypertext applications', in *Hypertext '91 Proceedings*, December, 1991.

APPENDIX SUMMARY OF Z NOTATION

Symbol interpretation

Δ	altered data file or state space
Ξ	unaltered data file or state space
$\widehat{=}$	defined as
?	input variable
!	output variable
$'$	altered variable
$\{\}$	set
$\mathbb{P}$	power set
dom	domain
$\in$	element of
$\notin$	not an element of

$\cup$	union
$\wedge$	and
$\vee$	or
$\rightarrow$	function
$\nrightarrow$	partial function
$\mathbb{N}_1$	positive integers
==	equivalence
:	from the set of
::=	free type definition
$\frown$	concatenation