An Evaluation of the Validity of the Dexter Hypertext Reference Model: A Case Study

Bilge Karal Say and Fazlı Can

BİLKENT UNIVERSITY

Department of Computer Engineering
and
Information Science

Technical Report BU-CEIS-94-25

An Evaluation of the Validity of the Dexter Hypertext Reference Model: A Case Study

Bilge Karal Say and **Fazlı Can**[1]
Department of Computer Engineering
and Information Science
Bilkent University
Bilkent, Ankara 06533, Turkey
say@bilkent.edu.tr and *fcan@bilkent.edu.tr*

Abstract:

Various formal models have been proposed for defining hypertext systems. Among them, the Dexter Hypertext Reference Model is gaining wide acceptance as a base for the design of future hypertext systems and interoperability tools. However, this model has yet to be extended and validated against the hypertext systems it took as its basis. This paper is an attempt in the latter aspect, namely considering a widely used hypertext system and validating it against the original model. For this purpose, HyperCard, which was indeed one of the Dexter Hypertext Reference Model's "reference" systems, is chosen. For validation, we use the concept of refinement with Z, a formal specification language.

1 Introduction

Hypertext[2] is gaining world-wide usage throughout the computer users. Builders of the next generation of hypertext tools will certainly benefit from models bringing together current hypertext concepts. Existence of sufficiently formal models will facilitate the comparison of different hypertext systems and building interoperability tools to enhance multi-hypertext collaborative environments. This concept can be viewed as a common semantic representation where world's languages can be translated to each other by first converting into this common form. Moreover, the usage of formal specification languages in comparing models of hypertext systems may be subject to automation via reasoning tools. This paper aims to serve such a purpose by making an attempt to compare a specific hypertext system, HyperCard[3] with the Dexter Hypertext Reference Model (referred to as the Dexter model for brevity from here onwards).

The design of the Dexter model was initiated with a series of workshops bringing together a number of hypertext system designers. It is gaining popularity as a unifying framework and several researchers have either extended the model or built experimental systems based on it [Comms. ACM 94]. The model is specified formally in Z, as well as being attributed a clear informal definition [Halasz and Schwartz 90]. Z is a formal specification language based on set theory and logic [Spivey 89]. It also allows the usage of refinement and proof techniques, that makes it suitable for a validation study.

[1] On sabbatical leave from Department of Systems Analysis, Miami Univ., Oxford OH, 45056, USA.
[2] The term *hypertext* is used interchangeably with hypermedia within this paper.
[3] HyperCard is a trademark of Apple Computer,Inc.

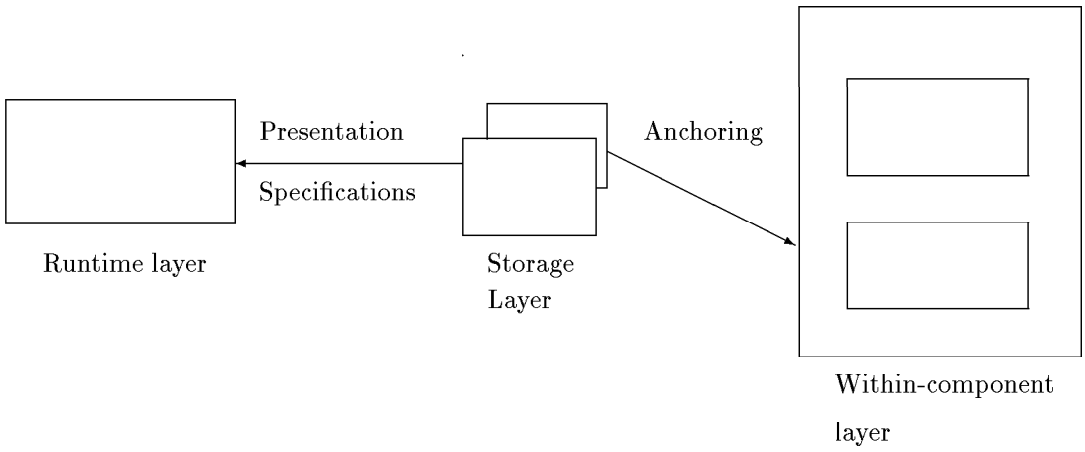

Figure 1: Layers of the Dexter Model.

Several other researchers have tried using formal notations including Z for modelling hypertext systems, but their models are not at the moment as generic as the Dexter model [Garg 88, Afrati and Koutras 90, Hitchock and Wang 92]. By taking the Dexter model as our generic model and building a validation specification for HyperCard, we aim to see how naturally and closely the Dexter model represents HyperCard, one of its "reference" models. Moreover, we want to see whether there can be some extensions for the model itself. We think that a validation specification will be valuable in both validating the power of the Dexter model and seeing how such a comparison works.

2 The Prequisites

2.1 An Overview of the Dexter Model

The Dexter model organizes the process and data information held about a hypertext system into three layers. It specifically concentrates on the *storage layer* which describes the general structure of the components that contain text, images, sound etc. and the network of links that characterize a hypertext system by making the browsing structure of a document non-sequential. The second layer is the *within-component* layer, which can accommodate the various forms a single component can take. The specification of this layer is left open to a great extent in the Dexter model, as there are implementation-dependent models that can describe the inside structure of a component. The interface of this layer with storage layer is realized in the form of an *anchoring mechanism* which give information about how components in general are related to their specific parts such as buttons or individual words. This mechanism facilitates the building of links between components.

The third layer is oriented to the dynamics of a hypertext system as the applications on it are run by users. A user can be presented with a hypertext application in several ways, as well as changing the contents of the hypertext application during a *session*, such as making annotations while reading a document. These features are incorporated into a *runtime layer*, which is again generically defined as the within-component layer. The storage layer presents information to the runtime layer in the form of *presentation specifications*, which are encodings of how the user is to be presented specific components, such as a special visual effect. The layer structure of the Dexter model is summarized in Figure 1.

We will benefit from concentrating on the structure of a component briefly for purposes of our

validation study. A component can be in various forms. An *atomic component* in the Dexter model is what is commonly known as a node in most hypertext systems, as the basic unit of information presented to the user. An atomic component is uniquely identifiable as well as having different attributes, contents, presentation specification and anchoring information. Anchors are uniquely identifiable within a component pointing to a specific value such as part of a screen, or a word in the text. *Link components* differ from atomic components in having no contents but various *specifiers*. Specifiers denote which components are linked by a link component. Finally, components can be composed of other components to form hierarchical composite structures. A more detailed description can be found in the Dexter model papers [Halasz and Schwartz 90, Halasz and Schwartz 94].

2.2 An Overview of the HyperCard Building Blocks

HyperCard is a hypertext authoring system working on Apple Macintosh computers, first released in 1987 [Michel 89]. It was made popular by the fact that it was for a long time distributed free with Macintosh operating system software.

A HyperCard application is called a *stack* and consists of *cards* which correspond to atomic components in Dexter terminology or nodes in general. Only one card is visible to the user at a time. A card can contain images and text, and be related with sound or animation. *Fields* are basic units for entering and displaying text. *Buttons* are the basic visual devices which are "pushed" by mouse clicks or keyboard keys and they are the basis for creating links to navigate. If several cards share same layout, buttons etc. these can be put on a *background card* for ease of programming. Buttons, fields, cards or stacks can be associated with different *properties* such as text size for fields, highlighting for buttons. They can also have their own *scripts* which are procedures written in HyperCard's scripting language, Hypertalk. Hypertalk can be used to create links among other things such as chunking expressions in fields, making calculations. A script is composed of *message handlers* which intercept events such as clicking of the mouse, and specify a set of commands for that event. Functions or commands can also be imported from other programming languages. There is a message passing hierarchy such that if, for example, a message can not be handled at a button level, it is passed to the higher level in the hierarchy, then a card, and higher to the stack if it is not handled at card level neither. Some more information will be given about HyperCard in the next section.

3 The Validation Specification

3.1 The Limits of the Specification

The purpose of what we call the validation specification is to develop an understanding of the degree of conformance of the design principles of HyperCard against the Dexter model as explained in the introduction section. It can also be seen as a formal attempt to check the validity of the Dexter model and propose ways to extend it. In this report, we try to keep the Z specification as complete as possible giving the emphasis to underlying the methods and concentrating on representative examples. More specifically, we choose to leave out:

- the run-time layer as it is generic enough to cover a HyperCard session easily,
- most of the within-component layer as it is left open in the Dexter model anyway,
- several aspects of the Hypertalk scripting language which are too procedural and away from the main philosophy of a hypertext system,

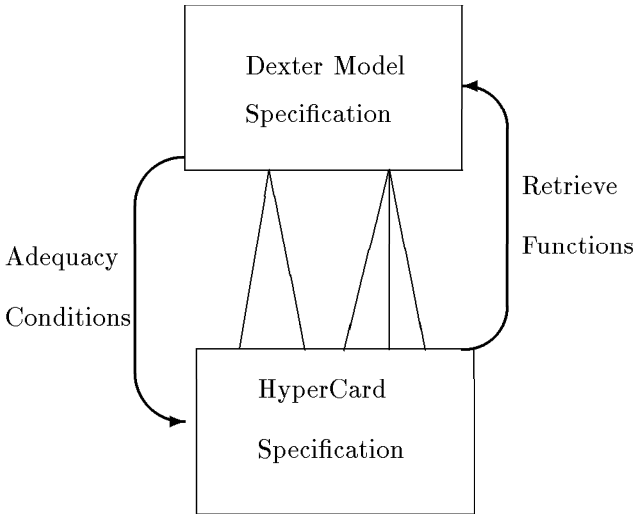

Figure 2: An Informal Description of the Refinement Process.

We also assume familiarity with Z and the availability of the original Dexter model paper [Halasz and Schwartz 90].

Thus, we are mainly taking a data-oriented approach concentrating on the storage layer of the Dexter model.

3.2 The Method

When making a formal specification of a system —usually from scratch—, one is encouraged to think in abstract terms first, for representing data and operations. Only when taking design decisions, we would rather start to think on more concrete, implementation-specific terms. Thinking of this procedure in reverse may help us compare the Dexter model with HyperCard, or any other hypertext system. The Dexter model is an abstract model of what hypertext systems are supposed to do and we want to show that HyperCard forms a concrete example of this model. The Dexter model is justifiably more powerful than HyperCard, but we ideally want to see which Dexter model constructs are expressible in HyperCard. Also in theory, every data type or operation actually realized in HyperCard should be traced back to a Dexter model construct for conformance to the model. Mappings between abstract specifications and concrete design structures are called reification [Jones 86] or more commonly in Z-related terminology *refinement* [Woodcock 91].

Let us explain the requirements of refinement more concisely before using the technique. We must be able to retrieve the original abstract data and constraints from the concrete structures by means of a *retrieve* relation[4]. This function (or relation) must be total, as it must be defined for each concrete structure. In Z, where the unit of specification is a schema, we would have the retrieve relation as a separate schema relating abstract and concrete states and data types of the system. In other words, this relation must provide an interpretation of the concrete design in terms of abstract specifications. The complementary condition for retrieve relation is the fact that there should be at least one concrete representation for any abstract value. This property is called *adequacy condition* and can be stated as several proof obligations or by informal argument. The refinement concept can be summarized as in Figure 2.

The argument above is basically related to data refinement rather than operation refinement.

[4] Usually, a function since a concrete representation has only one corresponding abstract representation.

We can simplify the operation refinement concept as a relation refinement process since useful operations are rather like relations between states as depicted fully in [Woodcock 91]. When a relation *Abs* is refined by relation *Conc*, domain of *Conc* is at least as big as that of *Abs*. Within the domain of *Abs*, *Conc* respects *Abs*, that is the relation that is deemed to be a refinement of the abstract one does not introduce further nondeterminism. When viewed as Z schemas, this concept can be defined as concrete operations having weaker preconditions and stronger postconditions than their abstract counterparts.

These refinement concepts give us several opportunities for proving statements made between abstract and concrete states and operations.

- Each possible initial concrete state must represent a possible initial abstract state.
- The concrete operation must terminate whenever the abstract operation is guaranteed to do so.
- If abstract operation is guaranteed to terminate then the state after concrete operation has ended must represent a possible abstract state in which abstract operation could terminate.

This technique briefly summarized here is mainly intended for refining specifications to design, though there is no reason why it should not be used for comparing whether a specific design (such as that of HyperCard) conforms to a generalized specification (such as Dexter model).

3.3 The Specification

We concentrate on the representative constructs of the storage layer of the Dexter model. We give the related data types and schemas of the HyperCard related constructs prefixed with a "*HC*". Non-prefixed data types and schemas are as taken from the original Dexter model paper [Halasz and Schwartz 90] unless they are retrieve schemas or otherwise specified.

A component in the Dexter model is characterized by a unique identifier, which is unique over the whole component space. There is a similar identifier number in HyperCard called "Card Id" but that number is only unique within a stack whereas an application may contain multiple stacks.

$$[HCCARDID]$$

$$UID == HCCARDID$$

$$\begin{array}{|l} \textit{Constraint}_1 \\ \hline stack_1, stack_2 : \mathbb{P}\, HCCARDID \\ c_1, c_2 : HCCARDID \\ \hline \neg\, ((c_1 \in stack_1 \land c_2 \in stack_2) \Rightarrow \\ \quad c_1 \neq c_2) \end{array}$$

In HyperCard a link's direction is usually *from* a button or a card *to* a card. The Dexter model also allows bidirectional or non-directional link end-points[5].

$$HCDIRECTION ::= NONE \mid TO$$

$$DIRECTION ::= FROM \mid TO \mid NONE \mid BIDIRECT$$

[5] In HyperCard, the NONE endpoint of a HyperCard link is actually a FROM endpoint that always points to the whole card.

In HyperCard, anchors can either be only in the form of buttons.

$$[HCBUTTON, HCBUTTONID]$$

$$ANCHORID == HCBUTTONID$$
$$ANCHORVALUE == HCBUTTON$$
$$ANCHOR == ANCHORID \times ANCHORVALUE$$

In HyperCard a component specification is either a card name or a card number denoting its sequence in the stack.

$$[HCCARDNAME, HCCARDNO]$$

$$HCCOMPONENTSPEC == HCCARDNAME \cup HCCARDNO$$

A presentation specification, on the other hand, is denoted by a series of values for visual effects.

$$HCPRESENTSPEC == \{BarnDoor, ZoomIn, .., Iris\}$$

The Dexter model relevant types can be instantiated to these definitions.

$$COMPONENTSPEC == HCCOMPONENTSPEC$$
$$PRESENTSPEC == HCPRESENTSPEC$$

A *specifier* specifies one end-point of a link.

SPECIFIER

$componentspec : COMPONENTSPEC$
$anchorspec : ANCHORID$
$presentspec : PRESENTSPEC$
$direction : DIRECTION$

A similar schema may be defined for HyperCard prefixing all components by HC. Now, let us think of how to retrieve the original Dexter model schema components from the schema components of HyperCard:

$RETRIEVE_1$

$SPECIFIER$
$HCSPECIFIER$

$direction = hcdirection \lor direction = FROM$
$\quad \lor direction = BIDIRECT$

$hcanchorspec = anchorspec$

$hcpresentspec = presentspec$

$hcdirection = direction$

We can see that retrieval process is trivial except for denoting that in the Dexter model, direction can have value of FROM or BIDIRECT.

A link in the Dexter model is a sequence of specifiers, of which there can be 2 or more (this denotes allowance for multiarity links). At least one of the specifiers must have a TO end-point.

$$
\begin{array}{|l}
\hline
LINK \\
specifiers : \text{seq}\, SPECIFIER \\
\hline
\#specifiers \geq 2 \\
\exists\, s \in \text{ran}\, specifiers \bullet s\ direction = TO \\
\hline
\end{array}
$$

A similar link schema for HyperCard must explicitly denote that links only with arity of 2 are allowed and their end-points are restricted.

$$
\begin{array}{|l}
\hline
HCLINK \\
hcspecifiers : \text{seq}\, SPECIFIER \\
\hline
\#hcspecifiers = 2 \\
hcspecifiers\ 1 \bullet direction = NONE \\
hcspecifers\ 2 \bullet direction = TO \\
\hline
\end{array}
$$

A second retrieve schema would state how the links of the Dexter model and HyperCard are related.

$$
\begin{array}{|l}
\hline
RETRIEVE_2 \\
LINK \\
HCLINK \\
\hline
specifiers = hcspecifiers \\
\hline
\end{array}
$$

We can now define what a component is, given the definitions above and the following types.

$$[ATOM], [HCCARD]$$

$$BASECOMPONENT ::= atom\langle\langle ATOM \rangle\rangle \mid link\langle\langle LINK \rangle\rangle \mid comp\langle\langle seqBASECOMPONENT \rangle\rangle$$

$$HCCOMPONENT ::= atom\langle\langle ATOM \rangle\rangle \mid link\langle\langle LINK \rangle\rangle$$

$$BASECOMPONENT = HCCOMPONENT \cup \{comp\langle\langle b \rangle\rangle \mid b \in BASECOMPONENT\}$$

We can observe that though the definition of a *BASECOMPONENT* in the Dexter model allows for composite components, such recursive types are not allowed in HyperCard.

In the Dexter model additional information relating to a component is called component information and includes attributes, all related anchor identifiers and presentation specifiers. In HyperCard this information is included in the script which in form of several statements corresponding to an event.

$$HCEVENT == \{MouseUp, MouseDown, .., OpenStack\}$$

$$HCSTATS = PRESENTSPEC \cup ATTRIBUTES \cup ANCHOR$$

$$
\begin{array}{|l}
\hline
HCCOMPINFO \\
hcscript : HCSTATS \nrightarrow HCEVENT \\
\hline
\end{array}
$$

The retrieve schema must map these two constructs.

$$
\begin{array}{|l}
\hline
RETRIEVE_3 \\
COMPINFO \\
HCCOMPINFO \\
\hline
\mathrm{dom}\ hcscript = attributes \cup presentspec \cup anchor \\
\hline
\end{array}
$$

The Dexter model Z specification [Halasz and Schwartz 90] defines several functions for readability which we skip here. Operations such as creating a new component are trivially refined by using the fact that there are no composite components in HyperCard. Extending the validation specification to include proof opportunities as explained in the previous subsection is at the moment seen to lengthy and detailed to be worthy. They might be rather worthwhile when developing a new hypertext tool conforming to the Dexter specification.

4 Conclusion

As can be seen from the previous specification, the validation process is quite straightforward in basic cases. We observe the points where the adequacy condition is broken in favor of the generality of the Dexter model, such as no multiarity links or composite structures being allowed. We also observe and document more subtle points such as the discrepancy between the Dexter model and HyperCard, in HyperCard having unique card identities only within a stack whereas this uniqueness is universal in the Dexter model.

As to possible extensions to the Dexter model, we have two points to make. First of all, although scripting is left open as a part of the within-component structure in the Dexter model, some commands in the HyperCard are closely related with other layers of the Dexter model. Anchors are not adequate since these commands do include important static features other than establishing links only. Another extended mechanism may be specified to create such a mapping. Second point is the specification of the message passing hierarchy in HyperCard. This hierarchy is not only limited to within-component layer but also affects other layers. It can, for example, cause creation of a new anchor. We think that this message passing hierarchy should be specified separately. Such a specification may as well be done in an object oriented version of Z, such as Object-Z [Carrington et al. 90].

References

[Comms. ACM 94] Communications of the ACM. Special issue on Hypermedia. 37(2). February 1994.

[Afrati and Koutras 90] F. Afrati and C.D. Koutras. A Hypertext Model Supporting Query Mechanisms. *Proceedings of First European Conference on Hypertext.* A. Rizk et. al (eds). pp. 53–65.

[Carrington et al. 90] D. Carrington, D. Duke, R. Duke, P. King, G. A. Rose and G. Smith. Object-Z: An object-oriented extension to Z. In S.Vueng, ed. *Formal Description Techniques II (FORTE '89)*, North-Holland:1990. pp.281–296.

[Garg 88] P. Garg. An Abstraction Mechanism on Hypertext. *Communications of the ACM.* 31(7). 1988.

[Halasz and Schwartz 90] F. Halasz and M. Shwartz. The Dexter Hypertext Reference Model. *Proceedings of the Hypertext Workshop*, NIST Special Publication 500-178, 1990. pp. 95–133.

[Halasz and Schwartz 94] F. Halasz and M. Shwartz. The Dexter Hypertext Reference Model. *Communications of the ACM.* 37(2). February 1994. pp. 30–39.

[Hitchock and Wang 92] P. Hitchock and B. Wang. Formal Approach to Hypertext System Based On Object-Oriented Database System. *Information and Software Technology.* 34(9). 1992. pp. 573–592.

[Jones 86] C.B. Jones. *Systematic Software Development Using VDM.* Prentice-Hall International, 1986.

[Michel 89] S. Michel. *HyperCard: The Complete Reference.* Osborne Mc-Graw-Hill, 1989.

[Spivey 89] J.M. Spivey. *The Z Notation.* Prentice-Hall International, 1989.

[Woodcock 91] J.C.P. Woodcock. An Introduction to Refinement in Z. In Prehn and Toetenel,eds. *Proceedings of VDM '91: Formal Software Development Methods.* Vol 2. Springer-Verlag, 1991. pp. 96–117.