記　号　処　理　60－1
プログラミング—言語・基礎・実践—　2－1
（1991. 6. 21）

視覚設計に遅延定義を許したハイパーテキストシステム

平井一路

東京工業大学情報工学科

ユーザインタフェースの見え方の部分を定義する際に試行錯誤による労力の損失を極力押えたハイパーテキストシステムを提案する。視覚設計は、実際に作成して自らの目で確かめないと評価しにくいという特徴があるために、構築以前に詳細な仕様を決定することが困難である。結果として、プロトタイプを作成して評価し、変更を加えることが多い。この際の損失を減らすために、クラス定義に似た概念を導入し、クラスの宣言を行ない、インスタンスを作成した後でもクラスに対する変更がインスタンスすべてに伝搬するようなメカニズムを持ち、また、プロトタイプで作成した例をクラスの定義に再利用できるようなハイパーテキストシステムを提案し、その評価を行なう。

KHS :
A Hypertext which allows to define the graphical interface lazily

Kazumichi Hirai
Department of Computer Science Tokyo Institute of Techonology

2-12-1 Ohokayama, Meguro-ku, Tokyo 152, Japan.

We propose a hypertext system that provides an environment where a designer can define the detail of graphical interface through trial-and-errors. For constructiong a graphical interface, a trial-and-error method is essential because the designed interface cannot be really evaluated until it is implemented and used by end-users. Thus, reasonable approach for designing such an interface is to make its protptype, check it, and modify it. Here we propose a hypertext system -KHS- that is suitable for such approach. KHS has the following features that makes easy to follow this approach: after declaring a class and making its instances, the modification on the class will be propagated. And an instance that is implemented as a prototype can be reused in defining the class.

1 Introduction

Today, a hypertext system is getting popular as one of information organizing tools. Comparing with other systems, the graphical interface of hypertext should be more emphasized because it is a tool to control the view of a text. To construct the graphical interface, trial-and-error method is essential because the designed interface cannot be really evaluated until it is implemented and used by its end-users.

In this paper, we propose a hypertext system *KHS* that provides an environment where a designer can define the detail of graphical interface through trial-and-errors. Here we first introduce the notions of "hypertext" and "hypertext system", and explain the background and motivation for proposing our hypertext system KHS. We then give the overview of KHS.

1.1 Hypertext and hypertext system

As one approach to computerized information management and representation, the notion of "hypertext" has been introduced [1][2][3]. The *hypertext* is a form of electronic document. It is an approach to information management and representation in which data is stored in a network of nodes connected together by links. The nodes, and in some systems the network itself, are meant to be viewed through an interactive browser and manipulated through a structured editor. By "the execution of a hypertext", we mean such an extraction of the information in the hypertext. Recently, a demand for processing a non-standard form of data has been increasing. Thus, it is often required that hypertexts can contain, in its nodes, not only text data or source code, but also graphics, audio, video, etc. Such hypertexts are called "hypermedia".

A *hypertext system* is a system for manipulating hypertexts. That is, a hypertext system provides both a developing environment and a executing environment, and it is used to develop hypertexts and execute them. Thus, when talking about a hypertext system, there are essentially two sorts of users: end-users and designers.

End-users are the users who want to get some information from a given hypertext. On the other hand, designers are those who use the hypertext system to design hypertexts; that is, a designer puts some information on his hypertext and wants to convey that information in comprehensible way. It means that the designer corresponds to the programmer in a conventional programming environment, and the end-users do the users of the program written by the programmer. Further more, hypertexts correspond to programs and thus, a hypertext system corresponds to both a programming environment and the programming language. In the following discussions, we use the term *hypertext* to refer a structured information system and use *hypertext system* to refer a system for constructing such information systems That is, designers construct hypertext on a given hypertext system and end-users execute a given hypertext obtain information from a given "hypertext."

After making this difference clearly, when we review the conventional hypertext, we can see that we tend to lay emphasis on interface between end-users and a hypertext as we can see in hypermedia. However, we claim that the interface between designers and a hypertext system is also very important. We can say that hypertext system is a method for a designer to send some information to end-users and the designer is some kind of specialists for a certain information. If a hypertext has a confortable environment to design, many specialists can be its designers. Then that increases the usefulness of the hypertext. Accordingly a hypertext system should be easy to understand and easy to use for those designers. Therefore, it is important for a hypertext system to have a good interface to a designer.

1.2 Designer-friendly hypertext system

What kind of hypertext system is easy to use for hypertext designers? This question should be considered in several aspects: from designing methodology to technical details.

We first point out that the "trial-and-error" construction is essential in constructing systems such as hypertexts, and thus claim that a hypertext system should provide an environment in which designers can achieve trial-and-errors easily.

We can regard a hypertext as a special and advanced form of "computer program". When writing a program (or making a system), we more or less have to do trial-and-errors. Suppose that a system is implemented in C language and that it is specified in details with much care. Then it might be possible to avoid a big modification after testing phases. (Of course, we always need small ones.) However, such detailed specification is almost impossible when the target system is

huge and/or it concerns with user-interfaces like hypertexts. When designing hypertexts, we have to consider how some information should be presented, e.g.,in which way it is displayed more comprehensively. We think that the easiest designing methodology for such systems is to construct some examples, test them, and choose the best one; that is, the trial-and-error construction.

When constructing a system by trial-and-errors, we have to be careful not to waste time and effort so much.

1.3 Features of KHS

In this paper we propose a new hypertext system – KHS – that provides a good development environment for hypertext designers.

Similar to Hyper Card[4][5], a hypertext developed in KHS is fully structured. Its primitive information unit is called a *box*. Information in a hypertext is stored as a collection of these boxes associated each other by linkage. And a hypertext provides some commands (to end-users) to extract necessary information from such a collection of boxes.

As we said, we can regard hypertexts as one type of "program". A hypertext is constructed by a designer and used by an end-user, which corresponds to the process that an ordinary program is made or written by a programmer and executed by its user. Thus, proposing a new hypertext system is similar to proposing both a new programming language and its developing environment. For a new hypertext system, one has to consider the following two aspects: (i) a way to specify hypertexts in the system (like a programming language) and (ii) a machinery by which hypertexts are constructed and used (like a programming environment). In the following we explain these two aspects for introducing our hypertext system, *KHS*.

In KHS, a designer can:

- specify a format, a connection to the other boxes, and available commands for a box in a uniform way,
- describe complex linkages between boxes and specify commands for searching through such linkages.
- use "class" to generate similar boxes,
- define "class" from its instances,
- specify a sequence of commands from monotony operations,
- test a developed hypertext easily,

More precisely, KHS has the following features:

- KHS handles only one object,*box*. From its appearance, a box is just a plain rectangle and all of its own data such as width, height, or dimension is administrated as the property in KHS. From its usage, a box contains properties and is connected by linkage. The linkage between *boxes* is also managed as property. The designer can name the linkage to distinct later. A *box* and its properties are defined on *KHS Editor*.
- KHS introduces the *Abstraction* mechanism. It helps to achieve "try-and-error construction".
- KHS is an "event-driven system" for an end-user. Certain action of end-users causes to trigger the corresponding event(s). To indicate what to do when an event occurs, KHS has the script language named *KTalk*[5]. And to behave as the script describes when an event occurs, KHS has the interpreter *KHS Event Driver*.

In KHS the designer can name the linkage as he wants. KHS has *Abstraction* mechanism to reduce the designer's routine such as making same object. It is similar to the class concept in object-oriented paradigm [6]. But it differs a lot from it.

KHS consists of three parts: *Abstraction Library, KHS Editor,* and *KHS Event Driver*. Both *Abstraction Library* and *KHS Editor* are the programming environment including its library of abstracted objects. and *KHS Event Driver* is the executing environment. *KHS Event Driver* is the place where the end-users use the hypertext.

A designer loads some information on a hypertext or defines the behavior of the hypertext working on *KHS Editor*. Sometimes he works on *KHS Event Driver* to check those events which he defined will be interpreted appropriately responding to the end-users action, then he gets back on the editor and change or correct some.

On the other hand, end-users get the hypertext and execute it on *KHS Event Driver*. They get some information by invoking some events on the hypertext and browsing on it. This is the outline of a lifecycle of a hypertext.

KHS can only handle sequence of character. So KHS is not a multi-media hypertext system. Linkage is managed as a property in KHS. All information about a *box* ,such as size of a *box*, is managed as properties as well as the linkage. All properties of a *box* is displayed on *property table*. A designer can edit the *property table* on *KHS Editor*.

Those are the overview of KHS. We discuss in detail in the following.

2 Requirements for a good hypertext system

In this chapter, we think what is needed for a hypertext system. In the first section, we show a basic function which is important for a hypertext system as an information organizing tool. As we discussed above, the essential character of hypertext system is very abstract. So we think what is demanded for a hypertext system through examples. That is a hypertext which is structured to reflect a paper's logical structure.

In the second section, we simulate how the designer think when he construct a hypertext. Obviously, there is no unique way how a designer construct a certain hypertext. Yet we can think of some common scheme that is often taken by hypertext designers. Here we see such scheme through an example of making hypertexts: *Schedule Chart*. Then we discuss about kind of features required to our hypertext system in this designing schema.

2.1 Schedule Chart

Schedule Chart is a hypertext that keeps and shows one's schedule of each month. In Fig. 1 (a), we illustrates the screen made by Schedule Chart.

From this screen, Schedule Chart may look like a system showing a calendar, but it has the following functions eventually:

- Display the schedule on the day indicated by mouse.
- Edit the schedule at a certain time on a certain day.
- Search free time specified with some method.

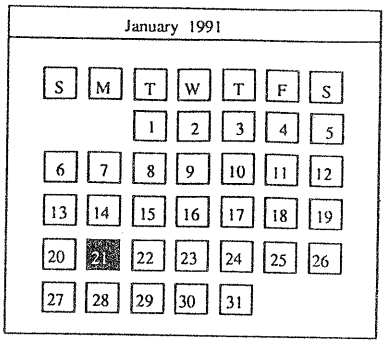
Now let us simulate the outline of the way how one designer would do when constructing this hypertext, Schedule Chart.

(1) First the designer wants to make a frame, *base-sheet*, which is used as a frame of the schedule management chart.

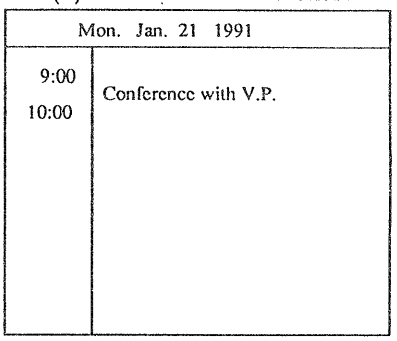
(2) He wants to make rectangles, *day-box*, as many as the days of the month. Each of them is labeled as the date and invokes to show the schedule on the indicated day when clicked by mouse.

(3) He notices that there may be some weekly schedule which depend on the day of the week, such as a meeting from 9 a.m. to 10 a.m. on *every* Monday. And he wants to manage these weekly schedule independently and merge both daily one and weekly one when the schedule of the day is to be displayed. Thus, the system is reorganized accordingly.

At step (2) the designer has to make about 30



(a) Screen of Schedule Chart



(b) Getting a schedule on a day

Figure 1: Schedule Chart

day-boxes. Note that those day-squares are similar; more precisely, they have the same function, but their appearance, i.e.,their position on the base-sheet, and the text, i.e., the date, written on them are different. In a conventional hypertext system such as HyperCard, the designer has to make such similar rectangles with the copy-and-paste operation; that is, he has to do $29 (\pm 1)$ copy-and-paste operations. It is a tedium work for him.

To avoid this routine work, we suggest the following two methods: (i)write a program that generates those day-boxes, or

(ii) define the "class" of day-box and instantiate it as many as the number of the days. In this example, using class concept seems to be better.

Then the designer wants to add new function on the day-frame (3). No matter which way did he select in the previous example(i or ii), he has to modify each of them by his hand or write a program for the work. Since the class has already defined. But if he had noticed the function before he defines the class, he could avoid this waste effort. And this is the best way in constructing

Schedule Chart as a result. Then you may say that he should have been more careful when defining the class. But it is difficult to determine the whole specification as we mentioned above.

Assume that the designer notices all functions that are convenient for Schedule Chart before he defines the class. But there are another factors that are hard to define before making all day-boxes. One of those is the size of them. The size of the day-boxes are to be determined with the balance of all day-boxes. It means the size of them should be fixed after making all of them and evaluating as a set of day-boxes.

3 KHS

In this chapter we propose a new hypertext system – KHS. This system is aimed to solve most of the problems discussed in previous section and provides a better environment for hypertext designers.

3.1 Overview of KHS

KHS is a hypertext system on which designers constructs hypertexts for end-users and also on which end-users execute constructed hypertexts.

We can regard hypertexts as one type of "program". A hypertext is constructed by a designer and used by an end-user, which corresponds to the process that an ordinary program is made or written by a programmer and executed by its user. Thus, proposing a new hypertext system is similar to proposing both a new programming language and its developing environment. For a new hypertext system, one has to consider the following two aspects: (i) a way to specify hypertexts in the system (like a programming language) and (ii) a machinery by which hypertexts are constructed and used (like a programming environment). In the following we explain these two aspects for introducing our hypertext system, *KHS*.

KHS has the following features:

- KHS handles only one object,*box*. From its appearance, a box is just a plain rectangle and all of its own data such as width, height, or dimension is administrated as the property in KHS. From its usage, a box contains properties and is connected by linkage. The linkage between *boxes* is also managed as property. The designer can name the linkage to distinct later. A *box* and its properties are defined on *KHS Editor*.
- KHS introduces the *Abstraction* mechanism. It helps to achieve "try-and-error construction".
- KHS is an "event-driven system" for an end-user. Certain action of end-users causes to trigger the corresponding event(s). To indicate what to do when an event occurs, KHS has the script language named *KTalk*[5]. And to behave as the script describes when an event occurs, KHS has the interpreter *KHS Event Driver*.

Now we think the examples in chapter 2 again. In the example "*Hyper Paper*", the paper is managed in many *boxes*. And the structure of the paper is described by the linkage. Then the end-user can read the paper in the sequential order by tracing the linkage. But in the paper, there may be references to other part or section of the paper. The cross reference is also described by the linkage. The relationship between the boxes connected by the former linkage differs from the one between the ones connected by the latter one. This could be a serious problem when the designer tries to install a searching tool which reflects the structure of the paper. Because if he does not pay attention to the difference of the relationship of connected boxes, he may be embarrassed in a labyrinth of linkage. This problem is solved in KHS by allowing the designers to name the linkage. It means that the former linkage is named "structure-linkage" and the latter "reference-linkage". Then the searching tool looks for a word by tracing only the "structure-linkage".

In the example "*Schedule Chart*", the order of the decision of the specification was the serious problem. It may cause the waste of the designers efforts. *Abstraction* mechanism solves this problem. It is similar to the class concept in object-oriented paradigm [6]. In the class concept, after a programmer defines the class, he instantiates the class. The property of the class is copied by the instance. Applying this concept to construction of hypertext obviously reduces the designers' effort but it can be a serious trouble if the designer wants to change the specification after instantiating the class. *Abstraction* mechanism also instantiate objects but does not copy the property. The instance of an abstracted object has the pointer to the property, instead of copying the value. At the same time, the abstracted object also has the pointer to the instance. So if the designer wants to change all instances of one abstracted object, he just change the abstracted object as he wants. Then the change to the abstracted object is propagated to the instances of it. Now if the designer makes clear what is the same among some object,

there is no fear of wasting his effort.

KHS consists of three parts: *Abstraction Library, KHS Editor,* and *KHS Event Driver. Abstraction Library* manages abstracted objects. Abstracted objects can be regarded as the materials of the hypertext. Designers construct hypertexts by combining these objects properly. Both *Abstraction Library* and *KHS Editor* are the programming environment including its library and *KHS Event Driver* is the executing environment. On the other hand, *KHS Editor* can be regarded as the factory where the material is turned into some products and the hypertext designer as both the planner and the laborer. Then the hypertext is the products which made by the material and *KHS Event Driver* is the place where the end-users use the products. The end-users corresponds to the consumer in this example.

A designer loads some information on a hypertext or defines the behavior of the hypertext working on *KHS Editor.* Sometimes he works on *KHS Event Driver* to check those events which he defined will be interpreted appropriately responding to the end-users action, then he gets back on the editor and change or correct some.

On the other hand, end-users get the hypertext and execute it on *KHS Event Driver.* They get some information by invoking some events on the hypertext and browsing on it. This is the outline of a lifecycle of a hypertext.

Talking about media of KHS, it can only handle sequence of character. So KHS is not a multi media hypertext system. Linkage is managed as a property in KHS. All information about a *box* ,such as size of a *box*, is managed as properties as well as the linkage. All properties of a *box* is displayed on *property table.* A designer can edit the *property table* on *KHS Editor.*

The figure 2 is a scene from a KHS, where a designer is constructing a hypertext. The upper rectangle is a *box* that the designer is now making. And the lower one is the *property table* of the *box.* End-users can not see the *property table* while they are browsing the hypertext. Only the designer can see it through *KHS Editor.* In the *property table*, "*name:*", "*absName*", "*xPos*", etc. are names of the properties. Similarly, "*clickLeft*" is one of properties but differs a little. It is the name of events that this object(*box*) can handle. Then the value of the *clickLeft* stores the script of procedure when the object(*box*) receives the event. It means that if an end-user clicks the left mouse button on this *box*, the event *clickLeft* sends to the *box*, then the contents of the property *clickLeft* is executed by *KHS Event Driver.* The language to describe the script of event is *KTalk*(see section 3.4).

Those are the overview of KHS. We discuss their detail in the following sections.

3.2 Boxes: Objects in KHS

As we mentioned above, KHS has only one sort of object called box. Each box can be connected to the other box by linkage and it can have as much linkage as the designer wants. Any two boxes connected by linkage have partial order. In other words, any linkage is directed. The linkage between two boxes can be devided into two types. These are *include* linkage and *child* linkage. They have difference when the connected box is appeared on the screen. When a box is displayed on a screen, the boxes connected by *include* is also displayed but the ones connected by *child* is not. So designer should instruct the box connected by *child* linkage if he wants to display it. Assume that there is a box named "FATHER" and it is connected with a box named "SON" by *include* linkage and a box named "LOVECHILD" by *child* linkage. When FATHER is displayed on the screen, SON is also displayed on the screen but LOVECHILD is not.

Each data of each object(*box*) in KHS is handled as a property: *name* of the object, *height* of the object, or *width* of the object,*etc.*

The table 1 is list of reserved properties that are arranged by the KHS for each purpose and all *box* has. Of course the designer can declare other properties than these.

The box can have as many properties as the designer wants. In KHS, every property is handled as character so he should be careful to use the property.

Beside this, KHS has event names which are invoked by the end-user with mouse or keyboard. Each script which indicates what to do when an event occurs is also regarded as a property. Assume that there's an event named *clickLeft* and one *box* has to *beep* when it catches the event. This is recognized that property *clickLeft* has the value *beep.* We can say it in other words "If an object gets an event, it executes the value of the event". The table 2 are the part of the events that the end-users can invoke. The designer can define new events but all of them are invoked by some of these events indirectly.

We will explain the language to indicate the action or to define what to do afterward.

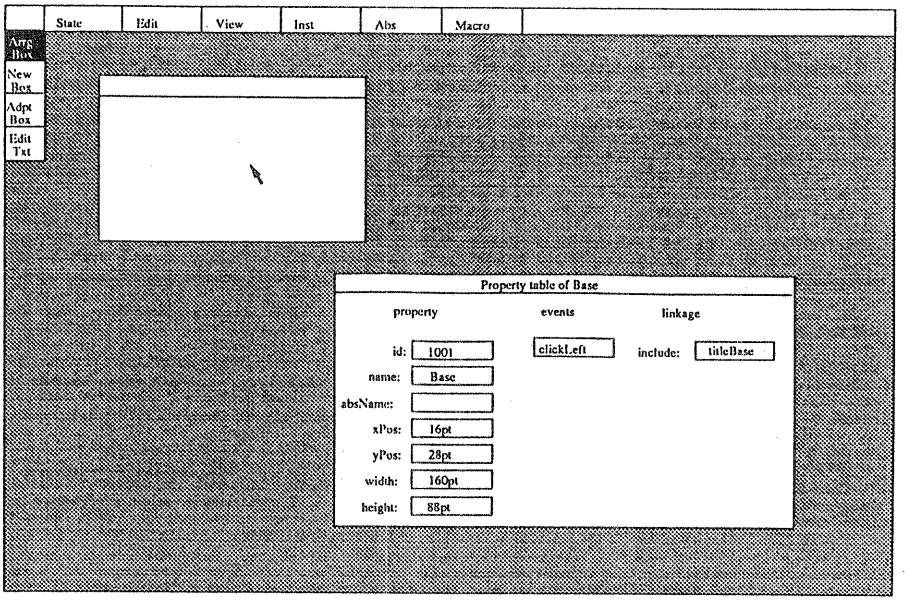


Figure 2: A designer is making a hypertext on KHS

property name	value
id	the ID number of the box, KHS sets this and one can not change this
name	the name of one box, it should be identical in whole system.
xPos	the number of the pixel from left edge of one box's parent to right
yPos	the number of the pixel from top edge of one box's parent to bottom
width	the number of the pixel of one box counted from left to right
height	the number of the pixel of one box counted from top to bottom
text	stream of bite which is displayed on the box.
absName	name of the abstracted box of one box (we will show its detail later)

Table 1: Reserved properties in KHS

event name	context
clickLeft	click the left button of the mouse.
clickCenter	click the center button of the mouse.
clickRight	click the right button of the mouse.
click	clickLeft or clickCenter or clickRight
dragLeft	drag the mouse with the left mouse button down
dragCenter	drag the mouse with the center mouse button down
dragRight	drag the mouse with the right mouse button down
drag	dragLeft or dragCenter or dragRight
doubleClickLeft	click the left mouse button twice in certain short timing
⋮	⋮
Key(control)	type the key "*Control*"
Key(return)	type the key "*Return*"
Key(a)	type the key "*a*"
⋮	⋮

Table 2: The part of events that end-users can invoke on the hypertext on KHS

3.3 Abstracted Boxes : Class Objects in KHS

Abstraction is a mechanism that aims to achieve lazy definition. The basic idea of *Abstraction* is the following:

When a designer constructs a hypertext, it sometimes happens that he knows that some properties are equal but he can not make decision about the value. For instance, think the example 1, there are as many boxes as the number of days in a month. But he wants to decide the dimension of them after making all boxes. That will be nice if he can define that all day-boxes have the same dimension beforehand and change the value afterwards. Let us explain more simple example, assume that the designer needs a pair of boxes which have the same dimension but the different text. In a comprehensive way, he makes one *box* and names it "A". Then he copies the *box*, places it, and names it "B". Both of "A" and "B" have the same width and height. Now, if he wants to change the dimension of "A" and "B", he must change both of them In this example, the value of dimension is managed in the system as expressed in figure 3(a). But, if he could define the property *width* of "B" as same as the width of "A" and there were some mechanism to maintain the consistency of the value, he had only to change the *width* of "A" It means that the width of "B" has the pointer to the width of "A". (see figure 3 (b)).

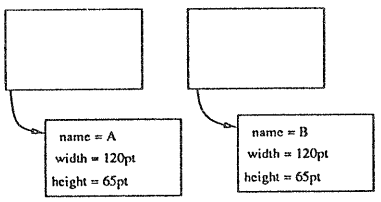


Figure 3: (a) Two independent objects.

Now if he make third and forth ones then can manage the dimension as well as "B". By the way, *box* "A" has the superiority to other *boxes* such as "B". Then we make a new ghost *box* that are superior to all of these *box*. It is *Abstracted box*(see figure 5).

3.4 KTalk : A Script Language

KTalk is the script language on KHS. Writing scripts in KTalk is different from traditional computer programming in many ways, but the most obvious difference is in modularity. The program

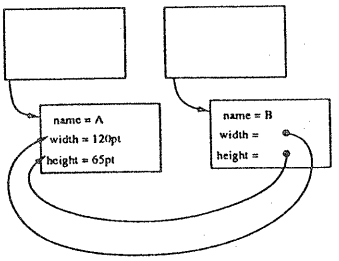


Figure 4: (b) Objects with dependency

listing in languages like Pascal or C can be printed out and viewed like a word processing document. But in KTalk, KHS objects (*boxes*) contain short scripts, which *KHS Event Driver* follows whenever events occur that affect those objects. For example, if an end-user clicks a button(*box*), *KHS Event Driver* looks into the *box*'s script to find out what to do, and operates objects as the *button*'s script.

The designer needs to write scripts only for the actions he wants the *box* to respond to. If he does not want anything special to happen when an event is happened, he does not need to do anything for the *box*.

End-users browse the hypertext invoking events on it. And as the end-users event, the *KHS Event Driver* does the script of the target event of the target object.

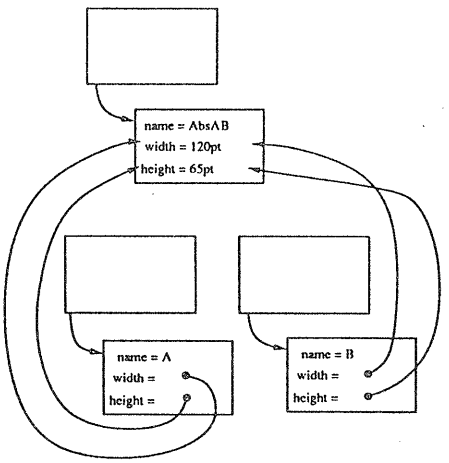


Figure 5: *Abstracted box* of both "A" and "B"

Like any language, KTalk has a vocabulary and rules of syntax. The KTalk vocabulary is similar to English, and so is the rule of syntax.

KTalk can be regarded as the sequence of imperative sentences. The basic syntax of a sentence is like this:

⟨*command*⟩⟨*arguments*⟩⟨*RETURN*⟩

⟨*RETURN*⟩ is the "end of line code". Most simple and frequently used command may be "view+". This command makes the box whose name is same as the argument display on the screen. For example, if the designer wants to display the box whose name is "section1.1" when a box is clicked then he may write the following script as the value of the click.

view+ "section1.1"

As the argument of command view+ the designer can other words to indicate a box, it means it KTalk has other vocabulary to indicate a box than the name. The designer can indicate a box by the following vocabulary.

- "⟨*name*⟩": bounded by double-quote.

  ex. view+ "section1.1"

- ⟨*property*⟩: one of the properties of the box which the event is received.

  ex. view+ child

  When *KHS Event Driver* interpret this script, it look into the value of the property "child" of this box. For this purpose the value of the property "child" should be the name of other box.

- whose ⟨*property*⟩ is ⟨*value*⟩:

  ex. view+ whose id is "1002"

  For this sentence *KHS Event Driver* finds out a box whose property "id" equal 1002.

3.5 KHS Editor and KHS Event Driver

3.5.1 KHS Editor

Almost all command in *KHS Editor* can be invoked by mouse. The designer selects a menu or a tool, or click the mouse to construct the target hypertext[7].

On this editor, the designer definds the *boxes*, and their properties. Some of these properties, like size or coordinates to the parent *box*, are put by the editor automatically. The designer can also put the value directly to any property exept *id*.

To define the *box*, the desiger may only choose a tool and drag the mouse pointer from left top point to right bottom point of the target *box*. Then the property *xPos, yPos, width, height* are calculated by the editor and the proper values are put in the property.

3.5.2 KHS Event Driver

KHS Event Driver is the interpreter on KHS. It interprets the script language *KTalk* one line by one.

KHS Event Driver watches the action of end-users with mouse or keyboard. And when an end-user push down the mouse button or hit the key on keyboard, it checks where the mouse pointer is and determine the target to the event. Then it looks into the target object(*box*) and look into the property(script). If the object has the corresponding property(script), *KHS Event Driver* execute the value of the property as the script responding to the event. If not, the event is passed to the parent of the target *box*. This process is repeated until it reaches the *root box* of the hypertext.. If the *root box* has no correspoinding property, it do nothing.

4 Conclusion

4.1 Summary

We proposed a new hypertext system that aims to decrease the designers' work and efforts when they construct hypertexts. We extracted the requirements for the hypertext at the view point of designers. At that point of view, the following features are needed to the hypertext system.

- It should be easy to make similar objects.
- It should be easy to change after making similar several objects. (It allow the designer to construct with trial-and-errors construction)
- It should be easy to make temporary prototype and the products should be reused in the final products as much as possible.
- It should define any complicated linkages between objects.
- It should give some mechanism to the designers to choose linkage from the script.
- It should offer a script language which is properly easy to understand and has as many functions as possible to access the internal data of objects.

According to the requirements above, we proposed KHS which consists of the following items.

- *KHS Editor* with abstraction mechanism and its library
- *KHS Event Driver* with its script language *KTalk*

4.2 Direction for future research

In this thesis, we determine the media that we manage as the sequence of characters. First we said we should pay much emphasis on the interface to designers, but it can be said that the interface to the end-users is good enough to put and display the necessary information on hypertext. Thus, we may not discuss about the hypertext without any idea of multi-media. Then we should expand our system to the one which handle multimedia such as music or animation, etc.

As the directions for future research, we ought to implement all of *KHS* and evaluate the interface to the designer in a practical use. And we ought to introduce other media than sequence of characters.

Acknowledgements

I wish to express my sincere thanks to all who have helped me to produce this paper. First of all, I would like to express my best appreciation to my supervisor Associate Professor Osamu Watanabe. Works presented in this material was not possible without intensive discussions with him, nor his advises.

I would like to acknowledge members of the Watanabe laboratory to discuss about my research activity in spite of the lack of common topics to theirs. The members of Katayama, Yonezaki, Tokuda laboratory also helped me to manage to write this paper.

I would also like to thank all the people who has encouraged me during my writing.

References

[1] Bush V. As we may think. In *Atlantic Monthly 176*, pages 101–108, July 1945.

[2] Stephen F. Weiss John B. Smith. HyperText. In *Communications of the ACM*, july 1988.

[3] Frank G.Halasz. Reflections on notecards:seven issues for the next generation of hypermedia systems. *COMMUNICATIONS of the ACM*, 1988.

[4] Danny Goodman. *THE COMPLETE HYPERCARD HANDBOOK.* Bantam Books, 1987.

[5] Danny Goodman. *THE COMPLETE HYPERCARD2.0 HANDBOOK.* Bantam Books, 1990.

[6] A. Goldberg and D. Robson. *Smalltalk-80: the language and its implementation.* Addison-Wesley, 1983.

[7] Shi-Kuo Chang. *VISUAL LANGUAGES AND VISUAL PROGRAMMING.* PLENUM PUBLISHING CORP., 1990.