計算機アーキテクチャ 77－4
（１９８９．７．１３）

IntelligentPad
アクティブ･メディア･オブジェクトの合成と管理のための ツールキット･システム

田中 譲、今滝隆元、長崎 祥
北海道大学工学部電気工学科

IntelligentPadはアクティブ･メディア･オブジェクトの蓄積と視覚的管理のためのツールキットである。このシステムでは、すべての物は紙として表現される。各紙はワードプロセシング、作図、作表などの機能を持つ。機能の違いにより異なる紙が用意される。紙を別の紙に貼ることにより、項目欄のレイアウト設計ができるとともに、これらの紙の機能が合成される。IntelligentPadは多数の紙を管理するのに、カタログ、ハイパーメディア･ネットワーク、フォームベース、パッドベースの4種類の方法を提供する。フォームベースは同一フォーマットの紙を蓄積するのに対し、パッドベースはすべてのフォーマットの紙を蓄積することができる。

IntelligentPad
A Tool Kit System for the Synthesis and the Management of Active Media Objects

Yuzuru Tanaka, Takamoto Imataki, Akira Nagasaki
Electrical Engineering Department
Hokkaido University
Sapporo, 060 Japan

IntelligentPad is a tool kit for the storage and visual management of active media objects. It represents everything as a pad. It associates each pad with a function such as word processing, line drawing, tabulation, graph drawing etc. Different functions define different pads. Pasting of pads on another pad defines a new pad that has both an arbitrary layout of fields and a new function composed of the constituent pads. IntelligentPad provides four ways of managing a large amount of pads, i.e., visual catalogs of pads, hypermedia networks, form bases, and pad bases. A form base stores pads with the same format, while a pad base manages all types of pads in the system.

1. INTRODUCTION

Some psychologists believe that a shape of a cup affords us how to handle it. This effect is referred to by 'affordance' [1]. We believe that formats of documents and the shape of deskwork tools also afford us how to manipulate and manage them. Conventional database systems, however, only deal with normalized tables. Their restricted representation structures and operations prevent their advanced applications to CAD/CAM, CAE, OA, and CAI. Recently, we see many proposals on various extensions of database concepts [2]. Among them are semantic databases, unnormalized databases, abstract-data-type databases, object oriented databases, and deductive databases. However, we still lack a general paradigm that associates each object not only with its operations but also with its visual form of presentation.

Some proposals have partially implemented such a paradigm. Office-By-Example (OBE) [3], developed as an extension of QBE (Query-By-Example), implemented this paradigm in OA by providing a relational database with good interfacing tools to business graphs, business letters, and electronic mails. FORMANGER [4] and FORMAL [5] are form base systems that provide visual definition, visual presentation, visual management, and visual retrieval of document forms. Media Room of MIT Media Lab. is a forerunner in the spatial management of data [6]. It hierarchically organizes related documents, and spatially arranges their access points at each level to map them on a visual planar space such as a picture, a diagram, a map, or a floor plan. These visual management systems, however, are not capable of visually defining new document processing or representing tools.

Visual definition of tools requires capabilities of visual program construction. Studies on visual system definitions are classified into three categories, i.e., (1) visual coaching, (2) visual programming, and (3) visual management [7]. Visual coaching constructs a program through showing what we want to do. Visual programming systems are further classified into two groups. Diagrammatic systems accept diagrammatic specifications of programs, while iconic systems uses icons for program composition. Visual coaching and visual programming systems are, however, concerned with visual synthesis of information processing mechanisms rather than with visual representation and manipulation of information itself.

Besides table and form systems like OBE, visual management systems include multimedia database systems [8, 9], and hypertext/hypermedia systems like NoteCards (Xerox), HyperCard (Apple), and Intermedia [10]. The concept of hypertext and/or hypermedia systems originated in V. Bush's paper published in 1945 [11]. D. C. Engelbart developed a prototype system in 1963 along this line [12]. T. H. Nelson was the first who used the word 'hypertext' [13]. These research results are mainly inherited by the researchers at XEROX PARC. Their primary concern was the development of a personal computer and an active media system running on it. Among them, A. Kay proposed Dynabook in 1977 [14]. Smalltalk-80 was developed as a language to implement active media systems.

Hypermedia systems span networks among documents by relating an item in a document to another document, and allow users to navigate the sea of documents along these links. They lack sufficient facilities, however, for the visual synthesis of media objects. Current programmable hypermedia systems use object-oriented languages for the media synthesis programming. NoteCards has Interlisp D, while HyperCard has Hypertalk. While Smalltalk-80 has introduced a lot of concepts and methods in the synthesis of active media, it lacks a front-end hypermedia system.

IntelligentPad is a visual management system based on the object orientation paradigm. It associates each object not only with its operations but also with its physical form. It is also a front-end hypermedia system for Smalltalk-80. IntelligentPad represents everything as a pad. Pads are all persistent, i.e., they continue to exist in the system unless deleted. Different pads are associated with different functions such as word processing, image editing, line drawing, tabulation, graph drawing etc. Pasting of pads on another pad defines a new pad that has both an arbitrary layout of fields and a new function composed of the constituent pads. It also allows us to overlap images given by different pads. IntelligentPad initially provides a set of primitive pads. Its users can define new pads by pasting existing pads together. Different from the previous hypermedia systems, it provides us with direct visual definition methods both for the propagation of operations among different media objects and for the functional composition of a new media object from existing ones.

The idea of using the metaphor of pasting pads originated in the Trillium system developed by A. Henderson in 1986 [15]. The Trillium system was, however, designed to rapidly prototype controls of photocopy machines, rather than large interactive systems. The pad metaphor was applied only to a restricted small set of objects with restricted functions. IntelligentPad, however, extensively applies this metaphor to a large variety of active media objects.

Pads can be bound to a book. A book is a pad with paging functions, a content page, and index pages. It works as a catalog used for browsing pads. A selection of an arbitrary item on the content page or the index pages opens the desired page. In CAD/CAM, CAE, OA, and CAI, we have to deal with various document types. Among them, we have text, drawings, images, forms in various formats, business graphs and charts, tables, memos and notes. They are all pads. A lot of deskwork tools also may be represented as intelligent pads. Some of them are scales, drawing templates, calculators, phones, computer displays, computer keyboards, video TV screens, meters, console panels, calenders, blackboards, cabinets, and book shelves.

IntelligentPad manages pads by providing four different ways of managing a large amount of pads, i.e., visual catalog of pads, hypermedia networks, form bases, and pad bases. A form base stores pads with the same format, while a pad base manages all types of pads in the system. A pad base is a database whose records are pads. Queries to a pad base are also specified by pasting pads.

Section 2 gives the design philosophy of IntelligentPad, while Section 3 describes its architecture. Section 4 describes some example pad compositions including a composition of a form base. Section 5 shows two more different ways of pad management, i.e., hypermedia networks and pad bases. Section 6 concludes this paper and shows some of our future research plans.

2. DESIGN PHILOSOPHY OF IntelligentPad

2.1 MVC Modeling and Active Media

IntelligentPad deals with two dimensional objects with pad forms. In IntelligentPad, we are concerned only with documents and deskwork tools. These objects are virtually two dimensional, since the arrangement of their access points such as buttons, levers, and books on shelves is necessarily planar in their user's view.

We believe that the modeling of an object needs to include two components, i.e., (1) its internal mechanism, and (2) its external mode of existence. The description of the internal mechanism consists of its state and its operation, while the description of the external mode of existence includes its appearance and its reaction. By its reaction is referred to its action against the outer stimuli.

In programming languages, data abstraction corresponds to the modeling. Historically, data types were first introduced

to describe the states of objects. Then, the introduction of abstract data types has enabled us to describe the state and operation of each object. These schemes describe only the internal mechanism of each object. The MVC triple in Smalltalk-80 has first enabled us to describe, for each object, not only its internal mechanism but also its external mode of existence. The model part describes the internal mechanism. The view part defines the appearance, while the controller part specifies the reaction.

Objects may have unobservable components of their states. The modeling of an object needs to distinguish the observable part from the unobservable part of its state. In this paper, the observable part of an object state is called the observable of the object.

2.2 Design Philosophy

IntelligentPad was designed based on the following three principles.

1. Everything is a pad.
2. Every pad is persistent.
3. Each pad has its observable value.
4. Each pad is associated with a function.
5. Each pad can be pasted on another pad.

In the object-orientation paradigm, everything is represented as an object with a state and a set of acceptable messages. In IntelligentPad, everything is represented as a pad, i.e., an object with a pad form. Each pad has not only its state and acceptable messages, but also a display view and a set of direct manipulations on this view. Pads are persistent in the system, i.e., they continue to exist in the system unless we delete them. Each pad has the observable part of its state. The value of this part is called the output of the pad.

Functional differences define various kinds of pads. Some of them are text pads with word processing functions, drawing pads with line-drawing functions, graph pads to represent a given data set in business graphs, n digit display pads for digital display of a numerical value, meter pads for analog display of a numerical value, switch pads for a binary value input, button pads for command sending, table pads for tabular representation and calculation of numbers, and binder pads to bind various pads together. These are all primitive pads initially provided by the system. Functions of these pads will be detailed in later sections.

These primitive pads have the following observable values. A text pad has as its observable the text string written on it. In case of a drawing pad, its observable is a list of the instances of geometrical primitives. A graph pad has as its observable a tabular representation of numerical values shown in the graph. Both a digital display pad and an analog meter have as their observable values a single numerical value displayed on these devices.

A paste operation gives a layout and a functional composition. For example, a personnel form consists of a photograph of a person, his name, his present address, his permanent address, his education history, and his job history. In IntelligentPad, we may associate each part of this form with a text pad or an image pad. We can make a personnel form by pasting these entry pads on a white form pad. Similarly, a calculator also has a layout of display board and buttons. We may also consider these parts as pads. The base pad used in this case has calculation functions. By pasting these pads on a calculation pad, we can implement a calculator on the display screen.

A pad becomes a subpad when pasted on another pad. Subpads are dependents of the pad on which they are pasted. A subpad becomes a pad when it is peeled off and placed on the desktop, namely on the screen background.

2.3 Pad Operations

IntelligentPad defines the following pad operations:

1. Open/close of a pad
2. Property change of a pad
3. Registration and Naming of a pad
4. Copying/deleting of a pad
5. Moving of a pad
6. Save/load of a pad
7. Paste/peel of a pad
8. Tree representation of a composition structure
9. Printing of a pad

Each closed pad has an iconic representation. We can design for each pad an arbitrary icon of an arbitrary size and an arbitrary painting. An icon may be transparent. A pad is alive even if it is closed. Closed pads also receive messages to change their states. A click on an icon opens it. Each pad has a property sheet on which we can select its properties from various property menus. Among its properties are its size, the shape of its boarder, and its shading and pattern. If its shading is specified transparent, the pad becomes transparent to show only the texts and pictures on it. We can not change the property of subpads.

Each pad has its identification code, its registration number, and its nickname. When a pad is created, the system gives it a unique identification code. We can register any pad to the system at any time. The registration of a pad gives it a unique registration number. System-defined pads are all registered pads. We can give any pad an arbitrary nickname at any time. Each pad keeps the latest nickname. When a pad becomes a subpad, it keeps its identification code, its registration number, and its nickname. We can make copies of any pad and any subpad. A nonshared copy can be used independently from its original., while shared copies of the same pad share the same state. A nonshared copy is automatically given a new identification code, while it inherits the original's registration number and nickname. Shared copies of the same pad share the same identification code, registration number and nickname with their original pad. A copy of a subpad becomes a pad with all the subpads pasted over it. The deleting of a pad deletes this pad together with all the subpads pasted over this pad.

We can move any pad to an arbitrary location on the screen. We can save and load any pad to and from a secondary storage device. We can paste a pad on another pad, and peel a subpad off a pad. These operations are also applicable to closed pads. The paste location can be arbitrarily chosen if the left top corner of the subpad lies in the underlying pad. A peeled subpad becomes a pad with a new identification code. It keeps its state before the separation, but breaks the functional composition with the other pad. Once a closed pad P_2 is pasted on P_1, the dependency between them is kept alive even after we open P_2. We may open P_2 at any location on the screen. Since closed pads may be transparent and arbitrarily large, the pasting of closed pads allow us to span navigation networks among pads.

For each pad, we can ask the system to show its composition structure as a tree. If the pad is primitive, the tree has only a root node with the nickname of this pad. Otherwise, the root of a tree shows the nickname of the base pad. Each node of the tree corresponds to a subpad over the base pad, and shows the nickname of this subpad. For a node corresponding to a subpad P, its sun nodes correspond to the subpads pasted directly on P. A mouse click at an arbitrary node of the tree selects its corresponding subpad. This enables us to select, from a composite pad, one of its subpads that might be hidden under another subpad.

We can print any pad on a sheet of paper.

3. IntelligentPad ARCHITECTURE

3.1 A Pad as an MVC triple

IntelligentPad was programmed in Smalltalk-80. Each pad was implemented as an MVC triple, where M, V, and C respectively stand for a model, a view and a controller. An MVC construct is a programming style developed in the Smalltalk-80 culture, though its concept might apply to other interpretive object-oriented languages as well.

For each pad, its three components play the following roles:

(1) Its model defines its state and behaviors.

(2) Its view defines how it looks on the screen.

(3) Its controller defines its mouse-interactions on the screen.

We have extended the class View in Smalltalk-80 to define its new subclasses. Each pad's view is implemented as an instance object of one of these subclasses.

In its MVC construct, each pad has an update-propagation dependency from M to V as well as message-sending paths from C to V and V to M. A state change of M automatically updates V. A message-sending path from an object O_1 to another object O_2 is denoted by $O_1 \rightarrow O_2$, while an update propagation dependency from O_1 to O_2 is represented by $O_1 \rightsquigarrow O_2$. These are two types of links between two objects.

IntelligentPad assumes that no object may have either message-sending links to more than one object or update propagation dependencies from more than one object. The first restriction simplifies the implementation of a message-sending link. We may provide each object with a reference pointer to another object. An update propagation from one object O_0 to other objects $O_1, O_2, \ldots, O_n$ is implemented as follows. We provide the object O_0 with a list $\{O_1, O_2, \ldots, O_n\}$ of its dependents. When the object O_0 changes its state, it also sends itself an **changed** message. This makes its receiver send another standard message **update** to all of its dependents.

3.2 Interrelations among Pads

Between two pads, we may consider two kinds of dependencies, i.e., an update dependency and a reference dependency. An update dependency $P_1 - \mathrm{u} \rightarrow P_2$ defines an automatic update propagation from a pad P_1 to another pad P_2, while a reference dependency $P_1 - \mathrm{r} \rightarrow P_2$ defines an access path through which P_1 accesses P_2. These two dependencies are respectively implemented by an update propagation link and a reference link from the view of P_1 to the view of P_2. As a special case, it is obvious that both $P - \mathrm{u} \rightarrow P$ and $P - \mathrm{r} \rightarrow P$ hold for any pad P.

IntelligentPad allows the following three combinations of dependencies between two pads.

(1) Both $P_1 - \mathrm{u} \rightarrow P_2$ and $P_2 - \mathrm{r} \rightarrow P_1$ hold.

(2) Only $P_2 - \mathrm{r} \rightarrow P_1$ holds.

(3) No dependency holds.

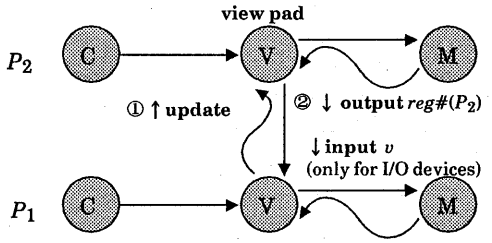
We can consistently define a relation $P_1 \mid\!\rightarrow P_2$ as

$P_1 \mid\!\rightarrow P_2$ iff $P_1 - \mathrm{u} \rightarrow P_2$ or $P_2 - \mathrm{r} \rightarrow P_1$.

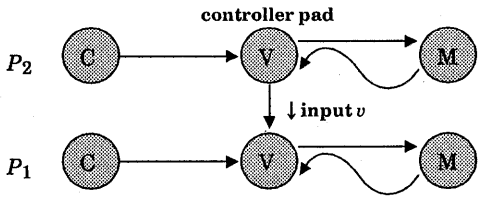
We say that there exists a pad dependency from P_1 to P_2 if $P_1 \mid\!\rightarrow P_2$ holds. Here, P_1 is the master and P_2 is its slave. As a design constraint, we impose antisymmetry on this relation. The pad dependency is a partial order relation.

In IntelligentPad, we can set a pad dependency from P_1 to P_2 by pasting P_2 on P_1. Each pad is classified into one of the three categories depending on the dependency set by its pasting on another pad. A controller pad, when pasted on another pad, sets the second combination of dependencies, and works as an input device of its master. A view pad, when pasted on another pad, sets the first combination of dependencies, and works as either an I/O device or an output device of its master pad. Pads that are neither controller pads nor view pads are model pads. When pasted on another pad, a model pad establishes no links to this pad. It is just pasted without any functional composition. IntelligentPad has MVC structures not only inside each pad but also among pads.

In IntelligentPad, pads of the same category have the same interface to their master pads. Every controller pad sends an **input** message with a parameter value to its master pad. Every pad sends all of its update-dependent pads an **update** message without any parameter. An **update** message to a view pad makes it send back an **output** message to its master pad to read its observable or a component of its observable as an output value from it. The **output** message is accompanied by the registration number of the sender pad. These messages define standard interfaces between pads (Fig. 3.1).



(a) A view pad P_2 works as an I/O device or an output device of P_1. (**output** is used to read out an output value from the master pad, while **input** v is used to send an input value.)



(b) A controller pad P_2 works as an input device of P_1.

Figure 3.1 Standard message interfaces between pads.

A pad may have more than one output value, i.e., its observable may have more than one component to read. When we try to paste a view pad on such a multioutput pad, it shows a list of its possible output items for us to choose one of them. The selected item is paired with the registration number of the pasted view pad, and stored in the multioutput pad. This mechanism allows us to paste multiple view pads on the same multioutput pad to display different output items simultaneously.

3.3 Primitive Pads and Their Categorization

New pads can be defined either by Smalltalk-80 programs or by compositions of existing pads. IntelligentPad causes no interface mismatch. Pads communicate with each other through the three standard messages, i.e., **input** v, **update**, and **output** *reg#*. Every pad accepts these three messages. The two messages **input** v and **output** *reg#* sent to a pad P are forwarded to its master pad if their methods are not defined in P. Similarly, an update message to a pad may be forwarded to its slave pads.

Fig. 3.2 shows some categories of basic pads. Pads in the same group are replaceable with each other. Replacement of a pad with one of the same group gives the same information in a different representation. A digital display pad and an analog meter are examples of mutually replaceable pads.

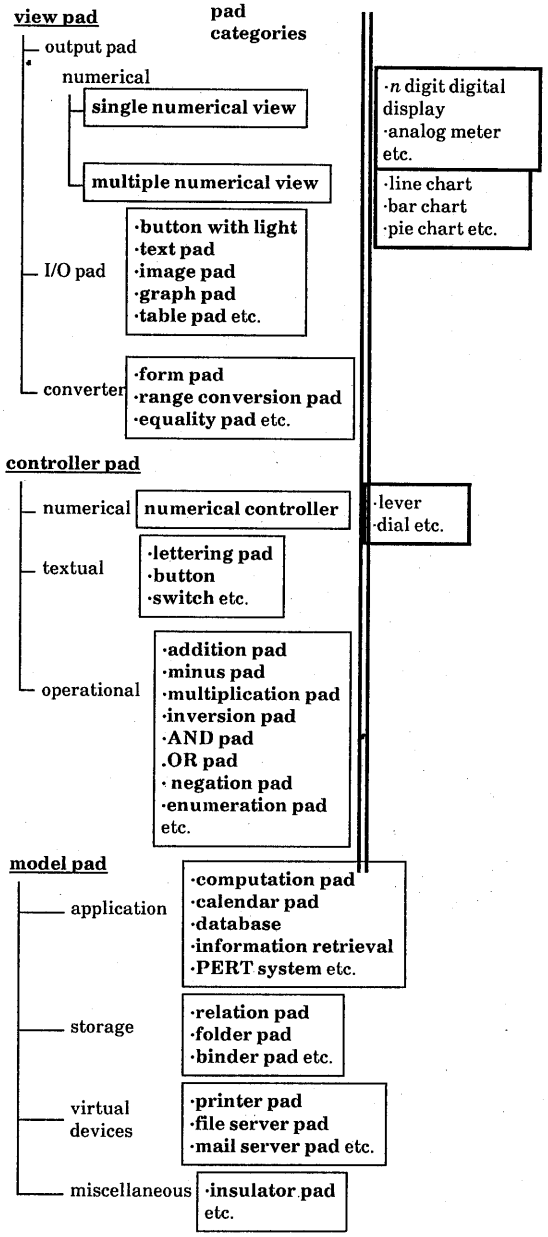


Figure 3.2 Some categories of basic pads.

A binder pad is used to bind pads together into a book. An insulator pad is inserted between two pads to break the dependencies between them. It is also used as a base pad on which we may put various pads without setting any pad dependencies from the base pad to them. Each page of a virgin white binder pad is an insulator pad.

4. EXAMPLE PAD COMPOSITIONS

4.1 A Composition of a Hand Calculator

In IntelligentPad, we can build a hand calculator by pasting several kinds of primitive pads on a computation pad as shown in Fig. 4.1. We first paste a digital display pad D, a switch pad S, and a set of button pads $\{B_i\}$ on a computation pad C, and then paste a set of lettering pads $\{L_i\}$ on the button pads, specifying each of them to be either number buttons or

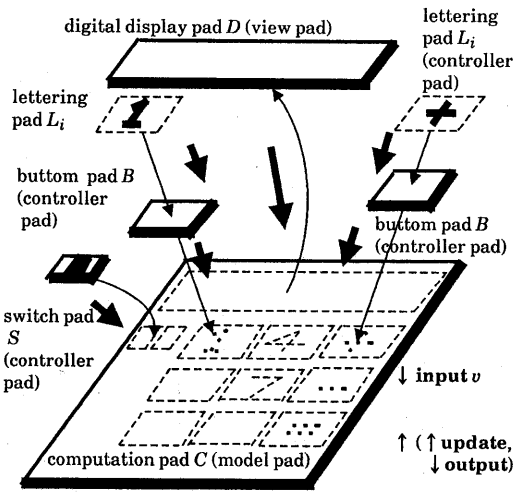


Figure 4.1 A composition of a hand calculator.

operator buttons. A computation pad represents the logic circuits of the calculator and the case to hold them. Its model sequentially receives a mathematical expression on numbers to calculate the result.

A switch pad, when pasted on another pad, resets its state to 'OFF'. When it is clicked, its state alternates between 'ON' and 'OFF'. The changed state is sent to its master pad C by an **input** message. The computation pad C enables its computation when receiving an **input** ON message, and disables it when receiving an **input** OFF message. A lettering pad for a numeral i, when clicked, sends a message **input** with i to its master pad B. The button pad B passes this to its master pad C. Similarly, a lettering pad for an operator "∘" sends this operator "∘" to the pad C through B. A computation pad C interprets the sequentially sent numerals and operators, and changes its state to the intermediate result of the computation. When C changes its state, this event is reported to its view pads through the update propagation link. In this example, the pad C has only one view pad D. The digital display pad D, when having received an **update** message, reads out the intermediate computation result of C by sending back a request message **output** *reg#*(D) to C. It displays the read out value on its view.

The digital display pad may be replaced with an analog meter pad. Intermediate computation results are then displayed on this meter. While a primitive meter has no range selection mechanism, we can easily attach this mechanism by providing a range selector pad under the meter (Fig. 4.2). It is a view pad that works as a master pad of the meter. Its model stores the current maximum scale. It consists of a base pad and several button pads. A new maximum scale is input by one of the button pads. Different button pads are associated with different maximum scales. When clicked, a button pad sends its maximum scale value to the base pad. When a calculator pad sends an **update** message to the analog meter, the range selector pad sends back an **output** message to read the output value. The range selector pad calibrates this value with respect to the current maximum scale, and keeps the result as its output. Then it sends an **update** to its master, i.e., the meter pad. This makes the meter pad read out the output of the range selector pad. The read out value is then displayed on the meter pad. We can also easily design an automatic range selector pad. These manual and automatic range selectors are applicable to any kind of analog meter pads. These examples show the highly generic modularity of each pad.

4.2 A Composition of a Chart Board

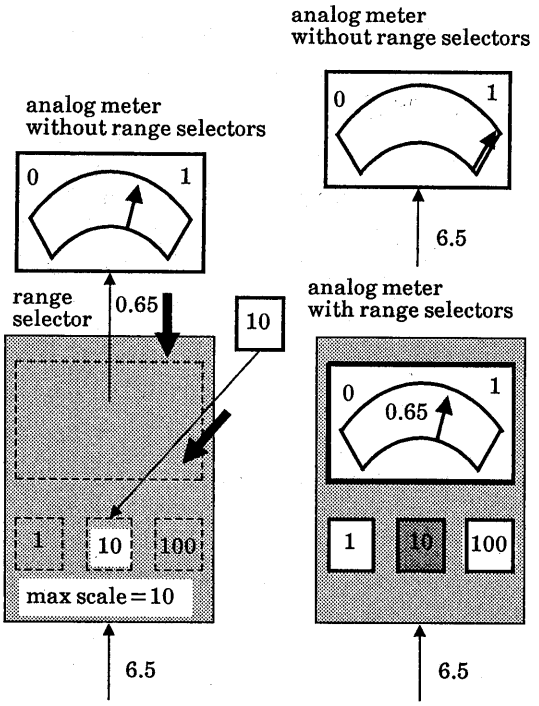


Figure 4.2 An analog meter and a range selector.

We show in Fig.4.3 a hard copy of the display. On its lower right area, it shows a composed hand calculator pad. On its

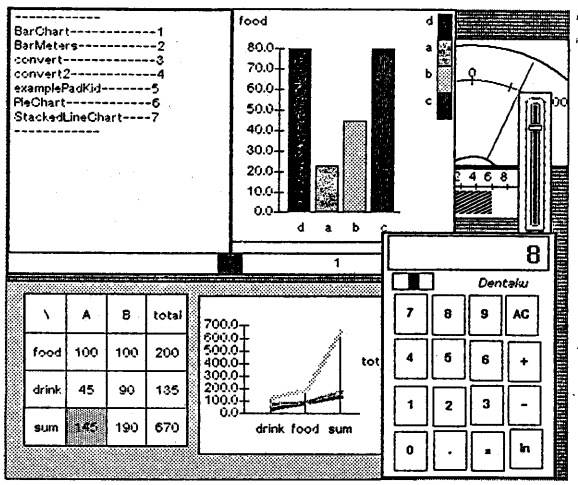


Figure 4.3 A hard copy of the display where we composed a hand calculator.

upper left area, the figure shows a book pad. The book shows its content page and its first page holding a bar-chart pad. On its upper right area, the figure shows two analog meter pads and a lever pad.

This figure also shows another example of a composite pad. This composite pad works as a line-chart board that is capable of presenting a numerical table in a line chart. It consists of a numerical relation pad, a numerical table pad, and a line chart pad. A numerical table pad gives a tabular view. Each column is a tuple of numerical values, while each row represents an attribute. The leftmost column gives attribute names, while the top row gives a name to each tuple. A numerical relation pad has a blank pad view and a model capable of storing a numerical relation. When pasted on a numerical relation pad, the table pad visualizes the relation stored in the relation pad. We can transpose the table if necessary. The table pad also works as a data entry device for the relation pad. When we insert a value into some table entry on the table pad, this pad sends this value to the numerical relation pad to update its model. When the relation in the relation pad changes, the table pad immediately reflects this change on its view.

A line chart pad gives a line-chart representation of the data set stored in its master pad. When pasted on a numerical relation pad, it displays, for each tuple, connected line segments showing each attribute value. Its X coordinate represents a set of attributes, while its Y coordinate measures numerical values. Instead of a line chart pad, we may paste a bar chart pad, a pie chart pad, or more than one of these chart pads. Chart pads pasted on the same pad will simultaneously present the same data in different chart forms. After we fill in the table, we automatically obtain its graphical presentation in the chart pad. If we want, we can freeze the chart pad, peel it off, and move it to any other place. Freezing allows us to keep the current state of the pad.

4.3 Composition of a Form Base Pad

A form base pad is also worth mentioning. It allows us to design a specially formatted form through pasting text and picture pads on it. Its model stores a collection of key-record pairs. When a pad is pasted on a form base pad, its registration number and its contents are paired and added to this collection in the form base pad. The registration number works as the key for its paired value. Fig. 4.4 shows this mechanism. Paste operations give an arbitrary layout of the

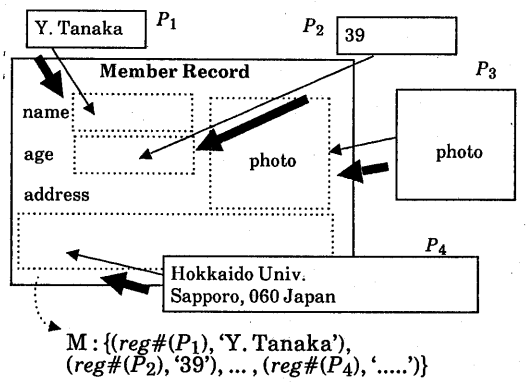


Figure 4.4 A composition of a form.

entries. A mouse click selects an entry to fill in. The selected entry pad receives the following keyboard inputs until it receives a return code. Then it sends **input** to the form base pad to insert a pair of its registration number and the input text into the collection in the form base pad.

Pads in the file category represent files. They are called folders. Printers, floppy disks, network servers, and disk files are all considered as files. Folders are model pads. If we place an icon of a pad over a folder icon, this pad is sent from the desktop to the corresponding file.

Among various folders, we have a form folder. This is a disk file that stores forms of the same type. We can specify the associated form type by pasting a registered form pad on the form folder pad. Suppose that this form pad has n entry item pads $P_1, P_2, \ldots, P_n$ on it. This form folder stores forms $\{F_i\}$ with these entry item pads. Forms of other types would be rejected to store in this folder. A form F_i is stored as a

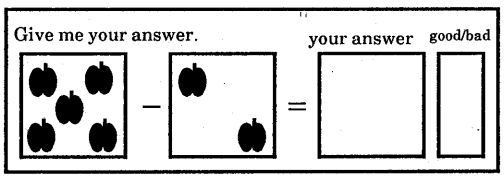
collection $\{(reg\#(P_1), v_{i1}), (reg\#(P_2), v_{i2}), ..., (reg\#(P_n), v_{in})\}$, where v_{ij} is the contents of the P_j pad on the form pad F_i. A form pad, when clicked, sends its master pad its association list $\{(reg\#(P_1), v_1), (reg\#(P_2), v_2), ..., (reg\#(P_n), v_n)\}$ by an **input** message. A form folder pad, when sent an association list $\{(reg\#(P_1), v_1), (reg\#(P_2), v_2), ..., (reg\#(P_n), v_n)\}$, searches the set $\{F_i\}$ of stored forms for such a form F_i in which, for any j satisfying $v_j \neq nil$, the entry pad stores the same value as v_j. It keeps the searched list $\{(reg\#(P_1), v_{i1}), (reg\#(P_2), v_{i2}), ..., (reg\#(P_n), v_{in})\}$ as its output, and sends its slave view pad an **update**. Then the form pad pasted on the form folder reads out this searched list. It keeps this list as its output and sends an **update** to its slave view pads. Each entry pad P_j sends back **output** $reg\#(P_j)$ to the form pad. When receiving **output** $reg\#(P_j)$, the form pad searches its output list $\{(reg\#(P_1), v_{i1}), (reg\#(P_2), v_{i2}), ..., (reg\#(P_n), v_{in})\}$ for the pair $(reg\#(P_j), v_{ij})$ with a keyword equal to the parameter of the output message. It returns the value v_{ij} to the entry pad P_j, which displays this value v_{ij}.

This process implements a mechanism that searches a set of stored forms for the one having, for every specified field, the same value as the form shown on the folder pad. To retrieve a specific form from the form folder, we need only to specify the retrieval condition on the form pasted on the folder pad. A mouse click on the specifying form starts the search process. The retrieved form is displayed on the same form pad. We can make a copy of this form pad to use it elsewhere.

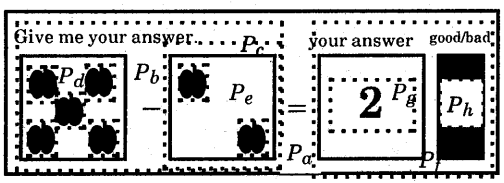
A form folder with a form pad pasted on it, thus, naturally implements a form base system in the IntelligentPad paradigm. It allows us to use any pads as parts of a form. Thus, me may define, for example, a form with a meter as one of its parts to implement a form base for such forms.

4.4 Arithmetic Card

IntelligentPad has also potential applicability to CAI systems. Let us consider, for example, an implementation of a simple educational card with an arithmetic question as shown in Fig. 4.5 (a). This card is a self-learning tool that



(a) arith. card teaching subtraction



(a) arith. card teaching subtraction

Figure 4.5 An arithetic card and its composition.

teaches subtraction. It requests a pupil to give a numerical answer to a subtraction of two apples from five apples. If he gives a correct answer, he will get a circle. Otherwise, he will get a black mark.

In IntelligentPad, this card may be implemented as shown in Fig. 4.5 (b). It consists of 15 pads including 7 apples. The pad P_a is a model pad with no function but storing an observable value. The controller pad P_b is an addition pad that calculate the sum of its subpads' observable numerical values. The controller pad P_c is a minus pad that changes the sign of its subpad's observable numerical value. The two controller pads P_d and P_e are enumeration pads that count the number of the subpads pasted directly on them. Each apple is also a transparent controller pad with an apple picture on it. Thus pad P_b calculates $5-2$ to obtain 3 as its observable. The view pad P_f is an equality pad that examines if the input value from its slave controller is equal to the observable value of its master pad. If the equality holds, the observable of the pad takes the true logical value. Otherwise, it takes the false value. The pad P_g is a number pad. When a numerical value is written on it, P_g sends this value to its master pad. In this example, the master pad of P_f has 3 as its observable value, while the slave controller of P_f gives 2. Thus, the observable of the equality pad becomes false. The view pad P_h displays either of the two logical values. It is actually a button-with-light pad. Its display patterns can be arbitrarily designed with the bit-map editor. The letters and words on this card are written on text pads pasted on insulator pads, and pasted on the card. Pupils can delete or add apple pads on the left hand side to make a new question.

5. MANAGEMENT AND RETRIEVAL OF PADS

5.1 Definition and Navigation of Hypermedia Networks

Hypermedia systems such as NoteCards and HyperCard span networks among documents by relating an item in a document to another document, and allow users to navigate the sea of documents along these links. They use two types of links, i.e., button-links and word-links. A button-link associates a media object with a button on another media object. A click on this button gets the associated media object on the screen. Buttons may have different sizes. A word-link associates a media object to an appearance of a specified word or word string in a text media. A click on this word or word string gets the associated media on the screen.

IntelligentPad allows us to define these links among pads. A button-link from a pad P_1 to a pad P_2 is implemented by pasting a shared copy of the iconic representation of P_2 on P_1. IntelligentPad allows us to change the size of the icon and, if necessary, to make it transparent. A double click on the icon opens P_1. Since we can use arbitrary number of shared copies of the same pad, we can associate the same pad with arbitrary number of different button areas on different pads. We can also use nonshared copies wherever they are required.

A word-link mechanism is already built in the text pad as its primitive mechanism. It provides two modes of association. Context-sensitive association associates a pad with a specified word at a specified location in the text, while context-free association does not specify the location. In the latter association, wherever we click the specified word, we get the associated pad on the screen.

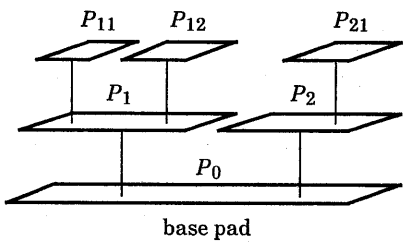
5.2 Pad Base and Its Search

Pads are all persistent in IntelligentPad. We need efficient mechanisms both for the management of pads and for the retrieval of specified pads. IntelligentPad provides three kinds of access methods, i.e., browsing, navigation, and quantification. The binder pad allows us to organize pads and icons in a catalog form. We can browse such catalogs to find desirable pads. The capability of spanning two kinds of hypertext links among pads allows us to organize pads in networks, through which we can navigate to find the pads in search. A form base allows us to find required pads by specifying the quantification condition satisfied by these pads. Each form base, however, stores pads with the same format. Thus form bases cannot meet our needs to manage all types of pads in an integrated way. IntelligentPad provides a special folder called a pad base to meet such needs.

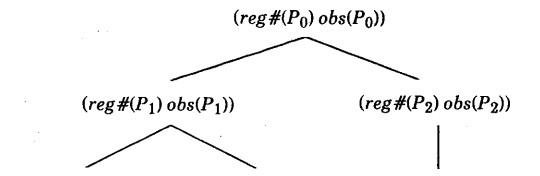
A pad base is a database in which stored records are all pads. IntelligentPad may define any number of mutually exclusive pad bases. Pad retrieval from each pad base requires some method to quantify the pads to find. IntelligentPad has

adopted the Query-By-Example approach for the quantification.

Each pad has its nickname, registration number, composition structure, and the observable values of its constituent subpads. Each of these may appear in retrieval conditions. The registration number of a pad specifies its type. We will define the signature of a pad as follows. It is a tree with a pair of a registration number and a value at each node. Its root node has a pair of the registration number and the observable of the base pad. Each son of its root has a subtree representing the signature of a composite pad that is directly pasted on the base pad. Thus, a composite pad in Fig. 5.1 (a) has its signature as shown in (b).



(a) composition structure



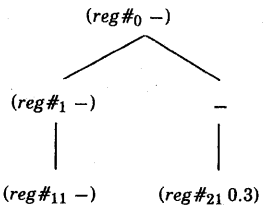
reg#(P) : the registration number of P
obs(P) : the observable value of P

(b) pad signature

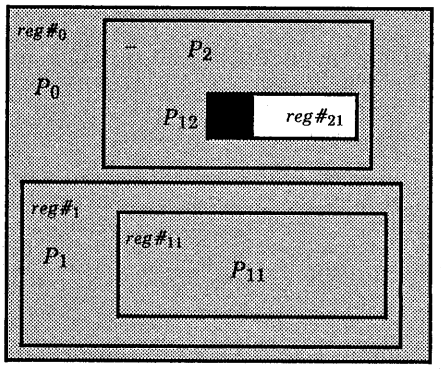
Figure 5.1 a composite pad and its signature.

The pads to retrieve can be quantified by partially specifying their signatures. Such a specification partially specifies a signature tree, remaining some of its nodes, subtrees, and/or the observable values of some nodes unspecified. Figure 5.2 (a) gives an example partial pad specification, while Figure 5.2 (b) shows a pad representation of the same partial specification. In Fig. 5.2 (b), each pad P_i except P_2 must be selected among those with the same registration number $reg\#_i$. The pad P_2 is a special pad only used in queries. It specifies no condition except the existence of a pad at its level. We can find these pads P_i by specifying their nicknames or by browsing a pad catalog. The shaded pads in the figure indicate that their observable values are not quantified. We can shade any pad by a mouse click on it. The pad representation of a partial specification is called a basic pad query.

For example, a shaded computation pad with a shaded digital display pasted on it may be used as a query to retrieve every composite pad with a digital display pad pasted on a base computation pad. The retrieval result includes various hand calculators with different sizes and layouts. If we further specify the digital display pad value to be 5.0 in the above query, the retrieval result excludes such a pad with its display value different from 5.0. IntelligentPad uses such a pad representation as a basic query to each pad base. It retrieves all the pads with signatures satisfying the partial specification.



(a) a partial specification of signatures



(b) pad representation of the partial specification

Figure 5.2 a composite pad and its signature.

For an insertion of a pad into a pad base, we only need to place the icon of this pad over the pad base icon. For issuing a query, we may open the pad base and paste the pad query on it. We can also relate a pair of two basic pad queries to satisfy some conditions. These two queries may or may not be issued to the same pad base. IntelligentPad allow us to specify either an equality/inequality relation between an observable in one pad query and an observable in the other, or a dependency relation between the two pad queries by pasting the icon of one of them on the other. In both cases, the system applies the nested loop algorithm to perform such complex queries. Update queries can be also specified in similar ways.

6. CONCLUSION

In this paper, we have proposed a system for visual synthesis and visual management of active media objects. IntelligentPad represents everything as a pad. It associates each pad with some function. The pasting of pads on another pad visually defines a new pad with an arbitrary layout of fields and a new function composed of the constituent pads.

Each pad is implemented as an MVC triple. Its model defines its state and behavior. Its view defines how it looks on the display screen, while its controller defines its mouse-interaction. Pads are classified into three categories. Controller pads, when pasted on other pads, work as their input devices. View pads, when pasted on other pads, work as their output devices or I/O devices. The remaining pads are model pads.

Through some examples, we have shown the highly generic modularity of pads. Each pad works as a standard part or attachment. It works as a generic software module that modifies the state, the behavior, or the visual image of another pad. The capabilities of IntelligentPad will further increase as we compose more pads.

IntelligentPad provides four ways of managing a large amount of pads, i.e., visual catalogs of pads, hypermedia networks, form bases, and pad bases. A form base stores pads

with the same format, while a pad base manages all types of pads in the system.

IntelligentPad may also be considered as a system integration tool. We may consider any application program as a model, and provide this with an appropriate view and an appropriate controller. We can, thus, defines a new pad. We call these view and controller a pad interface. Provision of an appropriate pad interface will allow us to use an arbitrary application program in the integrated environment of pads.

This system integration capability indicates a further extension of IntelligentPad. The model of a pad may not necessarily be implemented in Smalltalk-80. Smalltalk-80 running on UNIX allows us to implement any pad model in another language such as C. We may also consider pads as an extended window system.

ACKNOWLEDGEMENT

We would like to express our thanks to Dr. Adele Goldberg of ParkPlace Systems for her discussion with us. We also thank Messrs Kazushige Oikawa, Seiichirou Kamishiro, and Hayaki Watanabe of Fuji Xerox for their introduction of Smalltalk-80 Culture to Japan and to our laboratory.

REFERENCES

[1] Gibson, J. J., *The Ecological Approach to Visual Perception*, Houghton Mifflin Comp. Boston, 1979.

[2] Arav, G, and Clifford, J. (eds.), *New Directions for Database Systems*, Ablex Publishing Corp. NJ (1986).

[3] Zloof, M. M., 'QBE/OBE: A Language for Office and Business Automation,' *IEEE Computer*, Vol. 14, No. 5 (May 1981), pp. 13-22.

[4] Yao, S. B., Hevner, A. R., Shi, Z., and Luo, D., 'FORMANAGER: An Office Forms Management System,' *ACM Transactions of Office Information Systems*, Vol.2, No.3 (July 1984), pp. 235-262.

[5] Shu, N. C., 'FORMAL: A Forms Oriented Visual Directed Application Development System,' *IEEE Computer*, Vol.18, No.8 (Aug., 1985), pp. 38-49.

[6] Negroponte, N., 'Media Room,' *Proc. of the Society of Information Display*, Vol. 22, No. 2 (1981), pp. 109-113.

[7] Shu, N. C., *Visual Programming*, Van Nostrand Reinhold, NY, 1988

[8] Christdoulakis, S. et al., 'Multimedia Document Presentation, Information Extraction, and Document Formation in MINOS: A Model and a System,' *ACM Transaction on OIS*, Vol. 4, No. 4 (1986), pp. 345-383.

[9] Woelk, D., et al. 'Multimedia Information Management in an Object-Oriented Database System,' *Proc. of the 13th VLDB Conference*, Brighton (Sept. 1987) pp. 319-329.

[10] Meyrowitz, N., 'Intermedia: The Architecture and Construction of an Object-Oriented Hypermedia System and Applications Framework,' *ACM OOPSLA '86 Conference Proceedings* (1986), pp. 186-201.

[11] Bush, V., 'As We May Think,' *The Atlantic Monthly*, Vol.176, July 1945, 101-108.

[12] Engelbart, D.C., 'A Conceptual Framework for the Augmentation of Man's Intellect,' In *Vistas in Information Handling* (Howerton & Weeks Eds.), Vol.1, Washington, D.C., Spartan Books, 1-29.

[13] Nelson, T.H., 'A File Structure for the Complex, the Changing, and the Intermediate,' *Proc. of the ACM National Conference*, 1965, 84-100.

[14] Kay, A. and Goldberg, 'A., Personal Dynamic Media,' *IEEE Computer* 10 (3), March 1977, 31-42.

[25] Henderson, A., 'The Trillium User Interface Design Environment,' *Proc. of CHI '86*, 1986, 221-227.