TR-87-005

A Manipulation Language for Visual Programmings

KUO-YANG CHENG

PART TWO

A Manipulation Language for Visual Programmings

摘　　要

最近幾年來電腦應用的性質已經在改變了，讓非程式設計人員去發展他們自己的應用程式，將是一項很自然的需求。視覺程式規劃是爲達到這個目的所提出來的一種做法。在此，我們提出一個視覺程式規劃合成器（VIPS），它讓使用者能以視覺的方式來描述他們的應用程式，於此視覺程式規劃合成器中，我們以視覺表格做爲它處理的基本對象，因爲視覺表格較接近於使用者的觀點，並且可以做爲程式合成中的一種媒體。

本論文提出一種視覺表格的處理語言（VML），VML 是一種視覺導向的非程序性語言，它可當做一個程式編輯器，讓使用者任意地處理視覺表格實例，VML 提供的功能包括顯示，插入，除去，修改，組合，及分離等運作。當使用者處理完視覺表格上的運作，則他所定義的應用可以自身合成成一個具有一致性的內部結構，此內部結構可以被一個視覺表格解譯器直接去執行。最後，我們以一個電腦輔助教學的例子來說明VIPS如何處理及合成一個應用程式。

ABSTRACT

The nature of computer applications have changed in recent years. Non-programmers are turning to spontaneous demands for developing their own application programs. A visual programming approach is proposed to achieve this goal. We present a VIsual Programming Synthesizer(VIPS) which allows users to describe their application in visual. In VIPS, V-Forms (visual forms) are used as the fundamental objects because they are more akin to the user's view point and they can be used as medium in the program synthesizing process.

In this report, a V-Form Manipulation Language(VML) is presented. VML is a visual-oriented non-procedural language and served as a program editor that can manipulate V-Form instances at will. VML provides the following operations: (1) Displaying, (2) Inserting, (3) Removing, (4) Updating, (5) Grouping, and (6) Degrouping. After operating on V-Forms, a user-defined application can be self-synthesized into a consistent internal structure which can be executed by a direct V-Form interpreter. A CAI example is given to illustrate how an application program can be synthesized and manipulated by VIPS.

CONTENTS

CHAPTER 1
INTRODUCTION

1.1 BACKGROUND

As the advances in computers and their usage, computerization in many applications, such as Computer Aided Instruction(CAI)[1,20,25] and Office Information System(OIS)[9,21,27,28,29,32], has become more and more needed. However, we do not have enough qualified programmers to write application programs, the application backlog has become a serious problem in recent years. Two approaches to reduce the backlog are apparent: one is to increase the productivity of the people who can program, and the other is to increase the number of such people. In the past, a lot of effort was directed to increasing the productivity of professional programmers. Useful methodologies and convenient tools for all phases of software development life cycle were developed.

Later, the nature of applications has changed. Stimulated by the decreasing cost of computing, the users are turning to spontaneous demands for information to assist them in their daily work. That is, the end users must be given the facility to computerize their own applications. As a result, various forms of user-friendly facilities have appeared in the market-place, such as, icons, pointing devices, and user friendly menus have taken the pain out of learning and remembering commands. Word processors, query facilities, and spread sheet programs have become important tools for text preparation, simple quiries, and financail analysis.

Nevertheless, the usefulness of these tools is limited to pre-designed categories of applications[26]. In the servey conducted by Rockart and Flannery of the Sloan School of management[28,29], it was found that about 50-56% of the end user applications are not of the pre-designed categories. For these kind of user-driven applications, one cannot derive the benefits of computerization without programming. However, programming is still a time-consuming and detail-intensive chore. It is inconvenient and inefficient for all end users to develop their application programs by using conventional programming languages. Most of the end users are NP professionals (Non-Programmers) which are skilled at their fields and know little or none of programming knowledge. Unless provided with an alternative, they will continue to depend heavily on programmers to develop applications.

For example, in CAI applications, three principal factors have driven the development of an ideal CAI system (authoring systems)[20]. The first factor is ease of use of, or access to, computers for instructional purposes. Because the necessity of learning how to program in order to develop courseware would draw back a significant number of teachers who might attemp to use a computer as part of their teaching activities. The second factor is the expense and time required to create computer-based curriculum. Because by eliminating or sharply reducing the time of programming and debugging, the development time and costs of courseware can be significantly reduced. The third factor is the transportability of courseware. Since sharing of courseware is one of the major arguments for the cost/benefits of computer-based education(CBE). Similarly, in office automation, data processing activities, such as updating inventory file, extracting and putting together of data from two or more sources, as well as keeping records of receipt and delivery, etc. should be easily manipulated, namely through some sort of automated procedure. In this area of applications, we assume that the office workers or school teachers are not computer specialists, but required to use computer to develop their own application programs. To them, a system that supports a user friendly interface is essentail. Because it is rather hard to implement such application under conventional programming environment which is designed for programmers to use. Also, it is still difficult to implement them under artificial intelligence(AI) programming environment alone, because it seems still very difficult to handle the facts and rules in CAI or OA very well[5,14,33]. Thus, a new environment is needed in such kind of applications.

The representation of information in CAI or in OA is usually in visual. For example, CAI courseware is frame-by-frame and office transaction is form-by-form. It looks that visual programming language, which can define and manipulate frames (or forms) and their relationships, may be a promissing approach to provide such an interface for non-programmers. In particular, in recent years that the visual programming approach on such kind of applications has become more and more popular shows the importance of research in this direction.

1.2 RELATED RESEARCH

Several systems have been developed for users to develop their applications. Most of them have been released as products. In this section, important related researches, such as QBE/OBE, FORMAL, STDL, and PegaSys will be described.

1.2.1 QBE/OBE

QBE (Query By Example) is an IBM product released in 1978[35,36]. It is a high-level data base management language and it is centered on office and business applications and is widely used in the areas such as distribution, finance, government, manifacturing, processing, and utilities. On the other hand, OBE (Office procedure By Example) is the extended version of the QBE.

The features of QBE/OBE are:

1. QBE/OBE emphasizes on relational DBMS interactive query and data maintenance. The operations are easy to learn and easy to use. Users only need to describe their applications directly to the computer.

2. The fundamental object of QBE is a two-dimensional, one-level skeleton table which also provides the programming environment. Initially, users are given a blank table skeleton.

Then, after key in the appropriate name into the table name field, the table heading is generated and the users can now "program" it by entering appropriate QBE/OBE commands.

3. Users with no knowledge of any formal programming language can, in a matter of several hours training, formulate QBE programs to retrieve, modify, define, and control the database. Psychological testing of QBE users has shown it to be a very friendly language.

The major drawback of it are:

1. Its data object is only a one-level flat table, more complex (hierarchical or graphical) structure is hard to implement.

2. No procedures, conditions, or actions can be incorporated within the table.

1.2.2 FORMAL (Form-ORiented MAnipulation Language)

FORMAL provides a powerful visual-directed facility for non-programmers to develop their data manipulation applications on computers[28,29].

The features of FORMAL are:

1. A form-oriented approach is employed in FORMAL. The form data model proposed by Shu defines the forms as a named collection of instances (or records) with the same data structure. The components of a form can be any combination of fields and groups. Fields is the smallest unit of data that can be referenced in a user's application, while group is a sequence of one or more fields and/or subordinate groups. The group is also called the subform of a form.

2. In this model, "form" is used as a two-dimensional representation of hierarchical data. Each form has a form heading which exploit the hierarchical structure within a form.

3. The form-oriented programming language is a two-dimensional non-procedural description language which uses forms as both the fundamental data object and program structure. Users can program within the form visually. The concept of what-you-sketch-is-what-you-get is also introduced.

However, in FORMAL, forms can not be recursively defined to form more complex structure(such as graphical structure).

1.2.3 STDL (State Transition Diagram Language)

The state transition diagram language uses diagram to describe algorithms to the computer. A visual programming environment for this language is currently being implemented on a SUN workstation [19]. The main features of STDL are:

1. Uses graphical representations to represent data objects as well as abstract objects.

2. The temporal sequence during the dialog between a user and the system are emphasized.

3. One diagram can call upon another diagram. This is one of the important features of this language. In this way, procedure call in traditional programming language can be simulated.

The major drawback of it are:

1. STDL is still suitable for programmers. It may be too difficult for non-programmers to use.

2. The design of a precise flow may be a burder for most users.

1.2.4 PegaSys (Programming Environment for the Graphical Analysis of System)

The main purpose of PegaSys is to facilitate the explaination of program design[24]. It encourages and facilitaties extensive use of graphical images as formal, machine-processable documentation.

The features of PegaSys are:

1. A program design is described in PegaSys by a hierachy of interrelated pictures. Each picture describes data and control dependencies among such entities as "subprograms", "process", modules", and "data objects", among others. The pictures also describe how algorithm and data structure fit together to form the design of a large program.

2. PegaSys exhibits its ability to: (1) check whether pictures are syntactically meaningful, (2) enforce design rules throughout the hierarchical decomposition of a program, and (3) determine whether a program meets its pictorial documentation.

3. Pictures have dual interpretation: graphical structure and logical structure. A graphical structure is composed of icons and their properties, such as size and location. Icons is a picture correspond to predicates in logical structure. And this logical structure captures the computational meaning of a picture, each predicate in this structure denotes a computational concept expressed by the picture.

4. Many graphical manipulations of pictures are provided. Such as the creation of a new-level in a hierarchy, refinement of active, passive entities, refinement of an interaction.

Although PegaSys emphesizes program designs in visual, it is still unpractical for non-programmers to develop their application programs.

1.3 OUR APPROACH

To provide a new programming environment for NP-professionals to develop their own application programs, a VIsual Programming Synthesizer(VIPS)[6] is proposed. VIPS is designed to achieve three goals: (1) provides a visual facility to achieve a better user friendly interface, (2) provides a visual language for non-programmers to be able to program easily, (3) provides visual forms as a medium to process complex and knowledgable objects.

In VIPS, V-Form(visual form) is the data representation of text(in particular, Chinese characters), static graphics(line drawing and bit-map), dynamic graphics(animation), rules, and voices. Although it is still in form style, but in order to distinguish them from office forms, they are called V-Forms.

Just like other form models[4,22,24,28,29,30,35,36], our V-Form model consists of a V-Form type and a V-Form instance. A V-Form type describes the structure of a V-Form system, while a V-Form instance is obtained after filling values into the contents of the described V-Form type. The V-Form instance is a program that fits the concept of embedded procedures for computational and controlling requirements. Hence, facilities for conditional and unconditional control switches are also included in the V-Form model.

Based on the V-Form model, VIPS offers a two-dimensional non-procedural language as a user-friendly interface language to define and manipulate V-Forms on the screen. This interface language includes two parts: a V-Form Definition Language(VDL)[18] and a V-Form Manipulation Language(VML). VDL is used to define the V-Form type(logical and visual strtucture of a V-Form) and V-Form instance. And VML can be invoked to display, insert, update, group, and degroup these V-Forms gotten from VDL. After procesing VDL and VML, a complete internal form for each application can be self-synthesized. V-Form instances that are text-only can be executed by a direct V-Form interpreter, while V-Form instances other then text are treated as a series of procedural calls delivered to graphics and voice system for execution.

VIPS has been implemented on VAX/750 and VAX/785 under both VAX/VMS and VAX/UNIX(Ultrix-32) operating systems. Although, at present time, VIPS is only an experimental system, it can be enhanced to include more software packages such as standard graphics systems, Chinese input/output, and voice system so that a more powerful and complete VIPS can be obtained.

1.4 ORGANIZATION OF THIS REPORT

In this chapter, we discuss the requirements of computer applications (such as CAI and OIS) for NP-professionals first. In section 2, several important related research tasks or released products are studied for guideness. In section 3, a new visual programming environment is described where a visual programming synthesizer(VIPS) is served as a vehicle for non-programmers to develop their own application programs.

Chapter 2, a formal definition of the visual language is given in section 1. The visual language is shown to be a formal language which is generated by an inferred grammar containing only objects sketched by users. In section 2, a visual programming environment is compared to the conventional programming environment and to the AI programming environment.

In Chapter 3, the system architecture of a visual programming synthesizer is proposed. In section 1, a V-Form model is proposed. In section 2, the informant presentation data which is represented by V-Form is proposed as a visual representation of data objects. The function behavior of each components of the system architecture will be discussed in section 3.

In chapter 4, we introduce the V-Form Manupulatin Language and its applications in section 1. In section 2, we define the VML formally. In section 3, syntax rules of VML is given.

In chapter 5, section 1 describes the basic functions of VIPS. The displaying, inserting, removing, updating, grouping, and degrouping operations as well as the instance handling functions will be described in the subsequent sections.

In chapter 6, section 1 gives an illustrated CAI example to show how a courseware can be developed by VIPS. In section 2, we present a direct V-Form interpreter to interprete the example.

CHAPTER 2
LANGUAGE ENVIRONMENT

2.1. DEFINITION OF VISUAL LANGUAGE

A visual language is a formal language which designated for non-programmers to develop their own application programs in visual. In this section, we shall give a definition for the visual language. But first, we need the following definitions [10].

[Definition 2.1] An information sequence of a language L, denoted by I(L), is a sequence of codes from the set $\{ +y \mid y \in L \} \cup \{ -y \mid y \in V_T{}^*-L \}$, A positive information sequence of a language L, denoted by $I^+(L)$, is an information sequence of L containing only codes from L, Similarly, a negative information sequence of a language L, denoted by $I^-(L)$, is an information sequence of L containing only codes from $V_T{}^*$-L.

[Definition 2.2] An information sequence of a language L is said to be complete if:

1. $I^+(L)$ contains all codes from L.
2. $I^-(L)$ contains all codes not in L. A positive information sequence $I^+(L)$ is said to be complete if each code in L appears in $I^+(L)$.

[Definition 2.3] A sample of a language L, denoted by $S_t(L)$, is defined to be the set $\{ +y_1, \ldots ,+y_t \} \cup \{ -y_1, \ldots ,-y_t \}$, where $S^+ = \{ +y_1, \ldots , +y_t \}$ is defined to the positive sample and $S^- = \{ -y_1 , \ldots ,-y_t \}$ the negative sample. A positive sample S^+ of a language L(G) is structurally complete if each production defined in G is used in the generation of at least one code in S^+.

Now, the visual language can be formally defined as follows.

[Definition 2.4] Let D be the domain of what-you-sketch and y_i be a sketched object, $y_i \in D$. Suppose y_i can be generated by an inferred grammar G_i, then a visual language is

$$L_v = \{ y_i \mid y_i \in D, \text{and } y_i \in L^+(G_i) \}.$$

where

$$L(Gi) = L^+(G_i) \cup L^-(G_i) = \{ \text{positive\_sample} \} \cup \{ \text{negative\_sample} \}.$$

The positive sample is an information sequence of $L(G_i)$ containing only codes from $L^+(G_i)$ which is a set of objects described by the user. On the other hand, the negative sample is the parasitical product of G_i and is not included in the visual language. And,the positive sample L_v of the language $L(G_i)$ is structurally complete. In other words, a visual language is a formal language that contains only objects sketched by the users.

As we know, there are three types of languages, i.e. regular, context-free, and context-sensitive languages that can be generated by an inferred grammar in the limit[11,16]. Among them, context-free languages are not powerful enough to describe the programming applications, but context-sensitive languages are very complex for analysis. Consequently, a compromise is made by generating a context-free grammar into programmed grammars. The visual language L_v, which is used to get the desired applications in what-you-sketch-is-what-you-get manner of operations, can be implemented on a context-free programmed grammar that can simulate the behavior of the inferred grammar G_i of L_v. In other words, the visual programming itself is an interactive process which allows the user to describe the inference algorithm by himself. Here, the inference algorithm is a process of describing the positive samples as designed by the user. That is, the visual language is to allow the user to get the what-you-sketched objects according to his application requirements. In formal language aspect, this inferred grammar G_i of L_v is a context-free programmed grammar and the class of languages which are generated by G_i can be a context-sensitive language.

2.2. PROGRAMMING ENVIRONMENT

A conventional programming environment concerns how to provide programmers some convenient and useful tools to develop their application programs. In this environment, programming itself is not only a time-consuming detail-intensive chore but also a specialized arduous task requiring detailed textual instructions that must adhere strictly to syntax rules. As for non-programmers, using programming languages to develop their own application programs is almost infeasible (if not impossible). Most of them don't have time or interest to program because of lacking programming knowledge, and it is even doubtful whether their time is well spent in the training of programming.

Artificail intelligence is the field whose goal is to automate the knowledge processing. Consequently, it involves some kind of knowlwdge (information) representations, and to tackle the tasks (such as problem solving, descising making, making inference, manufacturing goals, pattern recognition, scene analyzing, natural language understanding etc.) is to manipulate or process the knowledge. However, in the AI programming environment, building an intelligent application system is essentially coding the required knowledge into facts and rules in a high-level languages such as PROLOG or LISP-based languages[2,3,7,17,34]. At present, the coding is manually written, and is, like writing a computer program, also a laborious and tedious task in AI environment. Furthermore, because its friendly human interface has not get been well-developed, it seems very difficult for NP-professionals to use AI languages to handle their specialized knowledge very well.

Based on these observations, a new programming environment is needed for such NP-oriented applications. Here, we present a visual programming synthesizer. In this system, users with or without any programming language background, can describe their specialized knowledge through the self-synthesizing process to get the desired applications. We shall use an CAI courseware as an example to illustrate key features of VIPS and reveal the easiness for teachers (NP-professionals) how to develop their own coursewares.

In CAI applications, courseware development may be created with the high level languages (such as BASIC, PASCAL, FORTRAN etc.), its disadvantages are obvious, teachers will be infeasible to develop their own courseware without programming knowledge. Another way for the courseware development is to use the author languages (such as TUTOR, PLANIT, PILOT, NATAL, COURSEWRITER, TIP, CATO etc.), which are a family of special purpose, high order application languages which facilitate the writing of instructional programs. Authoring system, on the other hand, represent a high level interface intended to allow authors (instructors or instructional developers) to create courseware; even though they are non-programmers. So the development and use of authoring system is the implementation of automatic programming or metasoftware, which is a programming system that generates other programs, has been a popular idea in computer science. As a matter of fact, our visual programming synthesizer can also be served as an authoring system in CAI applications.

CHAPTER 3
SYSTEM ARCHITECTURE

3.1. V-FORM MODEL

3.1.1. V-Form External Structure

The visual programming synthesizer allows users to open several windows on a screen. Each window is treated as a V-Form. The informant presentation data in a V-Form may include text, static graphics, dynamic graphics, rules, and voices.

Let Π be a set of codes to be displayed, Δ be a set of alphanumerics, ς be a set of Chinese characters, β be a set of graphic codes, Ω be a set of drawing attributes, and $\Sigma = \Pi \cup \Delta \cup \varsigma \cup \beta \cup \Omega$. Then the domain of a V-Form belongs to Σ^*, where * denotes the Kleene closure[15]. All V-Forms of an application are subset of D and are considered as the positive samples of $L(G_i)$.

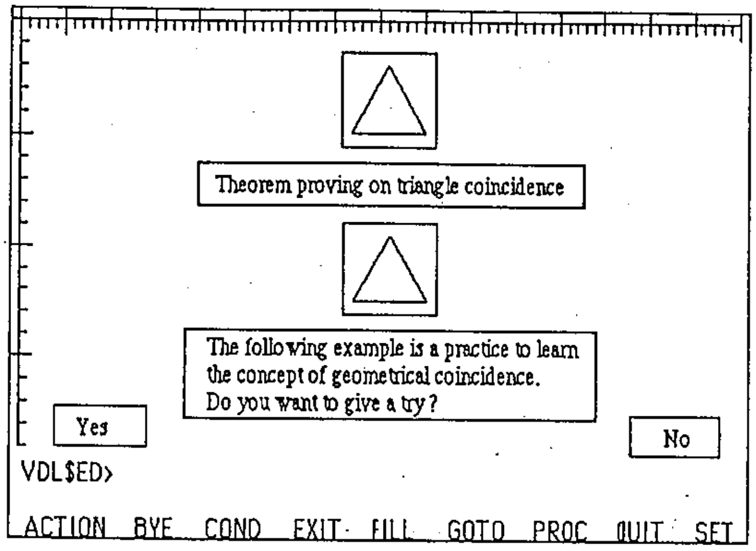
A **V-Form** is a pair of a **V-Form type** F and a **V-Form instance** I defined below. A V-Form type consists of a scheme S and a template T_S for S.

[Definition 3.1] A **Scheme S** is the logical structure of a V-Form. It can be recursively defined as:

```
<S>       ::=  <M> | <A>
<A>       ::=  <S> | <S>,<A>
<M>       ::=  <Type>:<Identifier>
<Type>    ::=  TEXT | GRAPH | BIT-MAP |RULE
               |ANIMATION | VOICE | VFORM
```

For example, consider the first V-Form of a CAI_Course shown in Fig.3.1. The scheme for this V-Form is defined in the following expressions:

```
CAI_Course= [TEXT:Topic, TEXT:Ask, ANIMATION:Pic1,
              ANIMATION:Pic2, VFORM:Yes, VFORM:No]
VFORM:Yes  = [ . . . ]
VFORM:No   = [ . . . ]
       . . .
```


Theorem proving on triangle coincidence

The following example is a practice to learn the concept of geometrical coincidence. Do you want to give a try?

Fig. 3.1 A typical V-Form screen

The hierarchical/graphic structure of the scheme for the CAI_Course is given in Fig. 3.2, where a circle means a V-Form which

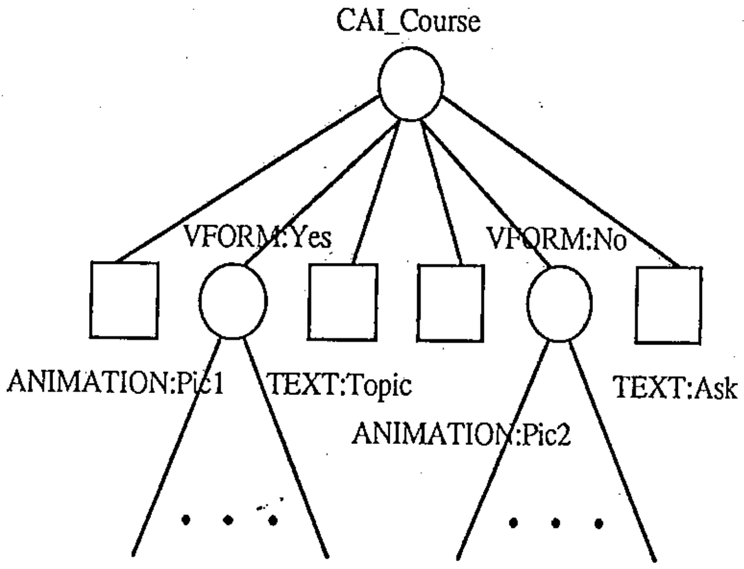


Fig. 3.2 Hierarchical structure for CAI_Course

contains another V-Forms, a box represents a minimum V-Form which contains no other V-Forms (also called an atom).

[Definition 3.2] A **template** T_S is a visual structure of a V-Form that represents a two-dimensional display format and visual properties of a scheme S. The visual properties of a V-Form are independent of V-Form instance.

The template for the scheme of Fig. 3.1 is given in Fig. 3.3, where the sub-VForms TEXT:Topic, TEXT:Ask, ANIMATION:Pic1, and ANIMATION:Pic2 are atoms, but VFORM:Yes and VFORM:No are not. As shown in Fig. 3.2, each V-Form may contain other sub-VForms and hence a hierarchical/graphic structure is formed.

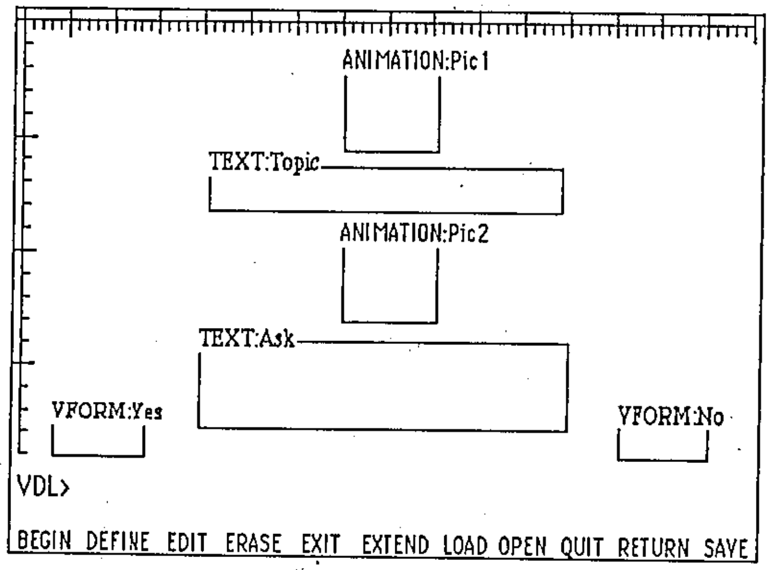


Fig. 3.3 A template for the CAI_Course

A V-Form F_j is said to be a sub-VForm of F_i if and only if F_i contains F_j. If F_j is a sub-VForm of F_i and is not equal to F_i, then F_j is said to be a proper sub-VForm of F_i. Two V-Forms F_j, F_k, which are the maximum proper sub-VForms of F_i, are called brothers. A template also describes the control flow of the V-Form node. For instances, when VFORM:Yes (sub-VForm of CAI_Course, here, CAI_Course is the V-Form system name which contains the first page of V-Form.) of Fig. 3.3 is selected then the display will be switched to that of its descendent V-Form node F2 (Fig. 6.1(b)).

[Definition 3.4] The **value** of a V-Form **y** of type **t** is **x**, such that $x \in \mathrm{DOM}(y_t) \subseteq \Sigma^*$ where

$$\mathrm{DOM}(y_t) = \begin{cases} (\Delta \cup \Pi \cup \varsigma)^*, & \text{if } t = \text{TEXT, RULE} \\ (\beta \cup \Omega)^*, & \text{if } t = \text{GRAPH, BIT-MAP, ANIMATION} \end{cases}$$

For example, the value of V-Form Topic of type TEXT in Fig. 3.1 is "Theorem proving on triangle coincidence".

[Definition 3.4] The **Characteristics** of a V-Form include the displaying attributes(NORMAL, BOLD, INVERSE, or FLASH vedio), the indicator(ON or OFF), the alias (give an alternative name to a given V-Form), and the embedded components (conditions, actions, and procedures) of a V-Form.

[Definition 3.3] A **V-Form instance** (I) for a V-Form type F is defined as a mapping which assigns a value to each atom of F. The value of a V-Form may have the following types: text, graph, bit-map, voice, and rule. One of the V-Form instance of Fig. 3.3 is shown in Fig. 3.1.

As stated above, a V-Form type can be filled with different contents to obtain different kinds of V-Form instances. Therefore, a V-Form type is just like a language, from which a user can write many programs, which, in this model, are V-Form instances.

3.1.2. V-Form Internal Structure

The internal structure of V-Form links four kinds of nodes, namely, system node, form control node, mode node, and action node. The system node (SN) keeps all information of the V-Form system. The system node connects a V-Form sub-system that may be two independent coursewares grouped together. The form control node (FCN) describes the characteristics and the display information of its corresponding V-Form (atom or non-atom). The mode node (MN) contains the content and the display attribute of an atom. The action node (AN) contains the embedded components, such as conditions, actions, or procedures.

In the following, the detailed information of these four kinds of nodes are specified.

1. System Node (SN):

id	:	sub-system id.
name	:	sub-system name.
ptr	:	a pointer pointing to the first corresponging V-Form page.
next	:	a pointer pointing to the next SN.

End.

2. Form Control Node (FCN):

id	:	form control node id.
name	:	V-Form name.
aliasptr	:	a pointer pointing to a MN which contains the alias of the corresponding V-Form (with type VFORM).
row	:	window row position in the screen.
col	:	window column position in the screen.
width	:	window width length in the screen.
height	:	window height length in the screen.
procptr	:	a procedure pointer pointing to an AN which specifies its embedded components.
phyaddr	:	a phisical address pointer.

type	:	a M/F flag, with type "M", this FCN represents an atom and its physical address pointer pointing to a MN containing its content; with type "F", this FCN represents a non-atom V-Form(with type VFORM) and its physical address pointer pointing to another form control table.
indicator	:	a ON/OFF display indicator.
next	:	a pointer pointing to next FCN.
link	:	a link pointer which is used to create the internal structure or link the available buffer.

End.

3. Mode Node (MN):

typename	:	type name of content.
attribute	:	display attribute.
type	:	a N/C flag, with type "N" means a normal content and type "C" means its content is a continued one.
content	:	represents the content of the corresponding V-Form or its embedded component.
link	:	a link pointer which is used to create the internal structure or link the available buffer.

End.

4. Action Node (AN):

condptr	:	a condition pointer pointing to MN which

		represents its condition part.
actptr	:	an action pointer pointing to a MN.
ptrtype	:	a M/F flag for actptr, with type "F", the actptr points to another FCT; with type "M", actptr points to a MN.
next	:	a pointer pointing to next AN.
link	:	a link pointer which is used to create the internal structure or link the available buffer.

End.

Accordingly, an atom is represented by a FCN which has a pointer pointing to a MN. A non-atom V-Form is also represented by a FCN but has a pointer (in FCN) pointing to a list of FCN which describes its sub-VForms. The list of FCN, also called a form control table (FCT), describes the V-Form structure within a window. Fig. 3.4 shows part of the internal structure of the V-Form system CAI_Course (only F1 of Fig. 6.1(a) is shown).

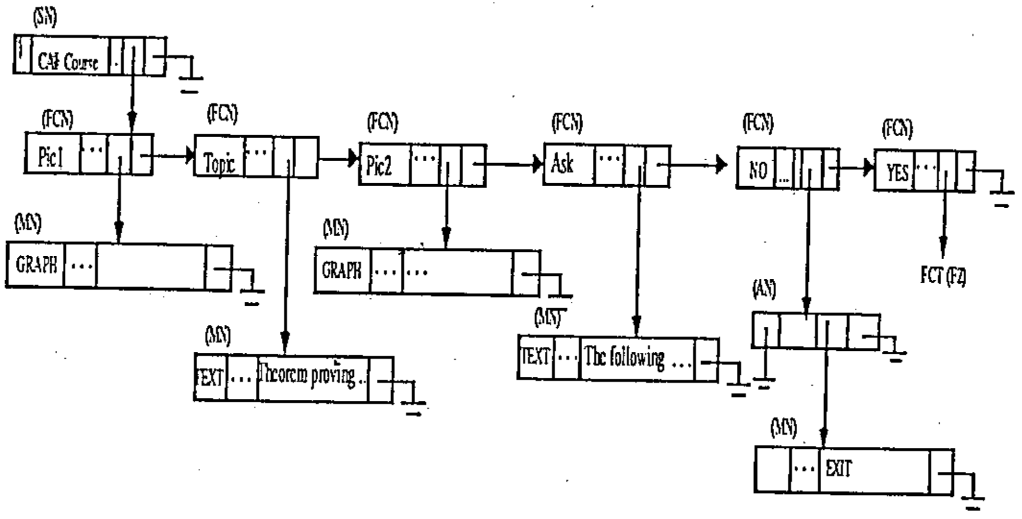


Fig. 3.4 The internal structure of the V-Form F1 of Fig.6.1(a).

In Fig. 3.4, the system node SN(CAI_Course) connects a V-Form system starting from the form control table FCT(F1), FCT(F1) is a linked list with the following elements: FCN(Topic), FCN(Ask), FCN(Pic1), FCN(Pic2), FCN(Yes), and FCN(No) etc. According to the V-Form created sequence, the relational position of these FCNs is obtained. The contents of atom TEXT:Topic, TEXT:Ask, ANIMATIOMN:Pic1, and ANIMATION:Pic2 are stored in the corresponding MN, respectively, which pointed by the FCN physical address(phyaddr) pointer. On the other hand, the content of VFORM:No is NULL and VFORM:Yes is another V-Form page (the physical address pointer of FCN(Yes) pointing to form control table FCT(F2)). An action node is pointed by the procedure pointer (procptr) of FCN(No), its condition pointer (condptr) now is NULL and action pointer (actptr) pointing to a MN whose content is "EXIT". It means that when the VFORM:No is selected, there is an unconditional action is executed which halts the execution. While, when the VFORM:Yes is selected, the display is switched to that of its descendent V-Form F2.

In this model, there are some other features about the embedded components, they are: (1). Automatical (unconditional) switching facility -- when the pointer type (ptrtype) of AN is "F" and condition pointer (condptr) is NULL, then the action pointer (actptr) points to another form control table(FCT), so an unconditional switching control is accomplished. (2). Conditional switching facility -- just like

(1) except now the condition pointer is not NULL, so the condition part is evaluated first, if it is satisfied, control is switched to another FCT, so a conditional switching control is accomplished. (3). Condition-action facilities -- when the ptrtype of AN is "M" , then the actptr points to a MN to specify the action part(such as display or other arithematic operations) and condptr also points to a MN to specify the condition part(expression evaluation). In this way, the condition-action facility is accomplished. (4). Embedded procedures -- just like (3), except now the condptr is NULL. Now it is served as an embedded procedure. (5). Graph structure -- V-Form can be recurssively defined to form any complex structure.

3.2. VISUAL REPRESENTATION

Requirements for visual representation are user visible, easy understandable, and executable in a visual programming environment. To fulfill these requirements, we use V-Forms (visual forms) as medium in manner of what-you-sketch-is-what-you-get. Here, the V-Forms dealt with contain text, static graphics (line drawing and bit-map), dynamic graphics (animation), rules, and voices.

There are several reasons for using V-Forms. First, after close examination of the non-programmers' data processing needs, we concluded that much of the data manipulation can be naturally expressed or thought of as form-format object processing. V-Forms hold the information needed in described objects and exhibit in their two-dimensional form-type format.

Second, V-Form describes the user-visible behavior and represents the cognitive structure of the user interface. In the last few years, a vast amount of work focusing on "forms" in the interest of office automation and office information system. For instances, forms have been used as templates for documents which are logical images of business paper forms. Form editors and screen management programs have been implemented to facilitate the entering and querying of data via forms. Query languages and data base systems have been extended to deal with forms and other types of documents.

Third, a VForm-based interface language offers an opportunity to drastically alter the nature of programming. Traditionally, programming is a specialized arduous task requiring detailed textual instructions which must adhere strictly to syntax rules. Both the structure of the traditional programming language and the data objects manipulated by them are geared toward the internal (computer) representations. Perception and convenience of non-programmers are sacrificed or ignored for the sake of machine efficiency. However, to encourage the user level programming, end user convenience must be considered. It is our belief-that V-Form is not only a convenient representation of data objects, but also a convenient representation of program structure. V-Forms are designed toward the external representation which are more akin to the user's point of view then the computer's.

In VIPS, the V-Form model is a two-dimensional representation of hierarchical/graphic data. It served as a visual aid to help users understand the high-level V-Form operations performed on their data. V-Form model is also a structured representation of various kinds of knowledgable objects. This structured V-Form is a fundamental concept that makes it possible to transform high-level explicit operations to truly non-procedural specifications.

3.3 SYSTEM ARCHITECTURE

In this section, a primary system architecture is proposed. It consists of the following components: the Informant Presentation Data(IPD), Visual Interface(VI), V-Form Definition Language(VDL), V-Form Manipulation Language(VML), Internal Structure(IS), Directed V-Form Interpreter(DVI), Supported Sub-system(SS), and Relational Data Base Management System(RDBMS)[8,12,23]. Fig. 3.5 shows a configuration of this system architecture.

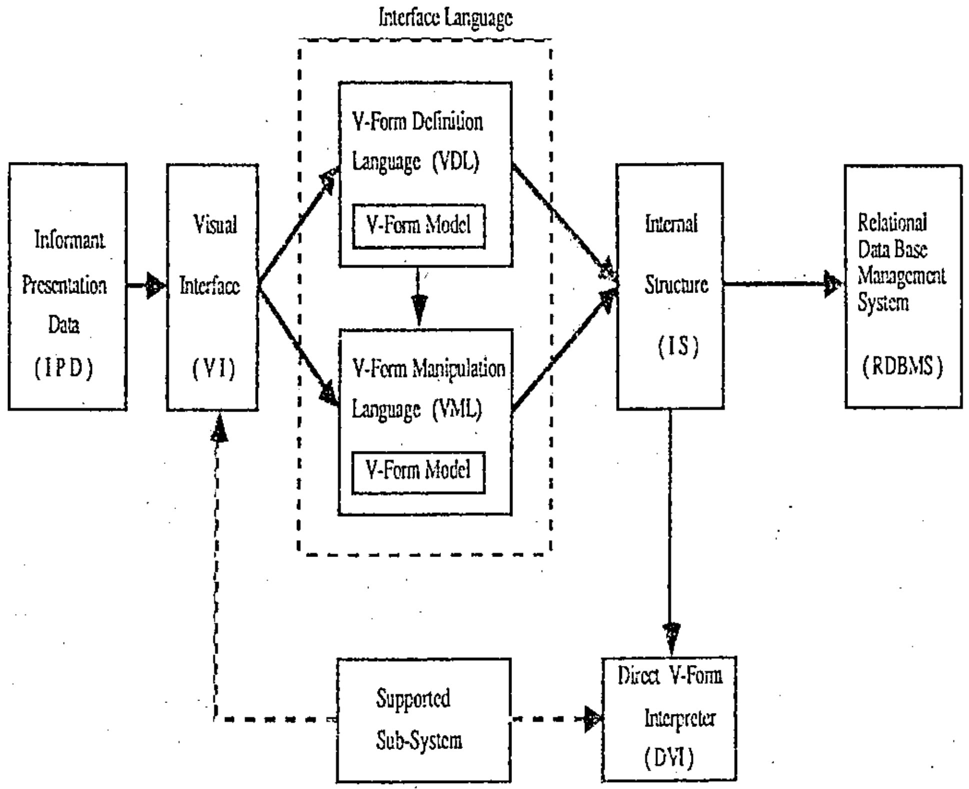

Interface Language
V-Form Definition Language (VDL)
V-Form Model
V-Form Manipulation Language (VML)
V-Form Model
Informant Presentation Data (IPD)
Visual Interface (VI)
Internal Structure (IS)
Relational Data Base Management System (RDBMS)
Supported Sub-System
Direct V-Form Interpreter (DVI)

Fig. 3.5 System Architecture of VIPS

The Visual Interface provides users an environment to communicate with VIPS. In this interface, many user-friendly devices, such as mouse or pointing devices, may be considered. But now in our system, a keyboard is used to simulate pointing devices. The type of the Informant Presentation Data in the communication includes text, static graphics (line drawing and bit-map), dynamic graphics (animation), rules, and voices, they are formatted in V-Forms. Based on the V-Form Model, a V-Form Definition Language and a V-Form Manipulation Language are designed as an Interface Languagre. V-Form Definition Language is used to define the V-Form type and edit the V-Form instance, while V-Form Manipulation Language accepts the V-Form type or V-Form instance from VDL, VML offers users several non-procedural operations to manipulate V-Forms.

After processing the VDL and VML, a consistent Internal Structure, which holds the properties of knowledgable objects and the execution flow of the desired applications, is self-synthesized. The Internal Structure is executed by the Direct V-Form Interpreter. The Supported Sub-system provides many graphic packages and voice utterances for users to generate their appliaction programs. That is, this Supported Sub-system supports the Visual Interface to communicate with users such that the VDL and VML can be used to define and manipulate V-Forms, and the Direct V-Form Interpreter to interprete the Internal Structure. Finally, in consideration of resources sharing, a Relational Data Base Management System is integrated with the visual programming synthesizer.

There are some design methods for visual programming languages. However, a systematic approach to a theoretical sound methodology is still under developing. Here, the visual programming synthesizer is interactive and application-oriented with the following features: (1) ease to use -- various user friendly facilities such as icons, pointing devices, and menus are included. End user perception and convenience are considered; (2) visual directed objects -- V-Form is used as the visual knowledge representation method because it is more akin to the user's view point; (3) non-conventional programming nature -- users only need to describe the external representation of objects instead of writing a series of instruction codes; (4) for non-programmers -- since application programs are generated autumatically by VIPS, users can concentrate in expressing their knowledge to obtain a better presentation; (5) portable -- V-Form system generated by VIPS is always consistent in its internal structure, to transport this system from one system to another is simply by means of an interpreter which interpretes the structure and its contents. (6) resource sharing -- intergrates VIPS with a relational data base management system will facilitate the utilization of resource data.

CHAPTER 4
V-FORM MANIPULATION LANGUAGE(VML)

4.1 INTRODUCTION

In many computer applications, such as office information system(OIS) or computer aided instruction(CAI), there is a close connection between activities and data processing. Most of the common business applications and the development of instructional system can be viewed as manipulation of forms. Here, V-Forms are taken as the fundamental objects which hold all information needed in objects description, they are displayed in a format of window forms. V-Form is not only a convenient data representation (in its external representation) but also a convenient representation of program structure. In the following, a V-Form manipulation language(VML) for the processing of V-forms is presented. It is a non-procedural language for users to do various operations visually.

Let us use a CAI courseware as an activity to explain the operation of VML. A V-Form type describes the program structure and various visual properties(such as displaying attribute, display format, display relationship, and content type etc.) of the defined courseware. And the original V-Form instance has been filled values into its V-Form type, its control flow has synthesized and attached with many embedded components(such as conditions, actions, and procedures). So, an original V-Form instance which conforms to one V-Form type is corresponding to an unwrought instructional program. The invocation of VML is to get the final V-Form instance (courseware) through various operations on V-Form. If the input object of VML is a V-Form type, VML can generate a complete V-Form instance by filling its contents. Otherwise, VML is served as a program editor to edit the existing V-Form instances. All of the information carried by V-forms can be modified by VML and new V-Form instances can be created by V-Form grouping and degrouping from existing V-Form instances. VML can be implemented as a menu-driven and prompt-oriented V-Form manipulator such that VML can operate in a what-you-see-is-what-you-get manner. Just like other form manipulations, VML provides users several convenient operations to manipulate V-Forms at will.

4.2 DEFINITION OF VML

[Definition 4.1] Let any two objects $x, y \in \Sigma$. The string transformation $T : \Sigma^* \rightarrow \Sigma^*$ such that $y \in T(x)$ has the following five transformations:

1. Inserting transformation

$$w_1w_2 \qquad\qquad w_1aw_2 \quad \text{for all } a \in \Sigma$$

2. Updating transformation

$$w_1aw_2 \qquad\qquad w_1bw_2 \quad \text{for all } a, b \in \Sigma, a \neq b$$

3. Removing transformation

$$w_1aw_2 \qquad\qquad w_1w_2 \quad \text{for all } a \in \Sigma$$

4. Grouping transformation

$$w_1w_2 \qquad\qquad w_1Aw_2 \quad \text{for all } A \in \Sigma^*$$

5. Degrouping transformation

$$w_1Aw_2 \qquad\qquad w_1w_2 \quad \text{for all } A \in \Sigma^*$$

where $w1, w2 \in \Sigma^*$.

[Definition 4.2] A **V-Form Manipulation Language** VML is a five tuples

VML = (O, I, F, T, S)

where

O is a set of the input objects. The object may be a V-Form type or an original V-Form instance.

I is a set of intermediate objects. Each intermediate object is a working V-Form.

F is the collection of final objects.

T is the string transformation. Each transformation carries an object $x \in O$ or $x \in I$ to a new object $\in T(x)$.

S is a strategy function, different strategies produces different results.

[Definition 4.2] A **V-Form process**, denoted by **VfP**, is defined as a sequence of transformations which gradually derive an input object to a final object.

Exampll 4.1. Given an input object $O = w_1aAbcw_2$, and a final result(object) $F = w_1adBcbw_2$, where $a, b, c, d \in \Sigma$, $w1, w2, A, B \in \Sigma^*$, then VfP may be the following sequence of transformations:

$O = w_1aAbcw_2$

$w_1adAbcw_2$ w_1adbcw_2 $w_1adcbcw_2$

$w_1adBcbcw_2$ $w_1adBcbw_2 = F$

[Lemma 4.1] Different VfP may obtain a same final object.

Example 4.2. Consider Example 4.1, a different VfP may be used to derive the same final object by the following transformation sequence:

$I = w_1aAbcw_2$

$w_1adAbcw_2$ w_1adbcw_2 $w_1adBbcw_2$

$w_1adBccw_2$ $w_1adBcbw_2 = F$

Fig. 4.1 shows a graphical interpretation of VfP for Example 4.1. Each horizontal branch indicates an updating transformation U, a right-upper diagonal branch indicates an inserting transformation I, a right-lower diagonal branch indicates a removing transformation R, a vertical upward branch indicates a grouping transformation G, and a vertical downward branch indicates a degrouping transformation D.

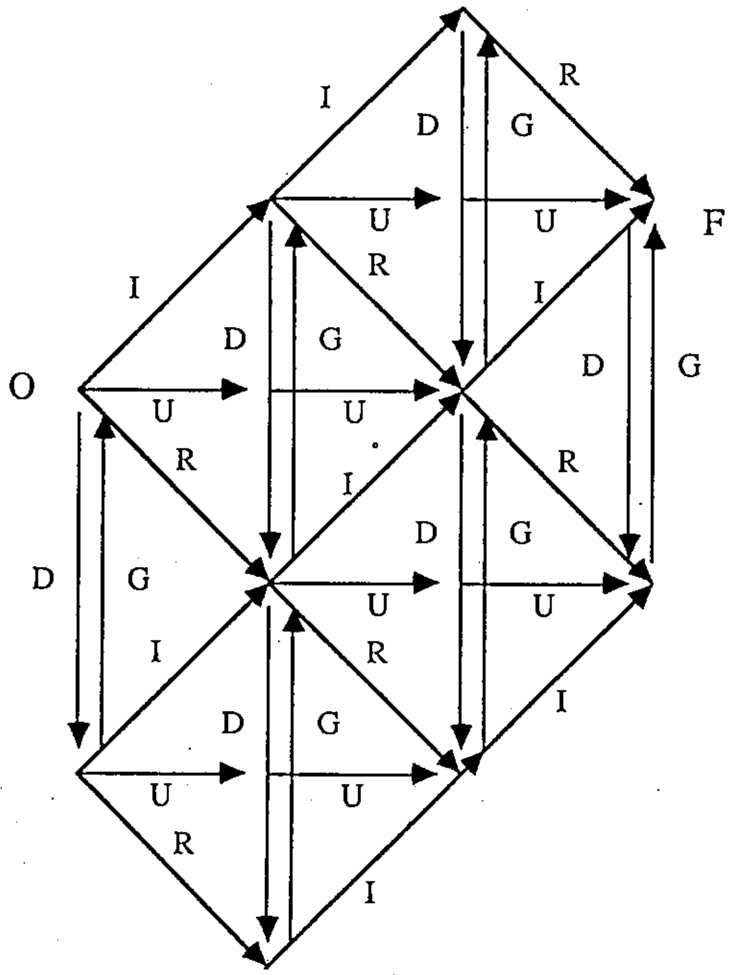


Fig. 4.1 A graphical interpretation of VfP for Example 4.1.

Each VfP which derives the initial object O to the final object F constitutes a path from O to F, but with the restriction that such a path contains just one vertical-upward branch and one vertical-downward branch. There exists many such paths as can be seen from Fig. 4.1.

[Definition 4.4] A **non-procedural language** is defined as a manipulator that obtains the same final object from any given input object by more than one VfPs.

[Lemma 4.2] VML is a non-procedural language.

4.2 SYNTAX RULES

As mentioned earlier, VIPS interface language is generated by an inferred grammar. The V-Form Manipulation Language is one part of this interface language, so VML is also generated by an inferred grammar. In this section, the syntax of V-Form Manipulation Language is designed. Table 4.1 shows the syntax of VML.

Table 4.1 Syntax of V-Form Manipulation Language

```
1  <VML>        :==  <Commands> <End>
2  <Commands>:==     <Command> <Commands>
                     |NULL
3  <Command>  :==    <Open>|<Close>|<Display>|<Insert>|<Update>
                     |<Remove>|<Group>|<Degroup>|<Save>
4  <Open>       :==  OPEN <Filename>
5  <Close>      :==  CLOSE <Filename>
6  <Display>    :==  DISPLAY <Identifier>
                     |DISPLAY <Identifier> <Character>
```

```
                           |DISPLAY <Identifier> <Embedded>
                            <Number>
7  <Insert>         :==    INSERT <Type> <Attribute> <Indicator>
                            <Identifier> AT <Position> WITH <Size>
                           |INSERT <Embedded> TO <Identifier>
8  <Update>         :==    UPDATE <Identifier>
                           |UPDATE <Identifier> <Character>
                           |UPDATE <Identifier> <Embedded>
                            <Number>
9  <REMOVE>         :==    REMOVE <Types> <Identifier>
                           |REMOVE <Embedded> <Number> FROM
                            <Identifier>
10 <Group>          :==    GROUP <Filename> <Identifier> AT
                            <Position> WITH <Size> [<Identifier>]
11 <Degroup>        :==    DEGROUP <Identifier> TO <Filename>
12 <Save>           :==    SAVE
                           |SAVEAS <Filename>
13 <End>            :==    EXIT|QUIT
14 <Filename>       :==    <Identifier>
15 <Character>      :==    ATTRIBUTE|INDICATOR
16 <Embedded>       :==    COND|ACTION|PROC
17 <Position>       :==    <Integer> <Integer>
18 <Size>           :==    <Integer> <Integer>
19 <Number>         :==    <Integer>
20 <Integer>        :==    <Digit>|<Digits>
21 <Digits>         :==    <Digit>|<Digits>
                           |NULL
```

```
22 <Digit>       :== 0|1|2|3|4|5|6|7|8|9
23 <Type>        :== TEXT|GRAPH|BIT-MAP|ANIMATION
                     |VOICE|RULE
24 <Types>       :== TEXT|GRAPH|BIT-MAP|ANIMATION
                     |VOICE|RULE|VFORM
25 <Attribute>   :== NORMAL|INVERSE|BOLD|FLASH
26 <Indicator>   :== ON|OFF
27 <Identifier>  :== <Letter> <Alphanums>
28 <Alphanums>:==    <Alphanum> <Alphanums>
                     |NULL
29 <Alphanum>    :== <Digit>|<Letter>
30 <Letter>      :== A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V
                     |W|X|Y|Z
                     |a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
                     |,|.|;|:|'|"|<|>|?|/|[|]|{|}N!|@|$|%|^|&|*|(|)|-|+|=|`|~
```

The inferred grammar is a context-free programmed grammar. The grammar G_m shown in Table 4.1 which is a context-free programmed grammar. For clearity, the definition of context-free programmed grammar is given in the following.

[Definition 4.5] A **programmed grammar** G_p is a five-tuple (V_N, V_T, J, P, S), where V_N, V_T, and P are finite sets of nonterminals, terminals, and productions, respectively. S is the starting symbol, S Œ

V_N. J is a set of production labels. A production in P of the form

$$(r)\, \alpha \rightarrow \beta \quad S(U) \quad F(W)$$

$\alpha \rightarrow \beta$ is called the core, where , $\alpha \in V^* V_N V^*$ and , $\beta \in V^*$. (r) is the label, $r \in$ J. U is the success field and W the failure field, U, W $\in$ J.

The programmed grammar operates as follows. Production (1) is applied first. In gereral, if one tries to apply production (r) to rewrite a substring , and the current string Ω contains the substring α, then (r) $\alpha \rightarrow \beta$ is applied and the next production is selected from the success go-to field U. If the current string Ω does not contain α, then the production (r) is not used(i.e. Ω is not changed) and the next production is selected from the failure go-to field W. If the applicable go-to field contains $\emptyset$, the derivation halts.

[Definition 4.6] If the core of the production of a programmed grammar is of the context-free form (i.e. $A \rightarrow \beta$), then the grammar is called a context-free programmed grammar.

The syntax of VML grammar G_m is a context-free programmed grammar. The success and failure fields of VML grammar is as follows. When its first production (1) is applied, its success field is the second production (2). Similarly, success field of (2) is (3). From production (4) to (12), all of their success fields are same, i.e. production (2), this forms a loop. The success field of production (13) is Ø which is a termination. All other productions have its right production label as its success field. The content of failure field for each production is obvious, so we do not bother to explain them further. It is shown that the language generated by a context-free programmed grammar can be a context-sensitive language. So the interface language generated by VML grammar G_m can also be a context-sensitive language.

CHAPTER 5
VML OPERATIONS

5.1 INTERACTION STYLES

As mentioned in Chapter 4, the V-Form manipulation language is a menu-driven prompt-oriented manipulator. Fig. 5.1 shows three regions of the screen: scratch-pad region, interactive region, and command region.

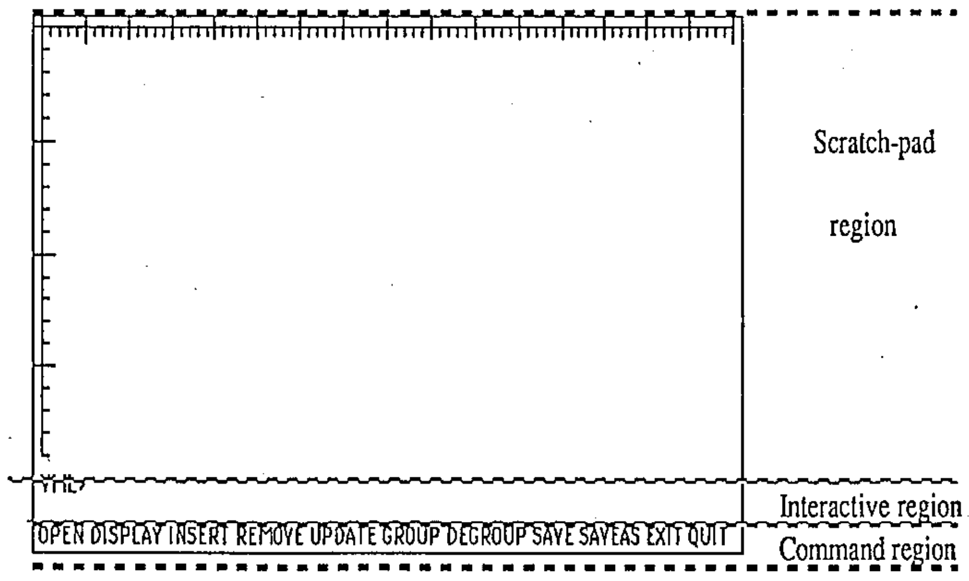

Scratch-pad region
VML>
Interactive region
OPEN DISPLAY INSERT REMOVE UPDATE GROUP DEGROUP SAVE SAVEAS EXIT QUIT
Command region

Fig. 5.1 Three regions of the screen.

All of VML commands (i.e. operations) are shown in the command region at screen buttom , called the command region. The scratch-pad region is the area in which a what-you-sketch-is-what-you-get manner operation is proceeded. In the interactive region, users can give commands in three styles: all-prompting, partial-prompting, and no-prompting. For example, the inserting operation in all-prompting style looks like --

```
VML> insert
_Type(TEXT/GRAPH/BIT-MAP/ANIMATION/RULE/COND/ACTION/PROC):
text
_Attribute (<N>:NORMAL / <I>:INVERSE / <B>:BOLD / <F>:FLASH): n
_Indicator (ON/OFF): on
_Naming: Test .i12;_Position (row <1-18>): 16
_Position (col <1-77>): 30
_Size (width <1-48>): 20
_Size (height <1-3>): 2
 /* Now fill in the data of TEXT:Test in scratch-pad region */
VML> ,
```

in partial-prompting style, it looks like --

partial-prompting example (1)

```
VML> insert text n on Test
_Position (row <1-18>): 16
_Position (col <1-77>): 30
_Size (width <1-48>): 20
_Size (height <1-3>): 2
 /* Now fill in the data of TEXT:Test in scrach-pad region */
VML>
```

partial-prompting example (2)

```
VML> insert text
_Attribute (<N>:NORMAL / <I>:INVERSE / <B>:BOLD / <F>:FLASH): n
_Indicator (ON/OFF): on
_Naming: Test TO 16 30
_Size (width <1-48>): 20
_Size (height <1-3>): 2
 /* Now fill in the data of TEXT:Test in scrach-pad region */
VML>
```

in no-prompting style, it looks like --

```
VML> insert text n on Test to 16 30 with 20 2
 /* Now fill in the data of TEXT:Test in scrach-pad region */
VML>
```

Notice that all commands (operations) can use first two characters. For example, users can key in:

```
VML> in text n on Test to 16 30 with 20 2
```

is also a valid command for inserting.

5.2 BASIC FUNCTIONS

VML in VIPS is a V-Form manipulator which provides a device for users to manipulate V-Forms. Some basic functions in VIPS are as follows.

Load function:

This function allows users to load V-Form types or V-Form instances from memory to system explicitly. When the V-Form type or instance is loaded, an entry in instance table is created and all information will be read in from file, the corresponding internal structure is built. Users can manipulate it later. For example, to load an instance(CAI.INS) into system, key in the command

```
VIPS> load cai.ins
```

Store function:

This function is an inverse of the Load function. It frees the entire corresponding internal structure and delete its corresponding entry in instance table. After a V-Form type or instance is stored, it is deleted from the system but can be load again if needed. For example, to store the instance CAI.INS , key in the command

```
VIPS> store cai.ins
```

Type function:

This function is a V-Form outline scanner which allows users to see the first V-Form page previously existed in system. There is no need to invoke VDL or VML, any V-Form type or instance can be viewed by this function. For example, to see the first V-Form page of CAI.INS on screen, key in the following command

```
VIPS> type cai.ins
```

VDL function:

This function is used to invoke V-Form Definition Language to define the V-Form types or V-Form instances. For example, key in

```
VIPS> vdl
```

entering into the environment of VDL which allows users to define the V-Form types or V-Form instances.

VML function:

This function is used to invoke V-Form Manipulation Language to manipulate V-Form types or V-Form instances. For example, key in

```
VIPS> vml cai.ins
```

entering into the environment of VML which provides several operations to manipulate V-Form instance CAI.INS.

Copy function:

This function allows users to copy V-Form type or V-Form instance to another one. For example, to get a new V-Form instance THEOREM.INS from V-Form instance CAI.INS, key in the command

```
VIPS> copy cai.ins to theorem.ins
```

Run function:

This function is used to invoke the direct V-Form interpreter to interprete the V-Form instance obtained from VDL or VML. For example, to interprete the V-Form instance CAI.INS, key in the command

```
VIPS> run cai.ins
```

Help function:

This function will show various kinds of information for users. It provides many help messages which assist users to understand the usage of the system. For example, to understand TYPE function, key in the command

```
VIPS> help type
```

Exit function:

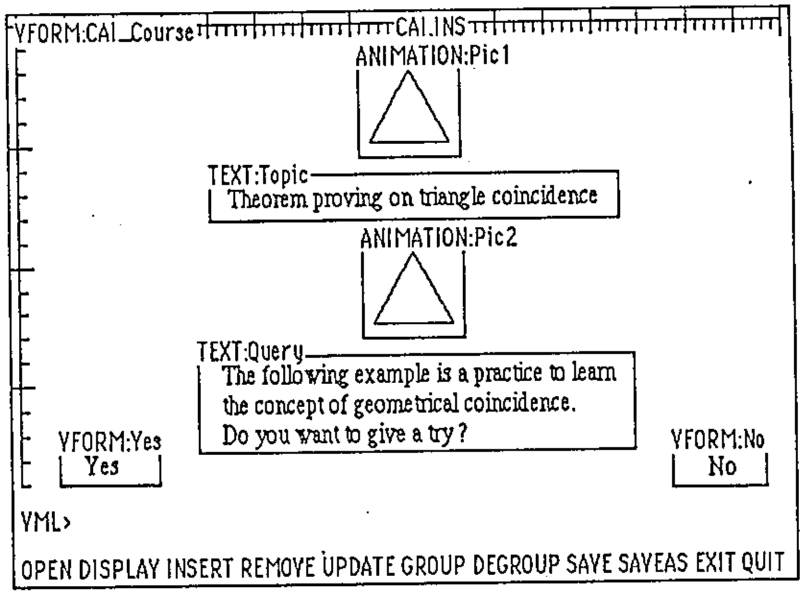
This function terminates the execution of the visual programming synthesizer system. After using EXIT command, the system returns to host operating system. For example, to terminate the execution of VIPS, key in the command

```
VIPS> exit
```

The above basic functions are built on the top of VDL, VML, and direct V-Form interpreter. These functions integrate V-Form components and manage their V-Form types and V-Form instances consistently before entering into the environment of VDL or VML. In the following sections, only the VML operations will be described. As for VDL operations, please consult the M.S. thesis of M. C. Lu[18].

5.3 DISPLAYING

A V-Form type or a V-Form instance is a graph structure with V-Form pages as its elemental nodes. Each V-Form page is defined as a V-Form window on the screen. The displaying operation is served as a page-tuner which allows users to tune each V-Form page in the V-Form type or V-Form instance. The V-Form page which is displayed and shown on screen is the current V-Form page. Each V-Form page has a instance name at the center of the first row on the screen, and a V-Form page name at the upper-left corner of screen. In a V-Form window, there is a V-Form name and its type that is shown at the upper-left corner for each V-Form(atom or non-atom), and the content of V-Form is shown within it. Fig. 5.2 shows a typical V-Form page, where CAI.INS is the instance name(the same as its file name), its V-Form page name is Cai_Course now. TEXT:Topic, ANIMATION:Pic1, ANIMATION:Pic2, TEXT:Query, VFORM:Yes, and VFORM:No are their sub-VForm type and name pairs, respectively.



Theorem proving on triangle coincidence

The following example is a practice to learn the concept of geometrical coincidence. Do you want to give a try?

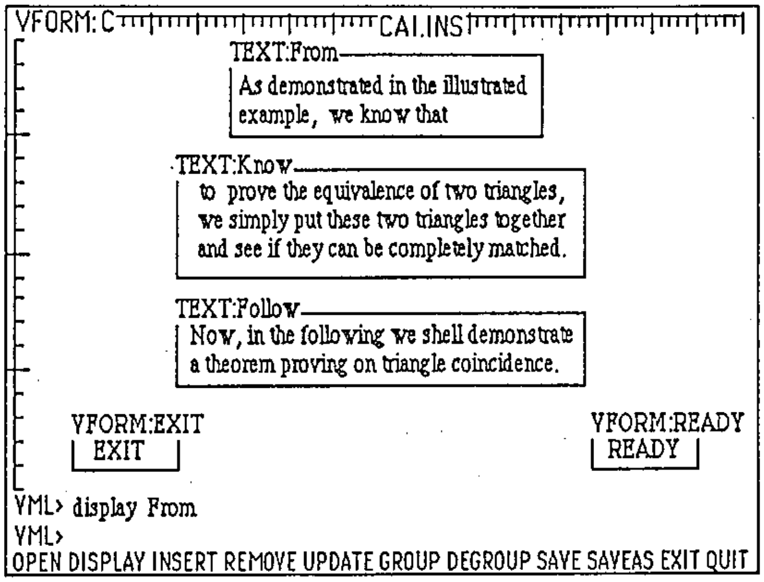
Fig. 5.2 A typical V-Form page (Cai_Course)

The V-Form page name inherits from its father's name. So, if one V-Form page has multiple father V-Forms, it may have different V-Form page names when different paths are used to reach it. The displaying operation searches for the destination V-Form page in the V-Form instance (or type) from the current V-Form page. The destination V-Form page may have two methods to identify it. One is to display its father V-Form name, and the other is to display any non-atom V-Form name which is included in this V-Form page. For example, the third V-Form page can be reached either by displaying its father V-Form name(C) or displaying its non-atom sub-VForm name(From). Fig. 4.2 shows an example for displaying operation by two displaying methods.

```
VML> display c
```

or

```
VML> display from
```


VFORM:C CAI.INS

TEXT:From
As demonstrated in the illustrated
example, we know that

TEXT:Know
to prove the equivalence of two triangles,
we simply put these two triangles together
and see if they can be completely matched.

TEXT:Follow
Now, in the following we shell demonstrate
a theorem proving on triangle coincidence.

VFORM:EXIT
EXIT

VFORM:READY
READY

```
VML> display From
VML>
```

OPEN DISPLAY INSERT REMOVE UPDATE GROUP DEGROUP SAVE SAVEAS EXIT QUIT

Fig. 5.3 An example of displaying operation

If the current V-Form page is now VFORM:Yes, the first command will get the third V-Form page. The second command can be used at any V-Form page to get the third V-Form page. Of course, the sub-VForm name "From" must be unique in this V-Form instance. Otherwise, the first V-Form page which include a sub-VForm name "From", will be displayed.

Because a V-Form instance is an application program which consists of several V-Form pages linked in a complex graph structure. Essentially, the V-Form displaying operation is a graph search and a displaying activity of V-Form. The graph search strategy adapted here is a bread-first traversal on each V-Form node. During the search process, an extra Queue is used to put the unsearched V-Form pages (form control tables), and a Traversed list is used to record those

V-Form pages traversed. The search process starts with the current V-Form page, traversing the remaining reachable part. If destination V-Form page is not found in the remaining part, then a secondary try begins from the rooted(start) V-Form page and to search down those parts untraversed in this V-Form instance. Beside the V-Form page searching and displaying, the displaying operation also contains the retrieval of all information held in V-Forms. The characters of each V-Form consists of its display attribute(may be NORMAL, INVERSE, BOLD, or FLASH) and its indicator(ON or OFF). The attributes and indicators of V-Forms are their displaying characters in V-Form interpreting. The following commands display the attribute and the indicator of V-Form TEXT:Ask.

```
VML> display ask attribute
The ATTRIBUTE of ASK is: NORMAL
VML> display ask indicator
The INDICATOR of ASK is: ON
```

The other information held in V-Forms are their embedded components, such as conditions, actions, or procedures. The displaying operation can retrieve all these components in the same way. For example, the first procedure of V-Form Exit in VFORM:Yes can be displayed through the command

```
VML> display  c  proc 1
The 1-th PROC of C is: DISPLAY  TEXT:Good.
```

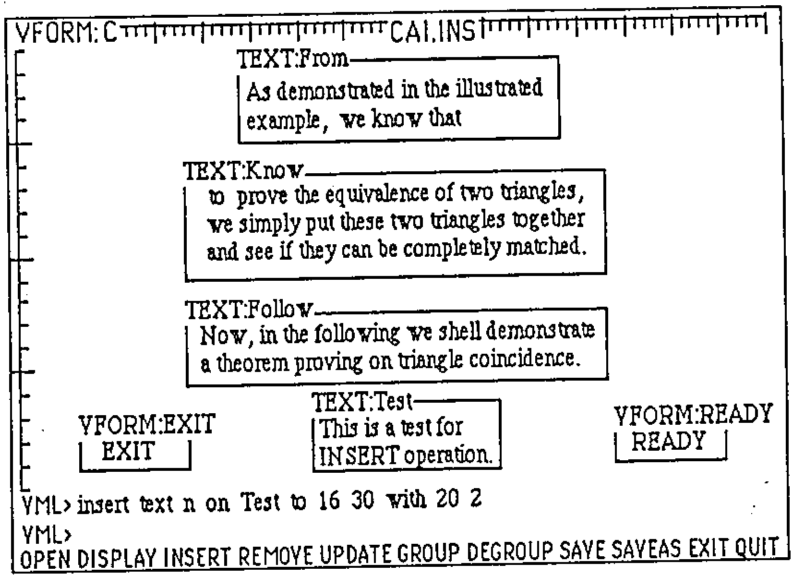
The displaying of conditions and actions are just the same.

5.4 INSERTING

Inserting operation is used to add an atom V-Form to the current V-Form page or to add embedded components(such as conditions, actions, or procedures) to some V-Form of the current V-Form page. All information of an atom V-Form must be specified in the INSERT command, this includes the content type, name, display attribute, indicator of V-Form, and its window coordinate and size. The following is an example. Assume the current V-Form page is VFORM:No. An atom V-Form with content of type text is now inserted into VFORM:No by using the command

```
VML> insert text n on Test to 16 30 with 20 2
/* Cursor is now at the beginning of TEXT:Test, user can now
key in the text "This is a test for INSERT operation." */
VML>
```

In this command, a text type V-Form TEXT:Test is added, its window position is at row 16 and column 30, its window size is width 20 and height 2. Besides, its attribute is NORMAL and indicator is ON. When complete this command, cursor will stay at the beginning of V-Form TEXT:Test waiting user to key in its text type content. After finishing this inserting operation, a new V-Form page VFORM:C, as shown in Fig.5.4 is obtained.



As demonstrated in the illustrated example, we know that

to prove the equivalence of two triangles, we simply put these two triangles together and see if they can be completely matched.

Now, in the following we shell demonstrate a theorem proving on triangle coincidence.

This is a test for INSERT operation.

```
VML> insert text n on Test to 16 30 with 20 2
VML>
```

Fig. 5.4 An inserting operation example.

Unlike displaying operation, inserting operation and the later ones will alter the internal structure of V-Form instance manipulated. In this example, when TEXT:Test is added, a form control node(FCN) which specifies this V-Form is added to the form control table(FCT) VFORM:C, the Mode Nodes(MNs) which records its attribute and content is linked by a pointer from the FCN of TEXT:Test. The logical structure of VFORM:C after inserting is shown in Fig. 5.5.

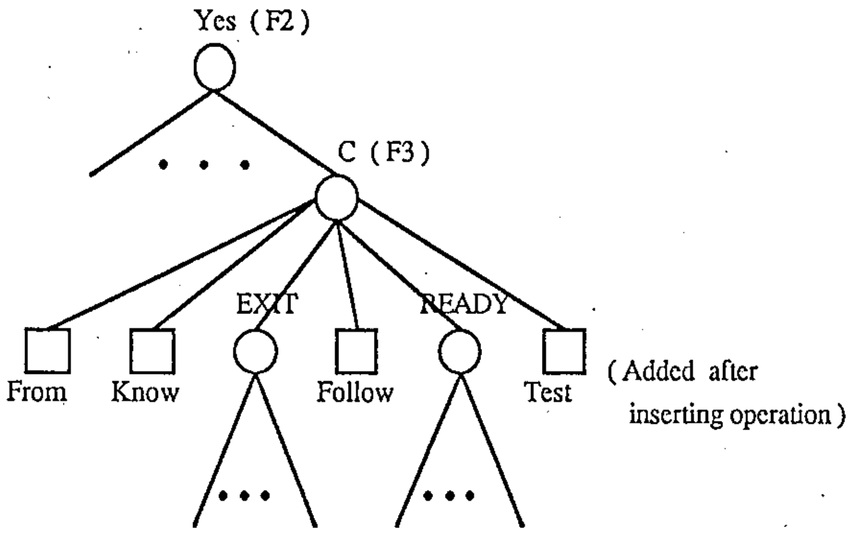


Fig. 5.5 The logical structure of VFORM:C after inserting TEXT:Test.

Another function of inserting operation is to add an embedded component to the V-Form under consideration. Each V-Form(atom or non-atom) may have several conditions, actions, and procedures. Condition must be added in a manner of condition-action pair, while action and procedure can be added alone. The inserting operations of condition-action pair and procedure are described in the following in line mode commands

```
VML> insert cond B
CONDITION: IF error_count LESS THEN 3
ACTION: ADD error_count 1
VML>

VML> insert proc C
```

```
PROCEDURE: DISPLAY TEXT:Good
VML>
```

The first INSERT command adds a condition-action pair to VFORM:C, it checks whether the query count is less then 3 or not. The second INSERT command adds a procedure to VFORM:C to update the V-Form page CAI_Course.

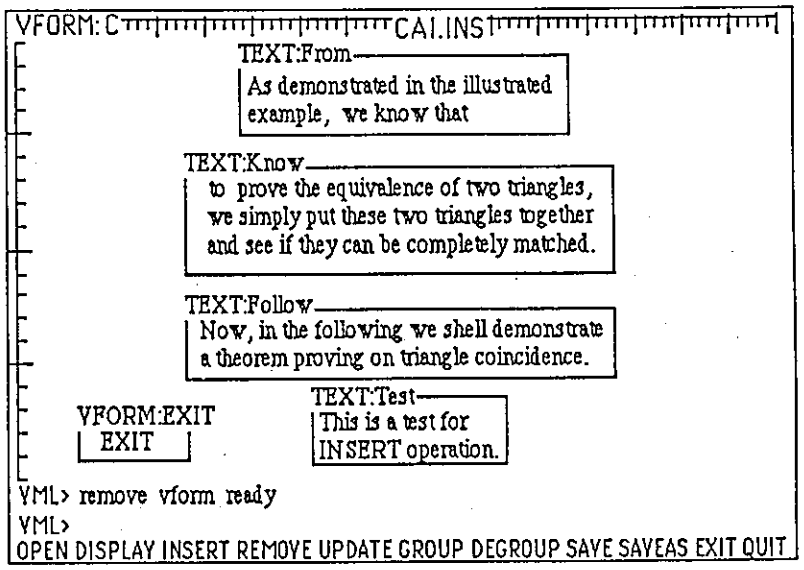
5.5 REMOVING

The removing operation deletes the link between the V-Form removed and its form control table. So this operation comes two effects on removed V-Form in the V-Form instance. First, it can be served to inverse the inserting operation. That is, it removes an atom from the current V-Form page to cause a deletion of the FCN corresponding to the atom from FCT. Then this FCN and all branches linked to it will be freed. Second, it removes an non-atom V-Form(with type VFORM) from current V-Form page. At this time, a structural relationship will be tested. If now they are structural unrelated, i.e. they are two independent V-Form instances, then removed V-Form instance will be freed, otherwise, it just breaks the link between this removed V-Form and the current V-Form page. Of course, when removing operation has completed, its corresponding V-Form window will be erased from current V-Form page. To removes an atom (TEXT:Test) from current V-Form page VFORM:C, key in the following removing command

```
VML> remove text test
VML>
```

Another removing operation example is to break the link between sub-VForm VFORM:Ready and the current V-Form page. After completing this operation, the resultant V-Form page as shown in Fig. 5.6 is displayed.

```
VML> remove vform ready
VML>
```


VFORM: C
CAI.INS
TEXT:From
TEXT:Know
TEXT:Follow
TEXT:Test
VFORM:EXIT
EXIT
OPEN DISPLAY INSERT REMOVE UPDATE GROUP DEGROUP SAVE SAVEAS EXIT QUIT

As demonstrated in the illustrated example, we know that

to prove the equivalence of two triangles, we simply put these two triangles together and see if they can be completely matched.

Now, in the following we shell demonstrate a theorem proving on triangle coincidence.

This is a test for INSERT operation.

```
VML> remove vform ready
VML>
```

Fig. 5.6 The V-Form page after removing sub-VForm Ready.

Another function of removing operation is to delete the embedded components of V-Forms in current V-Form page. The first procedure of VFORM:C is removed by using the following command

```
VML> remove proc 1 from C
VML>
```

The number of procedure which will be removed and the numbers of actions or conditions must be specified in REMOVE command

5.6 UPDATING

When the object manipulated by VML is a V-Form type, the updating operation is used to fill in the content of each V-Form. On the other hand, if the object manipulated by VML is a V-Form instance, the updating operation is used to modify the contents of V-Forms. The information manipulated by users in updating operation are all formatted data. So, the contents of V-Forms updated by this operation are represented externally. That is, the updating operation is performed in a what-you-sketch-is-what-you-get manner. The following command demonstrates the updating operation.

```
VML> update test
/* Cursor is now at the beginning of TEXT:Test, user can now
edit its content at will */
VML>
```

The embedded components are modified by updating operation in a prompting manner. That is, when embedded component is updated, old content is displayed and allowed to key in new content. For example, the first condition of VFORM:Yes in above example is modified by the following command

```
VML> update B cond 1
The 1-th CONDITION of YES is: IF error_count LESS THEN 3
_New CONDITION is: IF error_count LESS THEN 5
VML>
```

After finishing this operation, the first condition of VFORM:B has been changed to "IF error_count LESS THEN 5". If the embedded component that is modifying does not exist, the updating operation will be aborted and a error message is responded. The characters (such as display attribute or indicator) also can be modified by updating operation. For example, to change the attribute of TEXT:Query to INVERSE, key in the following command

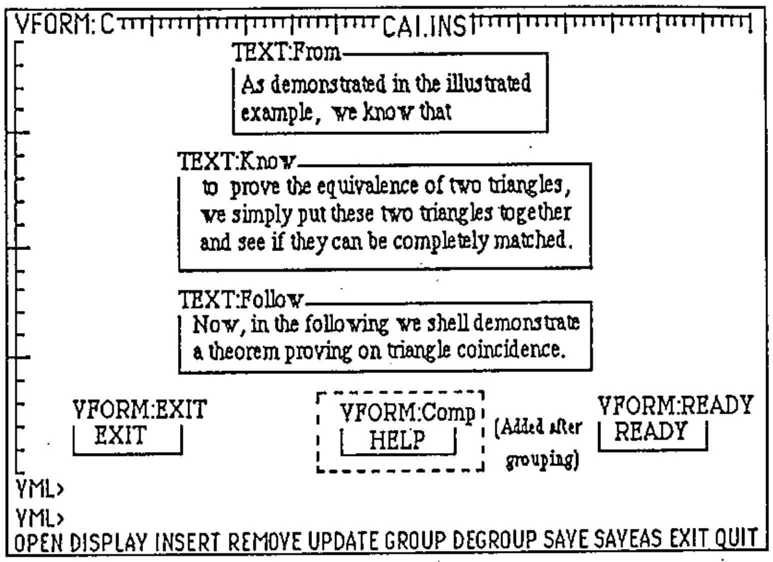
```
VML> update Again attribute
The ATTRIBUTE of QUERY is NORMAL.
_New attribute (<N>:NORMAL / <I>:INVERSE / <B>:BOLD /
                    <F>:FLASH):i
VML>
```

The indicator modification is just the same as attribute updating. Notice that, the updating operation will modify the contents of

V-Forms, but it cannot alter the internal structure of V-Form instance being manipulated.

5.7 GROUPING

Grouping operation is used to group two independent V-Form instances together. It takes the current V-Form page of manipulated instance as a destination V-Form page, and then takes any V-Form page in grouped instance as a root which will be inserted into the destination V-Form page. Finally, a new V-Form instance is created from the current manipulated instance. But it will leave the grouped V-Form instance unchanged. Fig. 5.7 shows an example of grouping. This grouping operation adds a complementary course, V-Form instance COMP.INS(in file COMP.INS), to the current V-Form page VFORM:C of the V-Form instance CAI.INS. Here, the VFORM:Comp in COMP.INS is used as a rooted V-Form page which is added to the current V-Form page. In this GROUP command, "HELP" is used as the alias of VFORM:Comp in VFORM:C. If the alias is absent, the content of VFORM:Comp will be "Comp". The grouped V-Form VFORM:Comp has a "FLASH" attribute just likes all the other V-Forms with type "VFORM", and an indicator with value "ON".



VFORM:C CAI.INS

TEXT:From
As demonstrated in the illustrated example, we know that

TEXT:Know
to prove the equivalence of two triangles, we simply put these two triangles together and see if they can be completely matched.

TEXT:Follow
Now, in the following we shell demonstrate a theorem proving on triangle coincidence.

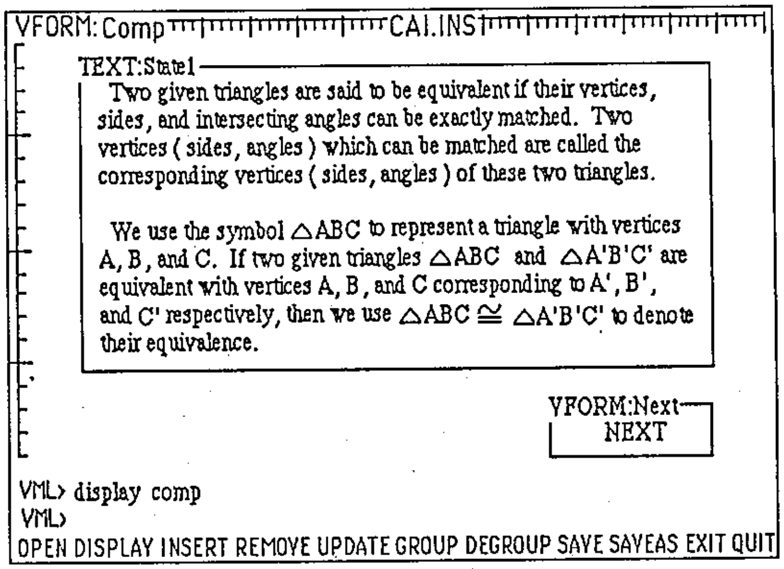
VFORM:EXIT
EXIT

VFORM:Comp
HELP
(Added after grouping)

VFORM:READY
READY

```
VML>
VML>
OPEN DISPLAY INSERT REMOVE UPDATE GROUP DEGROUP SAVE SAVEAS EXIT QUIT
```

(a) Current V-Form page VFORM:C (in frame F3).



VFORM:Comp CAI.INS

TEXT:State1
Two given triangles are said to be equivalent if their vertices, sides, and intersecting angles can be exactly matched. Two vertices (sides, angles) which can be matched are called the corresponding vertices (sides, angles) of these two triangles.

We use the symbol $\triangle ABC$ to represent a triangle with vertices A, B, and C. If two given triangles $\triangle ABC$ and $\triangle A'B'C'$ are equivalent with vertices A, B, and C corresponding to A', B', and C' respectively, then we use $\triangle ABC \cong \triangle A'B'C'$ to denote their equivalence.

VFORM:Next
NEXT

```
VML> display comp
VML>
OPEN DISPLAY INSERT REMOVE UPDATE GROUP DEGROUP SAVE SAVEAS EXIT QUIT
```

(b) Rooted V-Form page VFORM:Comp in COMP.INS.

Fig. 5.7 An example of grouping operation.

Fig. 5.7(a) shows a new structural relationship between two grouped V-Form instances. Fig. 5.8 shows the logical structure after grouping.

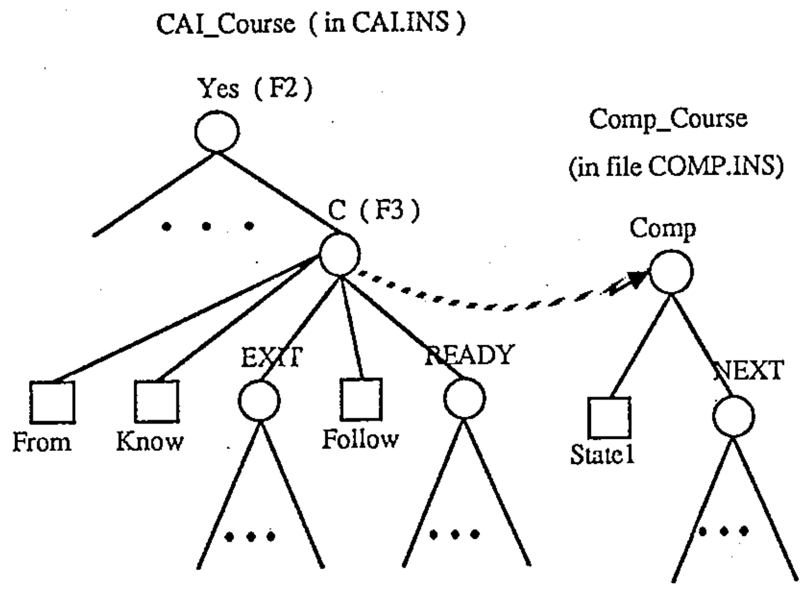


Fig. 5.8 Logical structure of CAI.INS after grouping operation.

5.8 DEGROUPING

Degrouping operation is just the inverse of grouping operation. The difference between removing and degrouping operations is that the degrouped instance will be extrally saved as a new one. Removing, grouping, and degrouping operations can all change the branches of control flow. A structural unrelationship must also be checked, just as removing a "VFORM" type V-Form, before performming this degrouping operation. The following command is used to degroup the VFORM:Comp from the current V-Form page VFORM:C in CAI.INS instance, and the degrouped object is saved in V-Form instance COMP.INS (COMP.INS file).

```
VML> degroup comp to comp.ins
VML>
```

It is an inverse action of the above grouping operation. The logical structure after degrouping is shown in Fig. 5.9, where two independent V-form instances are obtained.

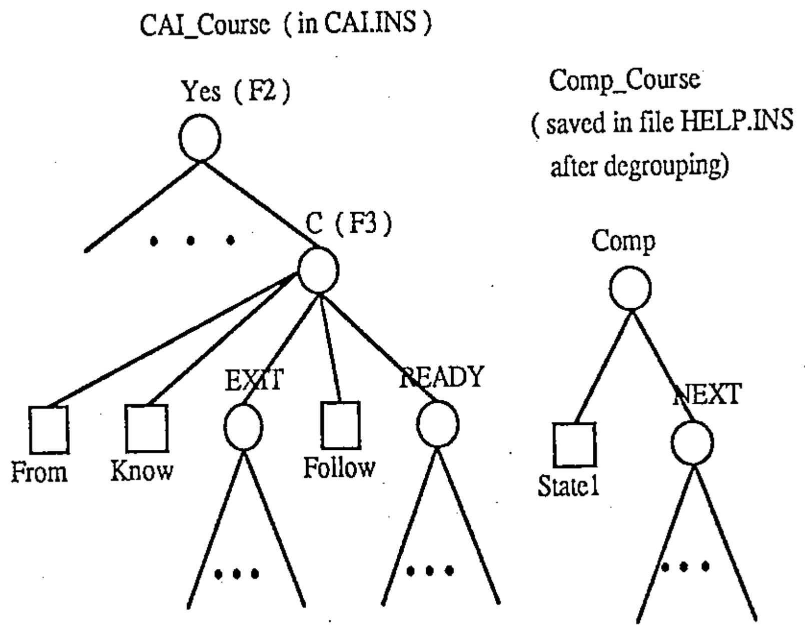


Fig. 5.9 Logical strutures after degrouping operation.

5.9 INSTANCE HANDLING

VML provides several instance handling commands for users to manage instance in this environment.

Save:

This command is used to save the immediate result of manipulated instance. When a V-Form instance is manipulated by VML, it is duplicated first and this duplicate is served as a working instnace for manipulation. SAVE command will overwrite an original V-Form instance with its working instance.

Saveas:

The difference between SAVE and SAVEAS commands is that SAVEAS command saves the immediate result to a new instance name instead of the original one. That is, it is served as an instance duplication for the immediate object.

Close:

CLOSE command closes the working instance. If there is any change during the manipulation, a query for saving its changes is given by system. When the user wants to save it, the working instance overwrites its original one and stops its manipulations. Otherwise, the working instance is aborted and its manipulation is stopped. If no change occurs, system deletes the working instance and stop its manipulation directly.

Open:

OPEN command can be used to open another V-form instance after CLOSE command for manipulation.

Exit and quit:

These two commands will cause to leave VML environment. The difference between them is that EXIT command will ask users whether to save the change or not before leaving it, while QUIT command leaves it right away.

CHAPTER 6
AN ILLUSTRATED EXAMPLE

6.1 COURSEWARE DEVELOPMENT

From previous discussion, an instant thought about the V-Form manipulation language is its application in CAI courseware development. Indeed, our visual programming synthesizer can be served as an authoring system, because it may exhibit the following functional characteristics:

1. Content creation -- the updating operation can be used to edit the contents of a specified V-Form type, just like the editing phase in V-Form Definition Language(VDL) which allows users to enter the data to be displayed or stored.

2. Lesson definition -- the inserting, removing, grouping, and degrouping operations can be used to specify the structure of individual lesson. They are related to the information display, student response, and branch control.

3. Courseware management -- VML allows a user to define an instructional course according to his strategy(e.g. drill, quiz, tutorial etc.). The direct V-Form interpreter can be used to simulate what his strategy will be.

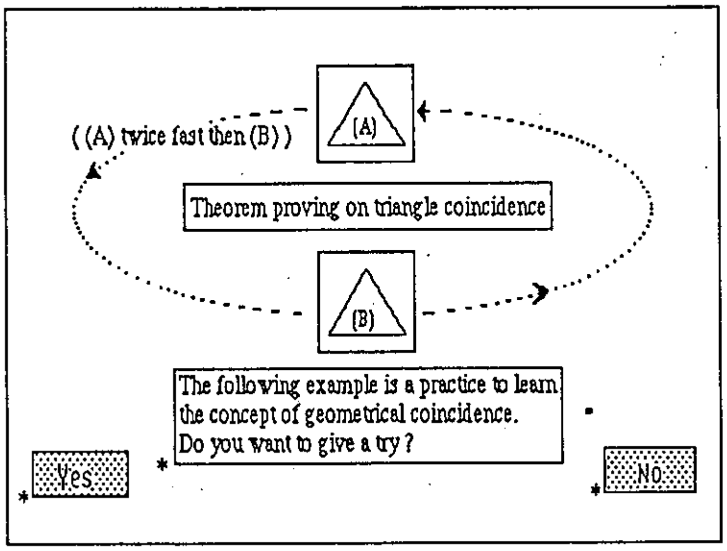
4. Provides alternate facilities for man-machine interaction -- VML provides different prompting styles for different level of users.

In the following, we will described how a CAI courseware can be easily created by VIPS. Fig. 6.1 shows a courseware for the purpose of demonstrating the concept of geometrical coincidence. Fig. 5.1(a) is the first frame of the courseware with two triangles circling around in different speeds, one is twice faster than the other so that both circling triangles will stop and coincide at the top one. A touch/selection the VFORM:Yes starts the practice to learn the concept of geometrical coincidence in the second V-Form page(the second frame F2). The third V-Form page(frame F3) exhibits the facility of delay displaying, the dotted line denotes the delay of a period of time between two consecutive V-Form.

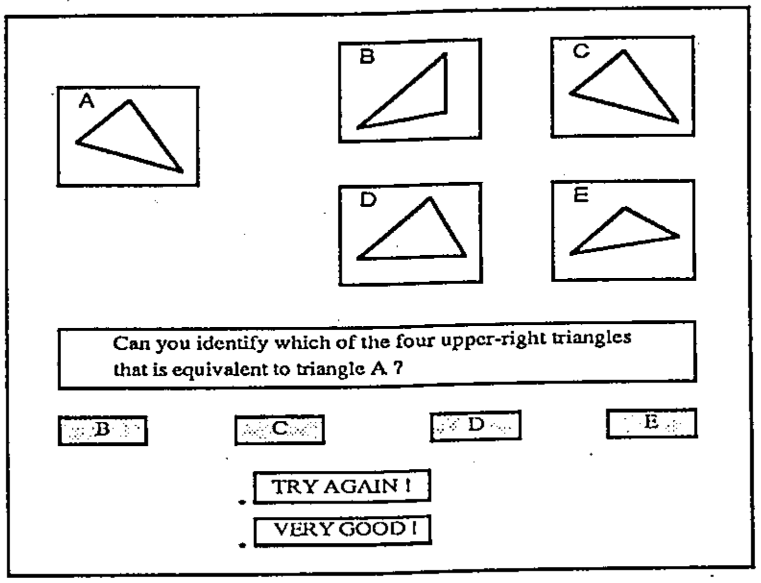
A theorem proving on triangle equivalence is demônstrated from F4 to F9. Notice that, in these frame, the whole screen is partitioned into two regions. the right-lower region is a window which allows a user to touch/select the appropriate answer within it. These answer-question activities are the theorem-proving process of triangle equivalence by using geometrical coincidence method. An asterisk lay on or beside the dotted line means the display sequence of V-Forms, and the display of the content depends on a proper selection of a fact. When a V-Form is marked with an asterisk at the left-lower corner, then the indicator of such a V-Form is now "OFF", indicats that the V-Form is not displayed on the screen initially in its interpreting, only to display when some kind of condition specified by users is satisfied.

To design such a CAI courseware, teachers with or without programming language background can concentrate in describing their specialized knowledge as the process of how to get the desired courseware. The interface language which provides some user-friendly facilities will allow them easily to implement the courseware.

Once a teacher has prepared his/ger own well-written sheets in V-Forms according to the courseware, he/she can interactively define the individual V-Form page with a sequence of commands in VDL, each non-atom V-Form(with "VFORM" content type) can be added by grouping operation or deleted by degrouping operation. In this way, an entire well-constructed V-Form instance (courseware) can be obtained. Furthermore, the displaying, inserting, removing and updating operations can be used to modify the V-Form instance at will. The embedded components can also be processed by these operations. Fig. 6.2 shows the self-synthesized control flow of the V-Form instance illustrated in Fig. 6.1.

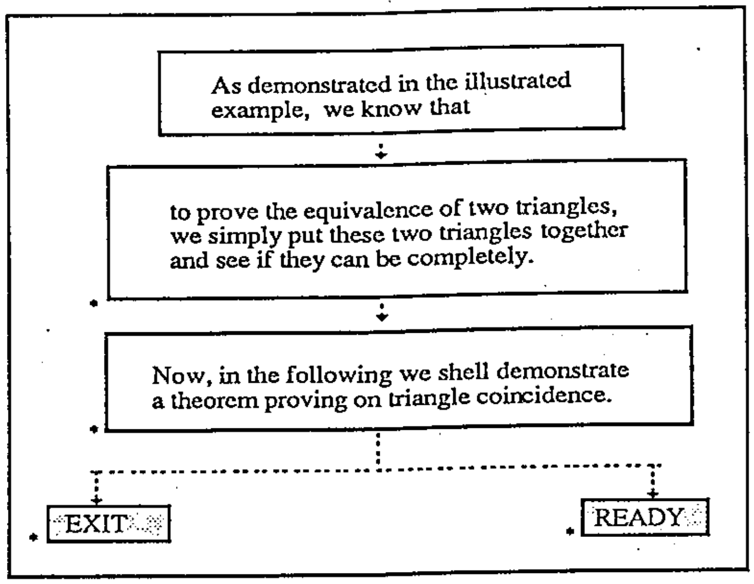

((A) twice fast then (B))
(A)
Theorem proving on triangle coincidence
(B)
The following example is a practice to learn the concept of geometrical coincidence.
Do you want to give a try ?
Yes
No
*
*
*

(a) F1



Can you identify which of the four upper-right triangles that is equivalent to triangle A ?

(b) F2



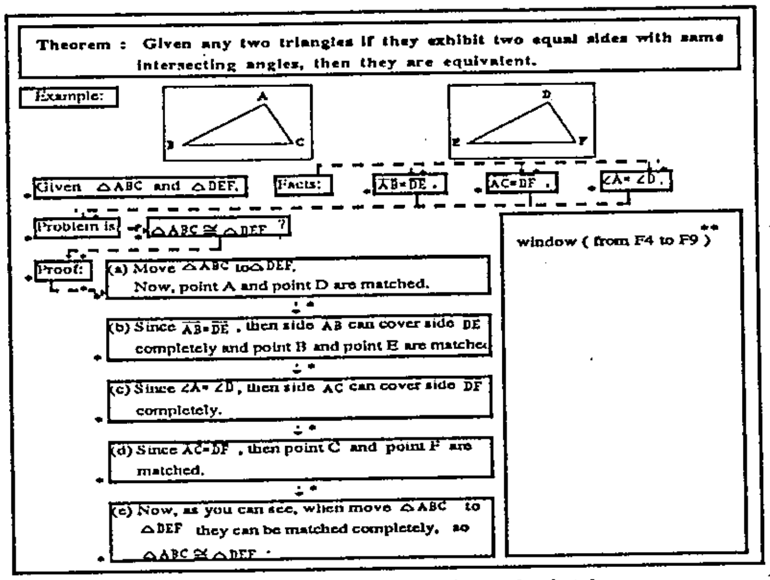
As demonstrated in the illustrated example, we know that

to prove the equivalence of two triangles, we simply put these two triangles together and see if they can be completely.

Now, in the following we shell demonstrate a theorem proving on triangle coincidence.

Note: $\dashrightarrow$ denotes delay of a period of time between two consequtive V-Form.

(c) F3



Theorem : Given any two triangles if they exhibit two equal sides with same intersecting angles, then they are equivalent.

Example:

Given $\triangle ABC$ and $\triangle DEF$.

Facts: $\overline{AB}=\overline{DE}$. $\overline{AC}=\overline{DF}$. $\angle A=\angle D$.

Problem is $\triangle ABC \cong \triangle DEF$?

Proof: (a) Move $\triangle ABC$ to $\triangle DEF$. Now, point A and point D are matched.

(b) Since $\overline{AB}=\overline{DE}$, then side AB can cover side DE completely and point B and point E are matched.

(c) Since $\angle A=\angle D$, then side AC can cover side DF completely.

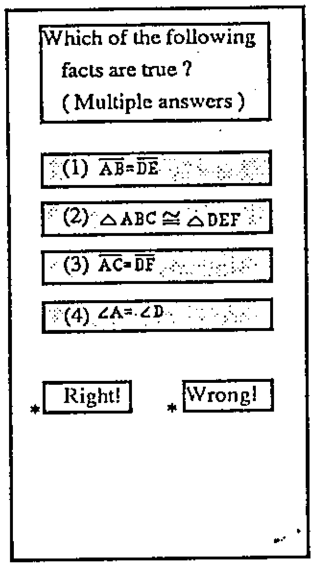
(d) Since $\overline{AC}=\overline{DF}$, then point C and point F are matched.

(e) Now, as you can see, when move $\triangle ABC$ to $\triangle DEF$ they can be matched completely, so $\triangle ABC \cong \triangle DEF$.

window (from F4 to F9)**

Note: * denotes the display of the content when the fact is properly selected.
** denotes the window of performing the answer-question from F4 to F9.

(d) F4



Which of the following facts are true ? (Multiple answers)

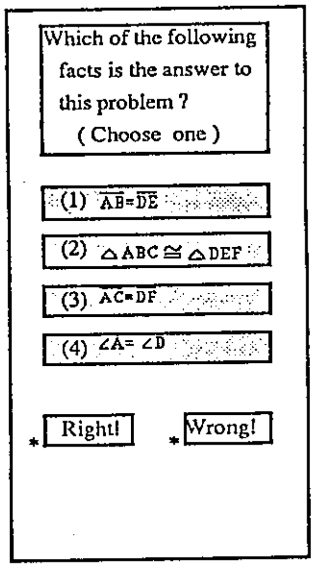
(1) $\overline{AB}=\overline{DE}$

(2) $\triangle ABC \cong \triangle DEF$

(3) $\overline{AC}=\overline{DF}$

(4) $\angle A=\angle D$

*Right! *Wrong!



Which of the following facts is the answer to this problem ? (Choose one)

(1) $\overline{AB}=\overline{DE}$

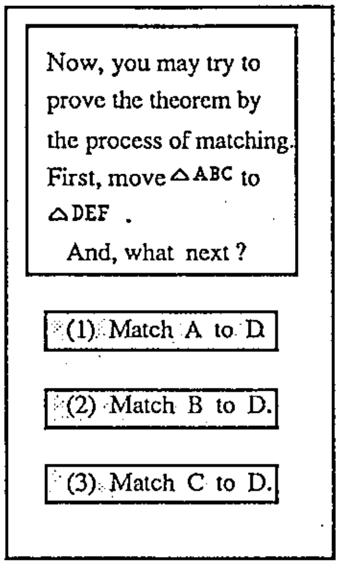
(2) $\triangle ABC \cong \triangle DEF$

(3) $\overline{AC}=\overline{DF}$

(4) $\angle A=\angle D$

*Right! *Wrong!

(e) the right-down corner of F4 (f) F5



Now, you may try to prove the theorem by the process of matching. First, move $\triangle ABC$ to $\triangle DEF$.

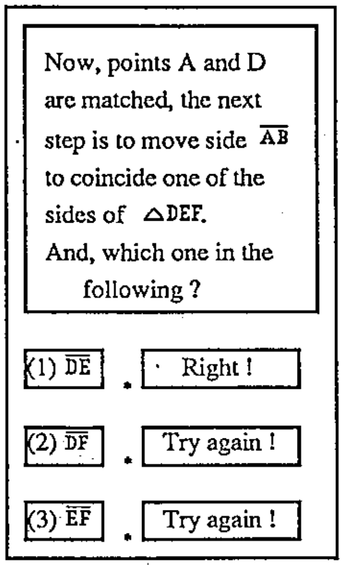
And, what next ?

(1) Match A to D.

(2) Match B to D.

(3) Match C to D.

(g) F6



Now, points A and D are matched, the next step is to move side $\overline{AB}$ to coincide one of the sides of $\triangle DEF$.

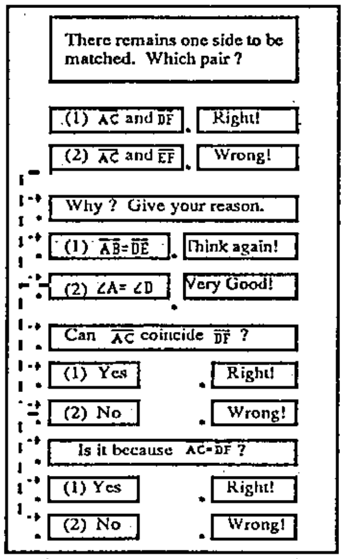
And, which one in the following ?

(1) $\overline{DE}$. Right !

(2) $\overline{DF}$. Try again !

(3) $\overline{EF}$. Try again !

(h) F7



There remains one side to be matched. Which pair ?

(1) $\overline{AC}$ and $\overline{DF}$. Right!

(2) $\overline{AC}$ and $\overline{EF}$. Wrong!

Why ? Give your reason.

(1) $\overline{AB}=\overline{DE}$. Think again!

(2) $\angle A = \angle D$. Very Good!

Can $\overline{AC}$ coincide $\overline{DF}$?

(1) Yes . Right!

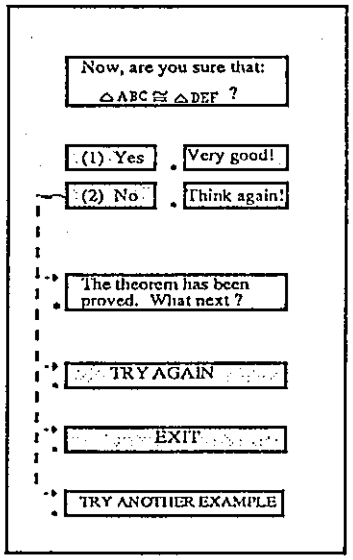
(2) No . Wrong!

Is it because $AC=DF$?

(1) Yes . Right!

(2) No . Wrong!

(i) F8



Now, are you sure that:
$\triangle ABC \cong \triangle DEF$?

(1) Yes . Very good!

(2) No . Think again!

The theorem has been proved. What next ?

TRY AGAIN

EXIT

TRY ANOTHER EXAMPLE

(j) F9

Fig. 6.1 V-Form instance of template CAI_Course

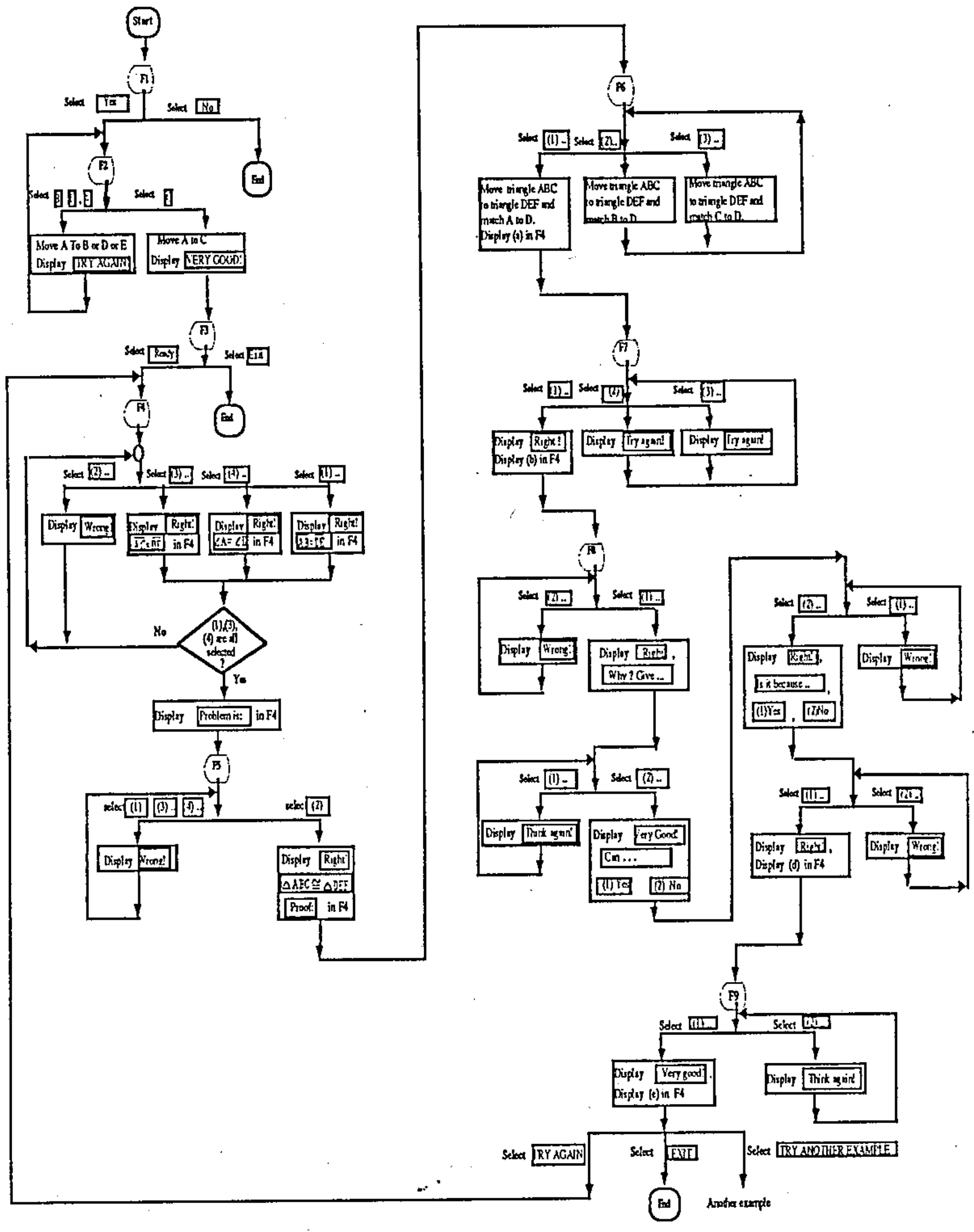

Start
F1
Select Yes
Select No
End
F2
Move A To B or D or E
Display TRY AGAIN
Move A to C
Display VERY GOOD!
F3
Select Ready
Select Exit
F4
Display Wrong
(1),(3), (4) are all selected ?
No
Yes
Display Problem is: in F4
F5
Display Wrong!
Display Right! △ABC ≅ △DEF Proof: in F4
F6
Move triangle ABC to triangle DEF and match A to D. Display (a) in F4
Move triangle ABC to triangle DEF and match B to D.
Move triangle ABC to triangle DEF and match C to D.
F7
Display Right! Display (b) in F4
Display Try again!
F8
Display Right, Why ? Give ...
Display Think again!
Display Very Good! Can ... (1) Yes (2) No
Display Right!, Is it because ..., (1) Yes, (2) No
Display Right!, Display (d) in F4
F9
Display Very good!, Display (e) in F4
Display Think again!
Select TRY AGAIN
Select EXIT
Select TRY ANOTHER EXAMPLE
End
Another example

Fig. 6.2 Control flow of CAI_Course

6.2 DIRECT V-FORM INTERPRETER

When a V-Form instance is created through the processing of VDL and VML, a Direct V-Form Interpreter can be invoked to executed such a V-Form instance. This interpreter views a V-Form instnace as an executable program and interpretes it directly. Because the internal structure of V-Form instance is consistent and complete, the interpreting algorithm is simple and easy to understand. We shall describe the interpretion of V-Form pages processing in the following algorithm (main program of the interpreter).

```
Get the first frame as the current V-Form page.
While ( the terminating condition is true )
    Display the current V-Form page.
    Execute the embedded components of the V-Forms with type
        non-"VFORM".
    Repeat
        Get input from user.
    until ( proper V-Form is selected ).
    Execute the embedded components of the selected V-Form.
    Get a new working frame.
    Erase current V-Form page.
    Set the new frame as current V-Form page.
end.
```

In this algorithm, the first frame is the rooted V-Form page in a V-Form instance. This interpreter interpretes a V-Form instance beginning with the rooted V-Form page(ie. the form control table(FCT) pointed by the system node(SN) of this V-Form instance). Then a form-by-form V-Form page displaying and processing is continued until the terminating condition is satisfied. Note that the termainting condition may be executed when a "EXIT" procedure is called or a forced termination by using Control-Z interrupt is encountered. Also, note that the selected V-Form must be a "VFORM" type V-Form else a beep is responsed.

The displaying of a V-Form page is supported by several sub-system, such as graphics packages and voice utterances. A text type V-Form is displayed directly by this interpreter, and the V-Forms with type static graphics(graph or bit-map) or dynamic graphics(animation) can be delivered to graphics sub-system. Similarly, the voice type V-Forms can be treated as a series of procedure calls which invoked a voice sub-system for execution.

The V-Form pages processing executes as follows: (1) tests whether the terminating condition is true or not, if it is true, stops the execution, (2) display the current V-Form page, (3) executes the embedded components of the V-Forms with type non-"VFORM", (4) wait for user's input for selection, (5) executes the embedded components of the selected V-Form, (6) creates a new frame for a working page and return to (1).

The embedded components of non-"VFORM" is often used to facilitate the management of V-Form displaying. For example, in the third frame F3 (VFORM:C in Fig. 6.1(c)), TEXT:From may have such procedures "DELAY 3.; DISPLAY TEXT:Know.". The first procedure comsumes three time segments for delaying and the second procedure displays a V-Form TEXT:Know because its indicator is "OFF". These two procedures cause a delay of a period of a period of time between two consecutive V-Forms TEXT:From and TEXT:Know. On the other hand, the embedded components of the selected V-Form is often used to control the switching of V-Form pages as well as many expressing. For example, the V-Form VFORM:B in the second frame F2 may have the following emebdded components: a procedure as "ADD error_count 1.", a condition-action as "IF error_count LESS 3. ; DISPLAY TEXT:Again.", and another procedure as "GOTO VFORM:CAI_Course.". So, when VFORM:B is selected, the first procedure is performed to increase the error count one, then test its condition whether the error count is less then 3, if yes, a V-Form TEXT:Again with indicator "OFF" is displayed, otherwise go to the first V-Form page VFORM:CAI_Course. The user's input may be entered by using convenient pointing devices, but now it is simulated by the entering of the window region coordinates. To create a new working frame is usually achieved by the selection of a proper V-Form or the "GOTO" procedure.

CHAPTER 7
CONCLUSION

VIPS provides a visual programming environment for users to develop their own application programs without going into detailed programming that must be adhered strictly to syntax rules. In VIPS, V-Forms are used as the fundamental objects because they are more akin to the user's view point and their contents may include text, static graphics, dynamic graphics, rules, and voices. The V-Form Manipulation Language provides displaying, inserting, removing, updating, grouping, and degrouping operations for users to manipulate V-Forms in a what-you-see-is-what-you-get manner. It can be used to generate complete V-Form instances by filling in its contents according to the V-Form types, or may be served as a program editor to edit V-Form instances at will. All information carried by V-Forms can be modified by VML, even some new V-Form instances can be created by V-Form grouping and degrouping from existing V-Form instances.

The self-synthesized internal structure which structurized from a V-Form instance can be viewed as a program for one particular application, and executed by a direct V-Form interpreter.

VIPS is only an experiment system in the study of visual programming and its applications. More knowledgable objects and their relations are expected to be included in VIPS in near future. To extent the functional capabilities of extensions may be proceeded for VIPS, there are two things could be done. One is to connect VIPS to a Relational Data Base Management System(RDBMS) so that the data records of internal structure can be stored in or retrieved from RDBMS. The other is to translate the internal structure of a particular application directly into an executable program, so that VIPS becomes trully an automatic program generator. From resource sharing point of view, VIPS integration with RDBMS should be done first. Besides, the technology of storing and accessing of an executable program codes within RDBMS still far to reach at the present time. However, still there are a lot of things to do. Among them, the first important thing is to dig out more of the theoretical understanding of visual perception so that a designer can devise knowledge representation for wider objects in a more general way.

APPENDIX REFERENCES

[1] Anderson, J. R., Boyle, C. F. and Yost, G., "The Geometry Tutor," IJCAI, LosAngles, U.S.A.,pp. 1-7 (1985).

[2] Bobrow, D. G., and Raphael, D. "New Programming Language for A. I. Research," Computing Surveys 1974.

[3] Campbell, J. A., ed., Implementations of Prolog, Ellis Horwood Series in Artificial Intelligence(1984).

[4] Chang, S. K. and Charisse, O., "The Interpretation and Construction of Icons for Man-Machine Interaction in an Image Information System," Proceedings of IEEE Workshop on Language for Automation, New Orleans (Nov. 1984).

[5] Chen, J. C., "A Knowledge-Based Environment on Virtual Workstation," M. S. Thesis, Graduate Institute of NTUEE, Taipei, Taiwan, R. O. C. (June 1985).

[6] Cheng, K. Y., Hsu, C. C., Lin, I. P., Lu, M. C., and Hwu, M. S., "VIPS: A Visual Programming Synthesizer," Second IEEE Computer Society Workshop on Visual Language, Dallas, Texas, U. S. A. (June 1986).

[7] Clocksin, W. F. and Mellish, C. S., *Programming in Prolog*, 2nd ed., Springer-Verlag, Berlin, 1984.

[8] Date, C. J. *An Introduction to Database System*, Third Edition, 1981.

[9] Ellis, C. A. and Nutt, G. J., "Office Information Systems and Computer Science," Computing Surveys, Vol. 12, No. 1, pp. 27-60 (March 1980).

[10] Fu, K. S. and Booth, T. L., "Grammatical Inference: Introduction and Survey -- Part I," IEEE Trans. on System, Man, and Cybernetics, Vol. SMC-5, No. 1, pp. 95-111 (1975).

[11] Fu, K. S., "An Introduction to Formal Language," Chap. 2. in *Syntatic Pattern Recognition and Application*, Prentice-Hall, Englewood Cliffs, N. J., 1982.

[12] Gray, M. D., *Logic, Algebra and Database*, Halsted Press, 1984.

[13] Harrison, M. A., *Introduction to Formal Language Theory*, Addison-Wesley (1978).

[14] Hayes, P. J., "The Logic of Frames", from Webber and Nilsson, Readings in Artificial Intelligence (Tioga, 1981).

[15] Hopcroft, J. E. and Ullman, J. D., *Introduction to Automata Theory, Language, and Computation*, Addison-Wesley, 1979.

[16] Hopcroft, J. E. and Ullman, J. D., "Formal Languages and Their Relation to Automa,", Addison-Wesly, 1976.

[17] Kowalski, R., *Logic for Problem Solving*, Amsterdam: North Holland, 1979.

[18] Lu, M. C., "A Visual Approach to Automatic Program-Synthesizer," M. S. Thesis, Graduate Institute of NTUEE, Taipei, Taiwan, R. O. C. (June 1986).

[19] Jacob, Robert J. K., "A State Transition Diagram Language for Visual Programming," IEEE Computer, pp. 51-59 (1985).

[20] Kearsley, G., "Authoring Systems in Computer Based Education," Comm. of the ACM, Vol. 25, No. 7, pp. 429-437 (1982).

[21] King, K. J. and Maryanski, F. J., "Information Management trends in Office Automation," Proceedings of t he IEEE, Vol. 71, No. 4, pp. 519-528.(1983).

[22] Kitagawa, H., Gotoh, M., Misaki, S., and Azuma, M., "Form Document Management System SPECDOQ - Its Architecture and Implementation," Second ACM-SIGOA Conference on Office Information System, Vol. 5, No. 1-2, pp. 132-142 (June 1984).

[23] Liu, K. C. "A Multi-Model Database System With A Unified Data Language," M. S. Thesis, Graduate Institute of NTUCSIE, Taipei, Taiwan, R. O. C. (June 1985).

[24] Moriconi, M. and Hare, D. F., "Visualizing Program Designs Through PegSys," IEEE Computer, Vol. 18, No. 8, pp. 72-85 (1985).

[25] Reiser, B. J., Anderson, J. R. and Farrel, R. G., "Dynamic Student Modelling in an Intelligent Tutor for LISP Programming," IJCAI, LosAngles, U. S. A., pp. 8-14 (1985).

[26] Rockart, L. A. and Flannery, L. S., "The management of End User Computing," Communications of the ACM, Vol. 26, No. 10, pp. 776-784 (Oct. 1983).

[27] Shu, N. C., Lum, V. Y., Tung, F. C., and Chang, C. L., "Specification of Forms Processing and Business Procedures for Office Automation," IEEE Transactions on Software Engineering, Vol. SE-8, No. 5, pp. 499-512 (Sep. 1982).

[28] Shu, N. C., "A Forms-oriented and Visual-directed application development system for non-programmers," First IEEE Computer Society Workshop on Visual Language, Hiroshima, Japan, pp.162-170 (Dec. 1984).

[29] Shu, N. C., "FORMAL: A Form-Oriented, Visual-Directed Application Development System," IEEE Computer, Vol. 18, No. 8, pp. 38-49 (1985).

[30] Sugihara, K., et al., "An Approach To The Design of a Form Language," First IEEE Computer Society Workshop on Visual Language, Hiroshima, Japan, pp.171-176 (Dec. 1984).

[31] Tsichritzis, D. C., "Form Management", Comm. Of the ACM, Vol. 25, No. 7, pp. 453-478 (1982).

[32] Tsichritzis, D. C. and Lochovsky, F. H. "Office Information System: Challenge for the 80's," Proceedings of the IEEE, Vol. 68, No. 9, September 1980.

[33] Winston, P. H., *Artificial Intelligence,* Addison-Wesley (1984).

[34] Winston, P. H., and Horn, B. K., *Lisp*. Reading, Mass.:Addison Wesley, 1981.

[35] Zloof, M. M., "Query-by-Example: A Data Base Language," IBM System Journal, Vol. 16, No. 4, pp. 324-343 (1977).

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