TR-86-007

Visual Programming

Approach to Computer Aided Instructions

K.Y.Cheng, C.C.Hsu, I.P.Lin, M.S.Hwu, and M.C. Lu

Institute of Information Science
Academia Sinica
and
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan, ROC

Visual Programming Approach to Computer Aided Instructions

K.Y.Cheng, C.C.Hsu, I.P.Lin, M.S.Hwu, and M.C. Lu

Institute of Information Science
Academia Sinica
and
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan, ROC

ABSTRACT

An ideal CAI system is to allow users(teachers) to be able to create their own coursewares even though they possess no knowledge of programming. In this paper, we propose a visual programming approach to achieve this goal. We present a VIsual Programming Synthesizer(VIPS) which allow users to sketch their well-written coursewares in visual. Through the interface language of VIPS, a user defined courseware can be self-synthesized into a complete internal strcuture. This internal strcuture holds properties of knowledgable objects and the execution flow of the courseware. We also present a mapping algorithm to transform the internal structure into a set of Horn clauses so that coursewares developed under VIPS can run on Prolog systems.

1. INTRODUCTION

Computer aided instruction (CAI) is one of the important issues in computer applications. An ideal CAI system[1,10,15] is to allow the end users (teachers) to be able to create their own application programs even though they possess no knowledge of programming. Such kind of demands in computer applications also can be widely seen in the area of office information system (OIS)[4,11,12,16,20]. However, in CAI, a smart courseware that can diagnosis and remedy student's errors and evaluate the progress of learning is essential. Consequently, friendly user-interface facilities as well as intelligent performance must be emphasized in CAI applications. It is hard to implement such a CAI system under conventional programming environment because such an environment is designed for programmers to use. On the other hand, under artificial intelligence programming environment alone, it seems still very difficult to handle the facts and the rules of colorful courseware expressions very well. Thus, a new environment is needed in CAI applications.

The major issues of visual programming are to provide a friendly environment for NP-professionals (Non-Programmers) to develop their own application programs visually[9,14,17,18,19]. In this paper, we present a VIsual Programmimg Synthesizer (VIPS) as a vehicle for teachers to develop their own coursewares. A simple courseware example of theorem proving on triangle coincidence is illustrated to demonstrate the features of VIPS. Under this environment, the forms dealt with contain text (in particular Chinese characters), static graphics (line drawing and bit-map), dynamic graphics (animation), rules, and voices. In order to distinguish them from office forms, we call them the V-Forms.

Just like other form models[17,18,19,20], 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 to define and manipulate V-Forms on the screen. This interface language includes two parts: a V-Form Definition Language (VDL) and a V-Form Manipulation Language (VML).

After processing VDL and VML, a complete internal form for each application can be self-synthesized. Then a mapping algorithm that transforms the self-synthesized internal structure into a program of Prolog clauses converts CAI courseware into an executable program. V-Form instances that are text-only can be executed by a Prolog interpreter[3,7,13], rules are also translated into Horn clauses, V-Form instances other than text are treated as a series of procedure calls which invoke graphics and voice subsystems for execution.

In the following, we discuss the language environment of VIPS in section 2, the V-Form model in section 3, the CAI courseware design through the interface language in section 4, and the interpreting environment in section 5.

2. VISUAL LANGUAGE ENVIRONMENT

We shall define the visual language formally as follows. 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 grammer G_i[5], then a visual language is

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

where

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

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. 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 that can be generated by an inferred grammar in the limit[6]. Among them, context-free languages are not powerful enough to describe the programming applications, but context-sensitive languages are very complex for analysis. Therefore, 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[6] 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. In formal language aspect, VIPS' inferred grammar is a context-free programmed grammar and the class synthesized by VIPS can be a context-sensitive language.

There are some design methods for visual programming languages[9,14,17,18,19]. However, a systematic approach to a theoretical sound methodology is still under developing. Here, we consider a visual programming synthesizer which is interactive and application-oriented with the following features : (1) easy to use -- various user friendly facilities such as icons, pointing devices, and menus are included; (2) visual directed objects -- V-Forms are used as the fundamental objects of VIPS because they are 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 automatically 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 VIPS from one system to another is simply by a mapping which transforms the internal structure to an executable program, such as Prolog.

3. V-FORM MODEL

3.1 External Structure

VIPS 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. They are represented in **V-Forms**.

Let Π be a set of codes to be displayed, Δ be a set of alphanumerics, Ç be a set of Chinese characters, ß be a set of graphic codes, Ω be a set of drawing attributes, and $\Sigma = \Pi \cup \Delta \cup$ Ç $\cup$ ß $\cup \Omega$. Then the domain of a V-Form belongs to Σ^*, where * denotes the Kleene closure[8]. 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 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. 4.3(a). The scheme for this V-Form is defined in the following expressions:

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

The hierarchical structure of the scheme for the CAI_Course is given in Fig. 3.1, where a **circle** means a V-Form which contains another V-Forms, a **box** represents a minimum V-Form which contains no other V-Forms (also called an **atom**).

[Definition 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 template for the scheme of Fig. 4.3(a) is given in Fig. 3.2, where the V-Forms TEXT:Topic, ANIMATION:Pic1, ANIMATION:Pic2, and TEXT:Ask are atoms, but VFORM:Yes and VFORM:No are not. As shown in Fig. 3.1, each V-Form may contain other V-Forms (A V-Form F_j is said to be a **sub-VForm** of F_i if and only if F_i contains F_j) and hence a hierarchical structure is formed. A template also describes the control flow of the V-Form node. For instances,

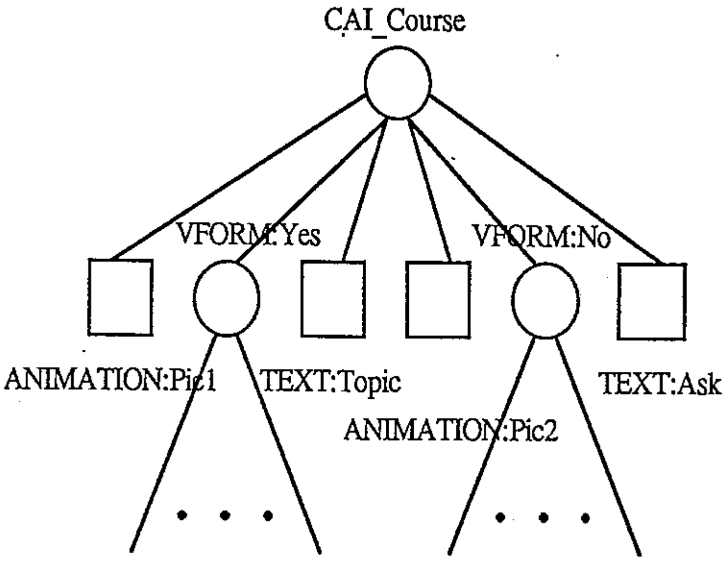


Fig. 3.1 Hierarchical structure for CAI_Course

when VFORM:Yes (sub-VForm of CAI_Course) of Fig. 3.2 is selected then the display will be switched to that of its descendent V-Form node F2 (Fig. 4.3 b).

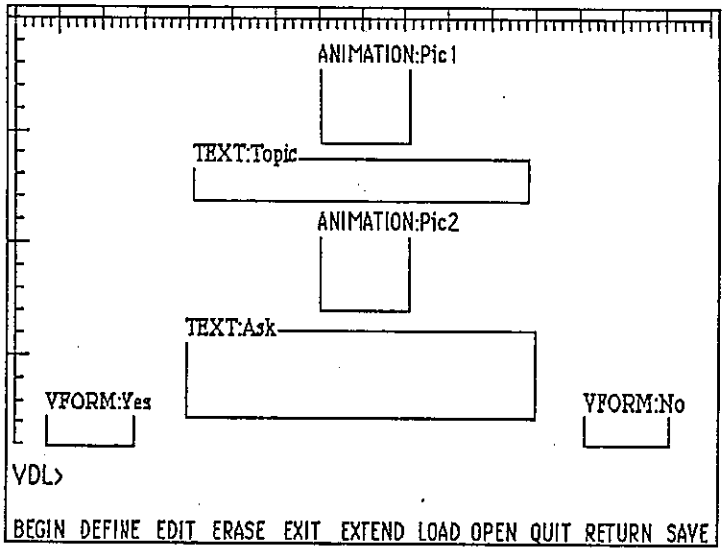


Fig. 3.2 A template for the CAI_Course

[Definition 3] A **V-Form instance** 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, ANIMATION, VOICE, and RULE. One of the V-Form instance of Fig. 3.2 is shown in Fig. 4.3(a).

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.

3.2 Internal Structure

The internal structure 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 form control node(FCN) describes the characteristics 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.

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. 3.3 shows part of the internal structure of the V-Form system CAI_Course (only F1 of Fig. 4.3 is shown).

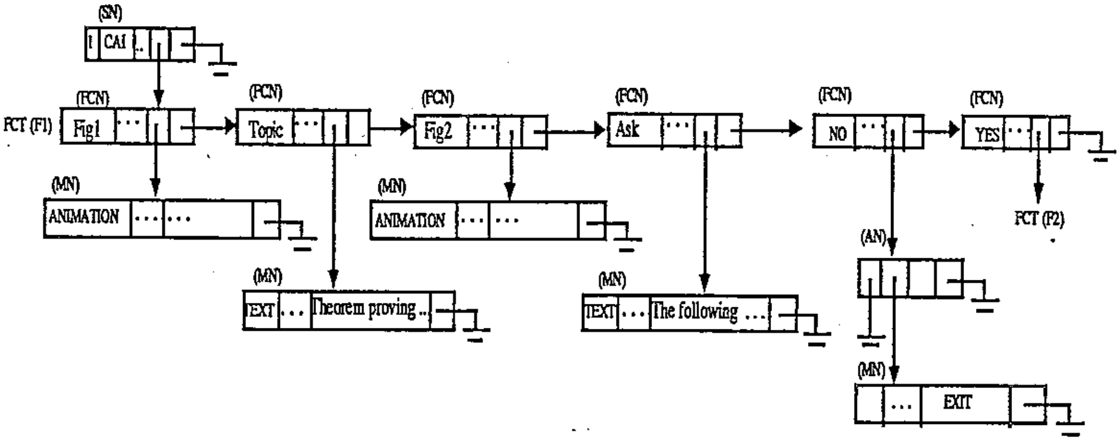


Fig. 3.3 The internal structure of the V-Form F1 of Fig. 4.1

4. CAI COURSEWARE DESIGN THROUGH INTERFACE LANGUAGE

VIPS contains an interface language which allows teachers to describe their own well-written courseware visually . In the following, we shall describe the Interface Language briefly and give an example to show how a CAI courseware can be designed through this language.

4.1 Interface Language

The interface language in VIPS is a two-dimensional non-procedural language which offers users to define and manipulate V-Forms on the screen. It has two parts : VDL and VML described as follows.

4.1.1 V-Form Definition Language

As mentioned in section 3, a V-Form consists of a V-Form type and a V-Form instance. Hence, V-Form Definition Language is divided into a **skeleton phase** and an **editing phase** to define V-Form type and V-Form instance respectively[2].

The skeleton phase is to define the V-Form type by manipulating the screen directly and visually. It provides many primitive operations which allow users to **sketch** what V-Form types shell be in their conceptual model. Table 1 shows a summary of the primitive operations of the skeleton phase in VDL.

Table 1. Primitive operations of Skeleton Phase in VDL

Category	Primitive Operations
Initialization	BEGIN
File Handling	LOAD, SAVE
Flow Manipulation	EXTEND, RETURN, SET
Window Management	OPEN, ERASE
Type Redefining	DEFINE
Instance Generation	EDIT
Convenience	COPY, HOME

The editing phase is to fill in each V-Form type with a V-Form instance. Table 2 shows a summary of the primitive operations of the editing phase.

Table 2. Primitive operations of Editing Phase in VDL

Category	Primitive Operations
Content Management	FILL, SET
Flow Manipulation	GOTO
Procedure Processing	CONDITION, ACTION
Convenience	ALIAS

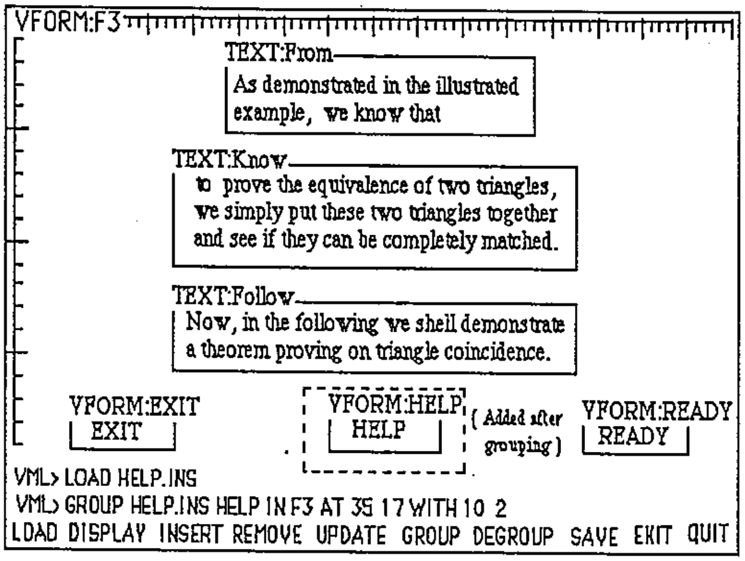
4.1.2. V-Form Manipulation Language

VML is a language to manipulate V-Form instances gotten from VDL. As described earlier, a V-Form instance can be regarded as a user-defined program. Thus, VML, similar to a program editor, can manipulate V-Form instances at will. It provides users the following opeartions: (1)displaying, (2)inserting, (3)removing, (4)modificating, (5)grouping, and (6)degrouping. A V-Form instance can be manipulated by the operations(1), (2), (3), and (4), while grouping and degrouping operations allow users to restructure the existing V-Form instances. The primitive operations of VML are shown in Table 3.

Table 3. Primitive operations of VML

Category	Primitive Operations
Dispalying	DISPLAY
File Handling	LOAD, SAVE
Atom Manipulation	INSERT, REMOVE
Content Modification	UPDATE
Instance Management	GROUP, DEGROUP

The operations of displaying, atom manipulation, content modification, and file handling are similar to many other form manipulations[16,17,18,20,21,22]. A slight difference for VML from others is to include the operations of grouping and degrouping. Fig. 4.1 shows an example of grouping. The grouping operation adds a complementary course, V-Form instance in HELP.INS file, to the



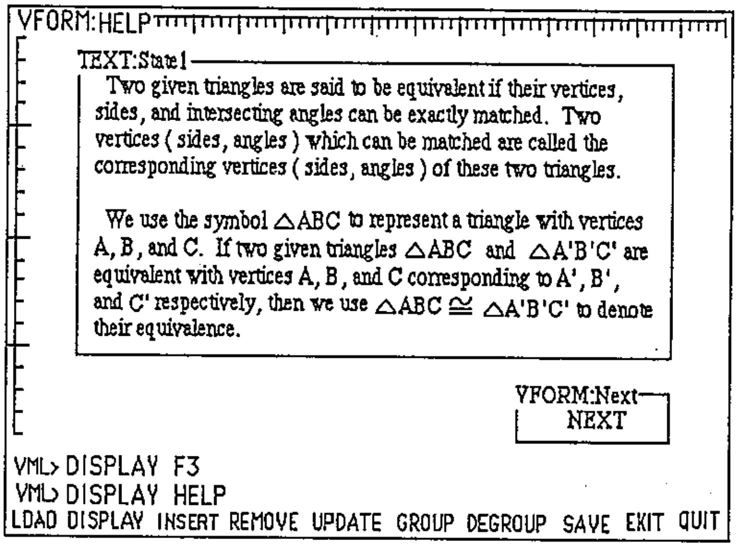
As demonstrated in the illustrated example, we know that

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.

```
VML> LOAD HELP.INS
VML> GROUP HELP.INS HELP IN F3 AT 35 17 WITH 10 2
```

(a)V-Form F3


VFORM:HELP
TEXT:State1
VFORM:Next
NEXT
LOAD DISPLAY INSERT REMOVE UPDATE GROUP DEGROUP SAVE EXIT QUIT

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 △ABC to represent a triangle with vertices A, B, and C. If two given triangles △ABC and △A'B'C' are equivalent with vertices A, B, and C corresponding to A', B', and C' respectively, then we use △ABC ≅ △A'B'C' to denote their equivalence.

```
VML> DISPLAY F3
VML> DISPLAY HELP
```

(b)V-Form HELP in HELP.INS file

Fig. 4.1 Example of Grouping

node F3 of the V-Form instance CAI_Course. Notice that, the operations of inserting, removing, and modifying can not change the branches of control flow, but the operations of grouping and degrouping can change them. The new structural relationship between two grouped V-Form instances is shown in Fig. 4.1(a). Fig. 4.2 shows the logical structure after grouping. Degrouping is just the reverse of grouping. Both operations can only operate on independent V-Form instances, i.e, the grouped/degrouped V-Form instances must be structurally unrelated.

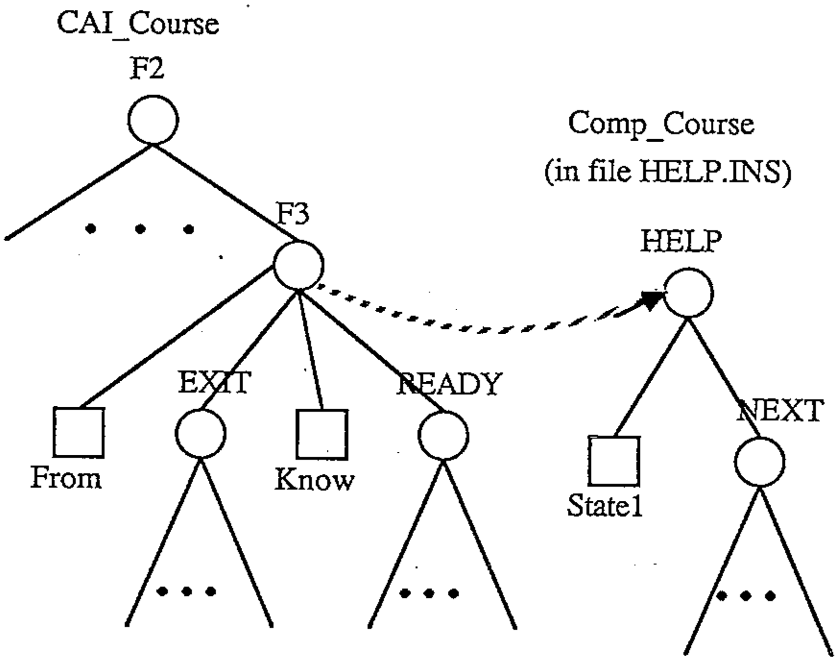


Fig. 4.2 Logical structure of CAI_Course after Grouping

4.2 CAI Courseware Design

Fig. 4.3 shows a coursewarefor the purpose of demonstrating the concept of geometrical coincidence. Fig. 4.3 (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 stop one. A touch/selection to the VFORM:Yes starts the practice to learn the concept of geometrical coincidence.

To design this 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.

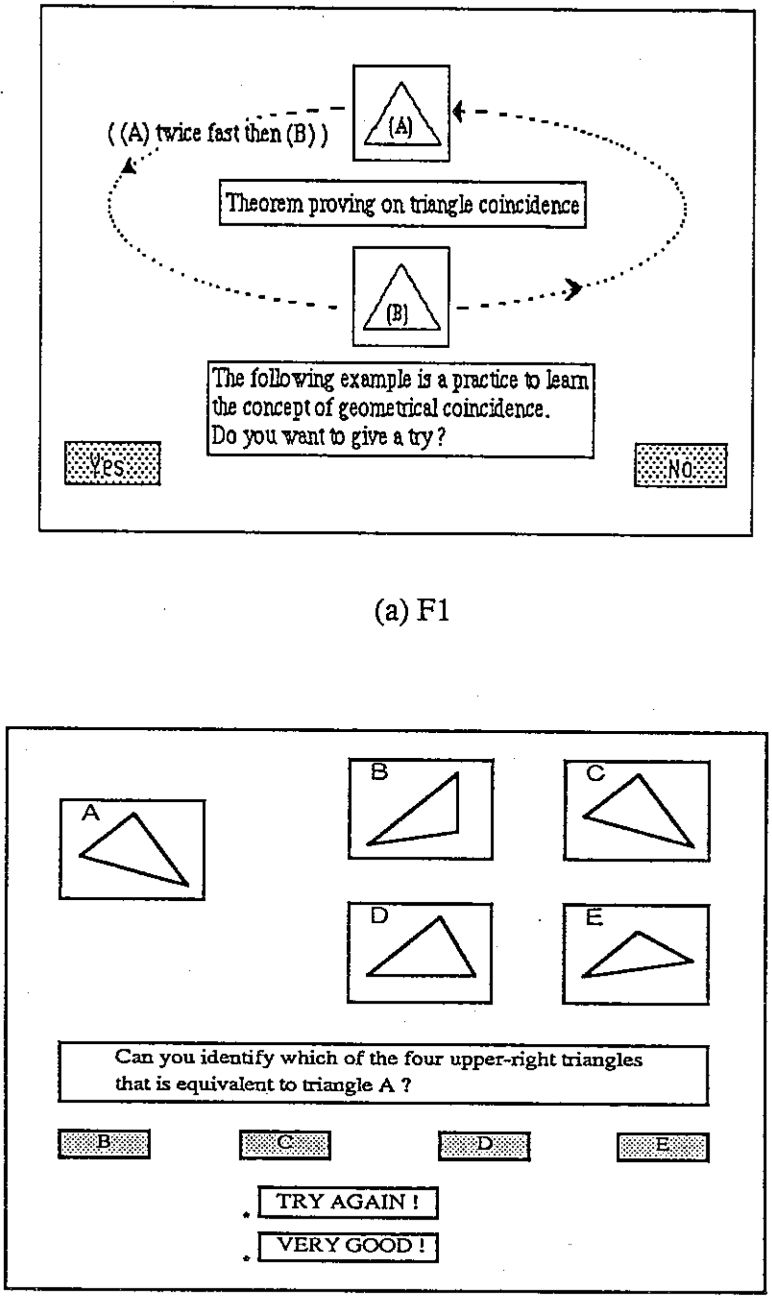
Once a teacher has prepared his/her own well-written sheets in V-Forms according to the courseware, he/she can interactively define the V-Form type with a sequence of skeleton phase commands in VDL. For example, we can define the V-Form type given in Fig. 3.2 by the following interactive sequence[2]:

```
VDL> BEGIN CAI Course
VDL> OPEN TEXT AT 6 21 WITH 38 1 AS Topic
VDL> OPEN ANIMATION AT 2 35 WITH 10 4 AS Pic1
VDL> OPEN ANIMATION AT 9 35 WITH 10 4 AS Pic2
VDL> OPEN TEXT AT 14 20 WITH 40 3 AS Ask
VDL> OPEN VFORM AT 17 4 WITH 8 1 AS Yes
VDL> OPEN VFORM AT 17 66 WITH 8 1 AS No
VDL>
```

When an entire V-Form type of the courseware is obtained, the operations of editing phase are used to fill in the contents of the V-Form type. For example, V-Form instance in Fig. 4.3(a) can be created from Fig. 3.2 through the following interactive sequence of commands:

```
VDL> EDIT Topic
VDL$ED> FILL  /* cursor is now at the beginning of V-Form TEXT:Topic,
                  user can now key in text 'Theorem proving on triangle coincidence' */
VDL$ED> SET ATTRIBUTE BOLD
VDL$ED> EXIT
VDL> EDIT Pic1
VDL$ED> FILL
           ...
VDL$ED> EXIT
VDL> EDIT Pic2
VDL$ED> FILL
           ...
VDL$ED> EXIT
           ...
```

If an atom has procedures embedded, then it is edited by the procedure processing and flow manipulation operations. Notice that, from F4 to F9 of Fig. 4.3, 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. V-Forms marked with asterisk are displayed only when correct facts are chosen. Fig. 4.4 shows the self-synthesized control flow of the V-Form instance illustrated in Fig. 4.3.



Theorem proving on triangle coincidence

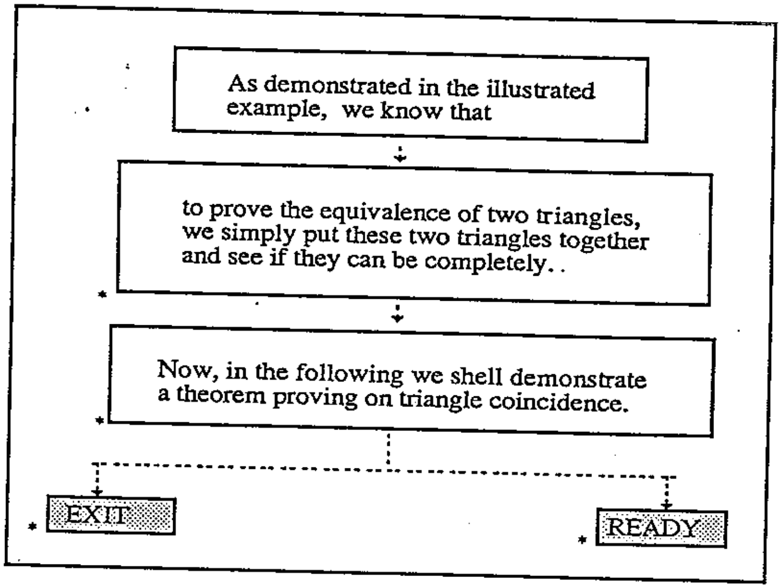
The following example is a practice to learn the concept of geometrical coincidence.
Do you want to give a try?

(a) F1

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

(b) F2

Fig. 4.3 V-Form instance of template CAI_Course



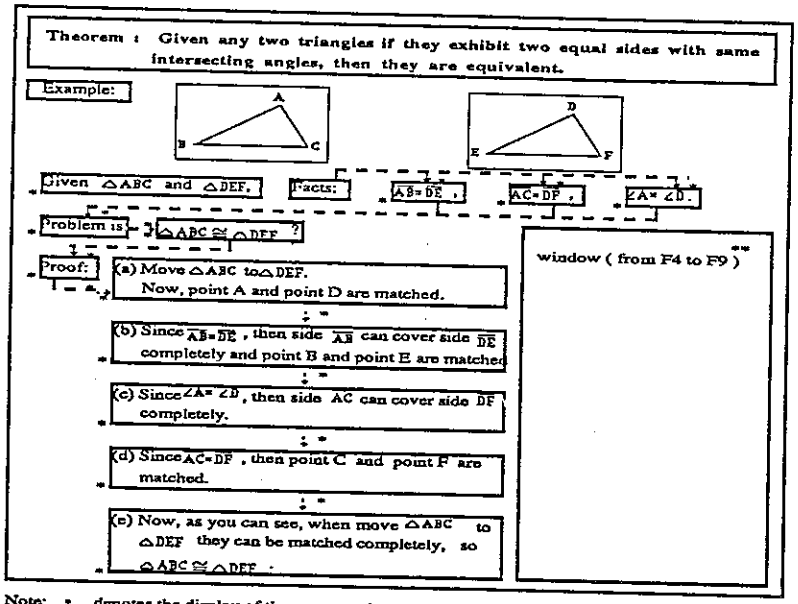
As demonstrated in the illustrated example, we know that

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: ---→ denotes delay of a period of time between two consecutive 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: $\overline{AB} = \overline{DE}$, $AC = DF$, $\angle A = \angle D$.

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 $\overline{AB}$ can cover side $\overline{DE}$ 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 $AC = DF$, then point C and point F are matched.

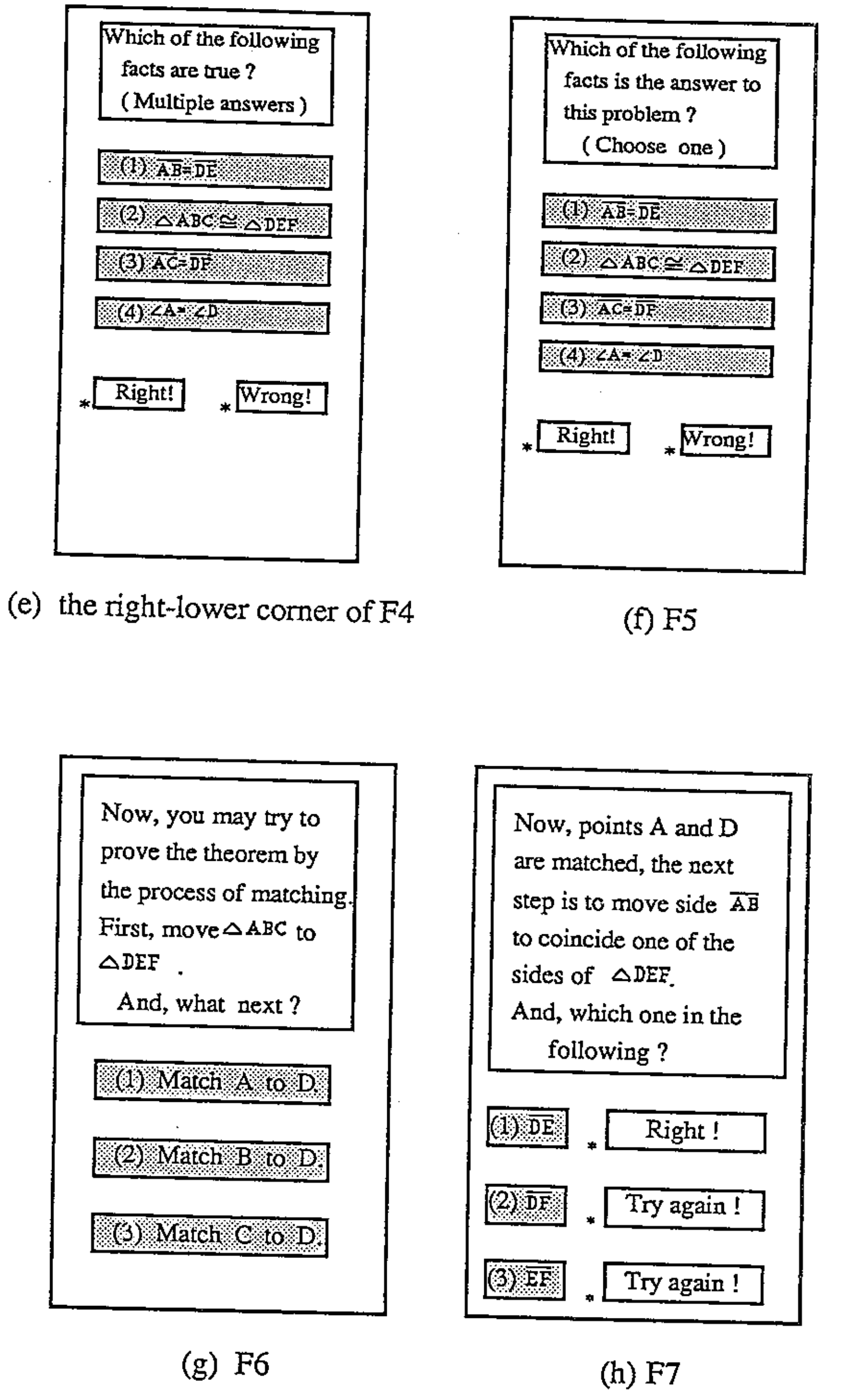
(e) Now, as you can see, when move $\triangle ABC$ to $\triangle DEF$ they can be matched completely, so $\triangle ABC \cong \triangle DEF$.

window (from F4 to F9)**

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

(d) F4

Fig. 4.3 V-Form instance of template CAI_Course (Continued)



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) $\angle A=\angle D$

(e) the right-lower corner of F4

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) $\angle A=\angle D$

(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 ?

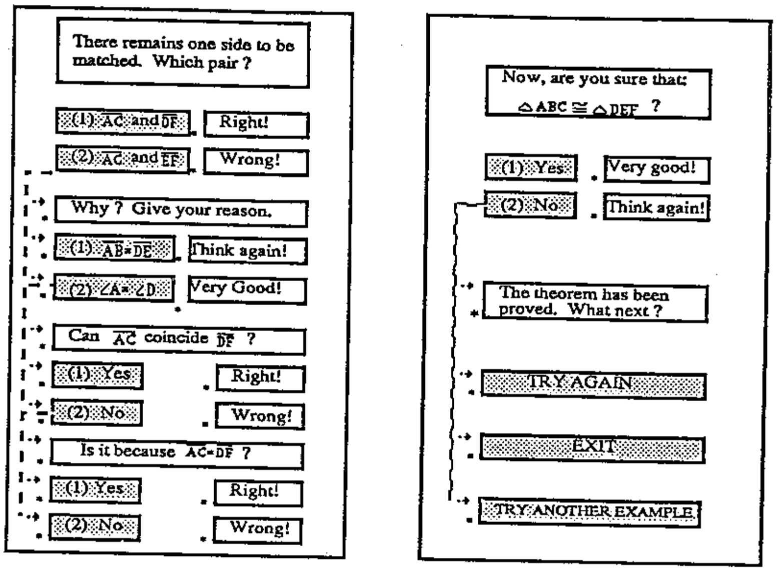
(1) $\overline{DE}$

(2) $\overline{DF}$

(3) $\overline{EF}$

(h) F7

Fig. 4.3 V-Form instance of template CAI_Course (Continued)


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!
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

(i) F8

(j) F9

Fig. 4.3 V-Form instance of template CAI_Course (Continued)

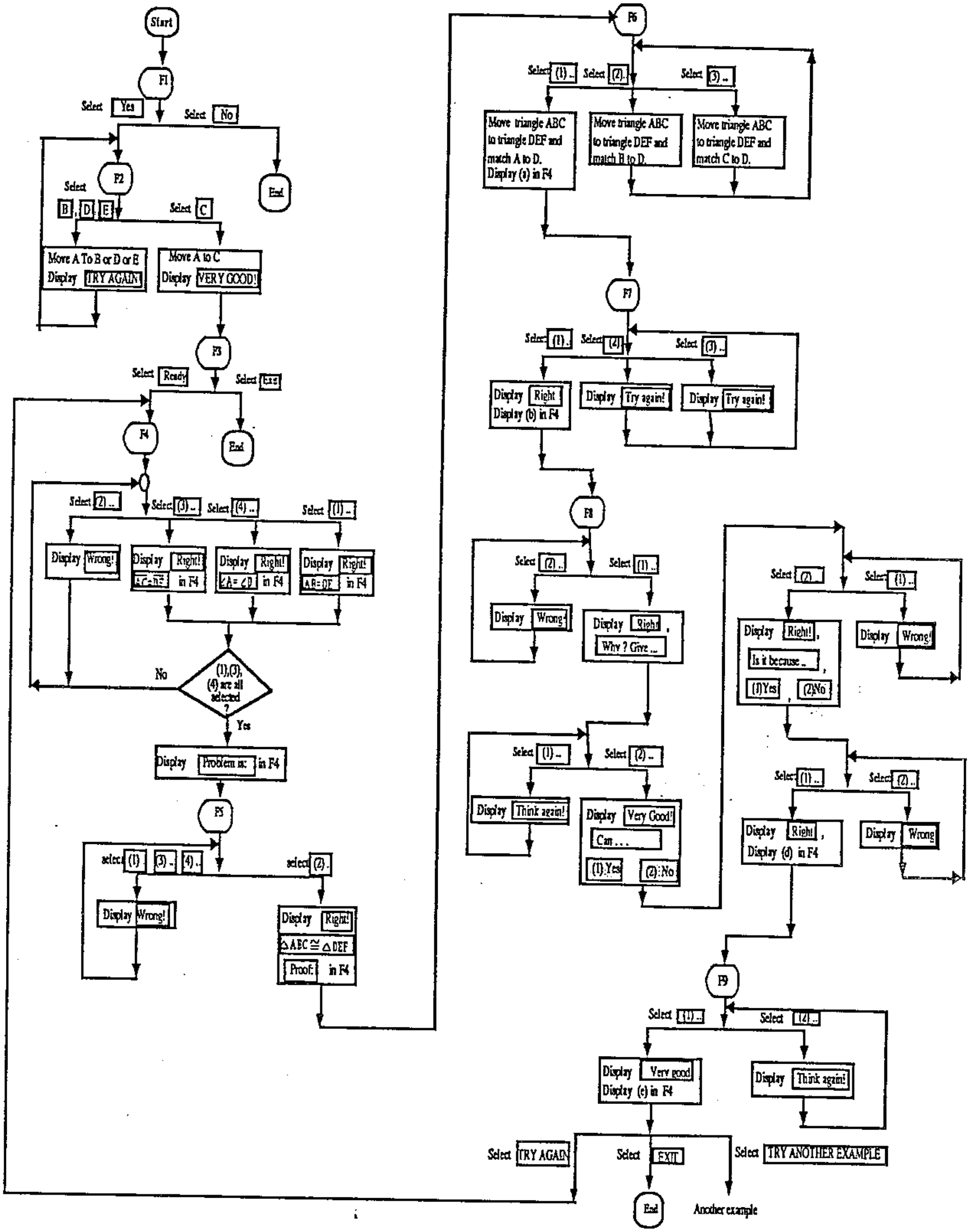


Fig. 4.4 Control Flow of CAI_Course

5. INTERPRETING ENVIRONMENT

After the completion of sketching the whole courseware, an internal structure as shown in Fig. 3.3 is automatically obtained. Then, the internal structure can be interpreted by a VIPS interpreter. There are two ways to interpret the internal structure, one is through a self-designed interpreter, the other is using an existing one (such as CProlog)[2]. In consideration of the portability and sometimes in need of unification and bactracking in the execution of a V-Form instance, a current existing Prolog interpreter is adopted. With this approach, a mapping algorithm is needed to translate the internal structure to Prolog facts and rules. Accordingly, VIPS can be easily ported to another system for execution if it has an available Prolog interpreter.

The **mapping algorithm** given below translates the internal structure synthesized by VIPS to Prolog facts and rules.

```
Step 1: Assert the goal
        ?- consult('syslib.pro').
Step 2: Repeat
            get a node from the resulting file generated by the interface language;
            if the node encountered is a SN then
                begin { Assert 4 facts and 1 rule }
                    vform('System-name',0,_,0,0,0,0,first_page_list).
                    coordinate(0,0).    { initial setup }
                    procedure(0).
                    workset([0]).
                    go:-valid(0), fcn(_,0,first_page_list).
                end;
            if the node encountered is a FCN then
                begin { Assert 1 fact and 1 optional rule }
                    vform(name,id,alias,row,col,width,height,sublist).
                    if this FCN is non-atom then
                        begin   { Assert 1 rule }
                            go:-valid(id), fcn(name,id,sublist).
                        end;
                end;
            if the node encountered is a MN then
                begin { Assert 1 facts }
                    content(name:id_of_FCN,type,characteristic,contents).
                end;
            if ·he node encountered is a AN then
                begin { Assert 1 rule }
                    procedure(id_of_FCN):-condition and action.
                end;
        until no_more_node;
Setp 3: Assert the goal
        ?-go.    { start execution }
```

In the algorithm, the information kept in FCN, MN, and AN are directly mapped to the predicates: **vform**, **content**, and **procedure** respectively. Predicates **coordinate** and **workset** offer the scope of selection during execution. The other predicates not mentioned above are all system predicates in the file **syslib.pro**, which are consulted in step 1. Table 5.1 shows a summary of these system predicates.

Table 5.1 Summary of system predicates in the file 'syslib.pro'

Category	System Predicates
Control Strategy	fcn, fired, valid, position
Database Manipulation	make_workset
Window Management	show
External Communication	get_response, system

At the beginning, the facts: coordinate(0,0), procedure(0), and workset([0]) give a unconditional display of the V-Form in Fig. 4.3(a) and then an unification is initiated by the predicate: **fcn**. This perdicate accepts user's input (from a mouse or a pointing device) to select one the next appropriate V-Forms. V-Forms with either TEXT or VFORM type are directly anipulated by window management predicate, while GRAPH, VOICE, ANIMATION, or BIT-MAP are delivered to a graphics or voice subsystem for execution by predicate: **system**.

Applying the mapping algorithm to the internal structure generated as shown in Fig. 3.3, we have:

```
?- consult('syslib.pro').

vform('CAI_Course',0,_,0,0,0,0,[topic:1,pic1:2,pic2:3,ask:4,yes:5,no:6]).
coordinate(0,0).
procedure(0).
workset([0]).
go:-valid(0),fcn(_,0, [topic:1,pic1:2,pic2:3,ask:4,yes:5,no:6]).

vform(topic,1,0,6,21,38,1,[]).
procedure(1).
content(topic:1,text,bold,'Theorem proving on triangle coincidence').

vform(pic1,2,0,2,35,10,4,[]).
procedure(2):-system("r circle1").
content(pic1:2,animation,normal,'...').
```

```
vform(pic2,3,0,9,35,10,4,[]).
procedure(3):-system("r circle2").
content(pic2:3,animation,normal,'...').

vform(ask,4,0,14,20,40,3,[]).
procedure(4).
content(ask:4,text,normal,'The following...').

vform(yes,5,0,17,4,8,1,[figA:7,figB:8,figC:9,figD:10,figE:11,
      query:12,ansB:13,ansC:14,ansD:15,ansE:16,again:17,good:18]).
procedure(5).
go:-valid(5), fcn(yes,5, [figA:7,figB:8,figC:9,figD:10,figE:11,
      query:12,ansB:13,ansC:14,ansD:15,ansE:16,again:17,good:18]).

vform(no,6,0,17,66,8,1,[]).
procedure(6):-halt.
go:-valid(6),fcn(no,6,[]).
```

6. CONCLUSION

VIPS provides a visual programming environment for teachers to develop their own coursewares without going into detailed programming that must be adhered strictly to syntax rules. 'hat they need is through what-they-sketch-is-what-they-get operations to imitate their teaching in assroom. Under this environment, teachers with or without any programming language background can describe their specialized knowledge through processing VIPS to get the desired applications. This will release teachers from doing arduous and tedious works.

VIPS has a V-Form Definition Language and a V-Form Manipulation Language to support the teacher interfacing with the system in visual. VIPS then structuralizes V-Forms into an unique internal structure. A mapping algorithm is deviced to translate the internal structure into Prolog predicates. These predicates can be executed by a CProlog interpreter under the environment of VAX/VMS on VAX 11/785.

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 V'PS in the near future. As a matter of fact, CAI courseware that can process the production rules will be proposed soon. But still, a theoretical understanding of visual perception is needed so that a designer can devise knowledge representations for wider objects in a more general way.

REFERENCES

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

[2] 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 (June 1986).

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

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

[5] 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).

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

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

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

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

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

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

[12] Kitagawa, H., et al., "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).

[13] Kowalski, R. A., "Predicate Logic as Programming Language," Proc. of IFIP 74, Stockholm (1974).

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

[15] 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).

[16] Shu, N. C., et al., "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).

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

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

[19] Sugihara, K., et al., "An Approach To The Design of a Form Language," First IEEE Computer Society Workshop on Visual Language, pp.171-176 (Dec. 1984). [20]Tsichritzis, D. C., "Form Management," Comm. of the ACM, Vol. 25, No. 7, pp. 453-478 (1982).

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

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

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