ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1992
Seria: MECHANIKA z. 108 Nr kol. 1161

International Conference on
COMPUTER INTEGRATED MANUFACTURING
Internationale Konferenz über
RECHNERINTEGRIERTE FERTIGUNGSSYSTEME
Zakopane, March 24-27 1992

Wojciech CHOLEWA

Chair of Fundamentals of Machine Design
Department of Mechanical Engineering
Silesian Technical University, Gliwice, Poland

# APPLICATION OF FRAMES IN AN EXPERT SYSTEM SHELL

Summary. A *frame* is a flexible representation of statements and rules. VV_SHELL is an expert system shell composed of a frame interpreter and tools for programmers. The main features of VV_SHELL have been presented in this paper.

## 1. Introduction

Most expert systems use a knowledge base. It has been assumed that a knowledge base is a collection of statements describing relationships between entities of the real world as well as abstract concepts. Such statements are variously called sentences, clauses, formulas and most often facts. Of course from the direct use of the notion *fact* there may result a lot of misinterpretations, because the particular statements are not independent, real existing facts. They are only some belief that something has happened or has been done.

A common way to represent statements is an object-attribute-value triple, used eg. in the pattern expert system MYCIN [5] dedicated to medical applications. Attributes are general characteristics of properties possessed by objects. The value specifies the particular nature of a property in a given situation. Of course, the sets of such triples are flat (they contain no underlying structure) and the maintenance of great sets is strongly difficult. A dozen of different idea of structuring the knowledge base has been proposed and implemented. Very

important is the idea of frames. It is an example of object-oriented programming, which makes it possible to generalize, classify and generate abstractions.

## 2. VV_SHELL

VV_SHELL [2] runs on the IBM PC family of computers. It is an expert system shell composed of the production system, frame interpreter, frame editor and debugger. The frame interpreter of VV_SHELL is a processing unit specially designated to handle different types of statement structures, represented by means of frames. It can also handle so called approximate statements. LISP-like frame description language allows to take into account different kinds of inheritance of frames.

Menus of VV_SHELL contain the following main items:

- FRAME EDITOR
  - Load frames from a text file
  - Save frames to a text file
  - Edit a file with frame description
  - Load from binary file
  - Save to binary file
  - View a frame
  - View an object
- UTILITIES
  - MEMORY UNITS
    - Memory description
    - Unit description
  - DICTIONARY
    - Description of a dictionary unit
    - Unit description
  - COLOURS OF TEXT WINDOWS
    - MODIFY COLOURS
    - Initialize the descriptions of windows
    - Load the descriptions of windows from a file
    - Save the descriptions of windows to a file
- Load from binary file
- Main task (goal)
- Memory description
- Toggle job log (on/off)
- History
- DOS Shell
- Toggle menu off
- End (Esc)

## 3. Frames in VV_SHELL

The notion of frames was introduced by Minsky [4] (see also [1] and [3]). His basic goal was concerned with the designing of a data-base containing encyclopedic knowledge, needed in commonsense reasoning. Frames offer high computational efficiency. They are an interesting tool for the designing of interfaces with users and with external sources of data (eg. measuring devices).

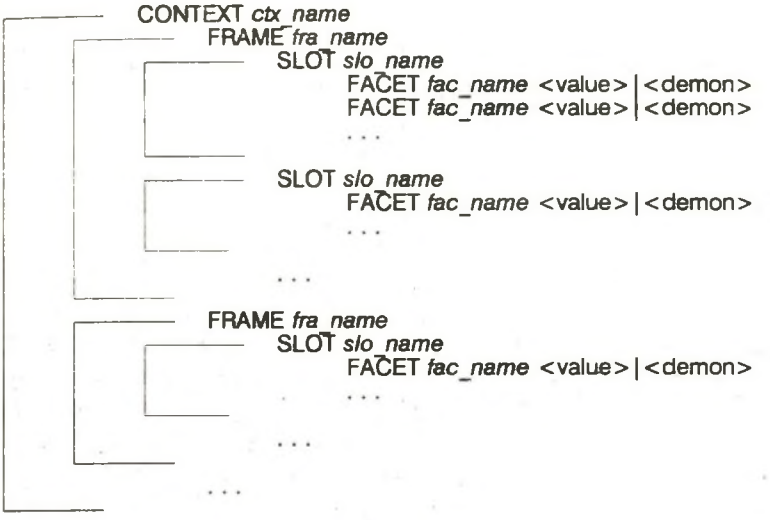


**Fig.1**. Elements of a frame

A frame is a description of a real or an abstract object. It is a special form of data and code structure. A frame contains slots representing attributes of the object. Slots may be interpreted as special representations of statements. Slots contain facets connected with values, default values and/or procedures (called demons) by which the values may be obtained (see Fig.1). It is important to point out that each facet can contain values or demons. Such an inclusion of demons in frames joins procedural and declarative representations.

Examples of special functions included in frame interpreter of VV_SHELL:

- frame processing functions
  `(COPY fac), (DEL fac), (FIX fac val), (GET slo),`
  `(SET val fac), (VIEW slo), (VIEW_FIX slo), . . .`
- task arranging functions
  `(EXIT errlev), (GOAL val . . .), (IF cond val1 val2),`

```
(RUN_GOAL ctx), (WHILE cond val), . . .
```

- list processing functions

```
(HEAD lst), (SEL lst pos), (TAIL lst), . . .
```

- uncertainty processing functions

```
(X_AGR sa sb), (X_AND sa sb), (X_IMP sa sb),
(X_NOT sa sb), (X_OR sa sb), . . .
```

- user interface functions

```
(CONFIRM txt), (DISPLAY val1 val2 ...),
(PROMPT typ val1 val2), . . .
```

All the elements of frames are identified by their names. The names ought to be locally unique. It means, eg., that all slots in a given frame ought to posses individual, different names. Global uniqueness of names is not required, i.e. we can set the same name for slots in different frames. Such an assumption results in a polymorphism - the names are shared and their meaning depends on the given context.

## 4. Encapsulation

Encapsulation is a property postulated by object-oriented-programming. It states that data structures and procedures which are to manipulate the data ought to be coupled together and isolated to some degree from direct access by other procedures. The frames enable us to control the degree of encapsulation. This is achieved by the use of demons (making no difference between data and code) and by the following two possibilities for the pointing of a slot:

- slot may be pointed out by its name,
- slot may be pointed out by its qualified name, i.e. by the pair composed of the name of the slot and the name of a frame to which this slot belongs.

In the first case we obtain an access to the slot (for which we are looking) in the current frame. In the second case access is obtained to a particular slot in a given frame. This allows us to use some slots as global and some slots as local (private) ones. In both cases all the assumptions about the inheritance are valued.

## 5. Inheritance

Inheritance is very powerful feature of frame structures. Frames may be arranged in hierarchical structures (see Fig.2, Fig.3) which make it possible to develop and process the idea about classes without being disturbed by details of any particular object. Such structures are given by links called AKO (a kind of) between superframes (parent frames) and subframes (derived frames). In VV_SHELL such links are listed as follows:

```
FRAME SubFrame_1                                    /* see Fig.2 */
   SLOT ako
```

```
        value = (Frame)

FRAME Frame                                          /* see Fig.3 */
   SLOT ako
      value = (SuperFrame_1 SuperFrame_2 SuperFrame_3)
```

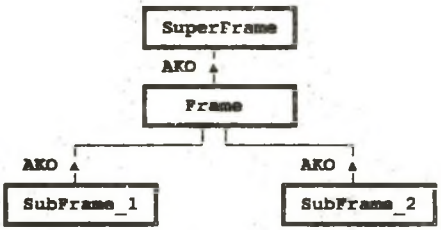


**Fig.2.** Simple hierarchies (there exists at most only one superframe for each frame)

Searching for a slot value for a frame is the *basic task* in frame systems. Special properties are assigned to the facets: *value*, *if_needed*, *if_added* and *if_removed*. The slot value is assigned to the facet *value*. When we are looking for the value of the slot, the content of the facet *value* is returned. If such a facet is missing, the facet *if_needed* points to the value or to a demon returning the value. Slots that are not present in a frame are inherited from superframes. Searching returns alternatively:

- a list containing values of the same slots in all superframes,
- only the first value is found; this value can override values in other superframes.

Inheritance is the most important feature of frames, which makes it possible to eliminate a redundancy of data and to handle exceptions. It can also be used to generate reasonable default data or assumptions in the case of incomplete information. Special facets, such as *if_added* and *if_needed* may be applied for forward and backward chaining, respectively.

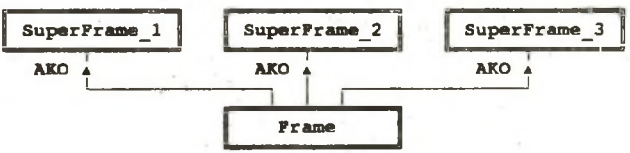


**Fig.3.** Multiple inheritance (each frame can posses a few superframes)

The simple hierarchy appears as a tree structure of frames (see Fig.2). For such structures a search path is given by the AKO links, starting from the selected node upwards and search time grows, at worst, linearly with the number of nodes. More advanced multiple inheritance results in a directed acyclic graph (see Fig.3), for which the strategies for *depth first* or

*breadth first* searching ought to be applied, where

- the results depend strongly on the arrangement of superframes,
- search time grows, at worst, exponentially with the number of nodes,
- redundant searches may be expected (for depth-first search).

## 5.1. An example

In order to illustrate the notion of inheritance let us consider the following listing of frames:

```
FRAME Root
     SLOT goal
          if_needed = (RET_YES (DISPLAY nl "Test VIEW_FIX")
                               (VIEW_FIX (data z))
                               (DISPLAY nl "Test VIEW")
                               (VIEW (data z))
                               (DISPLAY nl "Main goal is achieved."))
     SLOT x
          value = 5.55
     SLOT y
          value = 6.66

FRAME data
     SLOT ako
          value = (reading_input_data)

FRAME reading_input_data
     SLOT x
          if_needed = (SET (PROMPT FLOAT "x = ")
                           (reading_input_data))
          if_added  = (DISPLAY nl "Frame " (FRA) ":  x := ")
     SLOT y
          if_needed = (SET (PROMPT FLOAT "y = "))
          if_added  = (DISPLAY nl "Frame " (FRA) ":  y := ")
     SLOT z
          if_needed = (DISPLAY nl "x = " (VIEW x)
                               nl "y = " (VIEW y)
                               nl "(MAX x y) = " (MAX (VIEW x) (VIEW y))
                               nl "(MIN x y) = " (MIN (VIEW x) (VIEW y)))
```

as well as the following listing of messages obtained from the frame interpreter:

```
Test VIEW_FIX
Frame reading_input_data:  x := 1.11
Frame data:  y := 2.22
x = 1.11
y = 2.22
(MAX x y) = 2.22
(MIN x y) = 1.11
```

```
Test VIEW
x = 5.55
y = 6.66
(MAX x y) = 6.66
(MIN x y) = 5.55
Main goal is achieved.
```

The first step of the main task *(DISPLAY nl "Test VIEW_FIX")* results in the message *Test VIEW_FIX*.

For the next step *(VIEW_FIX (data z))* it is necessary to obtain the value of the slot *z* in the frame *data*. Such a slot is not included in the frame *data*. It is inherited from the frame *reading_input_data* pointed out in the slot *ako of the frame* data. The slot *z* does not contain a direct value. To obtain the value it is necessary to run a demon *if_needed* of this slot. The demon displays the message *x =* , and looks for the value of the slot *x*. Searching starts from the temporary root in the frame *data*, which is defined by the function *(VIEW_FIX ...)*. The slot *x* can be found in the frame *reading_input_data*. Its value is unknown and the demon *if_needed* is started. The demon asks for the current value by means of the function *(PROMPT FLOAT "x = ")*. Let us assume that the user responds with the value 1.11. The demon writes this value in the new facet *value*. Writing the facet starts the demon *if_added*, which sends the message *Frame reading_input_data: x := 1.11* . The value of the slot *y* is obtained in a similar way. The functions *(MAX (VIEW x) (VIEW y))* and *(MIN (VIEW x) (VIEW y))* calculate the returned values by means of data obtained from the new facets. It is not necessary to put any question to the user at this moment.

The third step of the main task *(DISPLAY nl "Test VIEW")* results in the message Test VIEW.

For the next step *(VIEW (data z))* of the main task the values of the slots *x* and *y* are found directly in the frame *Root*, because the function *(VIEW ...)* does not change the current root frame, as it was done by the function *(VIEW_FIX ...)*.

The last step of the main task results in the message *Main goal is achieved*.

## 6. Remarks

The investigations presented in this paper have been partially supported by the research project PB-132/9/91/KBN Poland. An educational version of VV_SHELL may be obtained free of charge, directly from the author. Of course the user's manual [2] exists at the moment only in Polish.

## 7. References

[1] BARTLETT F.C.: *Remembering.* The University Press, Cambridge 1932.

[2] CHOLEWA W. et al: *VV_SHELL User's Guide and Reference Manual* (in Polish). Report G-547/RMT-4/91, Technical University, Gliwice 1991.

[3] GOFFMAN E.: *Frame Analysis.* Harper & Row, New York 1974.

[4] MINSKY M.: *A Framework for Representing Knowledge.* [in:] Computers and Thought. [ed.:] WINSTON P.H.; McGraw-Hill, New York 1975, p.211-277.

[5] SHORTLIFFE E.H.: *Computer-based medical consultation MYCIN.* Elsevier, New York 1976.

ANWENDUNG VON RAHMEN IN EINEM EXPERTENSYSTEM

Zusammenfassung

So genannte Rahmen bilden eine ausreichend flexible Repräsentation von Ausssagen. VV_SHELL ist ein Expertensystemshell mit dem Interpreter von Rahmen und mit den Software-Tools für den Programmierer. Im vorliegenden Aufsatz werden einige Merkmale von VV_SHELL dargestellt.

## ZASTOSOWANIE RAM W SZKIELETOWYM SYSTEMIE DORADCZYM

Streszczenie

Ramy stanowią elastyczne reprezentacje stwierdzeń. VV_SHELL jest szkieletowym systemem doradczym, składającym się z interpretatora ram i narzędzi dla programisty. W pracy opisano podstawowe cechy systemu VV_SHELL.

Wpłynęło do redakcji w styczniu 1992 r. Recenzent: Ewald Macha