PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1984,
edited by ICOT.

Knowledge Representation and INference Environment: KRINE, --- An Approach to Integration of Frame, Prolog and Graphics.

Yutaka Ogawa, Kenichi Shima, Toshiharu Sugawara and Shigeru Takagi

Musashino Electrical Communication Laboratory
Nippon Telegraph and Telephone Public Corporation
3-9-11, Midori-cho, Musashino-shi, Tokyo, 180, Japan

ABSTRACT

It is well known that an expert system is an intelligent computer program intended to solve complex problems by means of inference using domain specific knowledge extracted from human experts in the domain. Especially, in order to develop an expert system for intelligent CAD, we will have much difficulty in describing the design knowledge including pattern matching procedures, in applying the design knowledge by trial and error or in displaying design objects dynamically, if only conventional object-oriented or frame-based knowledge representation tools are used. To cope with these problems, the following mechanisms were integrated with the frame-based knowledge representation system; (a) PROLOG programming functions with direct frame data unifier, (b) frame data recovery for backtracking of inference and (c) design object display functions by interpreting frame data directly. This paper describes the design philosophy, mechanisms and examples of integrated knowledge representation and inference environment (KRINE).

1. INTRODUCTION

Recently, many research efforts have been expended on software systems, that solve complex problems by means of inference using domain specific knowledge extracted from human experts (expert system). Tasks of expert systems range widely from diagnosis problems to design problems. In particular, in case of an expert system to solve design problems, hierarchical representation of design objects, representation and application control of design knowledge, and dynamic display and interactive editing of design objects (man-machine interface) are deemed essential.

UNITS (Smith et al. 1980, Stefik 1979), LOOPS (Bobrow et al. 1983) and Smalltalk (Goldberg et al. 1983) can offer functions related to hierarchical representation of design objects. However, because design knowledge requires explicit search and manipulation of hierarchical structure patterns among design objects and because simulations require dynamic display of the design object's structures, these conventional knowledge representation tools are not sufficient.

To cope with these problems, the authors have made a new knowledge representation and inference environment (KRINE) that integrates several knowledge representation paradigms. This is because different paradigms are appropriate for different purposes. KRINE has fundamental frame-based knowledge representation mechanisms (object-oriented, procedure-oriented and manipulation oriented) and also integrates the following mechanisms;

(i) PROLOG programming mechanisms with frame unification functions (logic-oriented) and rule representation mechanisms based on the PROLOG programming mechanisms (rule-oriented),
(ii) Frame-based graphic environment, where frame data are directly interpreted and dynamically displayed.

At first, this paper shows background, design issues and basic concepts of KRINE knowledge representation paradigms as the design philosophy. Next, it shows system structure and KRINE mechanisms. They are fundamental frame mechanisms, frame-based logic programming mechanism and frame based graphic environment. Finally, KRINE knowledge representation examples are mentioned.

2. DESIGN PHILOSOPHY

2.1 Background for KRINE

The problem solving tasks for expert systems can be classified into several categories, such as diagnosis problem, analysis problem, planning problem or design problem (Stefik et al. 1982). The design problem can be defined as the creation of an apparatus that consists of a lot of parts by applying design operations to its abstract specification. This problem is classified as one of the most difficult problems for expert systems.

To develop an expert system that solves design problems (Design Expert system: DE) (Takagi, Ogawa and Saito 1983, Takagi 1984), the following knowledge representation and inference mechanisms are required;

-- Representation of design objects

Design objects must be represented hierarchically according to parts-device hierarchy. For example, a computer hardware can be defined hierarchically; logical specifications, block diagrams, gate circuits or cell structures.

-- Representation of design knowledge

Design operations, such as the transformation or synthesis of design objects, have to be represented as design knowledge. These operations require pattern matching procedures that search for particular patterns in design objects.

-- Backtracking control for applications of design knowledge

Design operations are always applied by trial and error. When one operation fails, it is necessary to recover the previous design object's state in order to apply another operation.

-- Display of design objects (man-machine interface)

It is essential not only to display current design objects hierarchically but also to re-display them as soon as they are modified.

2.2 KRINE Design Issues

From the background viewpoint as mentioned in Section 2.1, the following KRINE (Knowledge Representation and INference Environment) design issues were decided upon.

(1) Multiple knowledge representation paradigms that are appropriate for their application methods or objectives.

It is difficult to represent various types of knowledge, such as design knowledge or design objects themselves, by means of only one knowledge representation paradigm. This requires the integration of knowledge representation paradigms.

(2) Extendability wherein a user can easily add user's specific functions to system functions.

This requires self-editing functions that can easily add or modify system functions by users.

(3) Pattern matching mechanism that searches for specified patterns in the knowledge base

(4) Knowledge base recovery functions

When one design operation fails, it is necessary to recover the previous design object's state in order to apply the next operation.

(5) User interface that visualize design objects

It is necessary to display design objects by means of direct interpretation of knowledge structures and to edit them by a screen editor.

2.3 KRINE Knowledge Representation Paradigms

To cope with the design issues described in Section 2.2, multiple knowledge representation paradigms and mechanisms were prepared based on frame data structures, because knowledge representation systems based on Frame theory (Smith et al. 1980, Stefik 1979) are suited for hierarchical knowledge. This section describes the basic idea regarding our six knowledge representation paradigms. Mechanisms that actualize these paradigms are described in Section 4.

OBJECT ORIENTED KNOWLEDGE REPRESENTATION

This representation consists of entities called "object" which has procedures and data aspects. Each procedure is activated by sending a message to it with some token. The procedure acts according to the token and gives a response. It is convenient to use such uniform protocols. As a feature, all objects are connected by an inheritance network; An object has the same property as the object of its super concept. KRINE identifies a frame with an object. A frame has the value called "inheritance role" to decide its inheritance mode.

PROCEDURE ORIENTED KNOWLEDGE REPRESENTATION

The knowledge of this representation method composes instructions. In this paradigms, data and procedures are separated in contrast with the object oriented knowledge representation. In KRINE, LISP (MacLisp) is used as a language for this paradigm.

MANIPULATION ORIENTED KNOWLEDGE REPRESENTATION

When the specified frame, slot or stored value, which is termed "active data" , is manipulated, the mechanism of this representation activates some procedures that have been specified in advance. This links implicitly frame manipulations to other procedures.

In KRINE, a user can create a special field to activate its slot or stored value (the special frame-field to activate frame) and can store procedures in the field. When the active data are manipulated, a specified procedure is invoked.

LOGIC ORIENTED KNOWLEDGE REPRESENTATION

This method represents knowledge with the first order logic (Horn clause) like PROLOG. It consists of the declarative clauses , axioms and goal clauses. If a goal clause is given, the mechanism of this paradigm searches for declarative clauses and axioms to prove it. This paradigm is appropriate for representing the knowledge that has logical features. KRINE has a prolog-like inference mechanism which is called "PRIME", and which can interpret frames as clauses and unify them.

RULE ORIENTED KNOWLEDGE REPRESENTATION

This represents knowledge with a pair of premise and action (If ... Then ~). KRINE has a rule interpreter and can set some rules in a frame. The rules are activated by sending a message to the frame.

FRAME-BASED GRAPHIC ENVIRONMENT

Since most design objects have modular and hierarchical structures, they are easily represented in KRINE frame data structures. Moreover, it is essential to display and edit current design objects. Therefore, KRINE manages figures of design objects by corresponding them to frame data structures. As a result, KRINE can uniformly manage figure's attributes, invocation of figure manipulation functions and hierarchical relations between figures, based on frame data structures. With this paradigm, a user can easily display figures of design objects. For example, a user can display simulation results dynamically, merely by changing the frame's values.

3. SYSTEM ORGANIZATION

KRINE offers six knowledge representation paradigms described in Section 2.3 on LISP processing system. Figure 1 shows KRINE system organization. All paradigms are based on frame data structures. Each knowledge is based on one paradigm and it can communicate with or refer to the knowledge based on the other paradigms by means of a message passing facility. So, users can easily develop expert systems that combine multiple knowledge representation paradigms, because this uniform protocol allows users not to mind what kind of paradigms are used.

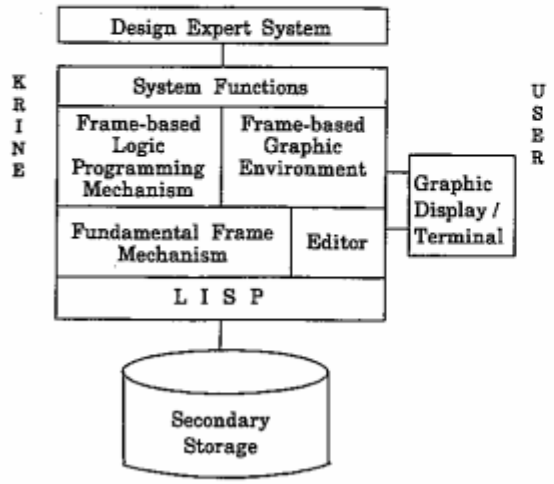

Figure 1 System Organization

4. KRINE MECHANISMS

This section describes the basic data structure, frame and mechanisms that offer six knowledge representation paradigms.

4.1 Frame Structure

Knowledge base is a set of frames that have hierarchical structures. Frames consist of the following entities (See Figure 2);

(1) Name : Name of the frame.

(2) Frame-fields : A frame-field is an area where an optional value is stored. This value is called "frame-field value (with respect to the frame)". One usage of this value is Manipulation Oriented Mechanism.

(3) Group : Group is an area to store special value that classifies frames to focus attention on them in the knowledge base.

(4) Slots : A slot consists of the following subentities;

(4.1) Name : Name of the slot

(4.2) Value : Stored slot value

(4.3) Datatype : This restricts the stored slot values: Only adaptable values are permitted to be stored. Datatype can be chosen among ATOM, FRAME, INTEGER, NUMBER, LISP, PROLOG, RULE, S-EXPR, TEXT, J-TEXT, TABLE and LIST (system's initial setting).

(4.4) Fields : A field consists of a field name and a stored value for the field. This is called "Field Value". The first field in a slot has the value termed "Inheritance Role" that decides the slot inheritance mode by progeny. See (Smith et al. 1980, Goldberg et al. 1983) and Table 1 about the inheritance role. Some special fields are used for Manipulation Oriented Mechanism.

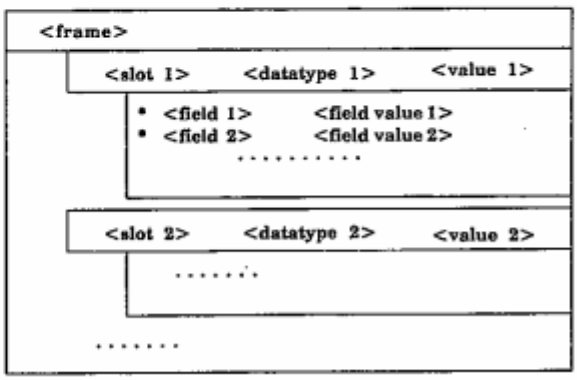

Figure 2 Frame Structure

Table 1 Inheritance Role

	Slot name, datatype and role inheritance	Value inheritance
U	All information is inherited.	The slot value is unique at each level in the hierarchy.
S	do.	Slot in the frame and its progeny have the same value.
R	do.	Slot in the frame can have the only value that the same slot in its parent frame have.
I	No inheritance	No inheritance

4.2 Fundamental Frame Mechanism

4.2.1 Basic Frame Handling

For the basic frame handling, KRINE's system functions and command editor, that is linked to EMACS, can be used. These facilities are almost the same as UNITS (Smith et al. 1980).

4.2.2 Special Frame Mechanisms

(1) Undo Mechanism for Frames

It is often necessary to make trial tentatively to find the best solution for the given problem. Some inference algorithms select one method among candidates which may lead to the best answer and make trial in turn until the best method is found. However, when one of them contains some changes in frames, slots or the stored values, the initial conditions for each trial are different from the previous conditions. Therefore, frame undo mechanism was attached to KRINE in order to do with this trouble.

KRINE has a system function "UNDO" for the undo mechanism. For example,

```
(undo <trial 1>)
(undo <trial 2>
   (undo <trial 2-1>)
   (undo <trial 2-2>))
```

trails (<trial 1> and <trial 2>) are made. Within the second trial, <trail 2-1> and <trial 2-2> are tried tentatively. So, it is possible to undo only a part of each procedure. This mechanism is used to implement the backtracking function in PRIME interpreter.

(2) System Frames

One of KRINE's design issues is to be able to add new facilities easily. To do this, KRINE prepares system frames which control most of the KRINE's facilities. This makes it possible to change the functions by editing the frames. The following entities are written in system frames.

(A) Editor commands and their functions.
(B) All kinds of datatypes
(C) The manner for making response to received messages
(D) Basic handling of virtual memory mechanism
(E) Definition of fields which is related to Manipulation Oriented mechanism.
(F) All kinds of inheritance roles and their inheritance manner
(G) Definition of focus of attention with group

4.3 Frame Based Logic Programming Mechanisms

This section describes frame-based logic programming mechanisms, such as PROLOG programming mechanism with frame unifier PRIME (PRolog wIth fraME unifier) and rule representation mechanisms.

4.3.1 Basic Structure

PRIME syntax is described in the following BNF (Backus normal form) notation;

<clause>::= {<plus literal>}{<minus literal>}:
Each clause can have at most one plus literal.

<plus literal>::= + (<predicate> {<term>})
This is used to represent the head of a clause (Prolog statement).

<minus literal>::= - (<predicate> {<term>})
Series of minus literals are used to represent the body of a clause (Prolog statement).

```
<term>::= <variable>| <constant>| <list>|
          (<function> { <term>'} )
<variable>::= * <symbol>
```

PRIME expands descriptions of PRIME's minus literal as the following three forms(each form is a list);

(1) Prolog predicate

All the pre-defined prolog predicates can be invoked by PRIME. Calling sequence for a prolog predicate is as follows;

```
(<predicate name> { <parameter> } )
```

(2) Lisp function

All the maclisp functions can be invoked by PRIME in the following form.

```
({call} <function name> {<parameter>})
```

The predicate name "CALL" is unnecessary unless the name conflict occurs. All the parameters must be bounded when the lisp function is invoked.

(3) Frame access

All the frames can be accessed by PRIME. The frame access is represented as the triple form

```
(<frame name> <slot name> <slot value>).
```

The above form is interpreted as the following term automatically by the system;

```
(fcall <frame name><slot name><slot value>).
```

The clause can be stored not only in the certain memory area controlled by PRIME (called "clause definition area"),but also in a slot in a frame. This is shown in Fig. 3. (*X,*Y,*Z and *W mean variables.) When the clause is stored in a frame, the frame with that clause name is accessed dynamically at runtime. When the same predicate name exists in both the clause definition area and a frame, the former is used.

```
    COUSIN
                    Prototype is PLPROC
 Slot               Top              Role  Datatype     Data
COUSIN:             *TOP*            (I)   <PROLOG>
  +(COUSIN *X *Y)-(PARENT *X *Z)
                  -(PARENT *Y *W)
                  -(BROTHER *Z *W):
BROTHER:            *TOP*            (I)   <PROLOG>
  +(BROTHER *X *Y)-(*X FATHER *Z)
                  -(*Y FATHER *Z)
                  -(EVAL (EQ *X *Y) NIL):
```

Figure 3. Clause Definition in frame.

4.3.2 Unification Mechanism

PRIME can represent invocations of a frame access, a lisp procedure and a prolog procedure as the same form. (See Section 4.2.1.) The invocation of a rule is performed by the message passing facility. These invocations are accomplished under PRIME's unification mechanism.

Each literal in clauses is processed in the following priority order;

(priority 1) Prolog predicate
(priority 2) Lisp function
(priority 3) Frame access

As usual, the literal can be processed in the above priority except that predicate names are "call" or "fcall". In the case of the frame access, triple forms (frame-slot-value) for unification candidates are produced automatically. If the value is not defined, it is treated as a variable. When the value is bounded, the frame access is executed in accordance with the slot datatype and the inheritance role.

To avoid fruitless access to knowledge base, frame grouping can be used. This is accomplish by dividing the knowledge base into several groups. As a result, searching space becomes limited and searching time is reduced.

4.3.3. Backtracking Mechanism

Each clause is executed according to input resolution. When an unification fails, PRIME releases the binding information, invokes the backtracking mechanism and searches for the next clause automatically.

When the lisp function result value is "NIL", the backtracking mechanism is invoked and PRIME begins processing on a new environment. Moreover, if the lisp function updates the frame information or structure and backtracking occurs, PRIME recovers the frame information up to the previous state automatically, using the undo mechanism (See Section 4.2.2 (2)). This undo mechanism is based on the variable substitution technique. PRIME saves only the minimal data needed to restore the old frame data.

4.3.4 Rule Mechanism

In the rule representation, a user can use the following functions;

(1) Reference to frame data structures.
(2) Invocation of lisp functions.
(3) Proof of prolog predicates.

Rule syntax is shown in the following BNF notation;

```
<RULE> ::= <PREMISE PART> ==> <ACTION PART>
<PREMISE PART> ::= {<RULE TERM>}
<ACTION PART>  ::= {<RULE TERM>}
<RULE TERM>    ::= ( <PREDICATENAME>
                       {<TERM ELEMENT>} ) |
                   ( <FUNCTIONNAME>
                       {<TERM ELEMENT>} ) |
                   ( <TERM ELEMENT>
                       {<TERM ELEMENT>} )
<TERM ELEMENT> ::= <VARIABLE> | <CONSTANT>
<VARIABLE>     ::= *<CONSTANT>
<CONSTANT>     ::= strings of alphanumeric
character
```

The premise part is used mainly in pattern matching for design objects. The action part is used for operations application. These can realize a production system (PS) based on frame data structures. In this PS, all frames are considered as working memory and rules are applied to them. A user can make hierarchical structures of rule sets. Figure 4 shows a rule example.

```
    SIMPLIFY-RULE
                Prototype is PRODUCTION
   circuit graph reduction rule.
 Slot           Top              Role  Datatype     Data
RULE:           PRODUCTION       (U)   <RULE>
   (*L1 TO *NOT1)
   (*NOT1 OUTPUT *L2)
   (*L2 TO *NOT2)
   (*NOT2 OUTPUT *L3)
   ==>
     (DELETE-FRAME *NOT1)
     (DELETE-FRAME *L2)
     (DELETE-FRAME *NOT2)
     (PUTVALUE *L1 TO *L3)
     (PUTVALUE *L3 FROM *L1)
```

Figure 4. Circuit Simplify Rule Example

4.4 Frame-based Graphic Environment

Frame-based graphic environment (GE) visualizes design objects by direct frame data structures interpreting. KRINE graphic primitives are based on ACM CORE standards (Computer Graphics 1979).

4.4.1 Frame-based Figure Management

KRINE manages a segment that is a unit for a CORE system's output primitive as one frame. KRINE stores all attributes of a segment, such as visibility, detectability or highlighting attributes. Also, KRINE manages modeling transformation parameters for each segment. These attributes or parameters are stored as slot values. When a user modifies an attribute by editing the value for the related slot, the KRINE manipulation-oriented mechanism automatically changes the figure on the graphic display at once.

4.4.2 Figure Manipulation by Using Object-oriented Facility

Figure manipulation functions are invoked uniformly by KRINE message passing facility. These functions can be classified into the following two categories;

(1) Meta level figure management functions

These management functions are figure creation and deletion functions. A user can invoke these functions by sending messages to the system frame (SHAPE_SEGMENT).

(2) Object level figure manipulation functions

Object level functions can be invoked by sending a message to the frame related to a figure itself.

- Figure transformation

  A user can transfer or arrange a figure by the following methods;

  (i) Modeling transformation parameters (rotation, parallel transformation and scale amounts)

  (ii) Arrangement of a figure on an arbitrary coordinate

  (iii) Rotation of a figure through an arbitrary amount around an arbitrary axis

- Setting up and modifying figure attributes

  This can be done by editing the related slot values.

4.4.3 Display of Design Objects Hierarchical Relations

The hierarchical relations for design objects in KRINE are classified into the following two categories;

(1) IS-A relation (prototype-instance relation)

This relation is used where many instances are created from several prototypes. For example, in digital circuits, definitions of circuit elements are prototypes (such as AND, NOT gate and so on) and individual elements that organize the circuit are instances of prototypes.

- Prototype figure definition

  A user can create a prototype figure by sending a message to the manager frame. This message contains the graphic information about the figure.

- Instance figure creation

  A user can create an instance figure by sending a message to its prototype figure frame.

(2) PART-OF relation

Individual parts are designed independently and have their own coordinates. Therefore, in order to make up a master apparatus, it is necessary to specify relative positions on the apparatus's coordinate (master coordinate) as the PART-OF relation. KRINE graphic environment offers mechanisms to set up this relation easily by the following methods;

- Relative rotation, parallel transformation and scale amounts through X,Y and Z axis of master coordinates (relative modeling transformation parameters)
- An arbitrary coordinate on the master coordinate
- A rotation amount around an arbitrary axis on a master coordinate

When a user changes the position of a master apparatus by figure transformation functions described in Section 4.3.2, all positions of figures on the master apparatus are automatically changed, where relative positions between the master apparatus and its slave figures are left unchanged.

5. EXAMPLE

This section describes design object and design knowledge examples of Design Expert System (Takagi, Ogawa and Saito 1983, Takagi 1984) for computer hardware logic design, based on KRINE.

5.1 Design Object Examples

Figure 5 shows a function block diagram as an example of design objects based on KRINE. Figure 6 shows a frame hierarchy example of Figure 5. In figure 6, common data path component concepts, such as register, ALU, multiplexor, or bus, are defined as prototypes. Every concrete elements, such as GR0, ALU1, MPX-1, or BUSA, are defined as instances. Instance frame examples (ALU1 and L1) are shown in Figure 7. The ALU1 frame has input/output terminals, its logical function and its graphical information. The shape information of ALU1 is inherited from its prototype BINARY-ALU (IS-A relation). The graphical information of ALU1, such as DISPLACEMENT, ROTATION, VISIBILITY,

HIGHLIGHTING and DETECTABILITY, can be uniquely defined. When this graphical information is changed by system functions or the frame editor, the shape displayed is automatically changed based on the data manipulation mechanism (the invoked function is set on "updated" field).

5.2 Design Knowledge Examples

Figure 8 shows design knowledge examples. Each design knowledge is represented in PROLOG and stored in a slot of a frame. This design knowledge is used to verify the consistency between a functional specification and its data path and, if inconsistency occurs, to analyze the cause of inconsistency and revise the data path. Since there are several ways to revise the data path, it might occur to have to take another revision method after a revision method is tried. This trial and error can be handled by KRINE UNDO mechanism.

6. CONCLUSION

This paper describes the design philosophy, system structure, mechanisms, and examples of Knowledge Representation and INference Environment (KRINE). KRINE features are as follows;

- Fundamental frame mechanisms
  KRINE offers hierarchical representation for design objects, procedural knowledge representation in LISP and invocation of LISP function, PROLOG program and RULE followed by frame manipulations.
- Frame-based logic programming mechanisms
  KRINE offers a PROLOG programming mechanism that can directly unify frame data in order to describe pattern matching procedures for design knowledge easily. KRINE also offers a frame data backtracking mechanism in order to apply design knowledge easily by trial and error.
- Frame-based graphic environment
  KRINE manages a segment as a frame to visualize design objects by directly interpreting frame data structures. KRINE also maintains hierarchical relations for design objects to allow arranging or editing them easily.

KRINE is now used to represent fundamental Design Expert system's knowledge. This runs under the DEC-20/MACLISP, the VAX/NIL and the Symbolics/ZETALISP environment. The program size for a current KRINE version is 20K lines (fundamental frame system: 10K, frame-based logic programming mechanisms (PRIME): 4K, graphic environment: 6K lines). The authors have a plan to convert KRINE to an ELIS/TAO lisp machine developed by the NTT laboratory and to make a frame-machine as a knowledge base machine.

ACKNOWLEDGMENT

The authors are grateful to K.Yamashita for his kind suggestion, and to project members for their helpful discussion.

REFERENCES

D.G.Bobrow and M.Stefik,"The LOOPS Manual," XEROX PARC Knowledge-Based VLSI Design Group Memo, KB-VLSI-81-13,1981.

"Status Report of the Graphics Standards Planning Committee," Computer Graphics 13 (3), August 1979.

A.Goldberg and D.Robson,"Smalltalk-80:The Language and Its Implementation," Addison-Wesley, 1983.

M.Minsky,"A Framework for Representing Knowledge," in Psychology of Computer Vision, McGraw-Hill, 1975.

R.G.Smith and P.Friedland,"UNIT package user's guide," Stanford Huelistic Programming Project Memo, Memo HPP-80-28,1980.

M.Stefik, "An Examination of a Frame-based Representation System," IJCAI, 1979, pp 845-852.

M.Stefik, J.Aikins, R.Balzer, J.Benoit, L.Birnbaum, F.Hayes-Roth and E.Sacerdoti,"The Organization of Expert Systems, A Tutorial," Artificial Intelligence,18 (1982),pp 135-173.

S.Takagi,"Rule Based Synthesis, Verification and Compensation of Data Path" Proc. of ICCD (1984).

S. Takagi, Y.Ogawa and M.Saito,"Examination for Design Expert System: DE-0 (in Japanese)," The 26th Convention (of IPSJ), 1983.

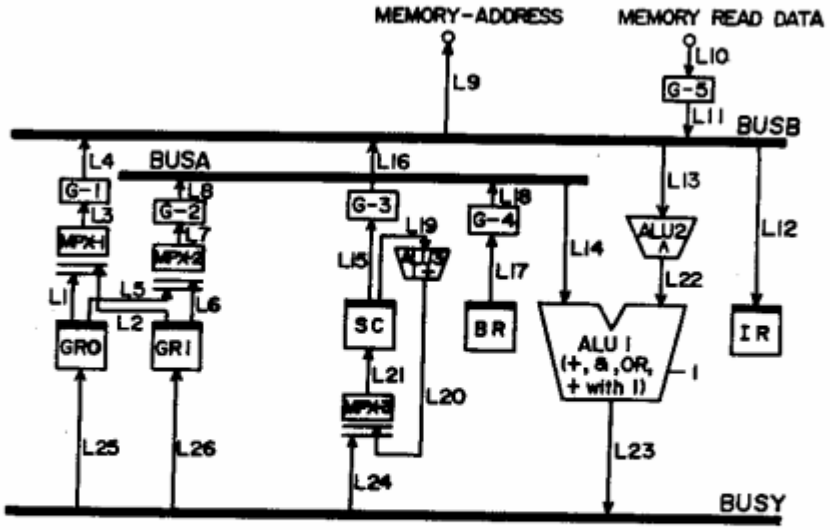

Figure 5. Design Object Example

```
/** DESIGN OBJECT FRAME HIERARCHY **/
HARDWARE
   *SAMPLE-HARDWARE*
   CONTROL-PART
      *SAMPLE-CONTROL-PART*
   DATA-PART
      *SAMPLE-DATA-PART*
     INPUT
        *MEMORY-READ-DATA*
     OUTPUT
        *MEMORY-ADDRESS*
     BINARY-ALU
        *ALU1*
     UNARY-ALU
        *ALU2*
        *ALU3*
     REGISTER
        *GR0*
        *GR1*
        *SC*
        . . .
     MULTIPLEXOR
        *MPX-1*
        *MPX-2*
        . . .
     BUS
        *BUSA*
        *BUSB*
        *BUSY*
     TRI-STATE-GATE
        *G-1*
        *G-2*
        . . .
     LINE
        *L1*
        *L2*
        . . .

/** DISPLAY CONTROLL FRAME HIERARCHY **/
SHAPE_ROOT
    SHAPE_PROTO
    SHAPE_LOCATE
    SHAPE_SEGMENT
    SHAPE_INTERACTION
    SHAPE_CLASS
    SHAPE_VIEW
```

Figure 6. Frame Hierarchy Example

```
    ALU1
          Prototype is BINARY-ALU
 Slot            Top          Role   Datatype      Data
NAME:            HARDWARE     (U)    <ATOM>        ALU1
INPUT1:          BINARY-ALU   (U)    <ATOM>        IN1
INPUT2:          BINARY-ALU   (U)    <ATOM>        IN2
OUTPUT:          BINARY-ALU   (U)    <ATOM>        OUT
FUNCTION:        BINARY-ALU   (U)    <PROLOG>
  +(FUNCTION ((ALU1 OUT) <- (ALU1 IN1) & (ALU1 IN2)) (ALU1 AND)):
  +(FUNCTION ((ALU1 OUT) <- (ALU1 IN1) OR (ALU1 IN2)) (ALU1 OR)):
  +(FUNCTION ((ALU1 OUT) <- (ALU1 IN1) + (ALU1 IN2)) (ALU1 ADD):
  +(FUNCTION ((ALU1 OUT) <- (ALU1 IN2)) (ALU1 THROUGH)):
SEGMENT_NAME     *TOP*        (I)    <INTEGER>     10001
DISPLACEMENT:    *TOP*        (I)    <S-EXPR>      (2000.0 3000.0)
   *UPDATED      TRANS$
ROTATION:        *TOP*        (I)    <S-EXPR>      (0.0 0.0)
   *UPDATED      TRANS$
VISIBILITY:      *TOP*        (I)    <ATOM>        ON
   *UPDATED      G$ATTR
HIGHLIGHTING:    *TOP*        (I)    <ATOM>        OFF
   *UPDATED      G$ATTR
DETECTABILITY:   *TOP*        (I)    <ATOM>        ON
   *UPDATED      G$ATTR

    L1
          Prototype is LINE
 Slot            Top          Role   Datatype      Data
NAME:            HARDWARE     (U)    <ATOM>        L1
FROM:            LINE         (U)    <S-EXPR>      (GR0 OUT)
TO:              LINE         (U)    <S-EXPR>      (MPX-1 1)
ROUTE:           LINE         (U)    <S-EXPR>
                                        ((POLYLINE ((-10000.0 3500.0)
                                                    (-10000.0 4500.0))))
SEGMENT_NAME     *TOP*        (I)    <INTEGER>     10002
 . . .
```

Figure 7. Design Object Frame Example

```
/** design method frame **/
  DATA-PATH-VERIFIER
          Prototype is PLPROC
 Slot           Top          Role     Datatype     Data
VERIFIER:       *TOP *       (I)      <PROLOG>
+(VERIFIER (*D <- *S1 *F *S2))
     -(OR ((VERIFY (*D <- *S1 *F *S2) *RESULT)
           (INFORM-RESULT *RESULT))
          ((WHY-FAILED? *TYPE (*D <- *S1 *F *S2) *FAILURE-INF)
           (REVISE-DATA-PATH *TYPE (*D <- *S1 *F *S2) *FAILURE-INF)
           (ASK-IF-ACCEPTABLE))
          ((NOT-VERIFIED (*D <- *S1 *F *S2)))) :
   . . .

  VERIFY
          Prototype is PLPROC
 Slot           Top          Role      Datatype     Data
VERIFY:         *TOP *       (I)       <PROLOG>
+(VERIFY (*D <- *S1 *F *S2) (*SUB-DATA-PATH *PATH1 *PATH2 *PATH3))
     -(FIND-FUNCTION (*Z <- *X *F *Y) *SUB-DATA-PATH)
     -(FIND-PATH *S1 *X *PATH1)
     -(FIND-PATH *S2 *Y *PATH2)
     -(FIND-PATH *Z *D *PATH3) :
  . . .
FIND-FUNCTION:  *TOP *       (I)       <PROLOG>
+(FIND-FUNCTION (*Z <- *X *F *Y) *SUB-DATA-PATH)
     -(FUNCTION (*Z <- *X *F *Y) *SUB-DATA-PATH) :
+(FIND-FUNCTION (*Z <- *X - *Y) (*SUB-DATA-PATH1 *PATH *SUB-DATA-PATH2))
     -(FIND-FUNCTION (*A <- ^ *Y) *SUB-DATA-PATH)
     -(FIND-FUNCTION (*Z <- *X + *B + 1) *SUB-DATA-PATH2)
     -(FIND-PATH  *A *B *PATH) :
   . . .
FIND-PATH:      *TOP *       (I)       <PROLOG>
+(FIND-PATH *FROM *TO *PATH)-(A-PATH *FROM *TO *PATH):
+(FIND-PATH *FROM *TO (*PATH0 . *PATH1))
     -(A-PATH *FROM *X *PATH0)
     -(A-PATH *X *TO *PATH1) :
   . . .
A-PATH:         *TOP *       (I)       <PROLOG>
+(A-PATH *FROM *TO *LINE)-(*LINE TO *TO)-(*LINE FROM *FROM):
+(A-PATH *FROM *TO *PATH)-(FIND-FUNCTION (*TO <- *FROM) *PATH):
   . . .

  FAILURE-ANALYSIS
          Prototype is PLPROC
 Slot           Top          Role      Datatype  Data
WHY-FAILED?:    *TOP *       (I)       <PROLOG>
+(WHY-FAILED? LACK-OF-FUNCTION (*D <- *S1 *F *S2) *MODULE)
     -(TYPE ALU *MODULE)
     -(DO-NOT-HAVE-FUNCTION *MODULE *F)
     -(FUNCTION (*? *Z *X *Y) (*MODULE *CTL))
     -(FIND-PATH *Z *D *PATH1)
     -(FIND-PATH *S1 *X *PATH2)
     -(FIND-PATH *S2 *Y *PATH3):
+(WHY-FAILED? LACK-OF-PATH-TO-DESTINATION (*D <- *S1 *F *S2) (*Z *D))
     -(VERIFY (*Z <- *S1 *F *S2) *PATH0)
     -(NOT (FIND-PATH *Z *D *PATH1)):
   . . .

REVISE-DATA-PATH
          Prototype is PLPROC
 Slot           Top          Role      Datatype    Data
REVISE:         *TOP *       (I)       <PROLOG>
+(REVISE-DATA-PATH *TYPE (*D <- *S1 *F *S2) *FAILURE-INF)
     -(CCALL (*TYPE (*D <- *S1 *F *S2) *FAILURE-INF)) :
LACK-OF-PATH-TO-DESTINATION: *TOP * (I) <PROLOG>
+(LACK-OF-PATH-TO-DESTINATION (*D <- *S1 *F *S2) (*Z *D))
     -(CONNECTED-TO-BUS *Z *BUS)
     -(NO-CONNECTION *D)
     -(CONNECT *BUS *D):
CONNECT:        *TOP *       (I)       <PROLOG>
+(CONNECT *FROM *TO)
     -(MAKE-INSTANCE LINE *LINE-NAME)
     -(PUTVALUE *LINE-NAME FROM *FROM)
     -(PUTVALUE *LINE-NAME TO   *TO):
   . . .
```

Figure 8. Design Method Frame Example