ICOT Technical Memorandum: TM-0809

TM-0809

An Object-Oriented and Constraint-Based Knowledge Representation System for Design Object Modeling

by
T. Yokoyama

October, 1989

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456 3191-5
Telex ICOT J32964

Institute for New Generation Computer Technology

An Object-Oriented and Constraint-Based Knowledge Representation System for Design Object Modeling

Takanori YOKOYAMA

Institute for New Generation Computer Technology
4-28, Mita 1-chome, Minato-ku, Tokyo 108, Japan
e-mail: yokoyama%icot.jp@relay.cs.net

Abstract

This paper presents an object-oriented and constraint-based knowledge representation system FREEDOM for design object modeling. An object model represented as a set of objects in this system is not a mere data structure but an active entity which works to solve design problems. Knowledge representation provided in the system, based on the object-oriented paradigm, makes it possible to describe constraints in declarative form. A class hierarchy is represented with *is-a* links and *includes* links, and a class-instance relation can be changed dynamically. These features are useful for top-down refinement. The problem solving mechanism of the system is based on constraint satisfaction techniques. Constraints are declared statically and can be added to objects dynamically. An object has a function to keep its state satisfying given constraints. By this function, we can find values of attributes and classes of objects that satisfy design requirements. The constraint satisfaction method is based on the technique of constraint logic programming and constraint propagation among objects. FREEDOM is implemented using ESP language on a PSI machine.

AI topic: Knowledge representation, Constraint-based problem solving

Domain area: Design

Language/Tool: ESP

Status: Research

Effort: 1.5 Person-years

Impact: Knowledge representation presented here makes it easy to build an effective design expert system.

1 Introduction

A number of design expert systems have been developed in several domains, such as VLSI design and mechanical design. The knowledge used in design expert systems can be classified into two types: knowledge about design objects and knowledge about design methods. Only knowledge about design methods has been regarded as important in conventional systems. But, recently, it is being recognized that knowledge about design objects is very important to realize practical expert systems [Ohsuga85].

The knowledge about design objects should be represented as a design object model in a design system. But object models used in conventional design systems are mere static data structures that represent attributes of design objects. They need to be interpreted and manipulated in terms of design tasks or procedures and the knowledge about design objects may be embedded in model manipulation procedures or design methods. In those design systems, it is difficult to make effective use of the knowledge about design objects.

To solve design problems effectively, it is important to represent all knowledge about design objects as an object model and to put it to practical use in the design process. Furthermore, we propose to give an active role to an object model; the object model has a function to change its state toward the state that satisfies given requirements.

We present a knowledge representation system based on an object-oriented and constraint-based paradigm for design object modeling. The system, called FREEDOM (a Framework for REpresenting and Elaborating Design Object Models), provides a framework for representing design objects in declarative form and facilities based on constraint satisfaction for problem solving.

In this system, a design object model is represented as a set of objects, and design requirements and most of the heuristics used in design can be represented as constraints. The object has a function to maintain its state to satisfy constraints when its structure or values of its attributes are modified or when a constraint is added to it. So the object model which represents a design solution can be built only by the addition of constraints as design requirements and heuristics.

The rest of this paper is organized as follows. First, we have a general discussion on knowledge representation frameworks for representing design object models in Section 2. Then knowledge representation of FREEDOM which integrates object-oriented and constraint representation is presented in Section 3. Problem solving of FREEDOM based on constraint satisfaction is discussed in Section 4. Implementation issues are described in Section 5. Finally, Section 6 briefly reviews related works and compares FREEDOM with them.

2 Design Object Model

A design object model represents information and knowledge about design objects, such as their attributes, shapes, structures, and so on. The objective of design is to build an object model that satisfies design requirements; it represents a solution. During the design process, a model whose properties satisfy given requirements is constructed.

A framework must be developed that represents knowledge about design objects and provides design process support facilities. The system should provide functions to calculate values of attributes and functions to build a design object model that satisfies design requirements: determining the kinds of parts, building the structure of a model, refining an object model, and so on.

A frame system [Minsky75] has been used to represent structures and attributes of objects in knowledge systems. Using a frame system, we can represent each element of an object in understandable modular form.

Recently, an object-oriented paradigm [Goldberg83] [Stefik86] whose concept is similar to a frame system has been generally used and also applied to design problems. An object-oriented language is suitable for representing structures, attributes and behaviors of objects. But it is difficult to describe relations between objects or between attributes.

These relations can be represented as constraints in declarative form. Design requirements can be regarded as constraints on design objects. But conventional object-oriented languages do not provide facilities for dealing with constraints on objects.

Methods by which to represent knowledge about design objects that introduce constraints have been investigated [Stallman77] [Sussman80] [Heintze87]. These provide efficient form of knowledge representation in terms of declarative description, but they are not suitable for the representation of large-scale, complicated objects because they lack structural representation.

So investigations have been made into introducing constraints suitable for describing relations to an object-oriented paradigm that describes structures and attributes [Borning81] [Borning86] [Harris86] [Struss87]. These have made it possible to represent properties of design objects in understandable form. However, only constraints on numerical attributes (instance variables) can be represented in these systems, and these constraints may be used only to calculate the values of attributes.

We have developed FREEDOM system based on object-oriented and constraint-based paradigm. The objective is to realize a system which provides adequate functions to build an object model that satsifies requirements. Objects in this system play an active role to solve a design problem. The features of FREEDOM are described in the following sections.

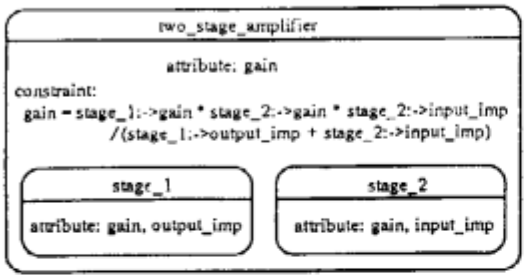

Figure 1: Example of constraints on objects

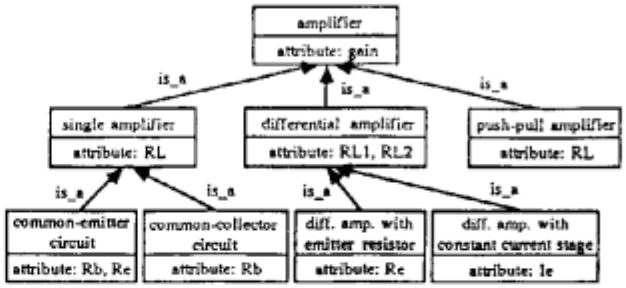

Figure 2: Example of class hierarchy

3 Knowledge Representation

3.1 Object and Constraint

Knowledge representation provided in the FREEDOM system is based on an object-oriented paradigm. Object-oriented representation is suitable for description of the structure of a design object.

Constraint satisfaction techniques play important roles in solving design problems. They reduce a combinatorial explosion, and values of attributes and structures of objects can be determined using them. Thus it is effective to introduce the concept of constraint to an object-oriented paradigm. It is possible to describe constraints in declarative form in FREEDOM.

Constraints on attributes are described in the form of a predicate or an equation. Figure 1 shows an example of constraint declaration. The constraint on the attribute of object *stage_1* and the attribute of object *stage_2* are declared in object *amplifier* which consists of *stage_1* and *stage_2*. Here, "stage_1 :-> gain" means *gain* of *stage_1*.

The part-whole relation, so-called *part-of*, is an important way to represent the structure of objects. The relation is classified into two: the first is that parts are needed to construct the whole; the second is that parts are not needed to construct the whole. For example, the relation of a rectangle and its four sides corresponds to the former case and the relation of a bookcase and books in it corresponds to the latter case. The former can be regarded as a structural constraint of an object and is called a *consists-of* relation in FREEDOM.

To specify the type of an element of a design object, the class name of an object corresponding to the element can be declared as a type constraint. The object must belong to the declared class or its subclass.

Because constraints may be generated dynamically during a design process, functions for addition or deletion of a constraint on an object are provided.

3.2 Class-Instance Relation

In a design process, parts that satisfy the design requirements must be searched and the values of their attributes must be determined. Sometimes, after the values of attributes of an instance that corresponds to a selected class have already been determined, the designer may want to perform an operation that changes the class to another class to which the instance belongs.

For example, Figure 2 shows a class hierarchy which represents a taxonomy of amplifier circuits. At the first stage of the top-down design of a circuit, attributes common to amplifier circuits such as gain must be determined before the kind of circuit is selected. So an instance of class *amplifier* is created and values of attributes are determined. Then the kind of circuit that satisfies the design requirements, for example *differential amplifier*, is selected.

In this case, in existing object-oriented languages, we must remove an instance that belongs to old class (*amplifier* in this example), create an instance that belongs to new class (*differential amplifier*), and copy the values of attributes common to these two classes (*gain*).

FREEDOM handles the relation between class and instance, the *instance-of* relation, which can be changed dynamically. This makes it possible to design effectively, because it is unnecessary to create a new instance or to copy the values of attributes. A dynamic change of a class is limited according to a hierarchy of *is-a* relations, and limited to the specified class and its subclasses to satisfy type constraints.

3.3 Class Hierarchy

A class hierarchy with multiple inheritance is not always represented using a relation between an abstract level and a concrete level, but using inclusion of a function declared in another class. It makes a class hierarchy too complex to understand.

In FREEDOM, *is_a* relations can be declared only when two classes belong to the same category, and they must be represented in the form of simple tree structures. A

```
fr_class  class name  has
     is_a  superclass name ;
     includes
          [ key name :: ] included class name ,
                        . . .                    ;
     consists_of
          element name [ ::= default class name  ],
                        . . .                     ;
     attributes
          attribute name [ ::= default value  ],
                        . . .                    ;
     constraints
          constraint ,
                      . . .                  ;
     meth(Obj,  message ) :-  body of method ;
                        . . .                    ;
fr_end.
```

Figure 3: Syntax of class definition

refinement in a design process corresponds to the search operation of a class that satisfies design requirements by referring to the *is_a* tree structure.

The multiple inheritance is useful for representing inclusion of functions, so a definition of the *includes* relation is introduced for representation of the class hierarchy.

3.4 Syntax

Figure 3 shows the syntax of class definition in FREEDOM. Only items needed for representing an object are to be declared, fragments in square brackets may be omitted. A key used to specify the view of an object can be added to an included class. A class name specified in an element definition is regarded as a type constraint of the element object. Only attributes of the object and attributes of its elements can be referred in constraint declarations.

In FREEDOM, methods are used to provide interfaces rather than to describe the behavior of objects. Properties and behavior should be represented using constraints.

System methods are provided and are used to create an instance, to change the class of an instance, to refine an object, to assign the value of an attribute, to add a constraint to an object, to delete a constraint of an object, and so on. An object model can be manipulated with these methods. A method is activated by message passing whose format is " :send(*receiver*, *message*) ".

Figure 4 shows an example of the class definition which represents an electronic circuit of an amplifier. Here, the notation x:->y means attribute y of element x.

A source program written according to the syntax is to be translated to the internal representation in ESP to be executed.

```
%%%  Voltage Amplifier  %%%

fr_class  voltage_amplifier  has
    attributes
        gain,  input_imp,  output_imp,
        max_output,  vcc,  vee,  i,  power;
    constraints
        power = (vcc + vee) * i;
fr_end.

%%%  Two Stage Differerential Amplifier  %%%

fr_class  two_stage_differential_amplifier  has
    is_a
        voltage_amplifier;
    consists_of
        stage_1 ::= differential_amplifier_unit,
        stage_2 ::= differential_amplifier_unit;
    attributes
        beta,  k,  open_gain,  amp_input_imp,
        r_nfb_1,  r_nfb_2,  r_input;
    constraints
        2 * open_gain = (stage_1 :-> output_imp
          + stage_2 :-> input_imp)
                * stage_1 :-> gain * stage_2 :-> gain,
        beta * (r_nfb_1 + r_nfb_2) = r_nfb_2,
        k * (1 + open_gain * beta) = 1,
        gain = k * open_gain,
        k * amp_input_imp = stage_1 :-> input_imp,
        input_imp * (amp_input_imp + r_input)
                                = amp_input_imp * r_input,
        output_imp = k * stage_2 :-> output_imp,
        max_output = stage_2 :-> max_output,
                . . . . . ;
fr_end.
```

Figure 4: Example of class definition

4 Constraint Satisfaction

4.1 Calculation of Value of Attribute

An object in FREEDOM has a function to maintain the values of its attributes that satisfy constraints declared statically or added dynamically. The function makes it possible to solve design problems effectively. The values can be obtained by solving constraints.

For example, suppose that there is an object representing an amplifier which consists of two amplifier units *stage_1* and *stage_2* and there are two constraints on attributes of them shown below.

$$total\ gain = stage\_1{:}{\to}gain * stage\_2{:}{\to}gain * \frac{stage\_2{:}{\to}input\_imp}{stage\_1{:}{\to}output\_imp + stage\_2{:}{\to}input\_imp}$$

$$stage\_1{:}{\to}gain = 2 * stage\_2{:}{\to}gain$$

If we have a heuristics that the value of *output_imp* of *stage_1* is much smaller than the value of *input_imp* of *stage_2* and the former is negligible, we can add the constraint below to the object.

$$stage\_1{:}{\to}output\_imp = 0$$

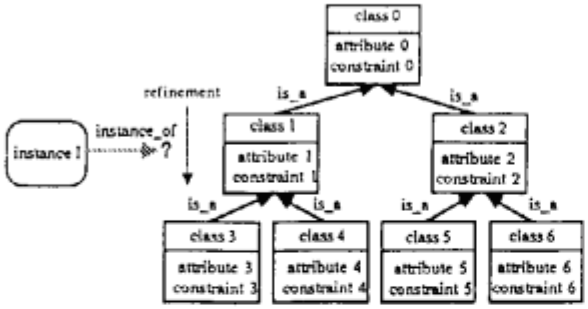

Figure 5: Class search based on constraint satisfaction

Now, if the value of attribute *total_gain* is given as a design requirement, attributes *gain* of *stage_1* and *gain* of *stage_2* are determined by constraint satisfaction.

As shown above, constraint satisfaction is a useful method in parametric design problems whose goal is to get values of parameters that satisfy design requirements.

4.2 Class Search

In general design problems, not only parameters but also the kinds of parts and the structures of objects must be determined. To find which kind of a part satisfies requirements, we must search a class hierarchy.

Search for a class that satisfies design requirements is realized using a constraint satisfaction mechanism in FREEDOM. When a structure or an attribute of an instance is modified, if constraint satisfaction fails in the class to which it belongs, the class may be changed automatically to another class to satisfy the constraints. An object in FREEDOM has a function to change its class for constraint satisfaction.

This function realize class search without describing a procedure. We call it *constraint-based taxonomic reasoning*. Figure 5 represents a class hierarchy with *is_a* relations. We assume an instance *I* which had belonged to *Class 0* was further specified and now it belongs to *Class 1*. The instance satisfies *Constraint 0* and *Constraint 1* in this state. Then, to refine it, or make it concrete, we send it the message to activate a procedure that tries to change its class to the lower class.

First, the class search mechanism tries to change the class of *I* to *Class 3* and solve the set of *Constraint 0*, *Constraint 1* and *Constraint 3*. If it succeeds, this procedure is terminated and the class of *I* is *Class 3*. If it fails, the procedure backtracks and tries to change the class of *I* to *Class 4* and solve the set of *Constraint 0*, *Constraint 1* and *Constraint 4*. If it succeeds, the class of *I* is determined *Class 4*. If it fails, the procedure fails and the class of *I* is not changed.

For example, a problem to determine the kind of an electronic circuit is considered. Figure 6 represents a

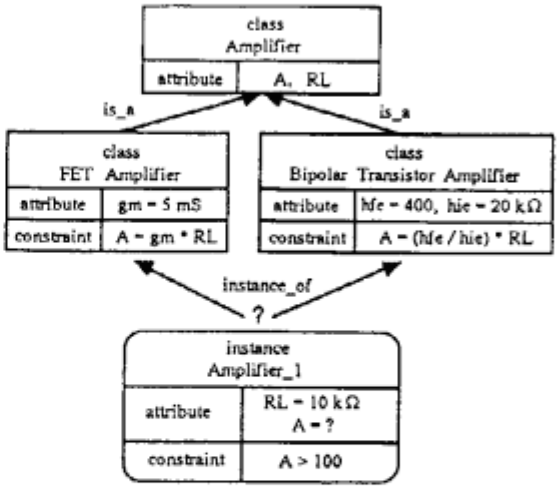

Figure 6: Example of class search

class hierarchy of simple electronic circuits: amplifiers. The problem is to determine the class of instance *Amplifier_1* that satisfies given requirements. At first, the class of *Amplifier_1* is *Amplifier*, and its class must be determined as either *FET Amplifier* or *Bipolar Transister Amplifier*. *Amplifier_1* has attribute gain *A* and load resistance *RL*, and it must satisfy the constraint on *A*. While the value of *RL* is not assigned its class cannot be determined uniquely. When the value of *RL* is assigned to *10k Ω* as shown in Figure 6, the class is determined *Bipolar Transister Amplifier*, not *FET Amplifier*, to satisfy constraint *A >100*.

This mechanism is useful for selecting a model of the device which depends on the condition in which the device is used. If the condition is declared as a constraint, a suitable model is selected automatically according to the condition by constraint satisfaction.

4.3 Constraint satisfaction technique

The constraint satifaction technique used in FREEDOM is based on constraint logic programming [Jaffer87] and constraint propagation. Each object has a constraint solver and constraints declared in the object are evaluated by the constraint solver in the way of constraint logic programming. The constraint solver can deal with constraints added incrementally.

An attempt is made to solve constraints among objects by constraint propagation. But when there is a cyclic dependency among constraints, they cannot be solved by local propagation that propagates values of attributes. So, in FREEDOM, constraints such as equations are propagated between objects.

Figure 7 shows an example of constraint propagation. Object *o* consists of object *a* and object *b*, and those ob-

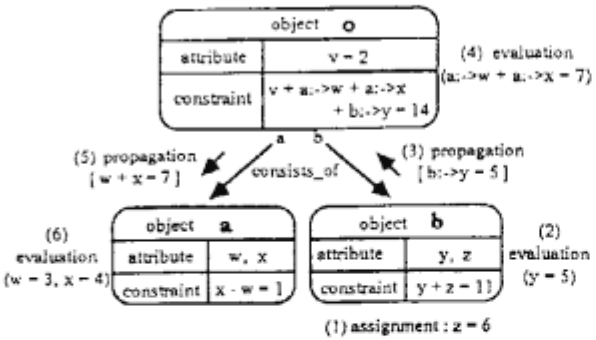

Figure 7: Constraint propagation

jects have the attributes and constraints shown. There is a cyclic dependency between the constraint in object o and the constraint in object a.

Now, we assume that the value of attribute z of element b is assigned 6 (1), then constraint satisfaction is executed in object b (2), the value of attribute y is determined and propagated to object o (3), constraint satisfaction is executed in object o (4), the constraint is propagated to object a because the constraint is on only the attributes of element a (5), and attributes w and x are determined by solving the constraints (6).

5 Implemantation

FREEDOM has been implemented using ESP language [Chikayama84] on a PSI machine [Taki84]. ESP is an object-oriented programming language based on logic programming and PSI is a sequential inference machine for logic programming. The class definition written according to the syntax of FREEDOM is translated to the ESP program.

An instance of FREEDOM is implemented as a set of instances of ESP, each of which corresponds its inherited classes to realize efficient class-change. The class of an instance of FREEDOM is changed dynamically by adding or deleting an instance of ESP corresponding to a difference between the class before it was changed and the class after it was changed.

Constraint satisfaction of FREEDOM is based on constraint logic programming and constraint propagation. The constraint solver of constraint logic programming language CAL [Aiba88] is used for constraint satisfaction in each object. CAL uses a Buchberger algorithm for solving simultaneous equations. Constraint propagation is implemented as message passing.

Only numerical constraints such as equations which are adequate for simple parametric design are dealt with in FREEDOM. But, of course, the domain of constraints dealt with in FREEDOM should be expanded to solve complicated design problems.

6 Related Work

Several systems or programming languages with object-oriented and constraint paradigm are presented. The first system of them is ThingLab, a constraint-based simulation laboratory developed by Borning [Borning81]. ThingLab is suitable for developing graphical user interfaces [Borning86]. SOCLE is a hybrid structured object and constraint representation language for knowledge representation [Harris86]. Other systems or languages of them have similar facilities using constraints [Shepherd86] [Struss87].

In these systems or languages, constraints are used to only calculate values of attributes. The constraint satisfaction mechanism makes it possible to calculate values of attributes automatically without describing procedure. This function is useful for parametric design. But in general design problems, not only parameters (values of attributes) but also the structure of a design object and the kinds of parts must be determined. So we must describe a procedure to search a class satisfying given requirements in these systems.

In FREEDOM, not only values of attributes but also a class that satisfies requirements can be determined by constraint satisfaction. As mentioned in this paper, this function is useful for solving design problems.

Constraint satisfaction mechanisms used in most conventional systems above are based on the local propagation and relaxation method. This method is unefficient when there is a cyclic dependency. Some systems use a global method that evaluates all constraints simultaneously. But in this case, as the number of constraints increases, computational complexity increases exponentially. A large-scale problem must be divided into subproblems to reduce computational complexity.

As shown in Section 4.3, the constraint satisfaction mechanism in FREEDOM divides a problem into subproblems naturally according to part-whole relations of objects. The constraint satisfaction method which propagates not values of attributes but constraints such as equations is efficient because it doesn't need relaxation.

7 Conclusion

A knowledge representation system for design object modeling is presented in this paper. This system, called FREEDOM, not only represents knowledge about design objects but also supports design tasks. The object model is not a mere design structure but an active entity for problem solving.

The knowledge representation framework of FREEDOM is based on object-oriented and constraint-based paradigm. It has features suitable for representing design object: constraints are represented in declarative form, a class-instance relation can be changed dynamically, and a class hierarchy is represented using *is_a* and *includes* relations.

An object in FREEDOM is kept in the state that constraints are satisfied; not only the values of its attributes but also its class are determined by constraint satisfaction. This function is useful to find values of parameters, types of parts, and the structure of design object that satisfies design requirements. The constraint satisfaction mechanism is based on constraint logic programming and constraint propagation technique.

According to the experience of applying FREEDOM to simple electronic circuit designs, we think FREEDOM is a powerful design support tool. But FREEDOM must be improved if it is to deal with more complicated problems. Though the domain of constraints is limited to numerical constraint in the system, the domain is being expanded to symbolic constraints and user-defined constraints.

Acknowledgement

I would like to thank Mr. H eto Sazuka at NTT Data Communications Systems C p. for his assistance in implementation of FREEDOM, and Dr. Koichi Furukawa, the deputy director of ICOT Research Center, and all members of the Fifth Research Laboratory at ICOT for their helpfull comments. I would also like to thank Dr. Kazuhiro Fuchi, the director of ICOT Research Center, and Mr. Kenji Ikoma, the chief of the Fifth Research Laboratory, for their support and encouragement.

References

[Aiba88] Aiba, A. et al. *Constraint Logic Programming Language CAL*, Proc. of International Conference of Fifth Generation Computer Systems 1988, pp263–276, (1988)

[Borning81] Borning, A. *The Programming Language aspects of ThingLab, a Constraint-Oriented Simulation Laboratory*, ACM Transactions on Program Languages and Systems vol.3, no.4, pp353–387, (1981)

[Borning86] Borning, A. and Duisberg, R. *Constraint-Based Tools for Building User Interfaces*, ACM Transactions on Graphics, vol.5, no.4, pp345–374, (1986)

[Chikayama84] Chikayama, T. *Unique Features of ESP*, Proc. of International Conference on Fifth Generation Computer Systems 1984, pp292–298, (1984)

[Goldberg83] Goldberg, A. and Robson, D. *Smalltalk-80: The Language and Its Implementation*, Addison-Wesley, (1983)

[Harris86] Harris, D. R. *A Hybrid Structured Object and Constraint Representation Language*, Proc. of AAAI-86, pp986–990, (1986)

[Heintze87] Heintze, N. et al. *CLP(R) and Some Engineering Problems*, in Lassez, J. L. (ed.) *Logic Programming, Proc. of 4th International Conference*, MIT Press, pp675–703, (1987)

[Jaffer87] Jaffer, J. and Lassez, J.-L. *Constraint Logic Programming*, Proc. 14th ACM Principles of Programming Languages conference, (1987)

[Minsky75] Minsky, M. *A Framework for Representing Knowledge*, in Winston, P. H. (ed.) *The Psycology of Computer Vision*, McGraw-Hill, (1975)

[Ohsuga85] Ohsuga, S. *Conceptual Design of CAD Systems Involving Knowledge Base*, in Gero, J. (ed.) *Knowledge Engineering in Computer Aided Design*, pp29–88, North-Holland, (1985)

[Shepherd86] Shepherd, A. and Kershberg, L. *Constraint Management in Expert Database Systems*, in Kershberg, L. (ed.) *Expert Database Systems*, Benjamin/Cummings, pp309–331, (1986)

[Stallman77] Stallman, R. M. and Sussman, G. J. *Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis*, Artificial Intelligence, vol.9, pp135–196, (1977)

[Stefik86] Stefik, M. and Bobrow, D. G. *Object-Oriented Programming: Themes and Variations*, AI Magazine, vol.6, no.4, pp40–62, (1986)

[Struss87] Struss, P. *Multiple Representation of Structure and Function*, in Gero, J. (ed.) *Expert Systems in Computer-Aided Design*, North-Holland, (1987)

[Sussman80] Sussman, G. J. and Steel G. L. Jr. *Constraints - A Language for Expressing Almost-Hierarchical Descriptions*, Artificial Intelligence, vol.14, pp1–39, (1980)

[Taki84] Taki,K. et al. *Hardware Design and Implementation of the Personal Sequential Inference Machine (PSI)*, Proc. of International Conference on Fifth Generation Computer Systems 1984, pp398–409, (1984)