ICOT Technical Memorandom: TM-1255

TM-1255

Multiple-Context Objects for a Parallel Cooperative Problem-Solving System

by
T. Yokoyama (Hitachi)

April, 1993

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191 ~ 5

Institute for New Generation Computer Technology

Multiple-Context Objects for a Parallel Cooperative Problem-Solving System

Takanori Yokoyama
Hitachi Research Laboratory, Hitachi, Ltd.

Contact information

Takanori Yokoyama
First System Dept., Hitachi Research Laboratory, Hitachi, Ltd.
1-1, Omika-cho 7-chome, Hitachi-shi, Ibaraki-ken, 319-12 Japan
Tel : 81-294-52-5111 ext. 3853
E-mail : yokoyama@hrl.hitachi.co.jp

Paper category

Research paper

Abstract

This paper presents multiple-context objects for a parallel cooperative problem solving system which consists of an object model and knowledge sources. The object model is a set of active objects which have functions for constraint satisfaction and multiple-context processing. A knowledge source has rules to solve a subproblem, and the rules access objects. Constraint propagation via objects supports cooperation between knowledge sources. An object contains a node network for multiple context management. The node network deals with combinations of multiple values of slots of an object. A prototype of a cooperative design system has been developed on a multiprocessor machine Multi-PSI using the concurrent logic programming language KL1.

1. Introduction

To solve a large-scale or complex problem, the problem is divided into subproblems. Partial solutions are got by solving the subproblems and then by integrating the partial solutions, a solution of the whole problem is got. For example, in functional design of a microprocessor, the problem is divided into subproblems, such as behavioral design, structural design, timing design, and so on. But, because the structure of the microprocessor is undefined and all subproblems share design data, designers must cooperate with each other to get a consistent solution in the actual design. This type of problem requires a cooperative problem solving technique.

A blackboard model is a cooperative problem solving model and cooperative design systems based on it have been developed [Rehak 85] [Bushnell 87] [Temme 88]. A blackboard system consists of a blackboard and several

knowledge sources. Each knowledge source has knowledge to solve a subproblem. The knowledge is represented as a set of rules in most of the systems. A knowledge source reads data from the blackboard and writes data to the blackboard. The data on a blackboard are integrated into a design solution.

The goal of our research is to develop a parallel cooperative problem solving system for design problems in which there are heavy dependencies between the subproblems. The architecture of existing blackboard systems is unsuitable for parallel cooperative problem solving. Though several parallel blackboard systems have been developed, the efficiency is not so good.

Knowledge sources can run in parallel in principle. But, there are problems to realize a parallel blackboard system. Most of the systems are built on sequential machines and cannot be executed in parallel. One of the major problems is scheduling. A scheduler of the system controls execution of knowledge sources and supports cooperation among them. To parallelize a blackboard system, the scheduler must schedule parallel execution of knowledge source. But, it is difficult to realize efficient parallel scheduling. Another major problem is implementation of a blackboard. As the blackboard is shared by knowledge sources, access to it may be a system bottleneck. Furthermore, it is difficult to implement the blackboard on a multiprocessor machine without a shared memory.

Cooperation among knowledge sources is important because of dependencies between subproblems. It is difficult to get a consistent solution by simply merging partial solutions after solving the subproblems. If a consistent solution is not obtained, knowledge sources must backtrack and generate other partial solutions. A parallel cooperative system needs a blackboard which can contribute greatly to cooperation among knowledge sources. Many candidates for partial solutions are generated and a combinatorial explosion may be caused in design problems. A mechanism is needed to prune the search tree. But a blackboard lacks this because it is simply a shared memory. An active blackboard is needed for parallel cooperative problem solving.

We present a multiple-context object for a parallel cooperative problem solving system in this paper. This system has an object model which consists of multiple-context objects. Multiple-context objects have facilities for constraint satisfaction and multiple-context processing. First, we discuss the parallel cooperative problem-solving model in section 2. Then, we present a multiple-context object with a constraint satisfaction facility and show cooperation among knowledge sources via multiple-context objects in sections 3 and 4 . Our prototype of a design system using multiple-context objects is described next in section 5 and section 6 discusses related work.

2. A Parallel Cooperative Problem Solving Model

2.1 A Problem Solving Model

We propose a parallel cooperative problem solving model composed of the object model and knowledge sources. Figure 1 shows the structure of the model. The object model consists of objects. In a design system, an

object model is a design object model which represents a design solution. We classify design knowledge into knowledge about design objects and knowledge about design methods [Nagai 88]. Knowledge about design objects can be represented as a design object model [Ohsuga 85]. We represent the knowledge about design objects using classes of objects. Class definitions represent knowledge about design objects, such as basic structure of design objects, available parts, constraints between parts, and so on. We can define constraints on objects in the declarative form [Yokoyama 90].

We also represent a design solution as a set of objects (instances) which satisfies design requirements. An object of the design object model provides facilities for cooperation among knowledge sources. Cooperation is based on consistency maintenance and constraint propagation by the multiple-context objects. Knowledge about design methods is divided and stored in knowledge sources. Each knowledge source has different design knowledge, including heuristics, for each subproblem which is used to solve the subproblem. All knowledge sources share a design object model and each can access part of the object model; that is, a knowledge source reads values of slots (instance variables) of objects of the object model, generates partial solutions, and puts them as new values of other object slots.

2.2 Approach

Objects can run in parallel in our system. Furthermore, to exploit parallelism, an object is implemented as a set of processes which can run in parallel. A knowledge source is a set of fine-grained processes. We can distribute processes of both objects and knowledge sources to processor elements of a multiprocessor machine.

The system should deal with a lot of data for efficiency. Ours can deal with plural candidates for a partial solution. Those candidates are generated and processed using pipeline processing. The interface between a knowledge source and objects is based on data-flow architecture using streams. We can get the effect of pipeline parallelism by processing data in a succession by processes connected by streams.

We have developed a prototype of the system using KL1 language on a Multi-PSI machine [Uchida 88]. KL1 is a concurrent logic programming language based on GHC language [Ueda 85]. In KL1, a stream is a chain represented as a list. Multi-PSI is a multiprocessor machine with distributed memories. KL1 is suitable for process-based concurrent programming. Further, fine-grained processes are realized with little overhead.

3. Multiple-Context Object

3.1 Object Definition

An object model is a set of objects. We represent knowledge about design objects as class definitions. Objects (instances) represent a design solution or an intermediate solution. And they store design data in their slots. Slot names and constraints on the slots are declared in a class definition. We can represent a constraint on

slots as a predicate, an equation, or an inequality. A knowledge source accesses slots of objects. But, in general, different knowledge sources access different objects or different slots according to their viewpoints. Figure 2 shows a class definition for a microprocessor. An object of the class "microprocessor" has slots, such as "alu", "internal_bus", behavior", and so on. The class definition describes the constraint of prohibition on a combination using a nogood predicate which shows that a combination of values of slot "internal_bus" and slot "control" must not be "1-Bus" and "Instruction-Prefetch". This means that the constraint prohibits combination of the one-bus structure as an internal bus structure and instruction prefetch as a control method. Another constraint on slot "performance", slot "average_number_of_steps" and slot "machine_cycle" is represented as an equation.

The translator of the system translates class definitions to KL1 programs. Each object is implemented as a set of processes. An object interprets messages and executes operations for the messages concurrently. Knowledge sources access an object model in parallel except for access to the same slot of the same object.

3.2 Constraint Satisfaction

An object in our system has a constraint satisfaction mechanism and keeps its state by satisfying given constraints. When a knowledge source sets a value in a slot of an object, the object evaluates any constraint on the slot. There are two types of constraint evaluation. One is to evaluate a constraint passively and the other, actively. Passive constraint evaluation tests whether a constraint is satisfied or not when an object gets all values of slots of the constraint. For example, when an object gets the values of x, y and z of constraint $x > y + z$, the passive constraint evaluation tests whether the set of values satisfies the constraint or not. Passive constraint evaluation is used to get a consistent solution. Maintaining consistency of intermediate solutions is important in cooperative problem solving and parallel processing. Consistency maintenance evaluates constraints passively and keeps the state of an object satisfying constraints. When knowledge sources set values to slots of an object, the object evaluates constraints on the slots and rejects combinations of values that violate the constraints.

Active constraint evaluation calculates a value of a slot using a constraint when an object gets the values of other slots. For example, when an object gets the values of x and y of constraint $x = y + z$, active constraint evaluation calculates the value of z. Active constraint evaluation is applied to the type of constraint that can determine a unique value of a slot. Active constraint evaluation realizes constraint propagation among knowledge sources via an object model without direct communication. A knowledge source influences other knowledge sources indirectly by constraint propagation and this effect is used for cooperation among knowledge sources. Figure 3 shows an example of constraint propagation. The value of slot c is calculated by evaluating the constraints with the value of slot a, and the value is propagated to other knowledge sources. Though this is an example using a constraint on slots of one object, we can use constraints between plural objects for propagation between objects.

Constraint propagation realizes cooperation among knowledge sources with no shared data. Execution

sequence does not affect cooperation, because the direction of propagation is determined dynamically during the run time.

3.3 Multiple-Context Processing

The system should simultaneously deal with plural candidates for a partial solution to exploit parallelism and get high efficiency. A multiple context object in our system has a mechanism to deal with multiple values of a slot. Context means the state of an object and the state is represented as a combination of values of slots.

Conventional systems represent multiple contexts as a context tree [Walters 88]. Figure 4 shows an example context tree. The tree structure grows as new values are set to slots. But, consistent contexts do not increase monotonously, because inconsistent contexts must be rejected. Inconsistent branches of a context tree should be pruned as early as possible for efficiency. Figure 4 also shows that plural contexts may become the same state, because execution sequence is nondeterministic and execution of knowledge sources is parallel and asynchronous. Those contexts should be unified immediately to remove redundant processing. Multiple context management facilities are needed for pruning and unification.

We present a multiple-context object which has a node network for multiple-context processing. A node of the network manages a combination of slots. Figure 5 shows an example node network. The node network forms a lattice. The head of the node network is the body of an object. Each slot of the body points to a node which deals with values for the slot. A following node deal with a superset of the set of slots dealt by the precedent nodes. The tail is a node for all the slots of an object.

A node stores constraints on slots dealt with by the node. It receives data from the precedent nodes and combines the data. It has functions for constraint evaluation. The node stores only the combinations of values that satisfy the constraints and rejects combinations that violate them. Figure 5 shows that the tail node stores the combinations of values of all the slots that satisfy all the constraints.

If all nodes for combinations of slots were generated, many nodes would be generated for an object. To reduce the nodes, the translator provides a facility to generate only the nodes for essential combinations. Two kinds of combinations are essential: the combinations referred to by constraints and the combinations referred to by knowledge sources. The translator analyze class definitions and knowledge source definitions and optimizes node networks.

When an object is created, its node network is generated. Each node is a process. An object is a set of processes that run in parallel. Node processes are connected with streams. The translator attaches demons to the node processes. The demons send referred combination data to knowledge sources.

4. Cooperation via Multiple-Context Objects

4.1 Knowledge Source Definition

A knowledge source is a forward chaining production system which we represent as a set of rules. Figure 6 shows an example of knowledge source definition. We define rules in the if-then form. A rule is fired when its condition part matches the data of objects of the object model. The rule in the example shows that if the class of an object is "microprocessor" and the value of its slot "behavior" is B, the value D is generated using a function called "generate_data_path" with value B and the value D is set to slot "data_path" of the object.

All matched rules are fired simultaneously without conflict resolution to exploit parallelism. If different rules are fired for the same slots of an object, different values are set to the same slots and the object has multiple contexts. A multiple-context object maintains consistency of its contexts.

A knowledge source executes inference according to changes of values of object slots. A knowledge source receives changed data from objects, generates intermediate data, and sets the intermediate data to the objects. Here, intermediate data mean a partial solution generated in the design process. Intermediate data are set to slots of objects. An object combines intermediate data. Combined data of an object mean contexts of the object. The constraint evaluation mechanism of an object tests if its contexts satisfy constraints and prunes inconsistent contexts.

The translator translates rules of knowledge sources to KL1 programs. When a rule sets data to a slot of an object, the rule must specify the context. The translator provides a facility for a rule to specify the context. For example, consider the rule shown below.

```
if
  class(Obj, microprocessor),
  slot(Obj, {slot_1, X}), X = 1
then
  set_slot(Obj, {slot_2, 2});
```

This rule means that if the value of slot "slot_1" of an object belonging to class "microprocessor" is at 1, value 2 is set to the slot "slot_2" of the object. The execution part of this rule is translated to the form shown below.

```
set_slot(Obj, [{slot_1, 1},
      {slot_2, 2}]);
```

Both the value of "slot_2" and the value of "slot_1" are set simultaneously. This translation is needed for consistency maintenance.

One rule is translated to one clause of KL1 and is executed as a process. We can distribute rules of a knowledge source to plural processors. Rules that match data of objects can be fired and executed in parallel.

4.2 Cooperative Behavior

The parallel cooperative problem solving system in this paper is based on the data flow architecture.

Knowledge sources and objects are connected with streams. Constraint propagation and consistency maintenance by multiple-context objects support cooperation between knowledge sources.

Figure 7 shows a simple example of the system which is composed of four knowledge sources and a design object model. The node network of the object of the design object model is optimized. We simplify the representation form of rules in the figure. First, when a design requirement is given, rules in knowledge sources 1 and 2 are fired in parallel without conflict resolution, and the values 1 and 2 are set to slot a and the values 3 and 4 are set to slot b. Those data are sent to node a and node b, respectively, stored there, and propagated to the node (a, b). Here, node a means the node which deals with the values of slot a, and node (a, b) means the node which deals with combinations of values of slot a and slot b. The node generates combination data that satisfy the constraints given to the node, stores them, and propagates them to the node (a, b, c). The node evaluates the constraints on slot a, slot b and slot c actively and gets the values of slot c. Then, the node stores combination data, and sends values of slot c to node c.

Then the rules of knowledge source 3 which refers to slot c are fired, and the combination of values of slot c and slot d are set to node (c, d) for consistency maintenance as mentioned in the previous subsection. Only the combinations that satisfy the constraint on slot c and slot d are stored. Both the combination data of node (a, b, c) and node (c, d) are propagated to node (a, b, c, d), and the node generates combinations of values of slot a, slot b, slot c and slot d. Note that the node generates only the consistent combinations. When plural precedent nodes deal with the same slot as an element of slot combinations, only the consistent combinations are generated. The value of the slot of one precedent node and the value of the slot of another precedent node must be equal in the consistent combination. Node (a, b, c d) generates only the combinations (1, 4, 5, 6) and (2, 3, 5, 6) as consistent combination data. Finally, knowledge source 4 outputs the data as solutions.

5. A Prototype Parallel Cooperative Design System

5.1 System Structure

We developed a prototype system for functional design of a microprocessor. We simplified the design process to allow the system to be built with a few rules and objects. The goal of prototype development is to evaluate the effect of multiple-context objects, not to build a design system. The system inputs design requirements, such as instruction set and performance requirement and outputs design solutions, such as behavior description and data path at the register transfer level. We analyzed the functional design of the microprocessor and divided the problem into five subproblems: architectural planning, behavior design, block design, data path design, and timing design.

The prototype system consists of a design object model and six knowledge sources: five knowledge sources for the subproblems and a knowledge source to output the design solutions. Class definitions for a design object model are represented in the form shown in Figure 2 and rule definitions of each knowledge source are represented

in the form shown in Figure 6. The number of rules is about 30 and most of the rules are for generating candidates for intermediate solutions.

The system has been implemented on Multi-PSI. We use static load balancing to allocate all the tasks before starting the job. We distribute knowledge sources and nodes of objects to processor elements of the multiprocessor machine.

5.2 Results

There are many candidates for intermediate solutions in this problem. Though there are only a few solutions that satisfy all requirements, if all the candidates were combined without consistency maintenance, about a thousand candidate solutions would be generated. Our system executes design cooperatively, pruning the candidates that violate constraints. Most nodes deal with less than ten combinations of candidates because the nodes generate only consistent combinations. Therefore the parallel cooperative problem solving system presented in this paper is effective for problems in which there are many candidates for solutions, though only a few of them are consistent.

We got speedups of 2 times with 4 processors and of 2.7 times with 8 processors. These were not as good as we expected. The number of active processors of the system decreases as processing advances. All processors are active early, but only one or two processors remain active in the last stage of processing. We need to apply the system to a large-scale problem and use efficient load balancing to get better speedup against the number of processors. We think that a dynamic load balancing strategy is needed.

6. Related Work

There are many concurrent object-oriented languages and systems, but few systems have a facility for multiple-context processing. There are also constraint-based object-oriented languages, but an object of the systems has only one context. We think that a multiple-context object is effective for parallel systems and it is not restricted to just the parallel cooperative problem-solving system presented in this paper.

Research on parallel systems for AI has been done in various domains. But developing an efficient parallel or distributed cooperative problem-solving system is difficult because of dependency among subproblems and nondeterminism of reasoning. A number of parallel or distributed blackboard systems have been developed [Engelmore 88] [Jagannathan 89]. But, it is not so easy to parallelize a blackboard system, particularly a system with a scheduler for control.

There are several approaches to exploit parallelism in a blackboard system: blackboard interaction operations, concurrent execution of knowledge sources, internal execution of a knowledge source, and so on [Corkill 89]. But getting high efficiency is difficult with only these approaches. It is important to exploit knowledge parallelism, pipeline parallelism, and data parallelism [Nii 88]. It has been reported that dealing with many data is effective for efficiency [Rice 89].

Poligon [Nii 88] [Rice 89] is an interesting approach to a parallel blackboard system. No scheduler exists in the system and rules can be invoked in parallel. Rules are associated directly with the nodes on the blackboard. The associated nodes and the rules are distributed in the multiprocessor. The association is executed at compile-time. Poligon has been developed with applications of real-time signal understanding and data fusion in mind. The Pligon approach is effective in these domains. But in design problems, our target domain, many candidates for partial solutions are generated and a combinatorial explosion may be caused. Our system provides facilities to reject candidates that violate constraints on a design object and to maintain consistency for pruning of the search tree and avoiding combinatorial explosion.

Our system provides facilities to manage consistent multiple contexts for efficient parallel cooperative problem solving. This is realized by a node network of a multiple context object. ATMS (Assumption-based Truth maintenance System) [de Kleer 86] provides a facility for maintenance of consistent multiple contexts. We can realize multiple-context management without redundancy using ATMS regarding values of slots as assumptions. But the number of nodes would be enormous if a value of a slot is regarded as an assumption. Though the form of the node network of our system is like the environment lattice of ATMS, the node of our system deals with a combination of slots, not a combination of values. Each node deals with plural combinations of values, namely, multiple contexts. Furthermore the node network is optimized by the translator. The number of nodes is minimal in our system for efficiency. Another problem to apply ATMS to node networks is that we must declare many nogoods to reject both combinations violating constraints and meaningless combinations (for example, the combination of slot_a = 1 and slot_a = 2 is a meaningless combination).

The node network of our system has a facility to get data that match condition parts of rules of knowledge sources. The node network has part of the functions of the RETE network [Forgy 82]. But the node of our system has facilities for constraint satisfaction. If our system is optimized, we may be able to get an efficient parallel production system.

7. Conclusion

We presents multiple-context objects for a parallel cooperative problem solving system. The system consists of an object model and knowledge sources which solve subproblems. The object model contains a multiple-context object which has functions for constraint satisfaction and consistency maintenance. Multiple context management is realized by a node network whose nodes deal with combinations of values of slots of objects. Cooperation between knowledge sources is supported by constraint propagation via multiple-context objects.

The system is based on data flow architecture. Nodes of node networks of objects and rules of knowledge sources are implemented as processes and they are connected by streams. We can distribute the processes to processor elements and execute them in parallel. The system is implemented using concurrent logic programming language KL1 on a multiprocessor machine Multi-PSI.

According to our experience in applying the system to a simple functional design of microprocessor, we think that the system is effective for problems, such as design in which there are many candidates for partial solutions. But we need to improve the system to get higher efficiency and an efficient load balancing strategy is also needed.

Acknowledgements

This research has been done in Institute for New Generation Computer Technology (ICOT). The author would like to thank Mr. Masayuki Ono at Oki Electric Industry Co., Ltd., Mr. Masahiro Wada at Sharp Corporation, and Hiroshi Ohsaki at Japan Information Processing Development Center for their contribution to building the prototype system. The author would also like to thank Dr. Katsumi Nitta at ICOT Research Center for his helpful comments, and Dr. Kazuhiro Fuchi, the director of ICOT Research Center, Dr. Koichi Furukawa, the deputy director of ICOT Research Center, Mr. Kenji Ikoma at NTT Data Communications System Corp. for their support and encouragement.

References

[Bushnell 87] Bushnell, M. L. and Director, S. W., "ULYSSES - a Knowledge-Based VLSI Design Environment", Artificial Intelligence in Engineering, vol.2, No.1, pp.33-41 (1987).

[Corkill 89] Corkill, D. D., "Design Alternatives for Parallel and Distributed Blackboard Systems", in [Jagannathan 89], pp.99-136 (1989).

[de Kleer 86] de Kleer, J., "An Assumption-based TMS", Artificial Intelligence, Vol.28, pp.127-162 (1986).

[Engelmore 88] Engelmore, R. and Morgan, T. (eds.), "Blackboard Systems", Addison Wesley (1988).

[Forgy 82] Forgy, C. L., "RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", Artificial Intelligence, vol.19, pp.17-38 (1982).

[Hayes-Roth 85] Hayes-Roth, B., "A Blackboard Architecture for Control", Artificial Intelligence, vol.26, pp.251-321 (1985).

[Jagannathan 89] Jagannathan, V. et al. (eds.), "Blackboard Architectures and Applications", Academic Press (1989).

[Nagai 88] Nagai Y. et al., "Expert System Architecture for Design Tasks", Proceedings of the Fifth Generation Computer Systems, pp.296-317 (1988).

[Nii 86] Nii, H. P., "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures", The AI Magazine, vol.7, no.2, pp.38-53 (1986).

[Nii 88] Nii, H. P. et al. , "Frameworks for Concurrent Problem Solving: A Report on CAGE and POLIGON", in [Engelmore 88], pp.475-501 (1988).

[Ohsuga 85] Ohsuga, S., "Conceptual Design of CAD Systems Involving Knowledge Base", in Gero, J. (ed.),

"Knowledge Engineering in Computer Aided Design", pp.29-88, North-Holland (1985).

[Rehak 85] Rehak, D. R. et al., "Architecture of an Integrated Knowledge Based Environment for Structural Engineering Applications", in Gero, J. (ed.), "Knowledge Engineering in Computer Aided Design", pp.89-117, North-Holland (1985).

[Rice 89] Rice, J. et al., "See How They Run... The Architecture and Performance of Two Concurrent Blackboard Systems", in [Jagannathan 89], pp.153-178 (1989).

[Temme 88] Temme, K-H. and Nitsche, A., "Chip-Architecture Planning: An Expert System Approach", in Gero, J. S., "Artificial Intelligence in Engineering: Design", pp.137-161, Elsevior (1988).

[Uchida 88] Uchida, S. et al., "Research and Development of the Parallel Inference System in the Intermediate Stage of the FGCS Project", Proceedings of the Fifth Generation Computer Systems, pp.16-36 (1988).

[Ueda 85] Ueda, K., "Guarded Horn Clauses", ICOT Technical Report, TR-103 (1985).

[Walters 88] Walters, J. R. and Nielsen N. R., "Crafting Knowledge-Based Systems", Chapter 15 "Knowledge Crafting with Multiple Contexts", John Wiley & Sons (1988).

[Yokoyama 90] Yokoyama T., "An Object-Oriented and Constraint-Based Knowledge Representation System for Design Object Modeling", Proceedings of the Sixth Conference on Artificial Intelligence Applications, pp.146-152 (1990).

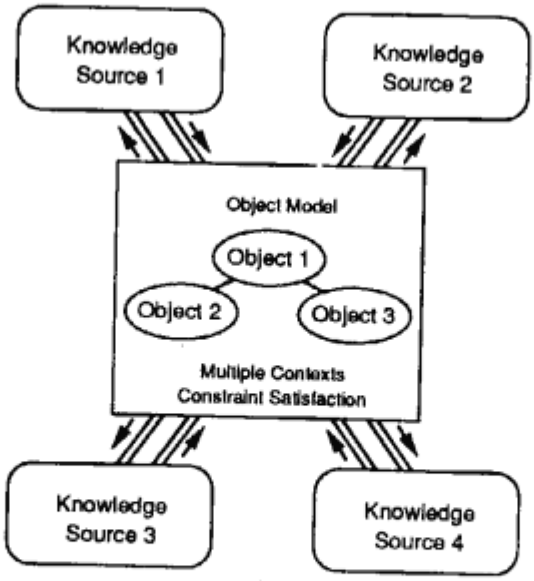

Figure 1. Parallel cooperative problem solving model

```
class microprocessor has
   slot
      alu, internal_bus, behavior,
      control, data_path, performance,
      machine_cycle,
      average_number_of_step, ..... ;
   constraint
      nogood( [internal_bus, control],
               ['1-Bus' , 'Instruction Prefetch'] ),
      performance
         = 1 / (average_number_of_step
                        * machine_cycle),
           ..... ;
end.
```

Figure 2. An example of class definition

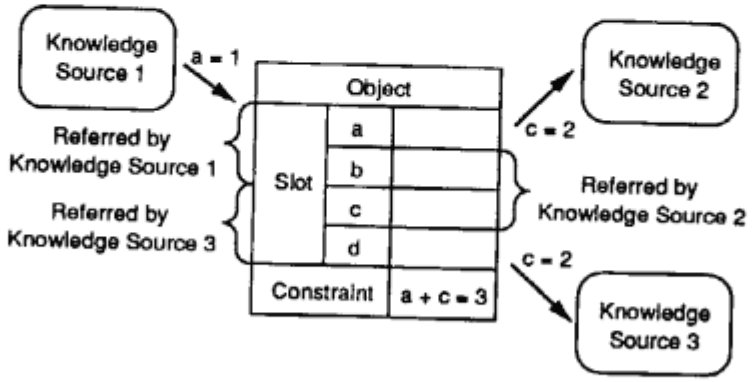

Figure 3. Constraint propagation

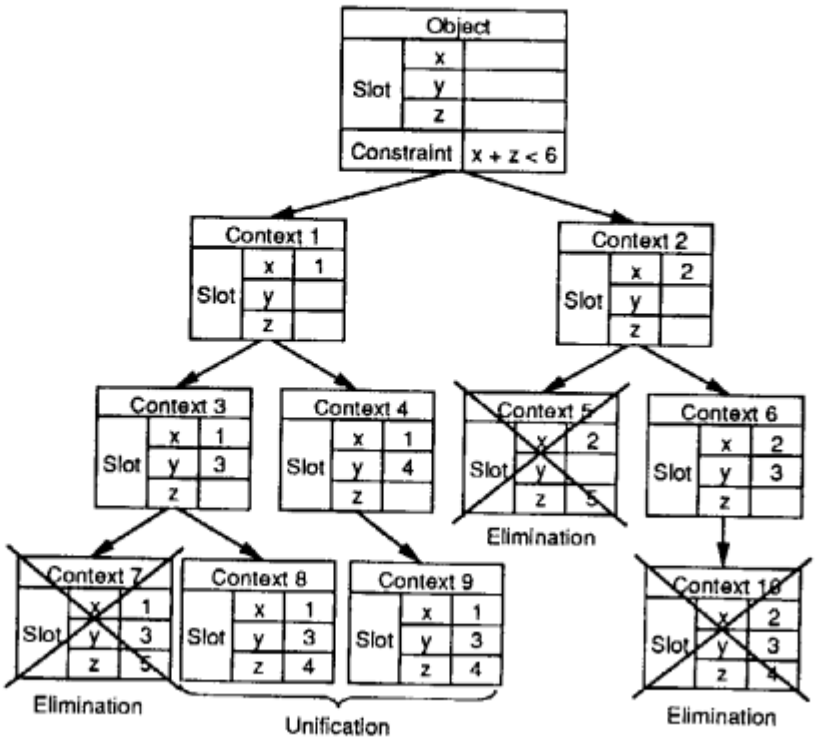

Figure 4. An example of context tree

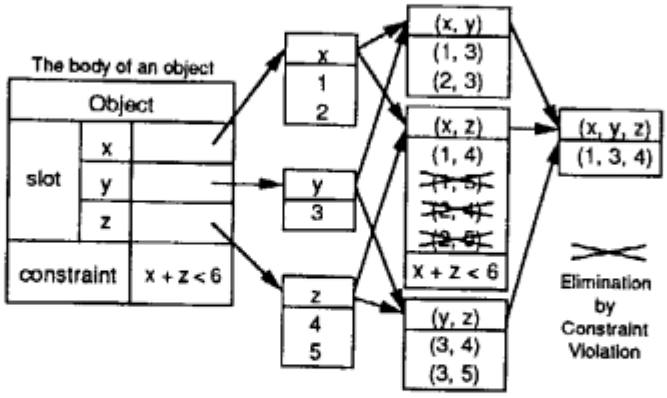

Figure 5. Node network for multiple-context management

```
ks data_path_designer has
   rule
      if      class(Obj, microprocessor),
              slot(Obj, {behavior, B})
      then  generate_data_path(B, D),
              set_slot(Obj, {data_path, D});
            ..........      ;
end.
```

Figure 6. Example of rule definition of a knowledge source

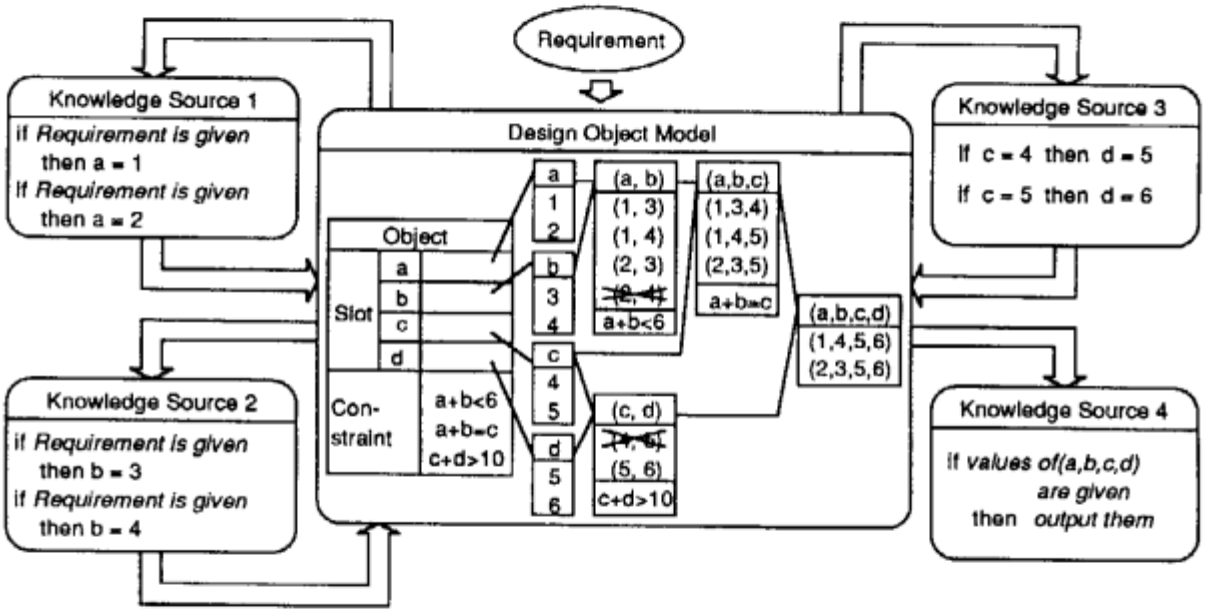

Figure 7. Cooperative behavior of the system