TM-1052

A Parallel Cooperative Problem-Solving System with Intelligent Blackboard

by

T. Yokoyama, M. Ono,
M. Wada & H. Ohsaki

May, 1991

A Parallel Cooperative Problem-Solving System with Intelligent Blackboard

Takanori Yokoyama*, Masayuki Ono**,

Masahiro Wada**, Hiroshi Ohsaki***

*Hitachi Research Laboratory, Hitachi Ltd.
426 Kuji-cho, Hitachi-shi, Ibaraki-ken, 319-12 Japan
e-mail: yokoyama@hrl.hitachi.co.jp

**Institute for New Generation Computer Technology
4-28, Mita 1-chome, Minato-ku, Tokyo 108 Japan
e-mail: {mono, mwada}@icot.or.jp

***Japan Information Processing Development Center
4-28, Mita 1-chome, Minato-ku, Tokyo 108 Japan
e-mail: j-ohsaki@icot.or.jp

Abstract

This paper presents a parallel cooperative problem solving system which consists of the intelligent blackboard and agents. The intelligent blackboard is not shared memory but a set of active objects which has functions for constraint satisfaction and multiple context management. An agent solves a subproblem accessing data on the intelligent blackboard. Cooperation between agents are supported by constraint propagation via the intelligent blackboard. To exploit parallelism, multiple context management is realized by a node network which deals with combinations of multiple values of slots of objects. The system is effective for design problems in which there is dependency between subproblems and there are many candidates for partial solutions. The system is based on data flow architecture. A prototype of cooperative design system has been developed on distributed multiprocessors Multi-PSI using concurrent logic programming language KL1.

1. Introduction

To solve a large-scale or complex problem, the problem should be divided into subproblems. The subproblems are solved and obtained partial solutions are integrated into a solution of the whole problem. Design is an interesting and important domain of knowledge engineering. Design is a problem to generate a solution that satisfies requirements and is regarded as a constraint satisfaction problem. In actual design by human designers, a large-scale or complex design problem is divided into subproblems which can be solved by one designer. This method is applied to knowledge-based design systems [Tong87].

For example, to design a microprocessor, we divide it into parts such as ALU, register, bus, and so on, then design the parts independently, and integrate them. But, it is not always possible to divide a problem into independent subproblems. Considering functional design of microprocessor, the problem can be divided into subproblems such as behavioral design, structural design, timing design, and so on. But, there are heavy dependencies between subproblems. This is caused by the structure of the microprocessor has not been defined and the design data must be shared by all the subproblems. So designers must cooperate each other to get a consistent solution in actual design. Cooperative problem solving technique must be applied to a design system for this type of problem [Klein89].

Cooperative problem solving systems have been developed in the domain called distributed artificial intelligence or distributed cooperative problem solving [Bond88]. There are many types of problem solving models such as blackboard model [Hayes-Roth85, Nii86], contract net model [Smith81], actor model [Hewitt77], and so on. The blackboard model is suitable for functional design because dependencies exist among subproblems.

Several cooperative design systems based on the blackboard model have been developed [Rehak85, Bushnell87, Temme88]. A Blackboard system consists of a blackboard and knowledge sources. Each knowledge source has knowledge to solve a given 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. A knowledge source may be called an agent.

Knowledge sources can be executed 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 executed in parallel. One of the major problems is scheduling. A scheduler of the system controls execution of knowledge sources and supports cooperation among agents. 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 the blackboard. As the blackboard is shared by knowledge sources, access to the blackboard may be a bottleneck of the system. Furthermore, to implement the blackboard on the multiprocessors without shared memory is difficult.

Many candidates for partial solutions are generated and a combinatorial explosion may be caused in the domain such as design. The mechanism to prune nodes or a search tree is required. And an architecture is also required in which different types of knowledge and reasoning mechanism can be used because the characteristics of the subproblems are different each other. So a new approach to cooperative problem solving is needed.

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. Parallel cooperative problem solving architecture that consists of an intelligent blackboard and agents is presented in this paper. Multiple-context objects which have constraint satisfaction mechanism represent a design object model on the intelligent blackboard. The problem solving architecture is applied to a simple design system.

2. Parallel Cooperative Problem Solving

2.1 Problem Solving Model

Cooperation among agents is important for solving functional design problems 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, agents must backtrack and generate another partial solutions. It is too inefficient. Therefore concurrent cooperative design by agents exchanging information, such as intermediate solutions and constraints, is required.

We propose a parallel cooperative problem solving model which composed of the intelligent blackboard and several agents. Figure 1 shows the structure of the model. A design solution is generated on the intelligent blackboard.

Cooperative problem solving requires modularity of knowledge. Design knowledge is classified into knowledge about design objects and knowledge about design methods [Nagai88]. Knowledge about design method is divided and modularized by agents. Each agent has different knowledge which is needed to solve each subproblem. Knowledge about design objects is stored on the intelligent blackboard and shared by all agents. agents have to generate one solution that is an instance of design object that satisfies design requirements. Knowledge about design object is represented as a design object model in a design system [Ohsuga85]. Dependencies between subproblems are caused by sharing a design object model.

The intelligent blackboard provides facilities to support cooperation among agents. A design object model is represented in the way of object-based representation. The intelligent blackboard is not mere shared memory but a set of active objects. Constraints on objects can be defined in the declarative form [Yokoyama90]. Cooperation among agents is based on consistency maintenance and constraint propagation by the active objects. General knowledge about design objects such as basic structure of design objects, constraints between parts to be satisfied, available parts, etc. is represented as class declarations. A design solution is generated as a set of instances on the intelligent blackboard in design process.

Each agent solves a subproblem using the knowledge about design method including heuristics. All agents share a design object model, but each agent accesses only part of the design object model. An agent reads data of the design object model and modifies it autonomously. To put it concretely, an agent reads values of slots (instance variables) of objects on the intelligent blackboard, generates partial solutions, and put them as new values of another slots of the objects.

Various types of reasoning can be applied to agents in this system. A system can contains different types of agents if the interfaces of those agents and the intelligent blackboard are the same one. In functional design, subproblems may require different types of reasoning.

2.2 Approach

It is difficult to parallelize a system for artificial intelligence. Architectural support is required for realizing an efficient parallel AI system because the paths to a solution is cannot be predefined and the control flow is nondeterministic. Though several parallel blackboard systems have been developed, the efficiency is not so good.

In our system, to exploit parallelism, an agent can be built with many fine-grained processes which can be executed in parallel. The objects can be executed in parallel too. Furthermore, an object is implemented as a set of processes and those processes can execute in parallel. Both agents and an intelligent blackboard can be distributed to multiprocessors.

Many data should be processed to realize an efficient parallel system. Plural candidates for a partial solution can be dealt within the system and those candidates are generated and processed in the way of pipeline processing. The interface between an agent and the intelligent blackboard is based on data-flow architecture using streams. The effect of pipeline parallelism can be obtained by processing data in succession by processes connected by streams.

We have implemented the system using KL1 language on a Multi-PSI machine [Uchida88]. KL1 is a concurrent logic programming language based on GHC language [Ueda85]. Multi-PSI is distributed memory multiprocessors. KL1 fits for process-based concurrent programming. Fine-grained processes can be realized with little overhead in KL1. A stream is a chain represented as a list in KL1.

3. Intelligent Blackboard

3.1 Object Definition

A design object model on the intelligent blackboard is represented in the way of object-based representation. General knowledge about general design objects is represented as class definitions. Instances are created and design data are set in their slots.

Slot names and constraints on the slots are declared in a class definition. A Constraint on slots can be represented as a predicate, an equation, or an inequality. A part of slots of an object can be accessed by agents. But, generally, different agents access different objects or different slots according to the viewpoints of agents. Figure 2 shows an example of class definition for a microprocessor. In this example, the constraint of prohibition on a combination is represented using *nogood* predicate and the constraint on the slot *performance, average number of steps* and *machine cycle* is represented as an equation. The former constraint forbids the combination of *one bus structure* as an internal bus structure and *instruction prefetch* as a control method.

Class definitions are translated to KL1 programs by the translator. Each object is implemented as a set of processes. Messages to an object are interpreted and executed concurrently. A design object model is accessed by agents in parallel except for the access to the same slot of the same object.

3.3 Constraint Satisfaction

An object in our system has constraint satisfaction mechanism and keep its state satisfying given constraints. When a value is set in slot by an agent, a constraint on the slot is evaluated. There are two types of constraint evaluation. One is to evaluate a constraint passively and the other actively.

Passive constraint evaluation is to test whether a constraint is satisfied or not when all values of slots of the constraint are given. For example, the values of x, y and z of constraint $x > y + z$ are given, whether the set of values satisfies the constraint or not is tested. Passive constraint evaluation is a basic facility to obtain a consistent solution.

Consistency maintenance of intermediate solutions is important in cooperative problem solving and parallel processing. Consistency maintenance is to evaluate constraints passively and keep the state of an object satisfying constraints. When agents set values to slots, constraints on the slots are evaluated and contexts (combinations of values of slots) that violate the constraints are rejected.

Active constraint evaluation is to calculate a value of a slot when the values of another slots are given. For example, when the values of x and y of constraint $x = y + z$ are given, the value of z is obtained. Active constraint evaluation is applied for the type of constraint that a unique value of a slot can be determined.

Constraint propagation among agents via a design object model is realized by active constraint evaluation without direct communication. An agent can influences another agents indirectly by constraint propagation and this effect can be used for cooperation among agents. Figure 3 shows an example of constraint propagation. The value of slot c can be calculated by evaluating the constraints with the value of slot a, and the value is propagated to another agents. This is an example using a constraint on slots of one object, but constraints between plural objects can be used for propagation.

Cooperation between agents which don't share data can be realized by constraint propagation via intelligent blackboard without direct communication. Cooperation is not affected by execution sequence because the direction of propagation is determined dynamically in run time.

3.2 Multiple Contexts

To exploit parallelism and get high efficiency, plural candidates for a partial solution should be dealt with simultaneously. The mechanism to deal with multiple values of a slot is realized by multiple-context objects in our system. Context means state of an object that is represented as a combination of values of slots.

In conventional systems, multiple contexts are represented as a context tree [Walters88]. An example of context tree is shown in Figure 4. The tree structure grows as new values are set to slots. But, consistent contexts don't increase monotonously because inconsistent contexts must be rejected. Inconsistent branches of a context tree should be pruned as early as possible for efficiency. Plural contexts may become the same state as shown in Figure 4, because execution sequence is nondeterministic and agents execute in parallel and asynchronously in the system. Those contexts should be unified immediately to remove redundant processing. The multiple context management with functions for pruning and unification is required.

We present a multiple-context object with a node network for context management. A combination of slots is managed by a node. An example of node network is shown in Figure 5. The node network forms a lattice. The head of node network is the body of an object. Each slot of the body points a node which deal 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. Constraints on slots are stored in the node which deal with the slots. A node receives data from the precedent nodes and combine the data. A node has functions for constraint evaluation. Only the combinations of values that satisfy constraints are stored in the node. Combinations that violate constraints are rejected. The combinations of values of all the slots that satisfy all the constraint is stored in the tail node in Figure 5.

If nodes for all combinations of slots were generated, a large number of nodes would be generated for an object. To reduce the number of nodes, the system provides a facility to generate only the nodes for essential combinations. Two kinds of combinations are essential: the combinations referred to by constraints given to the object and the combinations referred to by agents. The translator analyze class definitions and agent definitions and optimize node networks.

A node network is generated when an object is created. Each node is implemented as a process. An object is a set of processes that can be executed in parallel. Node processes are connected by streams. Demons for sending referred combination data to agents are attached to the node processes.

4. Agents and Cooperation

4.1 Agent Definition

Various kinds of reasoning which fit subproblems can be applied to agents in our model. But, knowledge is to be represented as a set of rules and agents execute as forward chaining production systems in the implemented system. This is because of representability of knowledge and easiness of implementation.

Figure 6 shows an example of agent definition. Rules are defined in the *if-then* form. A rule is fired when its condition part is matched to the data from the intelligent blackboard. All matched rules can be fired simultaneously without conflict resolution to exploit parallelism. Consistency is maintained in the way that the different states after different rules are fired are treated as different contexts. This is realized by facilities of the intelligent blackboard for multiple-context management and consistency maintenance.

An agent executes inference according to the changes on the intelligent blackboard. To put it concretely, an agent receives changed data from objects on the intelligent blackboard, generate new data, and set the data to objects. Intermediate solutions are integrated and tested by constraint evaluation every time a rule is executed. Inconsistent contexts are pruned by this mechanism.

Rules are translated to KL1 programs by the translator. When an agent sets data to slots of an object, the context must be specified to maintain consistency. But, this is realized by the translation not by a user. 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 *1*, set value *2* 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}]);
```

Thus, not only the value of *slot_2* but also the value of *slot_1* should be set in run time.

This translation is needed for consistency maintenance.

One rule is translated to one clause of KL1 and is executed as a process. Rules of an agent can be distributed to plural processors and the rules that matches data from the intelligent blackboard can be fired and executed in parallel.

4.3 Cooperative Behavior

The parallel cooperative problem solving system in this paper is based on the data flow architecture. Agents and the intelligent blackboard are connected by streams. Cooperation between agents is supported by constraint propagation and consistency maintenance by multiple-context objects on the intelligent blackboard.

Figure 7 shows a simple example of the system which is composed of four agents and the intelligent blackboard. The node network of the object on the intelligent blackboard is optimized. The representation form of a rule is simplified in the figure.

First, when the design requirement is given, rules in *agent 1* and *agent 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* of the object. Those data are sent to *node a* and *node b*, stored at the nodes, 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 combination data that satisfy the constraints given to the node are generated, stored at the node, and propagated to the *node (a, b, c)*. At the node, the constraints on *slot a, slot b* and *slot c* is evaluated actively and the values of *slot c* are obtained, the combination data are stored, and obtained values of *slot c* are sent to *node c*.

Then the rules of *agent 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)*. The reason the combination data are set to the node is to maintain consistency as mentioned at the previous subsection. Only the combinations that satisfy the constraints 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 combinations of values of *slot a, slot b, slot c* and *slot d* are generated. Note that only consistent combinations can be generated. When plural precedent nodes deal with the same slot as an element of slot combinations, the only combinations are generated that the value of the slot of one precedent node and the value of the slot of another precedent node are equal. Only the combinations (1, 4, 5, 6) and (2, 3, 5, 6) are generated as consistent data at the *node (a, b, c d)*. Finally, *agent 4* outputs the data as solutions.

5. A Prototype of Parallel Cooperative Design System

5.1 System Structure

We developed a prototype system for simple functional design. The target of the design is microprocessor. 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 as shown in Figure 8. But, the problem is simplified to realize the system with a few rules and objects. We analyzed the functional design of 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 the intelligent blackboard and six agents: five agents for subproblems shown above and an agent to output the design solutions. Class definitions for design object model are represented in the form as shown in Figure 2 and rule definitions of each agent are represented in the form as shown in Figure 6. There are about 60 rules and most of them are for generating candidates for intermediate solutions.

The system has been implemented on Multi-PSI. Static load balancing to allocate all the tasks before starting the job is used. Agents and nodes of objects are distributed to multiprocessors.

5.2 Results

There are many candidates for intermediate solutions in this problem. If all the candidates were combined simply, about thousands of candidates for solutions would be generated though there are only a few solutions that satisfy constraints. In our system, design is executed cooperatively pruning the candidates that violate constraints. The number of combinations of candidates dealt with at a node is below about twenty because only consistent combinations are generated. Therefore, the parallel cooperative problem solving system in this paper is effective for problems in which there are many candidates for solutions though a few of them are consistent.

Speedups of 2 times with 4 processors and 2.7 times with 8 processors are obtained. The result is not as good as we expected. Active processors decreases for execution process. All the processors are active at the first stage but only one or two processors are active at the last stage. The system should be applied to a large-scale problem and efficient load balancing must be used to get more speedup against the number of processors. We are studying on dynamic load balancing strategy.

6. Related Work

The research on parallel systems for AI have been done in various domains. But, to develop an efficient parallel or distributed cooperative problem solving system is difficult because of dependency among subproblems and nondeterminism of reasoning. A blackboard system fits for problems with dependency between subproblems because agents (knowledge sources) can share data on the blackboard. Several parallel or distributed blackboard systems have been developed [Engelmore88, Jagannathan89]. 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 agents, internal execution of an agent, and so on [Corkill89]. But, to get high efficiency is difficult by only these approaches. It is important to exploit knowledge parallelism, pipeline parallelism, and data parallelism [Nii88]. There is a report that to deal with many data is effective for efficiency [Rice89].

Poligon [Nii88, Rice89] 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 multiprocessor. The association is executed at compile-time. Poligon has been developed for applications such as real-time signal understanding and data fusion in mind. Pligon's 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 a 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 [deKleer86] provides a facility for maintenance of consistent multiple context. Multiple-context management without redundancy can be realized by applying ATMS to the intelligent blackboard regarding values of slots as assumptions. But, the number of nodes would be enormous if regarding a value of a slot 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 deal with a combination of slots not a combination of values. Each node manages 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 many *nogoods* must be declared to reject both the combinations that violate constraints and meaningless combinations (for example, the combination of *slot_a = 1* and *slot_a = 2*). ATMS is a powerful system for hypothetical reasoning but is not suited for our goal.

The node network of our system has a function to get data that match condition parts of rules of agents. The node network has part of functions of RETE network [Forgy82]. But, the node of our system has functions for constraint satisfaction. Though our system is not proper to a production system in principle, if the system is optimized for a production system, an efficient parallel production system may be realized.

7. Conclusion

A Parallel cooperative problem solving system for problems in which there is dependency among subproblems is presented. The system consists of the intelligent blackboard and agents which solve subproblems. The intelligent blackboard contains 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 agents are supported by constraint propagation via the intelligent blackboard without direct communication or explicit shared data.

The system is based on data flow architecture. Rules of agents and nodes of node networks of objects are implemented as processes and they are connected by streams. The processes can be distributed to multiprocessors and executed in parallel. This system is implemented using concurrent logic programming language KL1 on distributed memory multiprocessors Multi-PSI.

According to the experience of applying the system to a simple functional design of microprocessor, we think the system is effective for problems such as design in which there are many candidates for partial solutions. But, the system should be improved to get high efficiency. Particularly efficient load balancing strategy is necessary.

Acknowledgements

This research has been done in Institute for New Generation Computer Technology (ICOT). I would like to thank Dr. Katsumi Nitta, the chief of the Seventh Research Laboratory at ICOT, and all members of the Seventh Research Laboratory for their helpful comments. I would also like to thank 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

[Bond88] Bond, A. H. and Gasser, L. (eds.), "Readings in Distributed Artificial Intelligence", Morgan Kaufmann (1988).

[Bushnell87] 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).

[Corkill89] Corkill, D. D., "Design Alternatives for Parallel and Distributed Blackboard Systems", in [Jagannathan89], pp.99-136 (1989).

[deKleer86] de Kleer, J., "An Assumption-based TMS", *Artificial Intelligence*, Vol.28, pp.127-162 (1986).

[Engelmore88] Engelmore, R. and Morgan, T. (eds.), "Blackboard Systems", Addison Wesley (1988).

[Forgy82] 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-Roth85] Hayes-Roth, B., "A Blackboard Architecture for Control", *Artificial Intelligence*, vol.26, pp.251-321 (1985).

[Hewitt77] Hewitt, C., "Viewing Control Structures as Patterns of Passing Messages", *Artificial Intelligence*, vol.8, pp.323-364 (1977).

[Jagannathan89] Jagannathan, V. et al. (eds.), "Blackboard Architectures and Applications", Academic Press (1989).

[Klein89] Klein, M. and Lu, Stephen C.-Y., "Conflict Resolution in Cooperative Design", *Artificial Intelligence in Engineering*, vol.4, no.4, pp.168-180 (1989).

[Nagai88] Nagai Y. et al., "Expert System Architecture for Design Tasks", *Proceedings of the Fifth Generation Computer Systems*, pp.296-317 (1988).

[Nii86] 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).

[Nii88] Nii, H. P. et al. , "Frameworks for Concurrent Problem Solving: A Report on CAGE and POLIGON", in [Engelmore88], pp.475-501 (1988).

[Ohsuga85] 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).

[Rehak85] 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).

[Rice89] Rice, J. et al., "See How They Run... The Architecture and Performance of Two Concurrent Blackboard Systems", in [Jagannathan89], pp.153-178 (1989).

[Smith81] Smith, R. G. and Davis, R. "Frameworks for Cooperation in Distributed Problem Solving", *IEEE Trans. on Systems, Man and Cybernetics,* SMC-11-1, pp.61-70 (1981).

[Temme88] 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).

[Tong87] Tong, C., "Toward an Engineering Science of Knowledge-Based Design", *Artificial Intelligence in Engineering,* vol.2, no.3, pp.133-166 (1987).

[Uchida88] 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).

[Ueda85] Ueda, K., "Guarded Horn Clauses", *ICOT Technical Report,* TR-103 (1985).

[Walters88] Walters, J. R. and Nielsen N. R., "Crafting Knowledge-Based Systems", Chapter 15 "Knowledge Crafting with Multiple Contexts", John Wiley & Sons (1988).

[Yokoyama90] 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).

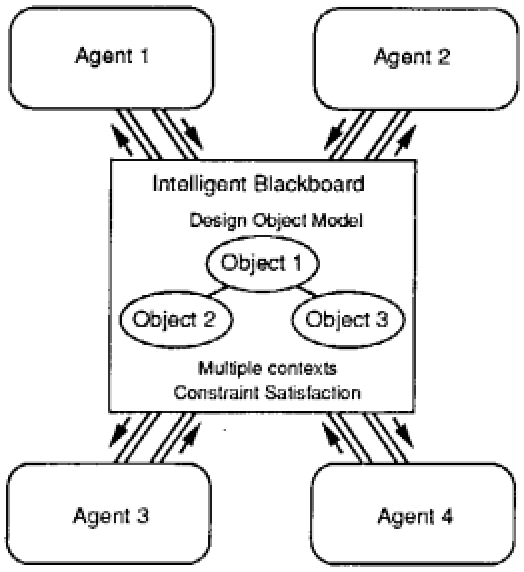


Figure 1. Parallel cooperative problem solving model

```
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. Example of Class Definition

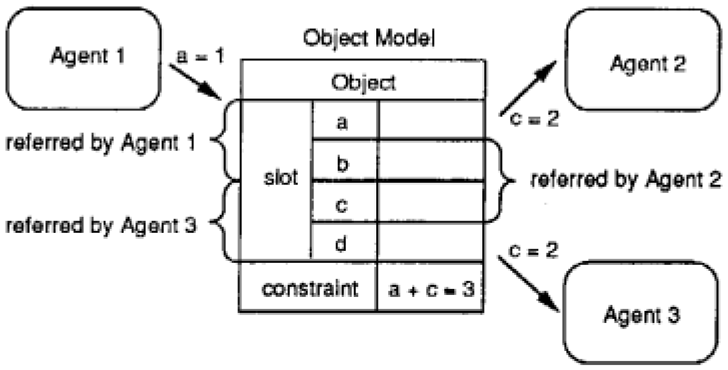


Figure 3. Constraint propagation among agents

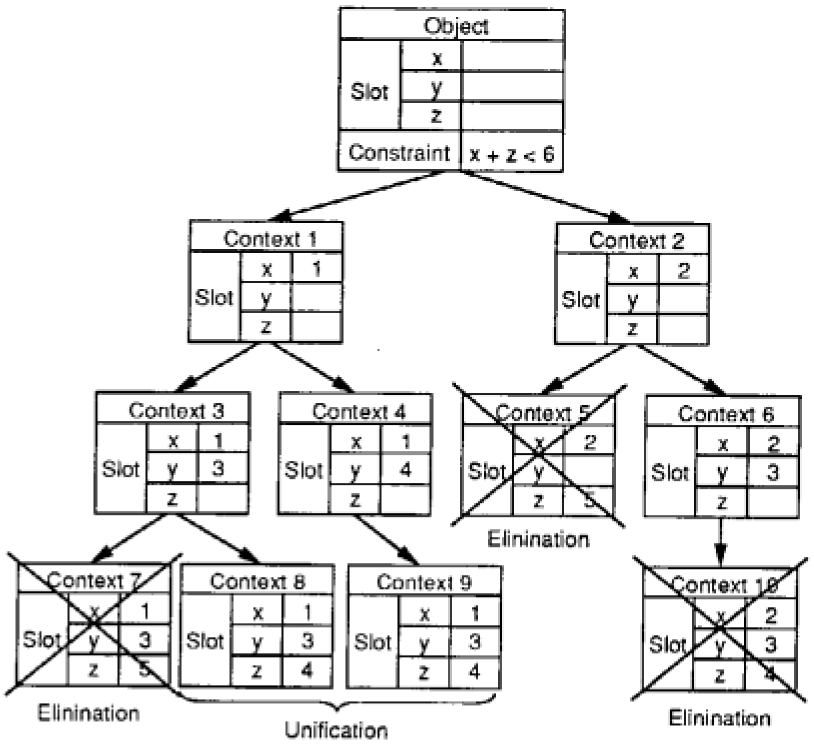


Figure 4. An example of context tree

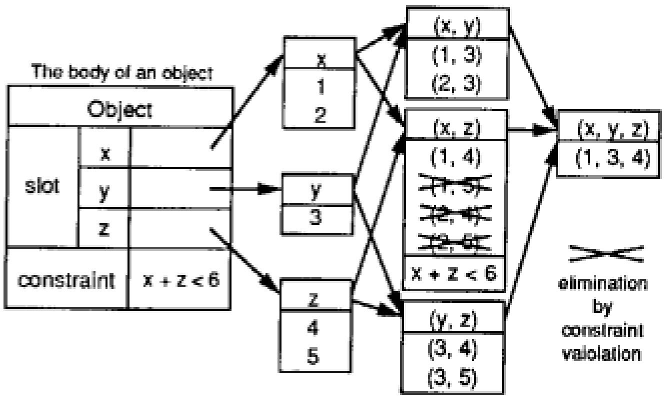
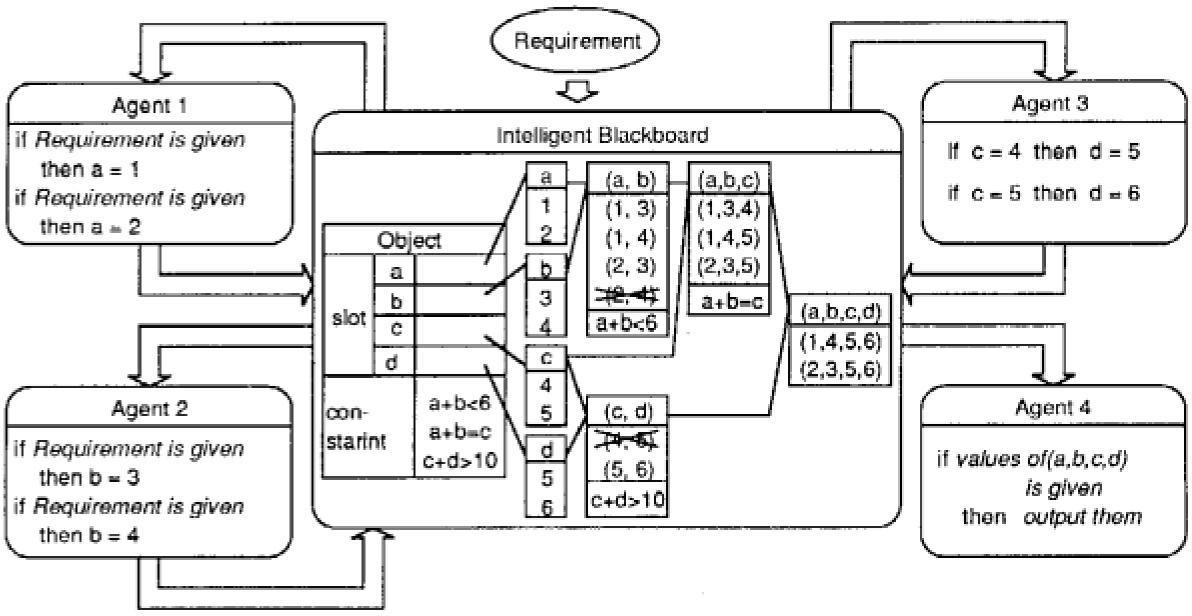


Figure 5. Node network for managgement of multiple contexts

```
agent  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 an agent



Requirement
Agent 1
if Requirement is given then a = 1
if Requirement is given then a = 2
Agent 2
if Requirement is given then b = 3
if Requirement is given then b = 4
Agent 3
If c = 4 then d = 5
if c = 5 then d = 6
Agent 4
if values of(a,b,c,d) is given then output them
Intelligent Blackboard
Object
slot
a
b
c
d
con-starint
a+b<6
a+b=c
c+d>10
a 1 2
b 3 4
c 4 5
d 5 6
(a, b) (1, 3) (1, 4) (2, 3) ~~(2, 4)~~ a+b<6
(a,b,c) (1,3,4) (1,4,5) (2,3,5) a+b=c
(c, d) ~~(4, 5)~~ (5, 6) c+d>10
(a,b,c,d) (1,4,5,6) (2,3,5,6)

Figure 7. Behavior of agents and the intelligent blackboard

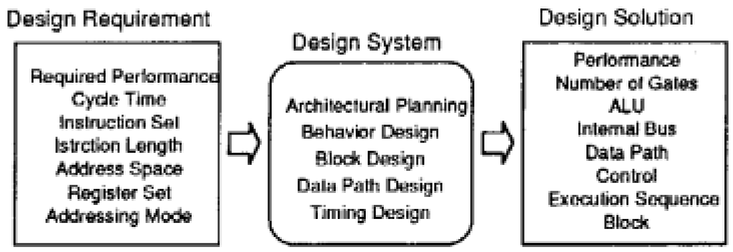


Fig. 8 Inputs and Outputs of Prototype Design System