Concurrent Engineering: Research and Applications (1993) **1**, 179–187

CONCURRENT ENGINEERING: Research and Applications

A Negotiation Platform for Cooperating Multi-agent Systems

Faruk Polat*†, Shashi Shekhar* and H. Altay Guvenir†

**University of Minnesota, Computer Science Department, Minneapolis, Minnesota 55455, USA and †Bilkent University, Department of Computer Engineering and Information Science, Bilkent, 06533 Ankara, Turkey*

Received 23 May 1993; accepted in revised form 26 June 1993

Distributed artificial intelligence attempts to integrate and coordinate the activities of multiple, intelligent problem solvers that come together to solve complex tasks in domains such as design, medical diagnosis, business management, and so on. Due to the different goals, knowledge, and viewpoint of the agents, conflicts might arise at any phase of the problem-solving process. Managing diverse knowledge requires well-organized models of conflict resolution. In this paper, a system for cooperating intelligent agents which openly supports multi-agent conflict detection and resolution is described. The system is based on the insights, first, that each agent has its own conflict knowledge which is separated from its domain-level knowledge; and, second, that each agent has its own conflict management knowledge which is not accessible to or known by others. Furthermore, there are no globally-known conflict-resolution strategies. Each agent involved in a conflict chooses a resolution scheme according to its self interest. The problem-solving environment allows a new problem solver to be added, or an existing one to be removed, without requiring any modification of the rest of the system, and therefore achieves open information system semantics.

Keywords: distributed artificial intelligence, conflict detection and resolution, conflict knowledge, open information system.

1. Introduction

Distributed artificial intelligence (DAI) is a subfield of AI, which attempts to integrate existing problem-solving methods used in classical AI in order to develop systems that benefit from multiple agents' point of view. A solution developed by multiple agents incorporates aspects of each agent's problem-solving capabilities and perspectives, rather than just the view of an individual agent's analysis of the problem [3], [4], [6], [17], [19], [20], [24].

Applications of DAI can be seen in human problem-solving tasks such as design, medical diagnosis, research, business management, and human relations. Several systems reflecting the DAI approach, such as Hearsay-II [5], Contract Net [24], Distributed Vehicle Monitoring Testbed [16], MDX [2, 10], Coop [23], etc. are described in the literature. Managing diverse knowledge is difficult because one has to take into account the problems which will arise in working out solutions in the face of conflicting goals, constraints, viewpoints, and knowledge of heterogeneous agents.

In this paper we describe a system in which a set of knowledge-based agents cooperate to solve design problems. The system is based on resolving of conflicting solutions that have been generated by agents having different goals, priorities, and evaluation criteria. Existing approaches to conflict management [1], [14], [15], [30] rely on coordinated resolution strategies which require resolution of a conflict based on a globally agreed-upon strategy. In existing systems, conflict-resolution knowledge is either maintained centrally or replicated by all agents. In any case, one of the disputants is given the power to take control of the conflict and use a resolution scheme known to all agents. In the approach described in this paper, however, agents are free to choose the most appropriate action, given their understanding of the global and local situations, as well as their own capabilities. They maintain their own set of conflict-resolution schemes which are not globally known. Using their own conflict knowledge, the participants may reach an agreement on a revised solution.

There are several reasons why conflicts need to be resolved by using agents' private conflict-resolution knowledge instead of global knowledge about conflict resolution. First of all, this task is very similar to the resolution of conflicts that occur among human beings in solving complex problem tasks in domains like design, diagnosis, business management, etc. When a conflict is detected, it is not resolved by a central authority using global conflict-resolution knowledge, but rather by specialists who are involved in the conflict and negotiate a revised solution that will be acceptable to all of them, using their own conflict-resolution knowledge and perspectives. Second, global conflict resolution requires consistent merging of the conflict-resolution knowledge obtained by each agent. This makes the maintenance of global knowledge difficult. Because when a new agent is added to, or removed, from the system, or the conflict-resolution knowledge of an agent is revised, the global conflict-resolution knowledge must be rebuilt accordingly. Lastly, distribution of knowledge leads to reliability and fault-tolerance to agent failures.

In Section 2 an overview of conflict management in cooperating intelligent systems is presented and the existing approaches are summarized. Section 3 describes the new system for cooperating intelligent systems. Section 4 explains how problem

Correspondence: Visiting Scholar, Faruk Polat, University of Minnesota, Computer Science Department, 4-192 EE/CSci Bldg, 200 Union St SE, Minneapolis, Minnesota 55455, USA

1063–293X/93/030179 + 09 $08.00/0

solving proceeds within the system. Section 5 describes how conflict detection and resolution takes place in the system. Section 6 includes a design example to illustrate how conflict detection and resolution take place in the system. Finally, the last section summarizes the new problem-solving environment, emphasizing its characteristics.

2. Conflict management among intelligent systems

When several specialized agents combine to solve a common problem, conflicts might appear as a result of incorrect and incomplete local knowledge, different goals, priorities, and solution evaluation criteria. When there are several conflicting proposed solutions for a (sub)problem, the agents involved in a conflict must either agree to choose one proposal, cooperatively revise one, or search for a new solution that will be acceptable to every agent.

A common practice in building knowledge-based systems is to avoid potential conflicting situations through analysis and consistency-checking of the knowledge base at development time [8], [18], [21]. This approach, though effective, becomes very costly as the amount and diversity of knowledge increases. Resolving all conflicts, no matter how unlikely, at development time can be prohibitively time-consuming. Moreover, dividing the domain knowledge into smaller internally consistent collections is difficult.

The problems encountered when resolving conflicts at development time can be avoided by allowing conflicts to occur and be resolved at run time. In other words, participating agents are allowed to generate conflicting solutions to the subproblems at run time. In the case of a conflict, a set of strategies could be used to resolve the conflict. Some examples of strategies include *backtracking, compromise negotiation* (a solution is iteratively revised by sliding a value, or set of values, along some dimension until a middle point is found that is mutually acceptable), *integrative negotiation* (the most important goals are found, and a solution is found which fulfils all of these goals), *constraint relaxation, case-based* and *utility reasoning methods, multi-agent truth maintenance*, etc. [1], [13], [14], [15], [22], [27], [28], [30]. Work in this class comes closest to providing conflict-resolution expertise with first-class status.

Next, we will summarize studies which emphasize the use of conflict resolution within the DAI paradigm. Klein and Lu [14] propose a model for cooperative design that emphasizes the parallel interaction of design agents. Given a design problem, design agents solve the subproblems relevant to their expertise. When a conflict is detected, the conflict-resolution agent takes control and tries to resolve it. Lander and Lesser [15] propose a cooperating expert framework (CEF) to support cooperative problem solving among sets of knowledge-based systems. All of the agents have a global knowledge of conflict-resolution strategies. Werkman [30] developed a system called the *Design Fabricator Interpreter* (DFI), which is a framework for distributed cooperative problem solving among construction agents. Conflicting recommendations issued by design agents are resolved by a *third-party arbitrator* agent. The arbitrator makes suggestions based on globally-known conflict-resolution knowledge. Adler *et al.* [1] discuss methods of conflict resolution in the domain of telephone network traffic control. A homogeneous group of agents has geographically-divided responsibilities with no overlap. The basic problem that the agents are to solve is excessive demand for resources in some parts of the network. Their research addresses how conflicts on the resource usage of resources could be resolved.

3. A negotiation-based environment for cooperating intelligent agents

The cooperating problem-solving environment is organized as a community of cooperating problem-solving agents, where each agent is represented as a fully-functional and autonomous knowledge-based system. The system is designed for solving problems in the domain of design and is based on the insights that each design agent has its own conflict-resolution knowledge separated from its domain-level design knowledge, and that this knowledge can be instantiated in the context of particular conflicts into specific advice for resolving these conflicts. The system allows a new problem solver to be added, or an existing one to be removed, without requiring any modification to the rest of the system. The system, therefore, can be considered as achieving open systems semantics* [11], [12] in the sense that it does not only allow scalability (the ability to increase the scale of commitment) but also robustness (the ability to keep commitments in the face of conflicts), which are two primary indicators in open systems semantics.

3.1. Architecture of the model

The cooperative design environment (Figure 1) is composed of a set of design agents which are fully-functional knowledge-based systems and are in a shared medium. The agents communicate by posting assertions in a shared language. This requires that the agents have translation capabilities.

3.1.1. Representation of problem

Definition (objects)

$O = \{o_1, o_2, \ldots, o_{No}\}$ represents the set of objects that contain information to be used by the agents in their design computations, Each object represents a separate element in the universe.

For each object $o_i \in O$, $1 \leq i \leq N_o$, $Attribute(o_i) = \{att_{i1}, att_{i2}, \ldots, att_{iMi}\}$ denotes a set of data attributes that include information in the form of either numerical/symbolic constants or procedures (methods) that yield numerical/symbolic values and constitute the derive attributes from basic data.

Definition (relationships)

$Rel = \{rel_1, rel_2, \ldots, rel_{Nrel}\}$ denotes the set of relationships between the objects involved in the problem to be solved. For $rel_i \in Rel$, $1 \leq i \leq N_{rel}$, each rel_i denotes hierarchical or logical relationships among certain types of objects.

Definition (tasks)

$T = \{t_1, t_2, \ldots, t_{Nt}\}$ represents the set of tasks to be performed. A task represents simply a goal at any level or granularity that must be satisfied by at least one of the agents in the problem-solving network. There exists a partial order over the tasks in T that the agents should follow.

* Open systems deal with large quantities of diverse information and exploit massive parallelism.

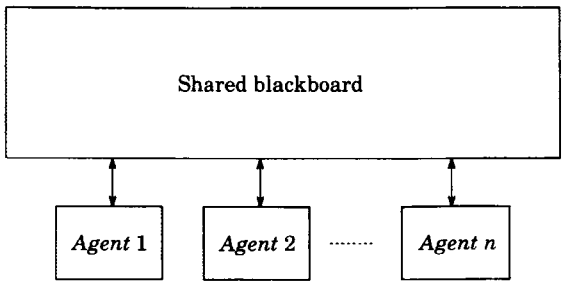


Figure 1. The architecture of the problem-solving environment.

Definition (actions and resources)

$Act = \{act_1, act_2, \ldots, act_{Nact}\}$ denotes set of allowable actions that the agents execute in achieving their assigned tasks. For $act_i \in Act$, $1 \leq i \leq N_{act}$, each act_i represents an action in the world which involves a set of objects. For instance, actions might request resources to accomplish problem-solving work.

Resources $= \{res_1, res_2, \ldots, res_{Nres}\}$ *denotes the set of resources.* $Res(act_i) \subseteq Resources$ denotes the set of resources required by an action $act_i \in Act$. $Amount(res_j, act_i)$ is a natural number to denote the amount of resource res_j to be used by act_i.

Actions can be defined differently in different domains. An action aims at establishing a relationship among a set of objects. In the engineering design domain, actions can be selecting appropriate objects, or adding/deleting/modifying a relationship that involves certain types of objects.

3.1.2. Shared medium

The shared medium is a public repository available to all agents. This permits the storage of "global" information, although the information can only be used locally by the agents. Alternatively, it would be possible to convey information directly through point-to-point communication channels or reserved-spot communication [31]. The shared medium is partitioned into four sections, allowing fast access, and delete and update operations of units. These sections are called *problem*, *solution*, *proposal*, and *conflict* areas.

The problem area of the shared medium contains the initial problem definition and overall requirements that must be taken into account by the design agents. A problem instance is a tuple of the form:

$$Problem - Instance = <a_o, T, C, I>,$$

where a_o denotes the problem originator, an agent that defines the problem to be solved; T is the set of tasks that must be satisfied for a design to be accepted; C is the set of constraints that design agents should not violate†; I denotes the initial problem information (such as the layout of a room if the problem is to design an office).

The solution area of the shared medium includes the evolving of design template, to which non-conflicting design commitments produced by the agents are added. The proposal area includes partial and incomplete solutions at several layers of abstraction produced by design agents. Design agents insert their solutions as proposals into this area. A proposal instance is a tuple of the form:

$$Q = < a_i, q_j, Act_{ij}, R_{ij}, C_{fij} >,$$

where a_i denotes the owner of the proposal; q_j is the id of the proposal; $Act_{ij} \subseteq Act$ is an ordered set of proposed actions to update the current design template; R_{ij} consists of the reasons justifying each of the actions in Act_{ij}‡; Cf_{ij} is the confidence factor that indicates the confidence of the owner in generating such a proposal. This information would be useful for other agents if the owner utilized inaccurate or incomplete knowledge in producing its solution.

The conflict area is the place where agents put their critiques related to a new design commitment. A portion of this area provides a communication medium with agents that are involved in a conflict situation. This area holds evaluation results and conflict resolution recommendations issued by design agents. An evaluation result instance is a tuple on the form:

$$ER = < a_i, q_j, Act - c_{ij}, Ra_{ij}, Re_{ij} >,$$

where a_i denotes the owner of the evaluation result tuple; q_j is the identity of the proposal evaluated; $Act - c_{ij}$ is the set of actions in proposal q_j that are criticized by agent a_i ($Act - c_{ij} \subseteq Act_{ij}$ in q_j); Ra_{ij} is the set of ratings (evaluation results) for each action in $Act - c_{ij}$; Re_{ij} is the overall result of the proposal q_j which is either a "conflicting proposal" or a "non-conflicting proposal".

A conflicting resolution instance is a tuple of the form:

$$CR = < a_i, q_j, Aref_{ij}, Rref_{ij} >,$$

where a_i is the owner of the conflict-resolution tuple; q_j is the identity of the related proposal; $Aref_{ij}$ is the set of refinements for those conflicting actions of q_j; and $Rref_{ij}$ includes the reasons for the proposed refinements.

There are also a few other tuple types used in negotiation dialogue among the agents involved in a conflict situation that are not described here.

3.1.3. Agents

Definition (agents)

$A = \{a_1, a_2, \ldots, a_{Na}\}$ represents the set of agents that will participate in the problem-solving process. For $1 \leq i \leq N_a$, an agent, a_i supports:

- A *database* that includes:
  1. $T(a_i)$ which denotes the set of tasks that agent a_i can perform ($T(a_i) \subseteq T$).
  2. *Facts.*
  3. *Historical information.*
- A *knowledge base* that includes:
  1. *Domain knowledge.*
  2. *Control knowledge.*
  3. *Conflict-resolution knowledge.*
- *Control procedures.*

At the start of problem solving, each agent examines the global task set, T, and selects those tasks that it can perform. An agent checks the objects and relationships to be established among the objects that have been introduced in a particular task definition. The agent may later decompose the task into several

† Some of the constraints may be violated through negotiation with the problem originator.

‡ R_{ij} could be empty, because an agent might not provide reasons for its proposal due to its inaccurate or incomplete knowledge.

subtasks (goals) according to its problem-solving capabilities. In part of their databases, agents also maintain two types of historical information related to the solution-generation phase and the conflict-resolution phase. This not only makes backtracking possible, but also allows case-based information to be used later for solving similar problems.

The knowledge base includes domain and control knowledge, just like in a classic knowledge-based system. In addition, it also contains conflict-resolution knowledge that can be used in cooperatively managing conflicts with other agents. This knowledge is not known globally, and it varies with respect to the agents' beliefs and understanding of the environment. The general controller includes procedures for generating and evaluating design commitments, managing conflicts, and translating messages into the common language.

The agents are actually heterogeneous in the sense that they might use different knowledge-representation techniques and inference mechanisms. Agents are assumed to generate proposals (solutions for subproblems) according to their knowledge. They cooperate to achieve the common goal of solving a global problem. The system can be augmented to support special agents such as database systems. For the time being we will assume that each agent is a knowledge-based system that makes an offer to solve subproblems (produce proposals) and cooperates with others in resolving conflicts through negotiation. Local knowledge is represented in whatever language desired but cannot be accessed by any agent except its owner. Several knowledge-representation techniques have been developed for the domain of design [7], [9], [25], [26], [29]. If the internal language is not the same as the shared language, translation procedures are incorporated within the agents. In the case of a conflict, agents might make some or all of their goals, constraints, and even knowledge available to others.

3.2. Problem-solving phases

Problem solving in the system is initiated by one of the agents asserting a problem definition $P = < a_o, T, C, I >$ into the problem area of the shared medium. Figure 2 outlines the basic steps in problem-solving phases. All the agents have the right to access the shared medium. When a problem definition is asserted into the shared medium, each agent examines the task set, T, to see which of the tasks it can perform. An agent a_i adds a task into its task set $T(a_i)$ if it has the knowledge to select particular object instances from its database and to establish the necessary relationships among the objects. When this process is completed by all of the agents, each agent knows the tasks it is going to perform. Some of the tasks might have been attempted by more than one agent, since the knowledge and problem-solving capabilities of different agents might overlap. However, one agent might have *deep* knowledge, whereas another might have *shallow* knowledge. In the negotiation phase, the proposal issued by the agent that has deep knowledge has more credibility than the proposals issued by other agents.

Each agent is also aware of the partial order among the tasks. Therefore, an agent can attempt the next task in its task queue if and only if all of its preceding tasks have been satisfied and agreed on. All interested agents, after examining the problem

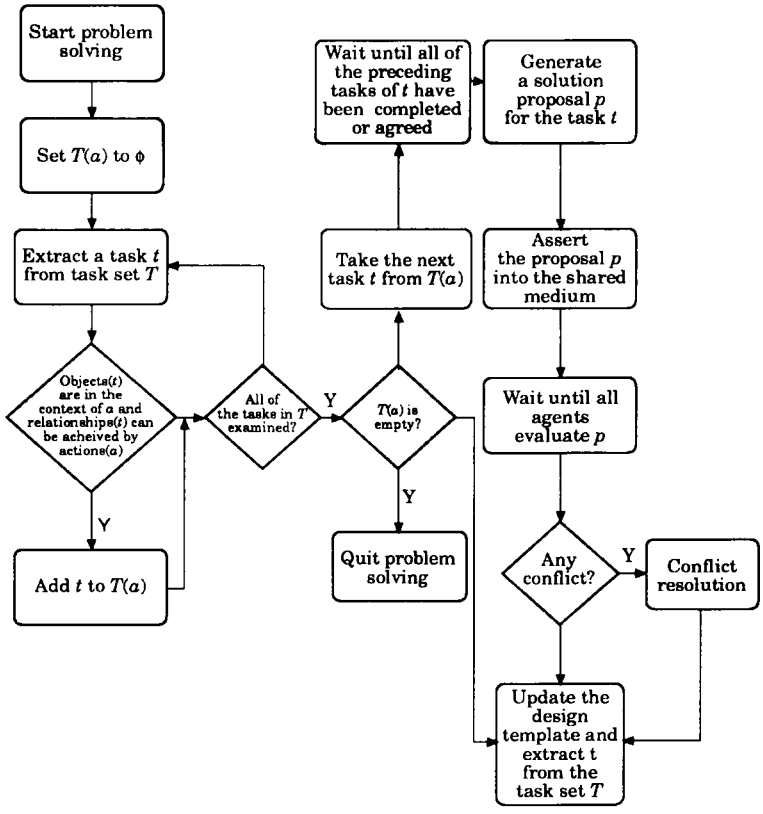


Figure 2. Problem-solving steps within agent *a*.

definition, are instantiated and they then start producing design proposals related to their expertise, knowledge, and viewpoints. When a design proposal is generated, it is put into the proposal area. The agent that produces this proposal also includes explanation regarding which of the agents' goals and constraints caused such a proposal. This explanation allows other agents to understand why such a proposal has been generated.

After a proposal has been generated, all of the agents are signalled. An agent does not interrupt its proposal generation process if it is already working on another proposal, but it immediately awakens another process, the evaluation process, that will run in parallel with the proposal generation process. The evaluation process first informs the owner of the proposal whether it is going to criticize the proposal. If not, the evaluation process will go to sleep and wait for another proposal to be asserted. If the agent is interested in the proposal, it evaluates it and posts the result in the conflict area of the shared medium. The owner of the own proposal also evaluates its proposal, usually as part of the solution-generation process. It is necessary for the owner to indicate its confidence in its own solution, because it might have used incomplete or inaccurate knowledge in producing that solution. The evaluation process results in a rating being produced which shows the "quality" of a solution with respect to the goal criteria. However, the agents use their internal evaluation criteria, and therefore may not share a common rating scale for their findings.

After all the interested parties have finished evaluating the newly-asserted proposal, those agents that have identified the proposal under consideration as conflicting with their beliefs come together to resolve the conflict (those interested agents that put *ER*s in which the R_e part is a "conflicting proposal"). Agents in our system do not have a global knowledge of conflict-resolution strategies. However, in existing systems, agents are assumed to be knowledgeable about the global conflict-resolution knowledge. In our system, each agent has its own conflict-resolution knowledge that allows it to participate in the process of conflict resolution. The result of conflict resolution is either revision or abandonment of the proposed solution.

When none of the interested agents detect any conflict related to a proposal, the partial design template residing in the solution area is updated by using the design contribution that exists in the proposal. This process continues until the design template meets the requirements specified by the agent that put the initial problem definition into the problem area of the shared medium. The design process may also be terminated, although the agent that put in the problem definition will not be satisfied. This may happen in cases where none of the agents can generate a non-conflicting design proposal.

4. Conflict detection and resolution

When an agent detects a conflict, it participates in the resolution process based on its own conflict-resolution knowledge. Each agent may utilize different conflict-resolution strategies. For example, suppose that we are given the problem of designing an office. The functionality agent suggests that the PC desk should be put close to the window, so that a PC user could have a look outside when (s)he is bored. On the other hand, the computer specialist, detecting a conflict, argues that sunlight could damage the PC. The computer specialist uses a conflict-resolution strategy which says "put electrical devices far away from windows". The functionality agent, however, uses a domain-independent resolution scheme the "try other subgoal alternatives". Eventually, the two agents revise the proposal, by using different resolution schemes, such that the PC desk is put into a place in the office which is not exposed to sunlight.

An agent, a_i, maintains an issue set, *Issues*(a_i), which includes domain issues used to evaluate partial plans proposed as solutions to tasks. For each issue in this set, a set of evaluation procedures is attached. These procedures are for detecting potential conflicts in the proposed solution from the agent's perspective on this issue. Figure 3 outlines the basic steps in proposal evaluation and conflict detection. When a proposal is asserted, each agent evaluates the solution based on these issues. However, it is not necessary for an agent to criticize a proposal from all of its issue perspectives since it may not have the knowledge to criticize the proposal from some perspectives. An agent may detect simultane several conflicts in a proposal based on different issues. Each such conflict is specified in the evaluation result tuple that will be put into the shared medium.

Upon detecting the conflict situation, an agent uses its conflict-resolution knowledge to overcome the conflict from its perspective. Conflict resolution knowledge is composed of general strategies (domain-independent) and specific strategies (domain-dependent). Domain-dependent strategies are gath-

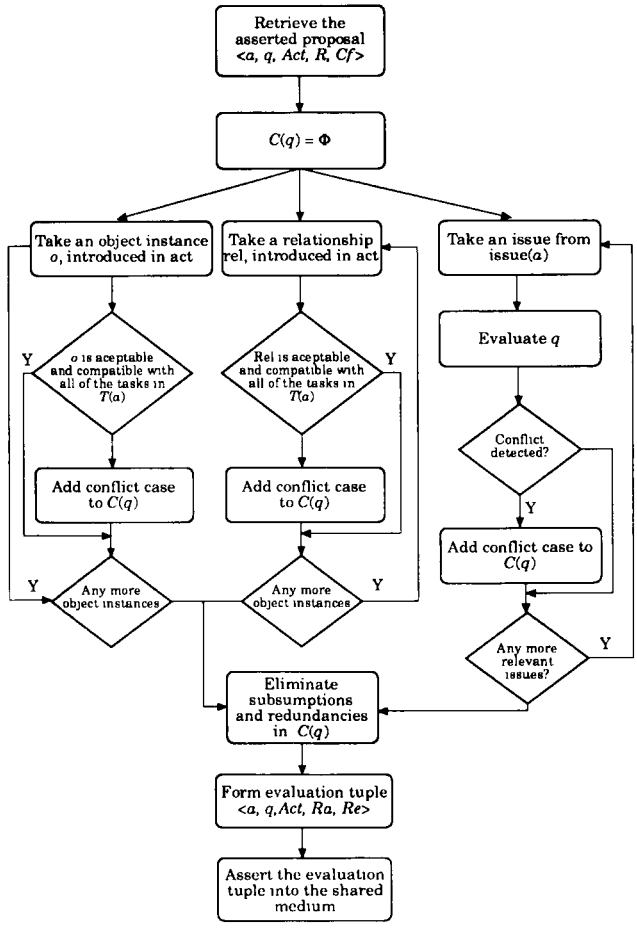


Figure 3. Proposal evaluation and conflict detection within agent *a*.

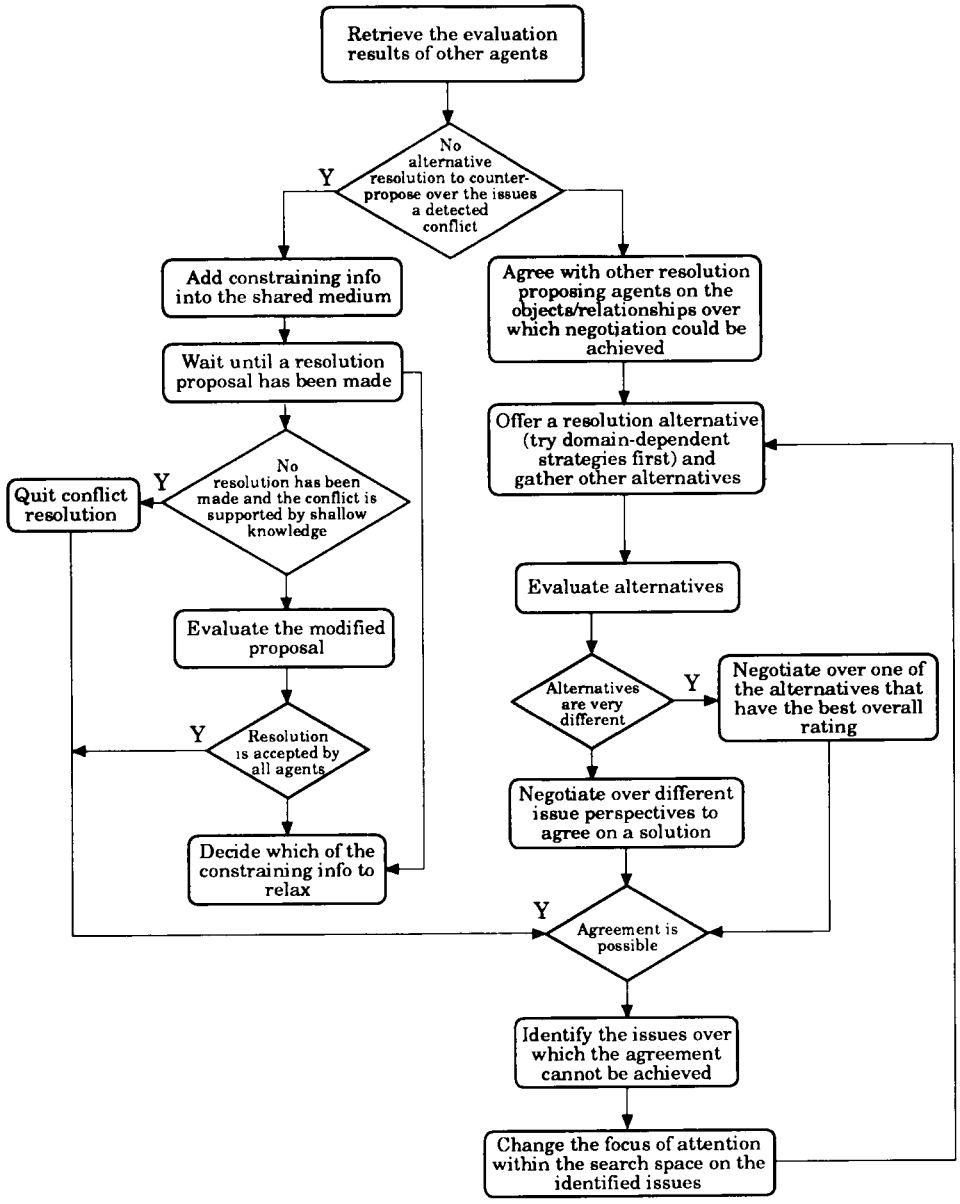


Figure 4. Conflict resolution steps within agent *a*

ered during the knowledge-acquisition phase. Agents prefer to use the most specific strategies first for resolving a conflict. General strategies are resorted to last, since they are computationally expensive and may lead to poor solutions.

Figure 4 outlines the basic steps in the conflict-resolution phase. Agents involved in the conflict situation start negotiating over the proposed solution. If an agent does not have a resolution alternative, it uses its perspective on the issue to constraint the search space of other agents. If it has the ability to counter-propose a resolution, then it tries its domain-dependent strategies first. Agents that detect a conflict from the same issue-perspective might use relaxation techniques so that an acceptable resolution could be generated even if the resolution alternatives they are proposing cannot be agreed upon.

If an agent detects a conflict and then choose a strategy to resolve the conflict, this does not mean that the agent may not alter the resolution strategy it has chosen. That is, upon observing the actions of other agents during the conflict-resolution phase, the agent may improve its understanding of the overall problem and the particular conflict encountered. This feature allows agents to alter strategies that they think will benefit from a change. When an agent proposes a revised solution based on its resolution scheme, it also explains why the new solution is a good candidate. This enables other agents involved in the conflict to choose the most appropriate action in the resolution process.

5. An engineering design example

The following example is taken from the domain of office design to exemplify the problem-solving process used by cooperating agents in our implementation. The reason for choosing this example is that it is in a concrete, rather than an abstract, domain and that it can be understood easily because of its suitability for simple, two-dimensional graphical representation. Here, we present a simplified layout problem for an office design, and we describe design agents and their interactions. A good office design draws from different areas of expertise such as aesthetics, functionality, energy efficiency, etc. In this example we have incorporated four agents in the problem-solving framework. They are the client agent, the functionality agent, the electricity agent, and the cost agent.

The client agent is the one that puts forth the problem definition which specifies general constraints and the global design goal to be satisfied. He may be the one to use the office being designed or he may be the department chairman who is having the office designed for a prospective faculty member. The functionality agent uses specific heuristic-search techniques in the area of space planning. The electricity agent is concerned with all of the electrical and electronic devices and wiring, including computers, telephones, facsimile systems, etc. The cost agent is required to control the overall cost of the design and to avoid the wasteful use of resources.

The client agent defines the problem and asserts the problem instance tuple into the problem area of the shared medium. The agents form their task sets and start producing design proposals by taking the partial order into account. Suppose that at a particular time, the electricity agent takes the next task in its task queue, which says to "put lamps into the slots in the ceiling". The electricity agent further decomposes this task into the following subtask:

```
task16 = ( (fix (:object lamp   :type lamp-type1)
                (:object layout :component light-plug1))
           (fix (:object lamp   :type lamp-type1)
                (:object layout :component light-plug2))
           (fix (:object lamp   :type lamp-type1)
                (:object layout :component light-plug3)) )
```

The electricity agent generates a solution for this task in which it proposes to fix three 100-W lamps into the plugs in the ceiling as follows:

```
< electricity-agent, q-4-16,
  ( (assert :relation fixed (:object lamp :instance lamp1)
                            (:object layout :component light-plug1))
    (assert :relation fixed (:object lamp :instance lamp2)
                            (:object layout :component light-plug2))
    (assert :relation fixed (:object lamp :instance lamp3)
                            (:object layout :component light-plug3)) )
  ( (have better-lighting) )
  good >
```

After examining the proposal, the cost and functionality agents detect conflicts and assert their evaluation results into the conflict area of the shared medium. The cost agent says that having a total of 300 watts of lighting is too costly, detecting a conflict from its *cost* issue perspective. The functionality agent evaluates the proposal and detects a conflict from the *effective-usage* issue perspective. The evaluation result tuples to be asserted are:

```
< cost-agent, q-4-16, (:all-actions : issues (cost))
  (not-acceptable), (conflicting-proposal) >

< functionality-agent, q-4-16, (:all-actions :issues
    (effective-usage))
  (not-acceptable), (conflicting-proposal) >

< client-agent, q-4-16, (:all-actions :issues (:all)),
  (acceptable), (nonconflicting-proposal) >
```

The electricity agent, cost agent and functionality agents come together to resolve the conflict. The electricity agent finds plenty of options to meet the *cost* concern of the cost agent by looking at its database and offering a total lighting package that can be accepted by the cost agent. However, the functionality agent, from its *effective-usage* perspective, has a good domain-dependent resolution alternative for the conflict, which says that the level of lighting accepted by both of the agents can be lowered further. Since the occupant of the office will be working alone most of the time, it is possible to lower the total lighting power consumption by introducing a desk lamp that might be put on top of the desk. After several iterations of these issue dimensions, the following solution is agreed upon by all of the agents:

```
( (assert :relation fixed (:object lamp :instance lamp6)
                          (:object layout :component light-plug1))
  (assert :relation fixed (:object lamp :instance lamp7)
                          (:object layout :component light-plug2))
  (assert :relation fixed (:object lamp :instance lamp8)
                          (:object layout :component light-plug3))
  (assert :relation on    (:object desk-lamp :instance desk-lamp5)
                          (:object desk :instance desk1)) )
```

In the resolution phase, the cost agent constrains the search space so that the total amount of lighting power should be less than 220 watts. The functionality agent uses a domain-dependent resolution alternative to augment a final solution that is acceptable to all of the agents. The design proceeds in this manner until it reaches the requirements specified by the client agent.

6. Conclusions

Distributed artificial intelligence has an important role in the field of artificial intelligence because many problems encountered in real life require the application of complex and diverse expertise. One of the important problems faced in a cooperating community of agents is how to detect and resolve conflicts that occur at any phase of problem solving. In this paper, a new cooperative problem-solving environment is described that openly supports multi-agent conflict detection and resolution. In this environment, each agent is free to choose the most appropriate action, given its understanding of the global and local situation and its own capabilities. Each agent has its own conflict-resolution knowledge which is not accessible or known by others. Furthermore, there are no globally-known conflict-resolution strategies. Each agent involved in a conflict chooses a resolution scheme according to its self interest. Agents might use different strategies of their own and might still agree on a solution.

The system achieves problem-solving flexibility, which is the most compelling argument for building modular multi-agent systems. A new agent can be added or an existing one can be removed without any modification of the rest of the system. This

characteristic of the system satisfies the requirements of open systems semantics. However, in the existing approaches, the addition or removal of an agent requires that global conflict-resolution knowledge be reformed accordingly. Currently, the system runs on a network of SUN-4 workstations. All of the problem solvers, agents, are modeled as processes running on different workstations that communicate over a network. The problem-solving environment will further be modified to support human participants as agents.

References

[1] M.R. Adler, A.B. Davis, R. Weihmayer, and R.W. Worrest, "Conflict Resolution Strategies for Non-hierarchical Distributed Agents", Distributed Artificial Intelligence, Vol. 2, M.N. Huhns, 1989, pp. 139–161.

[2] B. Chandrasekeran, "Decomposition of Domain Knowledge into Knowledge Source: The MDX Approach", Fourth National Conference of Canadian Society for Computational Studies of Intelligence, Canada,

[3] B. Chaib-Draa *et al.*, "Trends in Distributed Artificial Intelligence", Artificial Intelligence Review, No. 6, 1992, pp. 35–66.

[4] H.D. Durfee, V.R. Lesser, and D.D Corkill, "Trends in Cooperative Distributed Problem Solving", IEEE Transaction on Knowledge and Data Engineering, Vol. 1, No. 1, March 1989, pp. 63–83.

[5] L.D. Erman, F. Hayes-Roth, V.R. Lesser, and D.R. Reddy, "The Hearsay-II Speech Understanding Systems: Integrating Knowledge to Resolve Uncertainty", Computing Surveys, Vol. 12, June 1980, pp. 213–253.

[6] L. Gasser, "Social Conceptions of Knowledge and Actions: DAI Foundations and Open Systems Semantics", Artificial Intelligence, Vol. 47, 1991, pp. 107–138.

[7] J.S. Gero (ed), Artificial Intelligence in Engineering Design, Elsevier, UK, 1988.

[8] A. Gingberg, "Knowledge Base Reduction: a New Aproach to Checking Knowledge Bases for Inconsistency and Redundancy", AAAI, 1988, pp. 4–8.

[9] V. Goel and P. Pirolli, "Design within Information-Processing Theory: The Design Problem Space", AI Magazine, Spring 1989, pp. 19–36.

[10] F. Gomez and B. Chandrasekaran, "Knowledge Organization and Distribution for Medical Diagnosis", Technical Report, 84-FG-FGBC, Department of Computer and Information Science, The Ohio State University, 1984.

[11] C. Hewitt, "Offices are Open Systems", ACM Transactions on Office Information Systems, Vol. 4, No. 3, 1986, pp. 271–287.

[12] C. Hewitt, "Open Information Systems Semantics for Distributed Artificial Intelligence", Artificial Intelligence, Vol. 47, 1991, pp. 79–106.

[13] M.N. Huhns and D.M. Bridgeland, "Multi-Agent Truth Maintenance", IEEE Transactions on Systems, Man and Cybernetics, Vol. 21, No. 6, Nov/Dec 1991, pp. 1437–1445.

[14] M. Klein and Lu S. C-Y., "Conflict Resolution in Cooperative Design", Artificial Intelligence in Engineering, Vol. 4, No. 4, 1989, pp. 168–180.

[15] S. Lander and V.R. Lesser, "Conflict Resolution Strategies for Cooperating Expert Agents", International Conference on Cooperating Knowledge-Based Systems, Keele University, October 1990.

[16] V.R. Lesser and D.D. Corkill, "The Distributed Vehicle Monitoring Testbed: a Tool for Investigating Distributed Problem Solving Networks", Blackboard Systems, Edited by R.S. Engelmore and A. Morgan, Addison-Wesley, 1988, pp. 353–386.

[17] T.W. Malone and K. Crowston, "Towards an Interdisciplinary Theory of Coordination", Technical Report, CCS-TR-120, Center for Coordination Science, MIT, Cambridge, 1991.

[18] T.A. Nguyen *et al.*, "Verifying Consistency of Production Systems", Proceedings of the Third Conference on Artificial Intelligence Applications, 1987, pp. 4–8.

[19] J.Y.C. Pan and J.M. Tenenbaum, "An Intelligent Agent Frameworks for Enterprise Integration", IEEE Transactions on Systems, Man, and Cybernetics, Vol. 21, No. 6, Nov/Dec 1991, pp. 1391–1408.

[20] F. Polat and H.A. Guvenir, "Coordination Issues in Distributed Problem Solving", Proceedings of the Sixth International Symposium on Computer and Information Science, Elsevier, Vol. 1, Antalya, 1991, pp. 585–594.

[21] F. Polat and H.A. Guvenir, "UVT: A Unification Based Tool for Knowledge Base Verification", IEEE Expert, Vol. 8, No. 3, June 1993, pp. 69–75.

[22] F. Polat and H.A. Guvenir, "A Conflict Resolution Based Cooperative Distributed Problem Solving Model", Proceedings of AAAI92 Workshop on Cooperation among Heterogeneous Intelligent Agents, San Jose, California, July 1992, pp. 106–116.

[23] S. Shekhar and C.V. Ramamoorthy, "Coop: A Self-Assessment Based Approach to Cooperating Expert Systems", International Journal on Artificial Intelligence Tools, Vol. 1, No. 2, 1992, pp. 175–204.

[24] R.G. Smith and R. Davis, "Frameworks for Cooperation in Distributed Problem Solving", Readings in Distributed Artificial Intelligence, Edited by A.H. Bond and L. Gasser, Morgan Kaufmann, 1988, pp. 61–70.

[25] T. Smithers and W. Troxell, "Design is Intelligent Behavior But What's the Formalism", AI EDAM, Vol. 4, No. 2, 1990, pp. 89–98.

[26] D. Sriram and R. Adey, Applications of Artificial Intelligence in Engineering Problems, Vols 1–2, Springer-Verlag, 1986.

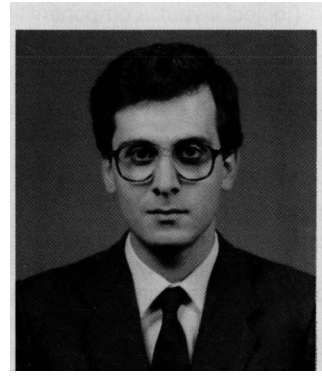
[27] K.P. Sycara, "Negotiation in Design", Proceedings of the MIT-JSME Workshop on Cooperative Product Development, MIT, Cambridge, Massachusetts, 1989.

[28] K. Sycara, S. Roth, N. Sadeh, and M. Fox, "Distributed Constrained Heuristic Search", IEEE Transactions on Systems, Man, and Cybernetics, Vol. 21, No. 6, Nov/Dec 1991, pp. 1446–1460.

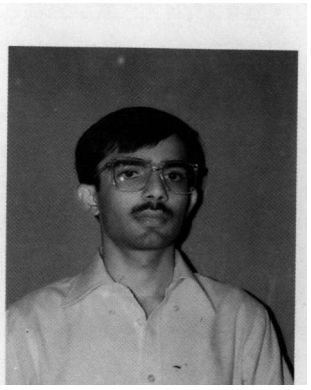
[29] M. Tokoro and Y. Ishkawa, "An Object-Oriented Approach to Knowledge Systems", Proceedings of the International Conference on Fifth Generation Computer Systems, 1984, pp. 623–631.

[30] K. Werkman *et al.*, "Design and Fabrication Problem Solving Through Cooperative Agents", NSF-ERC-ATLSS Technical Report 90–05, Lehigh University, Bethlehem, 1990.

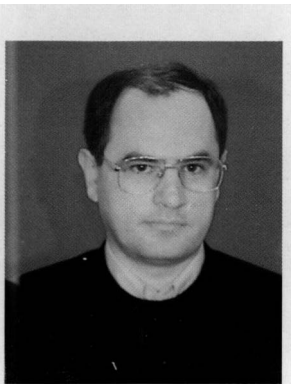
[31] P.H. Winston, Artificial Intelligence, Addison-Wesley, 1984.



Faruk Polat is a Ph.D. candidate in computer engineering and information science at Bilkent University in Ankara, Turkey. He has been working on his dissertation as a visiting NATO science scholar at University of Minnesota, Minneapolis, for the 1992–1993 academic year. His research interests include knowledge-based systems, knowledge base verification, and distributed artificial intelligence. He received his B.S. in computer engineering from the Middle East Technical University, Ankara, in 1987 and his M.S. degree in computer engineering and information science from Bilkent University in 1989.



Shashi Shekhar received the B.Tech. degree in Computer Science from the Indian Institute of Technology, Kanpur, India, in 1985, the M.S. degree in Business Administration, and the Ph.D. degree in Computer Science from the University of California, Berkeley, in 1989. He is currently an Assistant Professor in the Department of Computer Science of the University of Minnesota, Minneapolis. His research interests include databases, artificial intelligence, and software engineering with emphasis on important applications such as manufacturing. Dr Shekhar is a member of the IEEE Computer Society, ACM, and AAAI.



H. Altay Guvenir is an assistant professor of computer engineering and information science at Bilkent University in Ankara, Turkey. His research interests include expert systems, machine learning, and computational linguistics. He received his M.S. in electrical engineering from Istanbul Technical University in 1981 and his Ph.D. in Computer Science from Case Western Reserve University in 1987. He is a member of AAAI, ACM, ACM SIGART, and the International Association of Knowledge Engineers.