MULTI-AGENT COORDINATION AND COOPERATION IN A DISTRIBUTED DYNAMIC ENVIRONMENT WITH LIMITED RESOURCES

Nicholas V. Findler
Department of Computer Science and Artificial Intelligence Laboratory,
Arizona State University, Tempe, AZ 85287
and
Gregory D. Elder
Instructional Technology Division, U.S. Air Force Academy CO 80840

ABSTRACT

Coordination and cooperation are two major concerns in Distributed Artificial Intelligence (DAI) systems. How can a group of geographically distributed agents properly allocate a set of tasks among themselves while satisfying different types of constraints? Also, in an environment of limited resources, how can agents resolve resource conflicts and effectively accomplish tasks? We have examined these two problems and have developed new techniques to promote multi-agent coordination and cooperation. A novel method of negotiation allows agents to bid for tasks based upon the agents' capabilities. Further, the use of a threshold value ensures that only the best agents for a task become task commanders and renders some tasks to be renegotiated as agents improve their bids. To resolve resource conflicts, a technique called Hierarchical Iterative Conflict Resolution has been implemented. This allows conflicts to be resolved in an iterative manner, based upon a hierarchy of task priorities. Agents with higher priority tasks have the authority to borrow resources from agents with lower priority tasks. It ensures that higher priority tasks will be solved before those of lower priority. These new techniques were employed in a DAI testbed which simulates an air war environment.

KEYWORDS: Distributed Artificial Intelligence, multi-agent behavior, cooperative problem solving.

1. INTRODUCTION

Distributed Artificial Intelligence (DAI) involves collections of distributed computers (intelligent decision making agents) sharing information and resources in order to solve a common set of tasks. Such a distributed approach offers certain advantages for problem solving: faster response, increased flexibility, robustness, resource sharing, graceful degradation, and better adaptability.

Some of the research involving DAI have included distributed vehicle monitoring [Durfee, Lesser, & Corkill, 1987]; distributed air traffic control [Findler & Lo, 1986; Findler & Lo, 1991; Findler & Lo, 1993a & 1993b; Steeb et al., 1981]; distributed manufacturing control [Findler & Gao, 1987; Findler & Ge, 1989; Findler & Ge, 1993; Parunak, 1987; Parunak, 1990]; distributed control of street traffic lights [Findler, 1991; Findler & Stapp, 1992; Findler, 1993]; distributed problem solving for knowledge based vision [Decker, Durfee, & Lesser, 1989]; multi-agent motion planning [Fraichard & Demazeau, 1990]; and distributed control of moving resources to moving tasks [Findler et al., 1990; Findler & Sengupta, 1994].

A major concern with DAI systems is multi-agent coordination and cooperation. In what manner can geographically distributed agents behave so as to promote cooperative problem solving? Given a set of tasks, how can the agents assign among themselves the task responsibilities? Further, in a dynamic environment with limited resources, how can the agents share these resources to satisfy tasks effectively and minimize the impact of resource conflicts? Techniques for multi-agent coordination and cooperation have ranged from centralized approaches where a single agent assigns tasks to other agents, to negotiation strategies, such as the contract net [Smith 1980], the coordinator-coworker scheme [Findler & Lo, 1991, 1993a, 1993b] and Sycara's [1989] persuasive argumentation. Our research builds upon the idea of agent negotiation to enhance cooperative behavior. It allows agents as a group to determine task assignments. In addition, our research provides a new technique for resolving resource conflicts. This technique, Hierarchical Iterative Conflict Resolution, allows agents to resolve their conflicts in an iterative manner based upon a hierarchy of task priorities. These novel ideas were implemented in a DAI testbed for examining multi-agent behavior within an air war environment.

2. THE DISTRIBUTED AIR WAR TESTBED

It may be appropriate to introduce the characteristic feature of the Distributed Air War testbed first. It is a flexible computing environment for conducting studies related to the use of distributed intelligent agents for solving resource allocation problems within an air war environment. The testbed is fully developed and has been used to obtain the qualitative and quantitative results that prove the feasibility of our approach to dynamic resource and task allocation. As shown in Figure 1, the testbed consists of three main modules: the Battle Simulation Module, the User Interface Module, and the Distributed Problem Solving Module. The Battle Simulation Module simulates key aspects of an air war. The User Interface Module presents a graphical representation of the simulated air war environment to the user and allows the user to easily interact with the testbed. Finally, the Distributed Problem Solving Module provides a system of distributed agents to solve problems presented in the testbed. The Distributed Air War testbed was developed on a SUN workstation with the UNIX operating system. When running the testbed, the Battle Simulation Module and User Interface Module reside on one workstation while the Distributed Problem Solving Module resides on a series of networked SUNs. The Battle Simulation Module is written in C. The User Interface Module uses the X-Windows library, and the Motif toolkit for providing a mouse-driven, graphical interface. Lastly, the Distributed Problem Solving Module uses CLIPS (C Language Integrated Production System), an expert system development tool.

2.1. The Battle Simulation Module

The method of simulating air wars in the testbed is based upon the manner in which air war simulations are conducted at the United States Air Force Academy for the course on Air Power Doctrine and Strategy [United States Air Force Academy, 1990]. The Battle Simulation Module simulates the salient features of an air war. These features consist of the following:

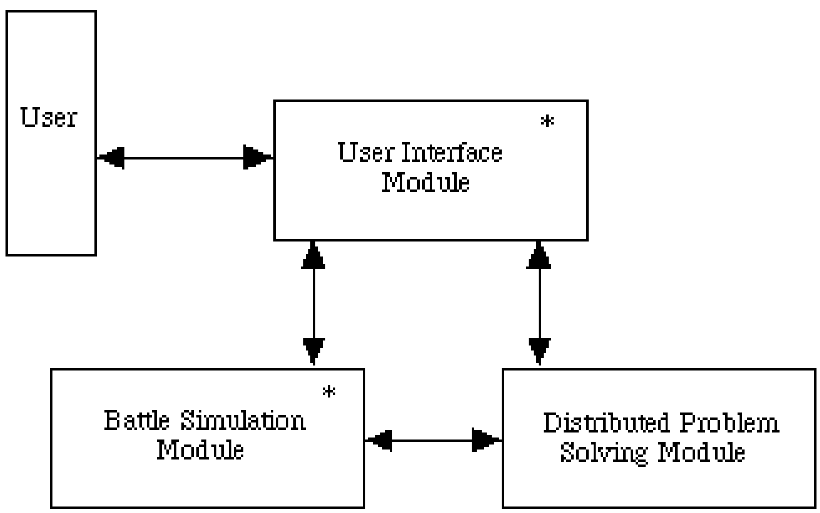

Figure 1 — Architectural view of the testbed. The modules marked with an asterisk reside on the same processor

• a map representing an area of conflict in which locations can be identified using some type of coordinate system;

• two forces which oppose one another in an air war;

• various types of fixed, land-based resources which may be viewed as potential targets by each force (command centers, airfields, munition depots, power centers, transportation centers, etc.);
• various types of air resources with differing capabilities, normally found in an air war environment (fighter aircraft, bombers, etc.);
• phases of aircraft operations to include maintenance, take-off, movement between waypoints (map coordinates or other reference points), combat, and landing;
• the fog and friction (unknown and unexpected events) associated with warfare; and
• ground-based air defense systems.

2.2. User Interface Module

The User Interface Module presents the user with a graphical display representing the simulated air war environment. Figure 2 shows a portion of an air war map as displayed to the user via the User Interface Module. The interface is mouse-driven and uses pull-down menus for ease of use. The User Interface Module provides the following functions to the user:

• a visual display of the air war map;
• specification of resources to be used in an air war simulation;
• deployment of resources to their initial map positions for the start of an air war simulation;
• simulations of air wars;
• the capability to interrogate the graphical display to obtain status information about resources.

2.3. Distributed Problem Solving Module

The Distributed Problem Solving Module implements the DAI methodology described in Section 2. It is an independent program, separate from the rest of the Distributed Air War testbed. The program is named *Agent*, as it represents an intelligent problem solving agent. It is run on each processor representing an agent for solving problems during air war simulations. The overall architecture of the Distributed Problem Solving Module is shown in Figure 3.

The following major components comprise the Distributed Problem Solving Module:

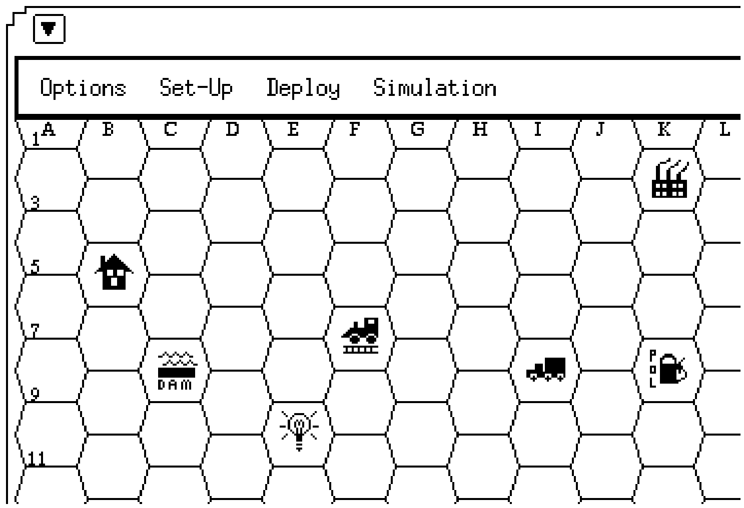

Figure 2 — A portion of the air war map displayed by the User Interface Module

- Communications Unit — Communicates with other problem solving agents.
- Message Handler — Handles incoming messages from other agents and prepares messages for transmission to other agents.
- CLIPS Knowledge and Rule Base — Maintains the knowledge base containing information about the state of the world and the rule base for use in decision making.
- Negotiation Unit — Negotiates the distribution and assignment of tasks with the other agents.
- Task Evaluator — Evaluates the agent's capability to handle a particular task; i.e., computes quality measures.
- Resource Allocator — Allocates aircraft squadron resources for task accomplishment.
- Flight Planner — Constructs flight plans for aircraft allocated by the Resource Allocator. Flight plans are sequences of waypoints for aircraft to follow in accomplishing missions.
- Solution Pad — Holds the solutions for each task assigned to this agent.

2.4. Testbed Operation

Figure 4 presents an architectural view of the testbed with multiple agent processes. During testbed operation, *Agent0* acts as a go-between for the other problem solving agents and the rest of the testbed. Once an air war scenario has been constructed, the user may input a set of tasks to the system. The agents will negotiate for task assignment, solve their tasks, and resolve any resource conflicts using the Hierarchical Conflict Resolution technique. The resulting task solutions are passed to the Battle Simulation Module for simulation. Simulated air wars may be conducted in the testbed between two opposing forces. One force may be controlled by a human decision maker, while the other force is under the control of the Distributed Problem Solving Module.

3. PROMOTING MULTI-AGENT COORDINATION AND COOPERATION

3.1. Background

In all but the simplest of cases, a problem may be looked upon as a set of tasks to be completed,

$$(T_1, T_2, T_3, ..., T_n)$$

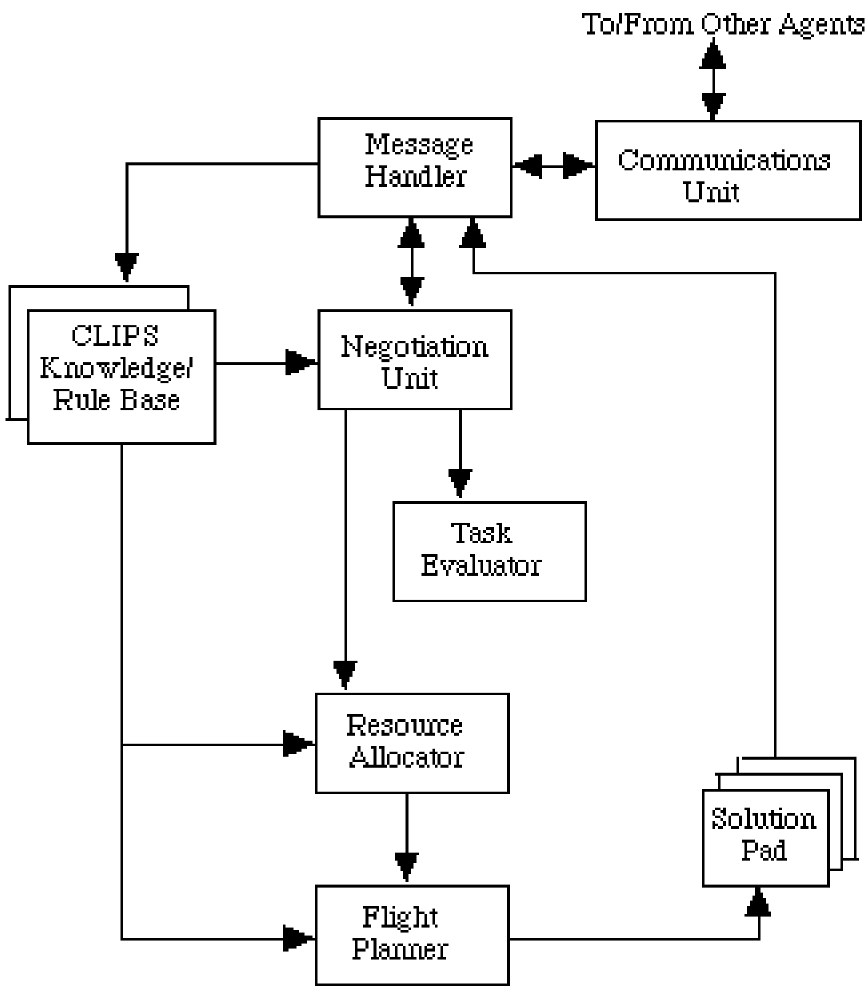

Figure 3 — Overall architecture of the distributed problem solving module

A single problem solver can develop an over-all solution by solving each individual task. The tasks may have certain constraints in regard to time, location, resource type required, etc. This fact may negate the approach of simply solving each task one at a time, in any convenient order. Accordingly, some tasks may require a temporal ordering, e.g., tasks T_3 and T_5 must be completed within the same time frame. Other tasks may have a constraint based upon a relationship with other tasks. For example, T_5 cannot be solved until T_4 is solved because a solution to T_5 depends upon the solution to T_4 Furthermore, a single problem solver may itself be constrained, in that it does not possess the required resources to solve all tasks while also satisfying the constraints.

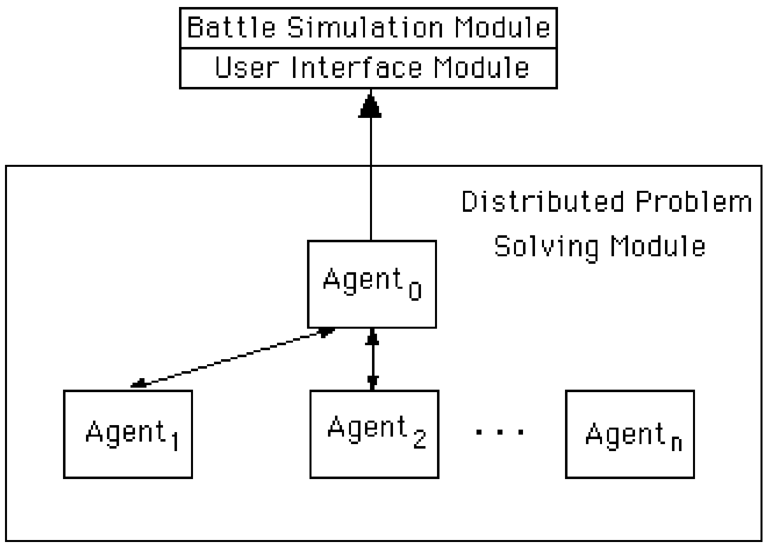

Figure 4 — Architectural view of the testbed with agent processes

In order to facilitate the solving of multiple tasks, multiple problem solving agents are to be employed,

$$(A_1,\ A_2,\ A_3,\ \ldots,\ A_m)$$

Each agent has at its disposal, a limited number of resources to use in completing tasks as, for example,

$$\begin{array}{ll}
A_1\text{'s resources:} & (R_{11},\ R_{12},\ R_{13},\ \ldots,\ R_{1n}) \\
A_2\text{'s resources:} & (R_{21},\ R_{22},\ R_{23},\ \ldots,\ R_{2o}) \\
A_3\text{'s resources:} & (R_{31},\ R_{32},\ R_{33},\ \ldots,\ R_{3p}) \\
 & \ \ \vdots \quad\ \ \vdots \quad\ \ \vdots \\
A_m\text{'s resources:} & (R_{m1},\ R_{m2},\ R_{m3},\ \ldots,\ R_{mq})
\end{array}$$

These resources may also be constrained based upon their functionality and availability. For example, some resources may be used only for specific tasks, while other, more general resources, can be used for any task. In addition, certain resources may be unavailable at certain times (undergoing maintenance, currently accomplishing another task, etc.).

Ideally, there would be unlimited agents and resources with each task mapped to a single agent. Each agent would have the capability and resources necessary for completing the task. Figure 5 illustrates this idea of one task per agent.

The ideal situation and the real world do not usually coincide: there are more tasks than agents. Moreover, individual agents may not possess enough resources or of the necessary types to complete a task. In this situation, agents must share resources with one another in order to accomplish the totality of tasks. Figure 6 represents this type of situation. Here there is one more task than the number of agents. Furthermore, the number and types of resources "owned" by each agent is different. Thus, agent A_2 is in charge of two tasks, and agent A_n must "borrow" resources which it does not have.

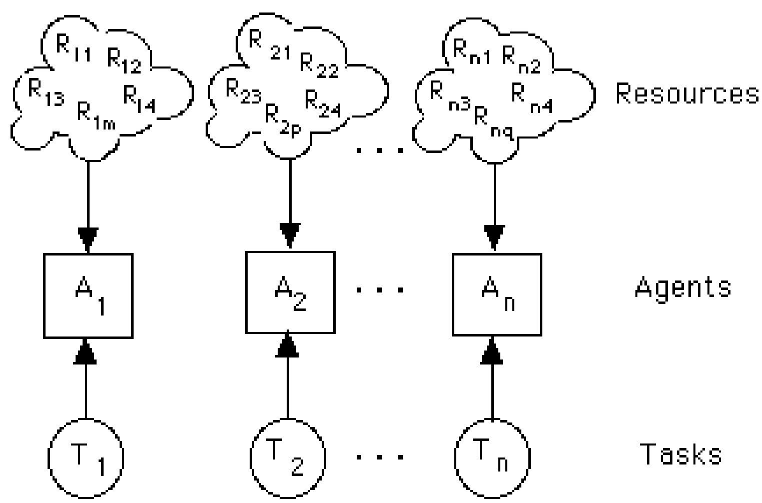

Figure 5 — One-to-one mapping of tasks to agents

The sharing of tasks and resources among a number of problem solving agents can provide a number of benefits:

• More tasks can be solved in a given time period using multiple agents than by using individual, isolated agents. A corollary of this is: a collection of tasks can generally be solved faster and at a lesser cost with multiple agents than with individual, isolated agents.

• Graceful degradation can be achieved using multiple agents; i.e., if one agent fails, the entire problem solving system does not stop. It continues to operate, albeit in a possibly degraded mode (more slowly and at a reduced level of quality). In a single agent system, the failure of the one agent can stop the entire problem solving process.

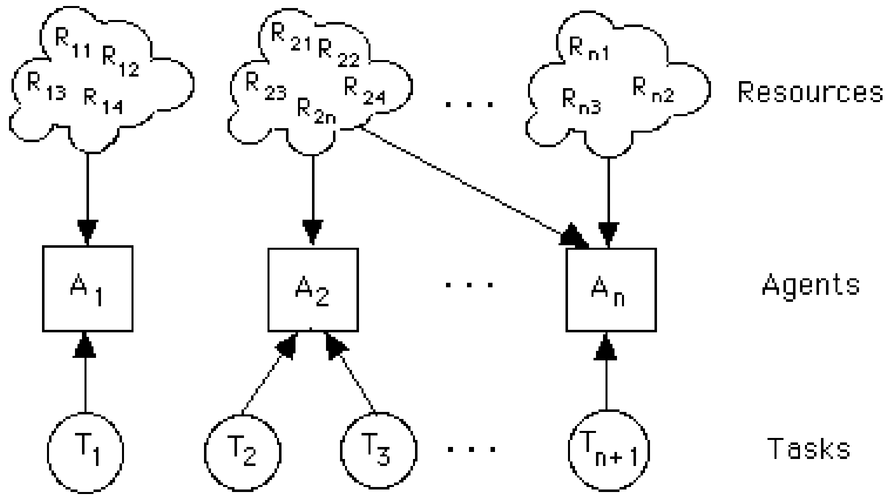

Figure 6 — Multiple tasks and resource sharing

• With multiple agents, task solutions may be more flexible and robust than single agent solutions.
• Having agents specialized in certain problem sub-domains, the quality of the overall solution should be better.

The sharing of tasks and resources is a complicated problem. Agents must distribute the tasks among themselves and determine which agents are responsible for which tasks at what point of time. As discussed before, various approaches to task distribution have been suggested and implemented. In some cases, a controlling or master agent may simply assign tasks to other agents. Other methods involve negotiation between the agents to determine an equitable assignment of tasks.

Once tasks have been distributed and the agents begin solving their particular tasks, conflicts may arise with respect to resource sharing. These conflicts can occur since each agent has a limited number of resources, and some resources have limited functionality. Conflicts will arise within a time window for task accomplishment. If the urgency of task accomplishment is high, and since concurrent usage of resources is not possible, conflicts develop. Two major types of conflicts can thus be present:

- A total of *n* resources are required to complete all tasks; however, less than *n* resources exist among all agents.
- Two agents require the use of a particular type of resource but only one resource of this kind is available.

These conflicts must be resolved in order to achieve a satisfactory solution to the tasks at hand.

3.2. Distribution of Tasks

A form of negotiation is used among the agents to distribute tasks. Given a set of tasks, each agent evaluates its ability and availability to accomplish each task in the set. In the usual negotiation strategies, this results in each agent providing a bid for each task based upon its estimated level of accomplishment, and the agent with the best bid is assigned the respective task. However, this does not take into account the agents' ability to improve their bids subsequently, using the knowledge of other agents' best bids for each task. For example, suppose an agent, A_1, has bids for tasks T_1, T_2, and T_3. Based upon its resources, A_1 has low bids for T_1 and T_2, and a high bid for T_3. (The lower the bid the better the prospect of being assigned the task.) Another agent, A_2, also has bids for T_1 and T_2, while a third agent, A_3, has a bid for T_3. As determined by the bids, A_1 has the best bid for T_1, A_2 the best for T_2, and A_3 the best for T_3. However, if A_1 had known that it would not have the best bid for T_2, it could have provided a better bid for T_3. This is because resources which it thought would be applied to T_2, could now be applied to T_3, and thus improve the quality of accomplishing T_3 (cost, timing, success rate, etc.). This negotiation approach is used in this research to allow agents to modify bids.

3.2.1. Task Evaluation

Agents in many domains are primarily concerned with solving those tasks that are geographically closest to them. In such cases, each agent first sorts the task list according to distance, i.e., the first task to consider will be the closest, the second task to consider is the one next in distance, and so forth. Using this sorted list, each agent then evaluates its ability to solve the task. The following function is used for the evaluation:

$$Q_{ij} = C_1\tau_i + C_2\pi_i + C_3\sigma_i + C_4\omega_j$$

where Q_{ij} is the quality measure of task i by agent j, τ_i is the timeliness for completing the task i, π_i is the cost of primary resources to be used, σ_i is the cost of support resources, and ω_j is the current workload of the agent. C_1, C_2, C_3, and C_4 are weighting factors. Each term in this function is calculated as follows:

• Timeliness is calculated as τ = D/S, where D is distance to the task and S is the speed of the slowest resource to be used for task accomplishment;

• The primary resource cost factor[1] will be an integer value within the range [0-2]. Resources have various capabilities. Some support specific types of tasks only. Other resources have general, multi-purpose roles. If an agent has resources available which specifically support the mission associated with the task, then the cost value is 0. If the agent can only use general purpose resources for this task, then the cost value is 1. If the agent does not possess any resources for this mission (and will have to borrow from other agents), the cost is 2.

• The support resources factor is calculated in a similar manner to primary resources. An integer value in the range [0-2] is used for support cost. Support resources are those which support the primary mission of a task. For example, in the air war environment fighter aircraft have a support role (bomber escort) for strategic bombing missions.

• The workload factor is simply the number of tasks which are currently assigned to the agent.

3.2.2. Task Assignment

The basic technique for task assignment involves a negotiation strategy. Agents evaluate their ability to accomplish each task (calculating quality measures) and then exchange these with one another. Agents whose *Q*s meet a specific criterion as the "best *Q*", become self-appointed Commanders for those respective tasks. Tasks for which no Commanders are assigned are renegotiated. (This procedure is heuristic in nature. A different "best bid" could be offered later by another agent that no longer submits bids for all tasks that it bid before. However, considering such possibilities would make the whole task assignment process prohibitively long in complex, real-life situations.) Using knowledge about which tasks were assigned, agents will recalculate *Q*s for the remaining tasks in order to improve their "bids". This process repeats until all tasks have Commanders. Since the agents do not simply use the best initial bids to assign all tasks, a better distribution of tasks among the agents should result. The following describes the task assignment process in detail.

Each agent evaluates its ability to accomplish each task as stated in Section 3.2.1. Since the task list is sorted by distance, the agents give primary consideration to those tasks that are closest to them. Therefore, a task which

---

[1] This and the following components are currently considered as qualitative measures but they can be easily mapped onto a quantitative scale whenever a more refined distinction is needed.

is farthest away may have a large *Q* due to the distance involved and the fact that resources are being considered for use with closer tasks.

After the agents have completed the evaluation of tasks, they exchange the task quality measures by sending a message to all other agents. The message is packaged in a simple data structure which lists each task's ID and the *Q* for each task. Having each agent's *Q* for each task, all agents can now determine which are the best "bids." (Best bids being the smallest *Q*s.) An agent becomes a self-appointed Commander for a task if the following hold true:

1. It has the smallest *Q* for the task.
2. Its *Q* must be better (less than) a certain *threshold* value. In addition, the best *Q* must be less than the next best *Q* by a certain value. This value is referred to as the threshold value. The idea here is that *Q*s which are relatively close to one another in value may not discriminate which is definitely *the very* best because of possible shortcomings of the evaluation function and in the system's assessment of the environment.

Agents notify all other agents when they become self-appointed Commanders. Some tasks may not initially have Commanders because the best *Q* did not exceed the threshold requirements. In this case, all agents will re-evaluate their *Q*s for those tasks, using the knowledge of which tasks have already been assigned to agents and which of its own resources have already been committed to a task. An agent may now be able to give a better *Q* to a task because it "lost" a bid to another agent and this made additional resources available for another task. This process of re-evaluating tasks continues until all tasks have a Commander using the criteria listed above, or a level of quiescence is reached. Quiescence is reached if the agents can no longer improve their *Q*s for the remaining tasks. In this case, the Task Commanders will be those agents with the smallest *Q*s. If two or more agents have the smallest *Q* for a task, then the agent with the least workload becomes the Commander. Finally, if two or more agents have the smallest *Q* and the same workload, then the Commander becomes the agent with the smallest ID. (Each agent has a unique ID value. This ID value is the final tie-breaker to use in assigning Commanders, if all other factors are equal.)

At this point, all tasks will have a Commander. All agents will then set about solving their particular tasks. In our domain, this will involve allocating specific resources and determining the best path for the resources to use in reaching the target associated with a task.

Consider the following scenario involving two agents, A_1 and A_2. We also have a set of available tasks: (T_1, T_2, T_3, T_4) and a threshold value of 2. Exemplary initial *Q*s calculated by each agent are shown in Table 1.

	Agent 1	Agent 2
Task #1	8.0	11.0
Task #2	7.1	6.0
Task #3	9.0	10.2
Task #4	6.3	7.1

Table 1 — Initial Quality Measures

If tasks were assigned simply based upon the smallest *Q*, then A_1 would be assigned T_1, T_3, and T_4 while A_2 would be assigned T_2. However, using the technique outlined in this chapter, A_1 is assigned T_1 since it has the smallest *Q* (8.0) and its *Q* exceeds A_2's *Q* (11.0) by more than a set value. The other tasks are not assigned since none of the agent's *Q*s for these tasks meet the criteria for task assignment. Therefore, T_2, T_3, and T_4 are renegotiated. Using the knowledge that it lost the bid for T_1, A_2 can improve its bids for the remaining tasks. (Resources that would have been used for T_1 can now be used with the other tasks instead and, in turn, A_1 can no longer count on those resources that are expected to be lost in performing T_1.) After a second iteration, the agents compute the following *Q*s shown in Table 2.

	Agent 1	Agent 2
Task #2	7.2	4.1
Task #3	9.0	9.0
Task #4	6.0	3.0

Table 2 — Quality Measures after a Second Iteration

A_2 is now assigned T_2 and T_4. T_3 must be renegotiated. Assume that after the third iteration, the agents achieved quiescence, i.e., they could no longer improve their bids. In this case, A_1 would be assigned T_3 since it has the same *Q* as A_2 but the smallest workload (one task versus two tasks). The final assignment of tasks is now A_1 with T_1 and T_3, and A_2 with T_2 and T_4.

3.3. Hierarchical Iterative Conflict Resolution

To resolve conflicts in a distributed problem solving system, we can take an iterative approach. Each task has a priority level, and tasks with a higher priority should be solved before tasks with a lower priority. This technique also eliminates the time expense associated with a contract net for resolving conflicts. Agents attempt to solve the specific tasks that are assigned to them. Conflicts are resolved based upon task priority. If the agent with the highest priority task has a resource conflict, that agent may "borrow" resources from any other agent to eliminate the conflict. This, in turn, could create a conflict with a lower-level priority task. If this is the case, the conflict resolution favors the agent with the task that has the next highest priority. In essence, an agent which resolves a conflict is deemed to make the following statement to the agent from which a resource is being borrowed:

"If my taking this resource affects your solving your task, then you must replan your solution (re-allocate resources)."

With this approach, conflicts are resolved iteratively based upon priority. Since some tasks are more important than others, the agents responsible for the higher priority tasks have the right to borrow and allocate resources owned by other agents. Using this scheme, either all conflicts will be resolved, or lower priority tasks will remain unsolved due to resource constraints. The general high-level algorithm for iterative conflict resolution is shown below:

Distribute tasks to agents

```
Each agent constructs a solution plan for its task
WHILE tasks not marked DO
  Agent with next highest priority task resolves its conflicts
  Mark this task solved or unsolvable
  Notify other agents
OD
```

While an agent is resolving resource conflicts, the other agents should not be idle — otherwise, this would diminish the benefit of distributed agents. Instead, all agents should be generating alternative, tentative solutions for their specific tasks. If an agent with a higher priority task must allocate a resource which affects others' solution, a list of alternatives will then be available for the affected agents. This will reduce the time needed for the latter to eliminate problems which were introduced by having to lend some of its resources.

In the air war environment, the tasks to solve involve allocating air units to attack, intercept, and destroy enemy resources (aircraft and land-based targets). The problem solving agents exist at the Wing Commander level in the Air Force command hierarchy and are located at the airfields. Each agent has a limited number of aircraft at its disposal. Furthermore, each type of aircraft squadron has a certain functionality and may be used only for specific types of tasks.

To solve a task, an agent must allocate aircraft squadrons with the desired capability. For destroying land-based targets, bomber aircraft must be employed. For suppressing enemy air defense systems, Wild Weasel aircraft are needed. For intercepting enemy aircraft, fighters must be used. For long range missions, tanker aircraft are necessary. Of course, combinations of the above may be needed for complex missions. As an air war progresses, resources will be lost to combat. Also, at any given time, some resources will be unavailable due to maintenance activities. Therefore, individual agents will, at times, lack the resources needed to solve a given task. This in turn leads to resource sharing between agents, which can cause further resource conflicts.

Task priority in this system is composed of two factors: a statically assigned importance value and a dynamically changing urgency level. The importance value is assigned by the Air Commander at the time tasks are generated. Urgency level relates to time. The sooner a task must be accomplished, the more urgent the task. Likewise, a task which can be put off until a later time is less urgent. For example, the task of intercepting enemy aircraft which are entering friendly air space is, in general, more urgent than the task of attacking a fixed, land-based enemy target. Over time, the enemy aircraft will move farther into friendly territory and thus increase the likelihood of inflicting harm to friendly resources. On the other hand, the fixed, land-based target will not change over time and the task of attacking it could be delayed if necessary. Thus, priority (P) is the product of importance (I) and urgency (U) (see [Findler, 1990; Findler et al., 1990]).

$$P = I * U$$

Applying the general Hierarchical Iterative Conflict Resolution algorithm to this air war environment, results in the following revised algorithm:

1. The Air Commander distributes the problem to the agents. The problem consists of a list of tasks with respective attribute values. Each ground-based target will have a statically assigned importance value associated with it. (Thus, the importance value of a task will be the importance value of the target associated with the task.) Under some conditions, certain tasks could have an extremely high urgency value. Furthermore, it is possible for several tasks to have the same priority.

In these cases, preference will be given to the task with the highest urgency level (smallest deadline value).

2. Each agent evaluates its ability to accomplish each task as discussed in Section 3.2.1.
3. Agents exchange their quality measures for each task.
4. Agents become Task Commanders (assigned tasks) as described in Section 3.2.2.

When all tasks have a Commander, the agents exchange information regarding their resources. This information consists of the status of resources (available, being maintained, on a mission, or returning to base). With Task Commanders identified and resource information exchanged, agents can construct detailed solutions (plans) for their tasks.

5. At this point, each agent begins solving (planning for) its particular task. The agents attempt to find the best solution possible using their own resources. When a solution is constructed, the respective agent notifies all other agents that it has a tentative solution for its task.
6. If conflicts arise, they will be resolved according to task priority.

a. The Task Commander of the highest priority task may take resources from other agents, if needed. This may create conflicts in other agents' plans for tasks which are of lower priority.
b. If so, such agents will have to replan for the affected tasks. Conflicts may arise and be resolved in lower and lower levels in a similar manner, in this iterative plan refinement process.
c. This process is repeated in task priority order for all tasks.
d. Agents with lower priority tasks do not sit idle, waiting for higher priority tasks to be solved. Instead, they will be constructing alternative, tentative solutions for their tasks. When notified that plans for higher priority tasks have been completed, they may then finalize their solution from their list of alternatives.

7. Either all conflicts can be resolved and all tasks taken care of or, if there are not enough resources, some lower priority tasks will remain unaccomplished.

As can be seen by the description of this process, it is hierarchical since we are planning for tasks based upon a hierarchy of priorities. All agents may not be able to solve all given tasks; however, this method ensures that the more important tasks will be solved (planned for) before tasks of lesser importance.

In resolving conflicts, certain heuristics will be adhered to in order to minimize the impact of borrowing resources from other agents:

• Resources with specific and limited functionality will be allocated first.
• When conflicts occur, available resources will be borrowed from agents which have "solved" their tasks, if possible
• When conflicts occur, resources will be borrowed from agents having the task with the smallest priority, if possible.

The next three figures illustrate the process of Hierarchical Iterative Conflict Resolution. This scenario involves three tasks (T_1, T_2 and T_3) and three agents (A_1, A_2, and A_3). Each agent has a specific set of aircraft squadron resources. These are specified with a type designator, followed by an identifier. For example, B-1 is bomber squadron #1. The other type designators are:

F - Fighter   FB - Fighter-Bomber   W - Wild Weasel

Figure 7 shows the assignment of tasks, the priority of the tasks, and the current tentative solutions (plans). In this case, A_1 has allocated resources B-1 and F-1 for its task. A_2 has allocated B-2, F-2, and W-1. A_3, on the other hand, has allocated F-3 but is lacking another fighter resource which is needed to accomplish its task. This is designated in the figure by substituting a question mark for the resource ID, i.e., F-?.

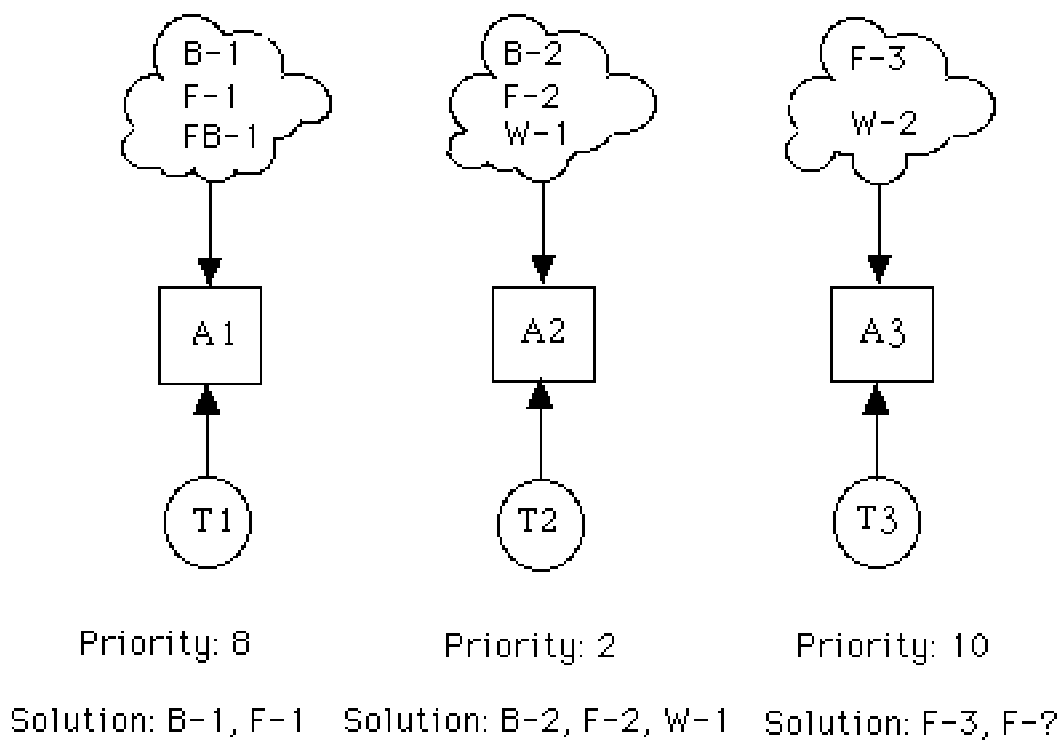

Figure 7 — Resource conflict with agent A_3 after the first iteration

After the second iteration of the conflict resolution technique, A_3 has borrowed resource F-2 from A_2. This is shown in the Figure 8 by the dashed arrow indicating that F-2 is being borrowed and utilized by A_3. Notice that A_3 has the task with the top priority (10), and that it has allocated a resource from the agent with the task having the least priority (2). Also, notice that A_3's allocation of F-2 has now caused a resource conflict in A_2's solution. A_2 must now attempt to borrow and utilize another resource in order to solve its task.

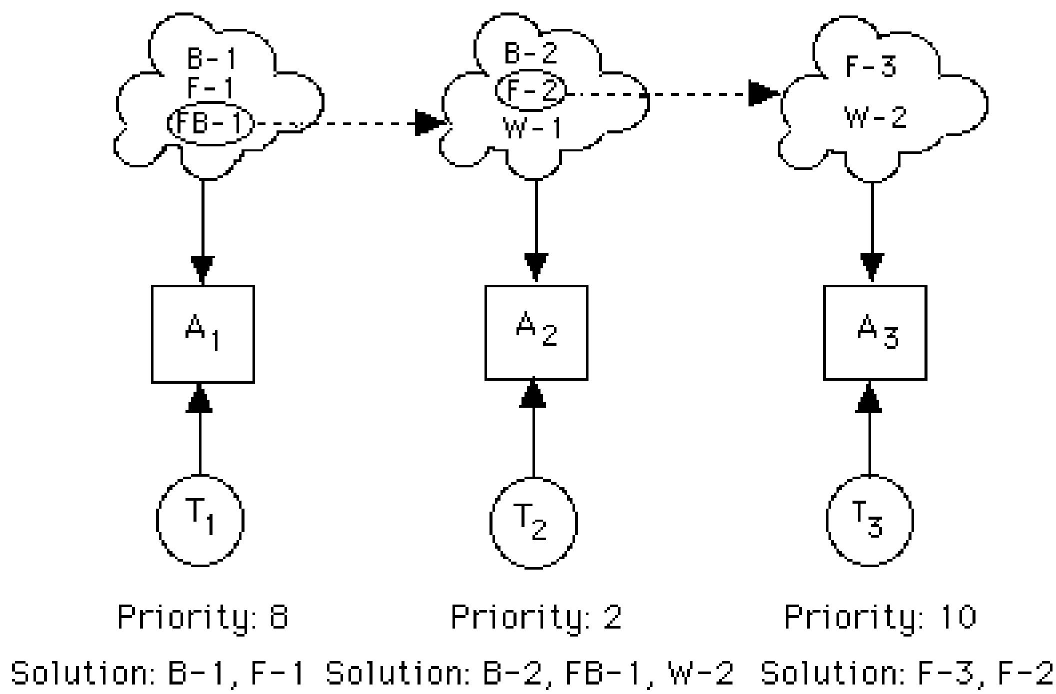

Figure 8— Resource conflict with agent A_2 after second iteration

Finally, Figure 9 shows that after the third iteration of the conflict resolution technique, all tasks have been solved. A_2 resolved its conflict by borrowing resource FB-1 from A_1 and allocating it. In this example, T_2 could use either a fighter or fighter-bomber. T_3 was constrained by the fact that only fighters could be used in its solution.

4. CONCLUSION

A major issue of concern in DAI involves coordination and cooperation of distributed problem solving agents. Given a set of tasks and a group of geographically distributed agents, how can the agents as a group assign tasks among themselves? Secondly, under conditions involving limited resources, how can the agents resolve resource conflicts in order to best accomplish a given set of tasks? These two general DAI problems provided the focus for this research.

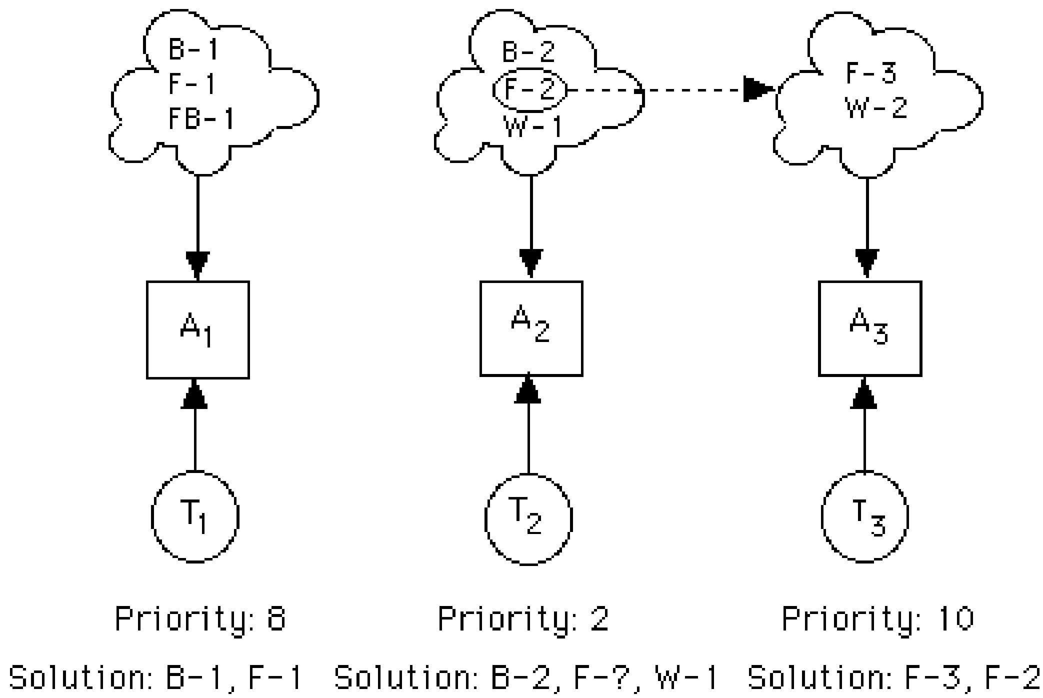

Figure 3.5 — All tasks solved after third iteration

To solve the first problem, a new negotiation technique was developed as an extension of the ideas of the contract net. Agents evaluate their abilities to accomplish tasks and then exchange “bids” with one another. Agents with the best bids exceeding a certain threshold are assigned those associated tasks. Tasks with no best bids are re-negotiated until bids are improved or a level of quiescence is reached. This method of negotiation allows the agents as a group to determine task assignments, rather than relying on one agent as a contractor to assign tasks.

Resource sharing occurs in a distributed system when one agent cannot satisfy (accomplish) an assigned task using only the resources under its jurisdiction. To solve such a task, the agent must borrow resources from other agents. A resource conflict can happen when more than one agent attempts to allocate the same limited resource for use during the same time frame. To solve this DAI problem, a technique called Hierarchical Iterative Conflict Resolution was developed. With this technique, agents with higher priority tasks (where priority is determined using an algorithmic method) may “take” resources belonging to agents with lower priority tasks. Borrowing resources in this manner may create additional conflicts but at lower priority levels. Conflicts are gradually resolved in this manner; thus, resource conflicts are resolved iteratively, based upon a hierarchy of task priorities. Either all tasks are solved, or if some tasks remain unsolved due to the lack of resources, the unsolved tasks will be those of lowest priority.

These two techniques have been implemented in a DAI testbed. The testbed simulates features of the air war domain which has the necessary characteristics of a distributed problem solving environment — geographically distributed agents (Wing Commanders), limited resources (aircraft squadrons), and distributed tasks (air missions).

It is also important to point out that many engineering problem environments may share some relevant characteristics with the environment used in our work. For example, electric power networks and telephone switching networks may may soon reach the *routine* stage when limited resources of different cost functions have to be allocated to the users on the basis of the urgency and importance of the tasks at hand. The process of negotiation necessary between the agents could well follow the paradigms developed in the present work.

REFERENCES

Applegate, C., Elsaesser, C. & Sanborn, J. (1990). An architecture for adversarial planning. *IEEE Transactions on Systems, Man, and Cybernetics*, **20**, 186-194.

Benoit, J. W., Elsaesser, C., MacMillan, T. R. & Tallis, H. C. (1990). *Planning for Conflicts in Multi-Agent Domains* (Technical Report No. MTR-90W00149). MITRE Corp.: Washington DC.

Cammarata, S., McArthur, D. & Steeb, R. (1983). Strategies of cooperation in distributed problem solving. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 767-770.

Corkill, D. D. & Lesser, V. R. (1983). The use of meta-level control for coordination in a distributed problem solving network. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 748-756.

Decker, K. S. (1987). Distributed problem-solving techniques: A survey. *IEEE Transactions on Systems, Man and Cybernetics*, **17**, 729-740.

Demazeau, Y. & Muller, J. P. (1990). Decentralized artificial intelligence. In Y. Demazeau & J. P. Muller (Eds.), *Decentralized AI: Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World*. 3-13. Elsevier Science: Amsterdam, The Netherlands.

Dunnigan, J. F. (1988). *How To Make War*. Quill: New York.

Durfee, E. H. & Lesser, V. R. (1988). Incremental planning to control time-constrained blackboard-based problem solver. *IEEE Transactions on Aerospace and Electronic Systems*, **24**, 647-662.

Findler, N. V. & Lo, R. (1986). An examination of distributed planning in the world of air traffic control. *Journal of Distributed and Parallel Processing*, **3**, 411-431.

Findler, N. V. & Gao, J. (1987). Dynamic hierarchical control for distributed problem solving. *Data and Knowledge Engineering*, **2**, 285-301.

Findler, N. V. (1989). Distributed knowledge-based systems in manufacturing. In G. Rzevski: *Artificial Intelligence in Manufacturing*. 17-33. Springer-Verlag: New York, Berlin, Heidelberg.

Findler, N. V. & Ge, Q. (1989). Perceiving and planning before acting — an approach to enhance global network coherence. *International Journal of Intelligent Systems*, **4**, 459-470.

Findler, N. V. (1990). *Contributions to a Computer-Based Theory of Strategies*. Chapter 6. Springer-Verlag: New York, Berlin, Heidelberg.

Findler, N. V., Sengupta, U. K., Bozsahin, H. C. Klesczewski, K. & Smith, J. (1990). PGMULES: An approach to supplementing simulation models with knowledge-based planning systems. *Proceedings of the AI & Simulation Workshop*, AAAI-90 Conference; Boston, MA; 89-94.

Findler, N. V. (1991). Distributed control of collaborating and learning expert systems. *Proceedings of the IFAC International Symposium on Distributed Intelligent Systems*, 145-150; Arlington, VA.

Findler, N. V. & Lo, R. (1991). A distributed artificial intelligence approach to air traffic control. *IEE Proceedings Part D, Control Theory and Applications*, **138**, 515-524.

Findler, N. V. & Sengupta, U. K. (1992). Multi-agent planning and collaboration in dynamic resource allocation. *Proceedings of the First International Conference on AI Planning Systems*, 305-306, University of Maryland, College Park.

Findler, N. V. & Stapp, J. (1992). A distributed approach to optimized control of street traffic signals. *Journal of Transportation Engineering*, **118**, No. 1, 99-110.

Findler, N. V. (1993). Knowledge-based approach to urban traffic control. In G. Rzevski, J. Pastor and R. A. Adey: *Applications of Artificial Intelligence in Engineering*, 235-248; Elsevier Applied Science: London, New York.

Findler, N. V. & Lo, R. (1993a). Distributed air traffic control — Part I: Theoretical studies. *Journal of Transportation Engineering*, **119**, 681-692.

Findler, N. V. & Lo, R. (1993b). Distributed air traffic control — Part II: Explorations in a testbed. *Journal of Transportation Engineering*, **119**, 693-704.

Findler, N. V. & Ge, Q. (1993). Distributed goal-oriented dynamic plan revision. *International Journal of Intelligent Systems*, **9**, 183-210.

Findler, N. V. & Sengupta, U. K. (1995). Multi-agent collaboration in time-constrained domains (In press, Artificial Intelligence in Engineering).

Fraichard, T. & Demazeau, Y. (1990). Motion planning in a multi-agent world. In Y. Demazeau & J. P. Muller (Eds.), *Decentralized AI: Proceedings of the First*

European Workshop on Modelling Autonomous Agents in a Multi-Agent World 137-153. Elsevier Science: Amsterdam, The Netherlands.

Galliers, J. R. (1990). The positive role of conflict in cooperative multi-agent systems. In Y. Demazeau & J. P. Muller (Eds.), *Decentralized AI: Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World* pp. 33-46. Elsevier Science: Amsterdam, The Netherlands.

Gasser, L. & Huhns, M. N. (1989). *Distributed Artificial Intelligence*. Vol. II. Pitman, Morgan, Kaufman: Los Altos, CA.

Gasser, L., Bragamza, C. & Herman N. (1987). MACE: A flexible testbed for distributed AI research. In M. N. Huhns (Ed.), *Distributed Artificial Intelligence*. 119-152. Pitman, Morgan, Kaufman: Los Altos, CA.

Huhns, M. N. (1987). *Distributed Artificial Intelligence*. Pitman, Morgan, Kaufman: Los Altos, CA.

Lesser, V. R. & Corkill, D. D. (1983, Fall). The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. *The AI Magazine*, pp. 15-33.

Lizotte, M. & Moulin, B. (1990). A temporal planner for modelling autonomous agents. In Y. Demazeau & J. P. Muller (Eds.), *Decentralized AI: Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World*. 121-136. Elsevier Science: Amsterdam, The Netherlands.

Parunak, H. V. D. (1990). Distributed AI and manufacturing control: Some issues and insights. In Y. Demazeau & J. P. Muller (Eds.), *Decentralized AI: Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World*. 81-101. Elsevier Science: Amsterdam, The Netherlands.

Smith, R. G. (1980). The contract net protocol: high-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, **29**, 1104-1113.

Smith, S. F. & Ow, P. S. (1985). The use of multiple problem decompositions in time constrained planning tasks. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1013-1015.

Steeb, R., Cammarata, S., Hayes-Roth, F. A., Thorndyke, P. W. & Wesson, R. B. (1981). *Distributed intelligence for air fleet control*. Rand Report No. R-2728-ARPA. Rand Corp.: Los Angeles, CA.

Steels, L. (1990). Cooperation between distributed agents through self-organization. In Y. Demazeau & J. P. Muller (Eds.), *Decentralized AI: Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World*. 175-196. Elsevier Science: Amsterdam, The Netherlands.

Sycara, K. P. (1989). Argumentation: Planning other agents' plans. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 517-523.

Sycara, K., Roth, S., Sadeh, N. & Fox, M. (1990). An investigation into distributed constraint-directed factory scheduling. *Proceedings of the IEEE Sixth Conference on Artificial Intelligence Applications*, 94-100.

United States Air Force Academy. (1990). *Airpower theory and doctrine*. Course No. PMS-220. United States Air Force Academy: Colorado Springs, CO.

United States Air Force. (1992). *Basic Aerospace Doctrine of the United States Air Force*. Air Force Manual 1-1. U. S. Government Printing Office: Washington, DC.

Werner, E. (1990). Distributed cooperation algorithms. In Y. Demazeau & J. P. Muller (Eds.), *Decentralized AI: Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World*. 17-31. Elsevier Science: Amsterdam, The Netherlands.