A Reconfigurable Multiagent Society for Transportation Scheduling and Dynamic Rescheduling

David Montana, Gordon Vidaver, and Talib Hussain
BBN Technologies
10 Moulton Street, Cambridge, MA 02138
Email: {dmontana,gvidaver,thussain}@bbn.com Phone: 617-873-2719/3358/6861

Abstract— *We have investigated the use of an agent-based system to automate the process of scheduling all the transportation assets for the U.S. military. For this problem, the creation of a fixed schedule based on static requirements is insufficient. Instead, we need to be able to maintain the schedule as a dynamic entity in the face of changing requirements, unreliable assets, and unexpected events during execution. Towards this end, we have focused on two main areas: dynamic rescheduling and dynamic load management. Dynamic rescheduling involves updating an existing schedule in response to changes in the scheduling resources and requirements, and is a special challenge in a multiagent environment. We discuss the approach we used to propagate and coordinate changes between the different agents. Dynamic load management includes the ability of the multiagent society to change its structure in response to changing computational requirements. We examine a set of techniques, including dynamic agent creation, to continually update the allocation of computational resources to the scheduling process.*

1. INTRODUCTION

The problem of scheduling all the transportation assets for the U.S. military is very complex. There are a huge number of assets to schedule, items/people to move, and tasks to perform. The heterogeneity of the assets and tasks necessitates the use of a variety of types of scheduling logic and procedures. Additionally, the interactions and dependencies between different parts of the large problem add to the complexity.

Military transportation scheduling is currently performed by a team of humans. It can take hundreds of people days or weeks to schedule a deployment of troops and equipment. When deviations from the plan occur due to unanticipated events (such as equipment failures or new requirements), the human planners usually fall back to the easiest solution, which is often extremely inefficient.

We have worked for nearly a decade on automating the process of military transportation scheduling. We use an agent-based system (whose structure is discussed in Section 2), which is a natural fit for such a large, interconnected problem. In previous work described in an earlier paper [8], we focused on building the initial transportation schedule to the lowest level of detail. However, the construction of a fixed schedule based on static requirements is not enough. We need to be able to maintain the schedule as a dynamic entity in the face of changing requirements, unreliable assets, and unexpected events during execution. Hence, we recently have focused on two main areas: dynamic rescheduling and dynamic load management.

Dynamic rescheduling involves updating an existing schedule in response to changes in the scheduling resources and requirements, as well as the passage of time. The changes can be as small as a vehicle needing repairs or as large as a major new deployment. Dynamic rescheduling is hard enough with a single scheduler. However, as we discuss in Section 3, with a whole society of scheduling agents, there are complex issues of interactions and propagation of changed schedules between agents. (The general nature of interagent dynamics has been discussed in places such as [10], while [4], [9], and [5] examine issues specifically dealing with dynamic multiagent scheduling.)

Dynamic load management refers to the process by which the multiagent society can shift and adjust its computational load in response to changing requirements. One property enabling dynamic load management is society reconfigurability, the ability of the multiagent society to change its structure in response to changing requirements. As described in Section 4, we draw upon features in the Cougaar architecture (presented in [2] and [7]), such as agent creation and agent persistence, to automatically adjust the mapping of computational resources to agents as the computational requirements change. (Such reconfigurability is also the goal envisioned by work such as [6] and [1].) Another important aspect of dynamic load management is the use of mixed-fidelity scheduling with adjustable time horizons. By changing how far into the future the scheduling agents perform detailed scheduling and aggregate-level scheduling, the society can adjust the load on the agents.

Our system was demonstrated on a full-scale problem using real-world military data, and we present the results in Section 5.

The work described here was performed under the DARPA UltraLog contract #MDA972-01-C-0025.

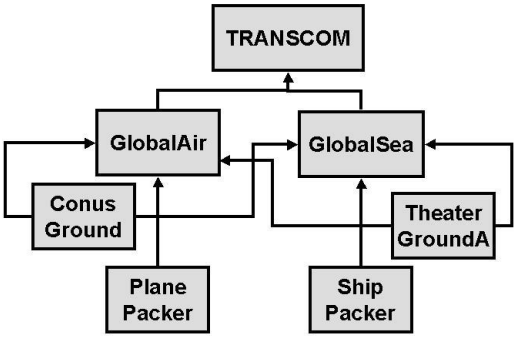

Figure 1. The initial seven-agent transportation scheduling community.

2. AGENT ARCHITECTURE

The Ultralog program has created a large demonstration multiagent society for military logistics planning based on the Cougaar multiagent infrastructure [3]. The transportation scheduling community is part of this larger society. Although small in terms of number of agents, the transportation community plays a central role. All supplies, personnel, and land vehicles in the plan must be transported from their original locations in the continental U.S. (CONUS) and Europe, to air or sea ports, across the ocean, and then across land to their final destinations.

The scale of the transportation problem is large enough that it must be divided into subproblems and distributed across multiple agents. Originally, this consisted of a relatively coarse decomposition into seven agents, as shown in Figure 1. Here, transportation requests enter at the highest level of the hierarchy of agents, the Transportation Command (TRANSCOM) agent. TRANSCOM's job is to route tasks: it chooses whether an item should go from the United States to a theater of operations by sea or air. This choice is represented as a Cougaar allocation of the task of transportation to one of two subordinates: GlobalAir or GlobalSea. The job of these agents is to orchestrate the planning of the legs of the journey, starting with the final leg and known due date and working backwards. These agents expand incoming tasks into three subtasks: theater ground transport, then sea/air transport, then CONUS ground transport. These tasks are then allocated to the agents whose job is to schedule these legs against physical assets (trucks, planes, ships). As each leg is scheduled, the results (a Cougaar AllocationResult) of when the task is scheduled to depart is sent back to the orchestrating agent, and then this agent creates the next task in the sequence with an arrival time preference requiring that it arrive no later than the next task starts.

Note there is a bi-directional flow of information: a downwards flow of requirements and decomposition of those requirements into manageable sub-problems, and an upwards flow of response information about how the subordinate agents satisfied the preferences of those requirements.

As we increased the scale of the logistics planning problem and with it the number of agents generating demand for transportation, the seven-agent transportation community became overwhelmed. With seven agents, the community could be distributed across at most seven machines. Furthermore, some of these machines were highly underutilized because the load was not equally balanced across machines. Our first step in addressing issue of scalability was to manually divide the community into 20 agents rather than seven. (Below we discuss how we later automated agent splitting.) If 20 machines are available, we could then distribute the workload over all the machines. If not, having finer-grained agents allowed us to better balance the load (originally manually and later automatically, as described below) by evening out the load across the existing machines.

Figure 2 shows this 20-agent society. In some cases, the problem handled by an agent was fully separable. For example, ammunition travels on special ammunition ships, so it was easy to split off AmmoSea as a separate agent from GlobalSea to specifically handle scheduling of ammunition on ships. In others cases, we needed to split an agent into two or more agents just because it was overloaded without a natural division. For example, Theater Ground A and Theater Ground B each handle half of the tasks and resources for ground transportation to the destination. In this case, we expect some decline in the optimality of the schedules due to the division of the problem into subproblems, but this is usually minor and is a necessary cost of scalability.

3. DYNAMIC RESCHEDULING

Rescinds are the basic mechanism that enable agents to cooperatively perform dynamic rescheduling. Rescinds allow one agent to withdraw either (i) a previous request to another agent to perform a task or (ii) a previous promise to another agent that it would perform a requested task. Rescinds generally force a ripple effect through a portion of the society, prompting the society as a whole to compensate for whatever change caused the rescheduling.

There are two basic types of changes that cause rescheduling: a change in requirements or a change in asset state. A change in requirements enters the transportation community at the top level (the TRANSCOM agent), and its effects filter downwards in the hierarchy. For example, consider the situation (shown in Figure 3) where a transport request from some external agent is modified so that the item must arrive a day earlier than previously specified. (Possibly the commanders have moved up the day of a planned strike.) This causes the TRANSCOM agent to rescind its previous request to the GlobalSea agent and then send a new request with the earlier date. This in turn causes GlobalSea to rescind its requests to its subordinates in the hierarchy and send new requests. Thus, a new revised schedule is created.

A change in asset state occurs at a low-level agent. If the low-level agent can adjust the schedule so that it can continue to

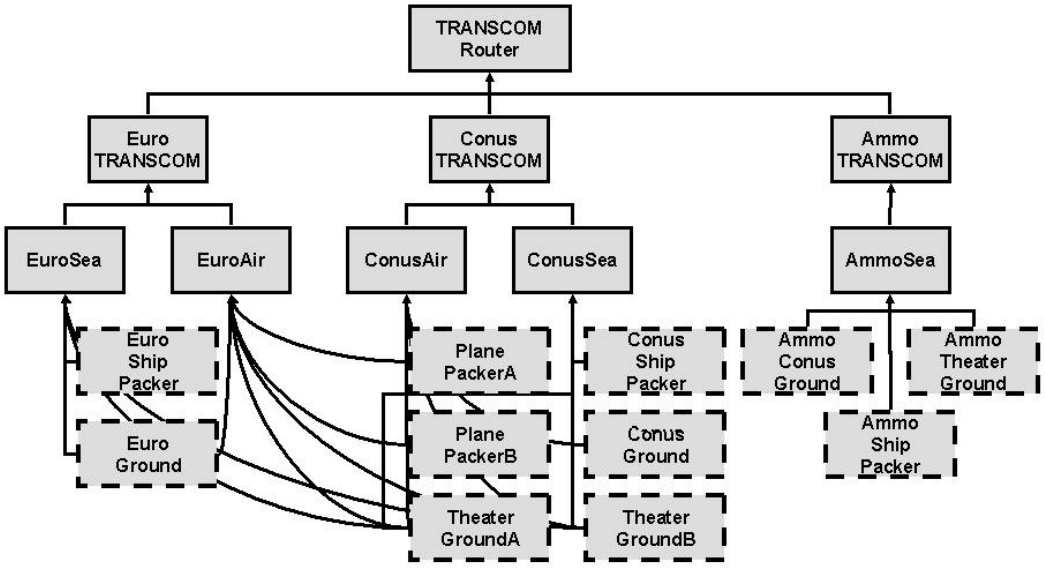

Figure 2. A 20-agent community of transportation scheduling agents. Additional agents have been added to route incoming tasks among distinct subproblems, and agents doing scheduling against physical assets (denoted with a dotted boundary) have been divided.

meet the requirements of the tasks requested of it despite the change in asset state, then no rescind is necessary. However, if the agent can no longer meet all the requests that it previously agreed to perform, then it must rescind one or more of the positive replies. This causes the agent's superior to attempt to modify its schedule in such a way that it can continue to meet the requirements imposed on it. If it can do so, it informs its subordinate agents of any schedule changes via rescinds. If it cannot alter its schedule to meet its requirements, it has to communicate this one level higher. Potentially, the rescind can get propagated out of the transportation community and back to the requesting agents, which then has to modify the request.

4. DYNAMIC LOAD MANAGEMENT

The transportation scheduling problem may impose significant changes in scheduling requirements that may affect a variety of different types of assets and to varying degrees. For instance, the needs of a mission may change and require large numbers of certain types of assets to be scheduled, and a small number of other types of assets.

In a static agent structure, e.g., with specific agents maintaining the responsibility to schedule specific types of assets with specific computing resources, certain changes in mission requirements may place a much heavier load on some agents and very little load on others. The result is that certain agents may become overloaded while others may underutilize the computing resources at their disposal.

To accommodate these asymmetrical loads upon the agents while also performing at high efficiency, one key capability that we incorporated into our agent system was adaptive load balancing to automatically adjust the mapping of computational resources to agents as the computational requirements change. The simplest way to do this is to just move an agent from one CPU to another less loaded CPU. Cougaar provides an agent mobility mechanism to save the state of an agent and reconstitute it on a different machine [7]. The service discovery mechanism allows connections with other agents to reform. [Note that this provides not only a mechanism for load balancing but also a means of enhancing survivability by ensuring the continuing operation of agents without losing state if a machine goes down.] The problem with this approach to load balancing is that it does not work if the agent has more load than can be handled by a single CPU.

For this situation, our system has the capability to spawn new agents of the same type to handle the additional workload [7]. A four step process is followed to initiate a rebalanced load. First, a *falling-behind* sensor detects instances when an agent is unable to process requests in a timely manner. For instance, the *falling-behind* sensor may notice that one of the leaf agents that plans against physical assets (e.g., PlanePackerB), is not allocating all its assets fast enough. Second, a new agent is spawned on a new (or underutilized) CPU and heap. Third, the current (overloaded) agent transfers copies of half its physical assets (e.g., planes) to the new agent, thereby splitting the load evenly across the two agents. Finally, the new agent broadcasts itself as a new scheduling agent of the same type as the current agent (e.g., AirTransportationProvider) via Cougaar Service Discovery. The leaf agent's superior (e.g., ConusAir), which orchestrates the planning process, gets this report of a new subordinate that provides this service. It can now do round-robin assignment of new incoming tasks across all its subordinates with that shared role.

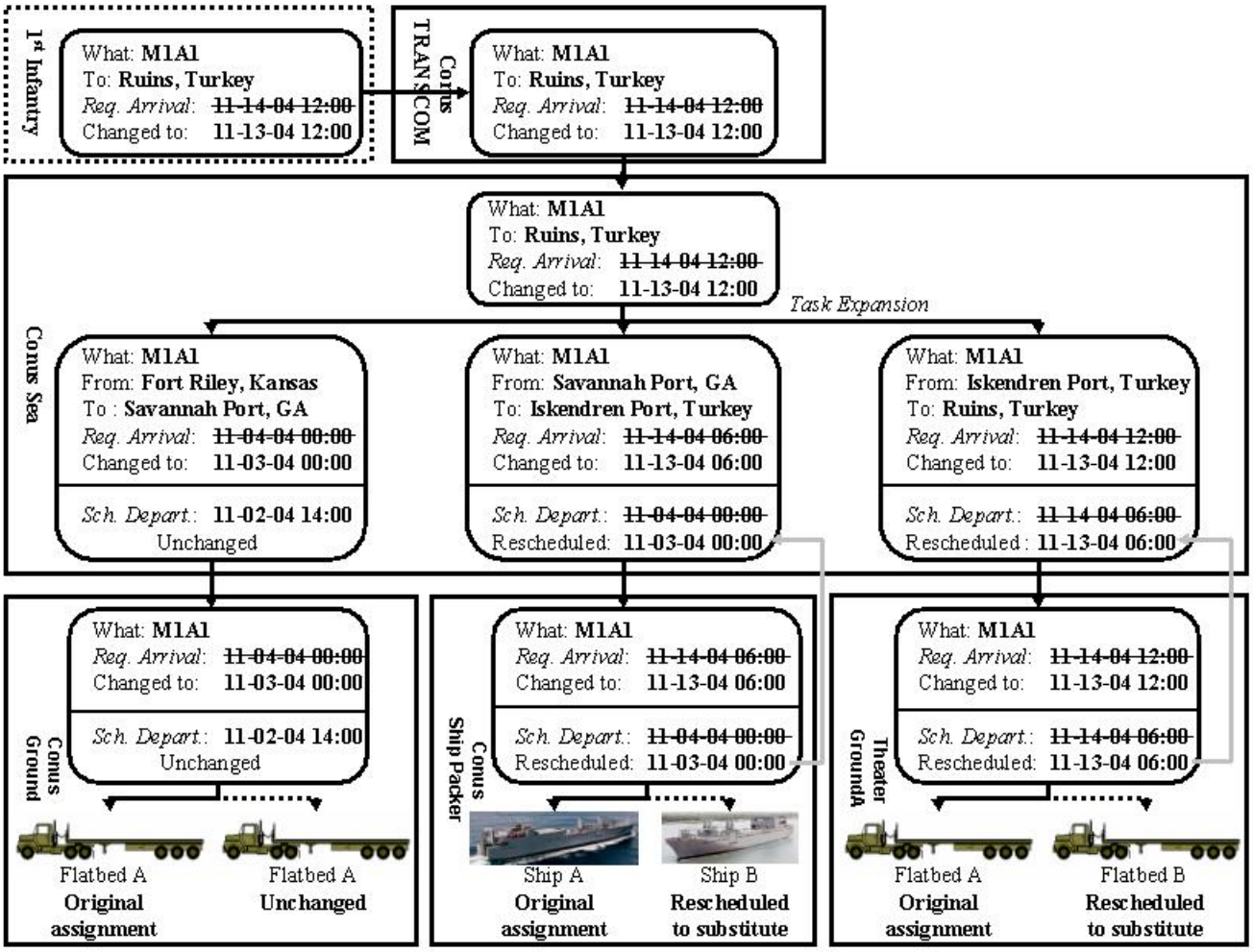

Figure 3. Cooperative dynamic rescheduling occurs when reactions to changes in requirements or changes in the assets' state ripple through the community via the mechanism of rescinds.

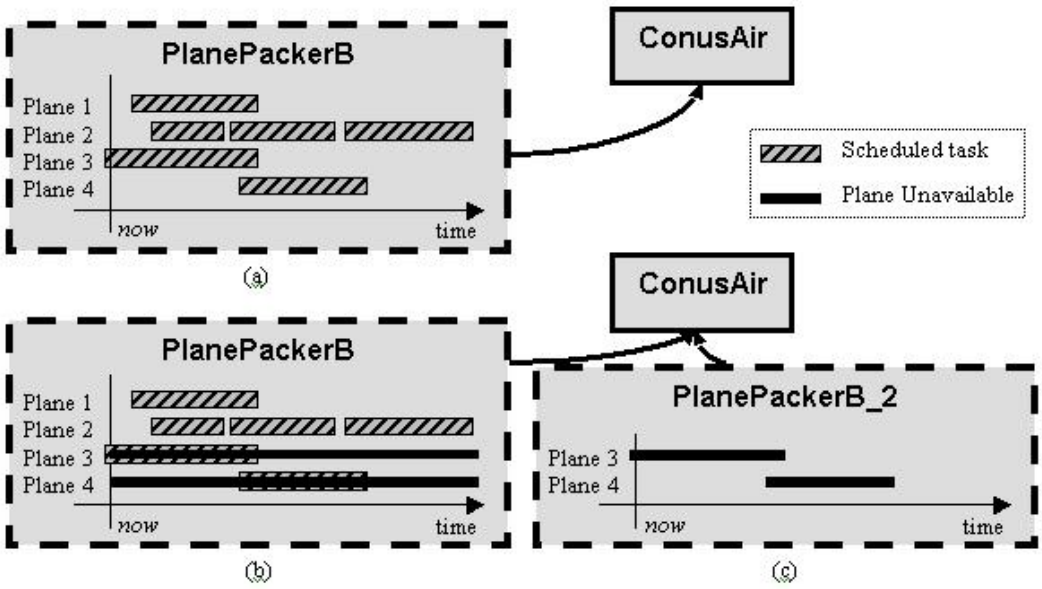

Figure 4. Illustration of scheduled tasks and unavailable assets before and after agent creation, for (a) original PlanePacker agent, (b) same agent after agent creation, and (c) newly created agent.

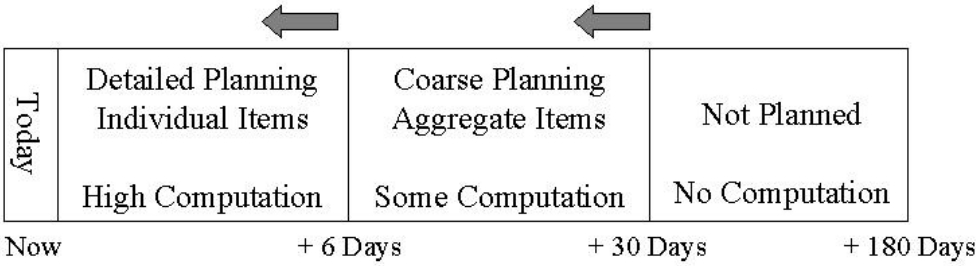

Figure 5. The time horizons for both detailed and aggregate-level scheduling can be adjusted in response to changes in load on a scheduling agent.

Once the load has been re-balanced, our system applies two restrictions to ensure consistent and appropriate planning (as shown in Figure 4). First, all the transferred assets are marked as *unavailable* in the new agent for those periods in which the assets have been previously been scheduled. This ensures that all plans made against the transferred assets are maintained. Second, all the transferred assets are marked as *unavailable* in the original agent. This ensures that new tasks in the original agent are not made against transferred assets (i.e., removes them from future consideration in planning within the original agent).

In addition to society reconfiguration, adjustable time horizons provide another equally important, although more mundane, technique for load management. Within a certain time horizon into the future, T_1, individual items are included in a detailed schedule/plan. Between time T_1 and some more distant time in the future, T_2, plans are created at a coarse level, based on large aggregates of items. Beyond T_2, no plan is made. This division of the future into three different levels of planning (i.e., *mixed-fidelity planning*) allows each scheduling agent to concentrate its efforts planning for the near future, which is both more urgent and less likely to change. If the *falling-behind* sensor indicates that the agent is not currently keeping up with its assigned workload, the agent can lessen its workload by changing the values of T_1 and T_2 so as to be planning less far into the future (as illustrated in Figure 5).

Note the use of mixed-fidelity planning requires the ability to perform dynamic rescheduling. As coarse plans transform to detailed plans due to the passage of time or a lightened load, the agents must adapt the schedule accordingly using the dynamic rescheduling mechanisms discussed above.

5. EXPERIMENTS

In our first experiment, our system was used to plan a solution to moving hundreds of different military units (their equipment and personnel) from the United States and Europe to Turkey over a 180-day period. There were a total of 22,118 equipment items and 34,058 people to move and 20 ships, 80 planes, and 265 trucks and railcars available for use across the transportation scheduling community. Three configurations of the community were used within a larger society of over 1000 agents. The first configuration used the original seven

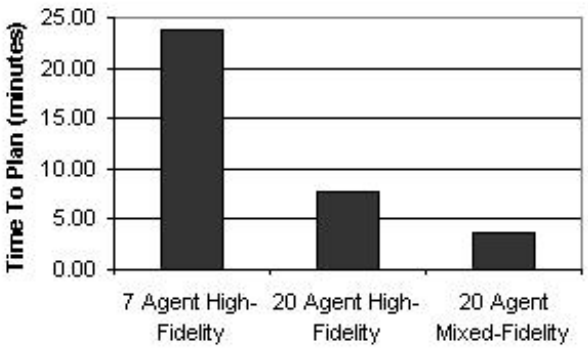

Figure 6. Results for static schedule creation.

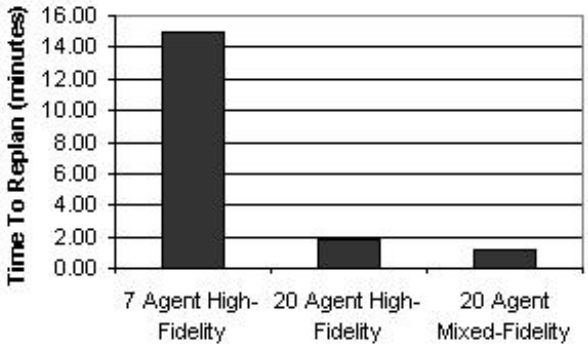

Figure 7. Results for dynamic rescheduling.

agent community, as illustrated in Figure 1, performing high fidelity planning over the entire 180 days. The second and third configurations used the improved twenty agent community, as illustrated in Figure 2, with the second one performing high fidelity planning over the entire 180 days and the third one performing mixed-fidelity planning, with a high-fidelity period of only two days. Figure 6 illustrates the difference in the time to plan the entire problem for each configuration. The twenty-agent mixed-fidelity case demonstrated the best response time.

In our second experiment, the same system configurations as above were applied to the same initial planning problem followed by an additional replanning stage, in which one brigade (7 units, comprising 1654 items and 3479 people) moved its arrival time requirement from 10 days to 15 days from the be-

ginning of the mission. Figure 7 illustrates the time required for the replanning stage only. Again, the twenty agent mixed-fidelity case demonstrated the best response time.

6. CONCLUSION

The problem of military transportation scheduling is inherently dynamic. New and changing requirements, as well as unanticipated events, necessitate the continual revision of any schedule. The combination of such a dynamically evolving environment with a multagent society presents a special set of challenges. Some of these challenges are algorithmic: agents need to cooperate on their updates so as to maintain a globally good schedule and not cause uncontrolled dynamics in the society. Others of these challenges are structural: as the computational needs shift along with the changing requirements, the society needs to reassign computational resources to meet these needs. We have developed a set of techniques to address both types of challenges, creating a multiagent society for transportation scheduling that robustly handles the dynamic aspects of this problem.

REFERENCES

[1] Bradshaw, J., N. Suri, A. Canas, R. Davis, K. Ford, R. Hoffman, R. Jeffers, and T. Reichherzer: 2001, 'Terraforming Cyberspace'. *IEEE Computer* **34**(7), 48–56.

[2] Brinn, M., J. Berliner, A. Helsinger, T. Wright, M. Dyson, S. Rho, and D. Wells: 2004, 'Extending the Limits of DMAS Survivability: The UltraLog Project'. *IEEE Intelligent Systems* **19**(5), 53– 61.

[3] Cougaar.org: 2004, 'Cognitive Agent Architecture (Cougaar) Home Page'. http://www.cougaar.org.

[4] Cowling, P., D. Ouelhadj, and S. Petrovic: 2003, 'A multi-agent architecture for dynamic scheduling of steel hot rolling'. *Journal of Intelligent Manufacturing* **14**(5), 457–470.

[5] Dahl, T., G. Sukhatme, and M. Mataric: 2002, 'Scheduling with Group Dynamics: A Multi-Robot Task-Allocation Algorithm based on Vacancy Chains'. Technical Report CRES-02-007, Center for Robotics and Embedded Systems.

[6] Hannebauer, M.: 2002, *Autonomous Dynamic Reconfiguration in Multi-Agent Systems: Improving the Quality and Efficiency of Collaborative Problem Solving*. Springer-Verlag.

[7] Helsinger, A., K. Kleinmann, and M. Brinn: 2004, 'A Framework to Control Emergent Survivability of Multi Agent Systems'. *Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*. pp. 28–35.

[8] Montana, D., J. Herrero, G. Vidaver, and G. Bidwell: 2000, 'A Multiagent Society for Military Transporation Scheduling'. *Journal of Scheduling* **3**(4), 225–246.

[9] Wagner, T. and V. Lesser: 2000, 'Design-to-Criteria Scheduling: Real-Time Agent Control'. *Proceedings of AAAI 2000 Spring Symposium on Real-Time Autonomous Systems*. pp. 89–96.

[10] Youssefmir, M. and B. Huberman: 1997, 'Clustered volatility in multiagent dynamics'. *Journal of Economic Behavior and Organization* **32**(1), 101–118.