Appl Intell (2006) 25:335–357
DOI 10.1007/s10489-006-0111-2

Achieving dynamic, multi-commander, multi-mission planning and execution

Eugene Santos Jr · Scott A. DeLoach · Michael T. Cox

Abstract The Multi-Agent Distributed Goal Satisfaction (MADGS) system facilitates distributed mission planning and execution in complex dynamic environments with a focus on distributed goal planning and satisfaction and mixed-initiative interactions with the human user. By understanding the fundamental technical challenges faced by our commanders on and off the battlefield, we can help ease the burden of decision-making. MADGS lays the foundations for retrieving, analyzing, synthesizing, and disseminating information to commanders. In this paper, we present an overview of the MADGS architecture and discuss the key components that formed our initial prototype and testbed.

Keywords Mobile multiagent systems · Mixed-initiative planning · Distributed mission planning and execution · Intelligent resource allocation · Agent oriented software engineering

1 Introduction and motivation

Real-time air mission planning and execution occurs in a highly complex and dynamic environment where new constraints and conditions frequently arise at all levels of operation from theater level planning to individual unit tasking and execution. Some new conditions are "discovered" as planning failures arise while completing local objectives (goals and sub-goals). Planning failures are caused by factors such as internal planning conflicts, irrevocable commitments from partial plan execution, and insufficient resources (number of planes, fuel, etc.). Other conditions are imposed from outside the system during planning and execution. Such conditions include command modifications in the mission objectives (human intervention—friendly and/or enemy) and continuous changes in the external environment (weather changes, equipment failure, etc.). Moreover, military operations planning is a distributed activity. Planners at all levels and in all services cooperate (and compete) to create and execute operations.

With all this in mind, the burden faced by our commanders/planners in the modern information rich battlespace is overwhelming to say the least. To be fully aware of the battlespace situation is simply not feasible. However, the strength of our commanders lies in what they know combined with what they can easily access in order to get the job done.

Unfortunately, most of the research to date on mission planning and execution has been based around the unrealistic assumption of a single "point of planning;" that is, all planning occurs at a single location with unlimited access to global information. Such an approach does not permit scaling and is unrealistic in today's information intensive environments. First of all, without the ability to scale to large problems, automated planning will not be able to fit into existing organizations, or to permit simultaneous planning with plan execution. A particularly important issue is that local dynamic conditions, as discussed in the preceding paragraph, can result in a rippling effect of plan failures all the way up

E. Santos Jr. (✉)
Thayer School of Engineering, Dartmouth College, Hanover, NH 03755
e-mail: Eugene.Santos.Jr@Dartmouth.edu

S. A. DeLoach
Department of Computing and Information Sciences, Kansas State University, Manhattan, KS 66506-2302
e-mail: sdeloach@cis.ksu.edu

M. T. Cox
Intelligent Distributed Computing Department, BBN Technologies Cambridge, MA 02138
e-mail: mcox@bbn.com

to the highest levels. Secondly, unlimited global access is simply not feasible given the size, reliability, and dynamic reconfigurability of our information network as well as the necessity of providing information security in such a critical environment. For mission planning and execution to work effectively, we must be able to reliably provide commanders the information when they "need to know" and guarantee the information security if they "need to know."

In the real-world dynamic and large-scale setting of air mission planning and execution, this can only be achieved in a distributed manner to provide sufficient levels of efficiency, robustness, and security. This has been the primary theme for our Multi-Agent Distributed Goal Satisfaction (MADGS) project.

In our approach, we set out to clearly identify the root causes of plan failures and when and where such failures can be best addressed by the human commanders. In particular, we posited that significant planning failures occur during execution when specific resource requirements for a given task cannot be satisfied because of the dynamic nature of our operational environment. Instead of forcing a costly re-planning of the mission, we redefined the problem in terms of satisfying resource requirements during mission execution. This allows us to avoid planning failures if alternative resources can be located (in a local fashion) for the human commanders to carry out their tasks. In the case where resources cannot be allocated, we now have up to date information on relevant resources and availability that can now be passed up the mission plan hierarchy. Combined with our developments in mixed-initiative planning through goal-transformations, we can better assist during re-planning but also limit the scope necessary of the overall re-planning effort.

Specifically, we have achieved the following: We (1) designed a model of distributed goal satisfaction to mitigate plan resource failures; (2) introduced mixed-initiative planning through goal transformations; (3) developed and deployed a scalable mobile multi-agent infrastructure for dynamically reconfigurable networks; and, (4) built a framework for automated software agent generation and validation. In the remainder of this paper, we lay out the MADGS architecture, discuss its key components, and finally present our evaluation of the MADGS prototype system.

2 Multiagent distributed goal satisfaction

The Multi-Agent Distributed Goal Satisfaction (MADGS) system is a JAVA-based mobile-agent system that facilitates distributed mission planning and execution in complex dynamic environments with a focus on distributed goal satisfaction [38]. The MADGS system consists of five key components: a Distributed Goal Satisfaction module (DGS), an agent-server framework (Carolina), a set of application specific mobile-agents, an agent generation capability (agentTool), and the PRODIGY planner. The primary issues we explored include robust and reliable communication protocols, agent design, and a system architecture that facilitates both agent and agent server autonomy.

The target real-world operational environment for the MADGS system is a network topology that consist of intermittent nodes and uncertain network connections that exist in a large-scale, multi-platform dynamic network. The resulting design developed for this environment addresses the communications issues faced when handling massive numbers of mobile-agents in such a topology. Our development process required the consideration of bandwidth capacities (minimal broadcasts if any), mobile-agent collaboration issues, and server awareness of available resources. In designing a system capable of handling an unknown but unrestricted number of communication and agent migrations over the proposed topology we made an in-depth examination of both agent and server responsibilities. From this examination we developed a premise that there exists a marriage between the functionality of the Carolina agent-server and the agents themselves despite their autonomy. For this reason the MADGS architecture was built around this marriage, maintaining autonomy for both without depreciating the security or performance of the system. The marriage is one built of necessity. In order to minimize agent size some functionality was better placed in the server and offered as a service to the agents.

MADGS mobile-agents are the workhorses of the system providing the functionality the system users require. Mobile-agents are injected into the system through the agentTool [16] component responsible for agent creation. The agentTool component is not a standard component present on all nodes. As an administrative tool, agentTool, is instantiated at predetermined points in the network expressly as a defined creation and entry point for mobile-agents into the MADGS system. This architecture provides an avenue for insuring a level of authenticity and thereby security to the system. If a new mobile-agent or a clone is needed, the requests for these are filtered through an agentTool component via an agentToolHandler agent. The DGS module will provide alternative resource configurations to facilitate the completion of a plan constructed via PRODIGY given constraint failure without backtracking or replanning. The DGS component will interact with the resource agent to maintain an accurate record of available resources using a distributed database of resource attributes and linear programming tools to make replacement determinations. The DGS component will rely on human operators to make resource substitution decisions. Alternative resource configurations for resources identified by the plan as mission-critical are given priority by the DGS component. PRODIGY [46] is a legacy planning system that will provide the initial master plan for operations. Figure 1 illustrates the interactions between agentTool, PRODIGY,

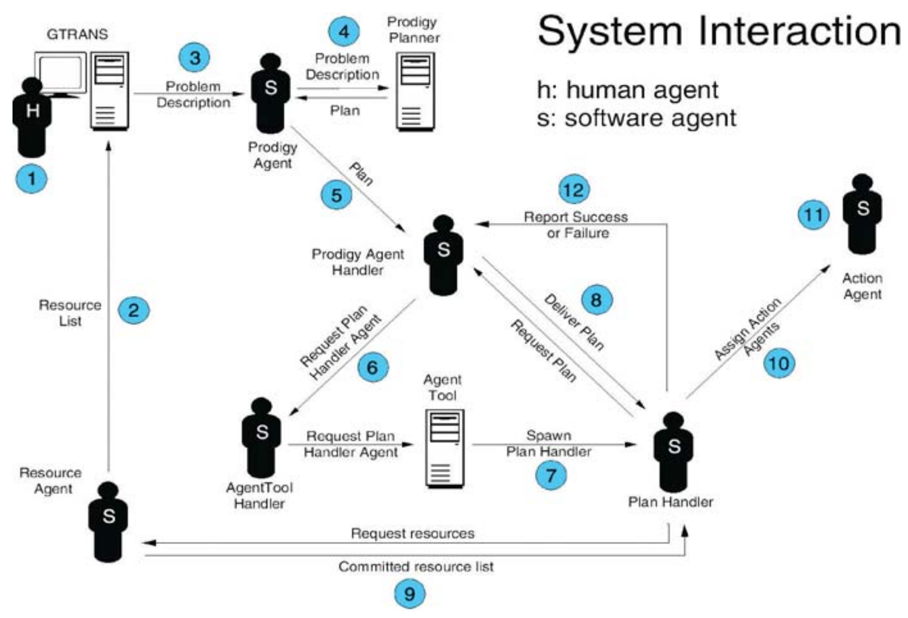


Fig. 1 MADGS system. Component/agent interaction flow

Carolina, and the DGS components of MADGS. In this figure, Carolina is the backplane responsible for the communications and agent operation environment.

Our MADGS system functions as follows (Fig. 1): For each human commander, there is an associated PRODIGY system complete with GTrans interface to assist the commander in formulating plans for their given missions. Also attached to the commander are a ResourceAgent, a PRODIGY agentHandler, and agentToolHander. When a plan is completed by the commander with PRODIGY, it consists of a sequence of specific execution tasks. The PRODIGY agentHandler now requests agentToolHandler to spawn a specific planHandler agent from agentTool for this given plan. In this case, the planHandler can be retrieved from a library of agents that agentTool has created off line. The plan sequence is now picked up by our newly created planHandler agent where a resource analysis is conducted to determine the resource requirements of each task in the sequence. Execution of each task is now commenced. For each task being deployed, MADGS attempts to satisfy the specific resource requirement by querying the commander's ResourceAgent and then negotiating with other ResourceAgents "nearby" if necessary. When the requirements are satisfiable, the planHandler then requests agentToolHandler to now spawn a specific task execution agent for the task to be executed. This continues until all tasks are executed or a failure in a task execution or resource requirement is encountered. Once encountered, the planHandler then gathers the details of the failures and attempts to overcome the failure to send to the prodigyAgentHandler and mixed-initiative replanning commences. Also, implicit in the diagram is the ability of the human decision maker to observe activities on the system.

In the following sections, we detail the individual MADGS components described above.

2.1 Distributed goal satisfaction

The general problem with the development of the MADGS system revolves around facilitation of real-time operation and plan failure handling. One of the common threads across the aforementioned domains and the general problem is the need for distributed goal satisfaction that can work cooperatively with legacy planning systems yet autonomously handle changes in constraints. The ability to autonomously handle changes in the constraints of a plan can mean the success or failure of any distributed operational mission/goal. The need to re-plan or backtrack due to constraint changes in any plan can mean a substantial resource loss; be it lost capital or life, the expense is real. Our approach seeks to mitigate a significant amount of this loss by preemptively expecting failure, defining alternative constraint configurations, developing delivery arrangements and in the event of a failure offering an instant solution to the user.

This distributed goal satisfaction (DGS) process is a primary background activity of the MADGS system. In the forefront, the system is providing an environment for general operation, sub-plan and sub-task execution, and user interface. It is this aspect of the MADGS system that facilitates the DGS process. The MADGS system represents every resource (including personnel) by at least one agent. However, this representation allows the MADGS system to facilitate the DGS process by maintaining a more complete view of the current state of the world. This world-view is constantly shifting in any operation especially large-scale operations.

For this reason alone communications becomes a crucial aspect of our system.

Even though some agent-based mission planning and execution systems have been developed [47] they do not fully use the power of agent programming. Our approach is to use the strengths of legacy systems in conjunction with the strengths of agent programming coupled with our own approaches to communications (as outlined above) and resource location and allocation. Most large-scale operations create a plan offline by formulating a problem or suggested outcome and then determining an optimal plan for the realization of this global goal. While an agent system could be used for such an operation there are legacy systems in existence optimized for this purpose. This is why PRODIGY is used to create a Master Plan for the MADGS system. This plan is then provided as input to the MADGS system. The MADGS system then uses command agents to decompose the plan (if necessary) into sub-plans which are further decomposed into tasks by sub-command agents (Command agents with a lower rank) that assign the tasks to subordinate task agents. This process is not dissimilar to those present in existing agent-systems. Our approach is only unique in how we communicate and plan for alternative courses of action in the event a plan fails during execution due to changes in the present state of the 'world'.

Resolving to one course of action (or plan) in a real-time system poses great difficulty due to the volatile nature of the constraints and the conditions a plan is based on. A change in a constraint or condition of a sub-plan could lead to its total or partial failure that in turn can lead to a rippling effect, thereby negating the validity of the initial plan. To overcome these points of failure, a robust and flexible planning system is needed. The DGS agent module seeks to provide a surrounding technique to improve the robustness and flexibility of the overall planning and execution process.

We accomplish this by acting on the resources required to accomplish a given goal, plan or task. A resource is any commodity that is necessary to facilitate the completion of a goal. (i.e.: Goal A requires resource X, quantity 3) The DGS agent module receives the local version of the plan and a list of the required constraints (primary resources). With this input a tree is constructed of the alternative resource configurations meeting a stated Tolerance level representing a cost-benefit function for each required primary resource. This data is then rated based on alternative resource availability to the local plan (taking into consideration other known pending or current local plans). This information is then stored and the DGS agent module monitors the primary resource statistics. In the event that a primary resource fails or is exhausted the DGS agent module suggests alternative resource configurations to complete the current plan. If accepted the resources are set into action in place of the primary (failed) resources. During this process DGS collects and records data on the selections that users make to give weight to certain configurations and more importantly to learn new resource alternatives and configurations when users manually manipulate the suggested configurations prior to commitment. This process when successful negates the need to replan or backtracking furthering the maintenance of a real-time system. In the following subsection, we present our formal approach to resource matching.

2.1.1 *Intelligent resource allocation*

As stated above, backtracking due to plan failure can result in substantial loss; whether it be loss of capital or or loss of life, the expense is real. For MADGS, we determined that we can mitigate a significant amount of this loss by preemptively expecting failure, defining alternative resource constraint configurations, developing delivery arrangements.

In the event of a failure, our goal is to offer alternative solutions to the human commander in an effort to assist them in accomplishing their mission. Our approach seeks to mimic and exploit the strengths of our on-site commanders by enhancing their own innate resourcefulness. In effect, we attempted to provide targeted resource information to assist commanders in resource substitution decisions and on the spot plan alterations.

At its most basic level, resource substitution can be simplified to: Can resource A be substituted by resource B given a set of mission requirements. While this is the minimum that is needed to assist the commander's "scrounging" decisions, we realized that many more factors must be captured. These include cost factors such as transportation costs, scheduled availability, production costs, etc. Also, another more problematic cost is the possibility of cascading plan failures when a particular critical resource is diverted toward solving another plan. This is especially unacceptable if the second plan is not critical to overall theater operations.

In addition, given the vast number of resources available in any theater, the number of alternative resource suggestions can actually overwhelm the human commander. Our goal is to take all these issues into consideration and develop a framework for intelligent resource substitution that can ultimately further provide global guarantees of mission satisfiability in the overall theater operations.

Clearly, resource substitution is very much related to logistics management (e.g., [44]). We can state such a problem as consisting of a set of suppliers and consumers. Suppliers provide resources, while consumers utilize some resources to achieve some goals, such as doing jobs or producing final products. After more than a decade's development on logistics management, many kinds of logistics management models have been proposed and implemented. Some models are stand-alone and centralized, while others use a client/server approach ([4, 24, 37]). In recent years, researchers have proposed multi-agent based models ([30, 33, 39, 48, 49]). However, most of the models regard logistics management as an auction, in which each entity tries to maximize its own benefit. Such an approach is only appropriate for inter-organizational logistics, which consists of competitive entities. However, this is clearly inappropriate for our problem. In our case, suppliers (bases, depots, etc.) and consumers (commanders) have a common goal to maximize the outcome of the entire organization (theater operations). People usually use intra-organizational logistics to describe this case. The point is that the maximized outcome does not mean the maximal benefit of each entity. Therefore, an optimal scheduling may be built based on sacrificing of some individuals' benefits. Unfortunately, the typical auction approach does not account for necessary self-sacrifice.

To address intra-organizational logistics, some researchers have developed coordinating multi-agent logistics management models. Sadeh et al. in [39] proposed MASCOT (Multi-Agent Supply Chain cOordination Tool), a reconfigurable, multilevel, agent-based architecture for coordinated supply chain planning and scheduling. Shen et al. in [43] proposed MetaMorph II architecture for enterprise integration and supply chain management, which is mediator-centric and agent-based. Fox et al. in [21] described the architecture of the integrated supply chain management system, in which each agent performs one or more supply chain management functions, and coordinates its decisions with other relevant agents. A KQML-based multi-agent coordination language was proposed in [1] for distributed and dynamic supply chain management. However, their approaches are ad hoc and lacking in precise optimization models.

To overcome their limitations, we believe that it is necessary to formulate the resource allocation problem formally. Luh et al. in [34] utilized Lagrangian Relaxation to remove couplings between constraints so that the original problem can be separated into subproblems. Ideally, these subproblems should be separable/independent. The separability property is good for us because we can allocate a different agent to solve each subproblem. If the solutions for these subproblems are compatible with each other, we are done. Otherwise, these agents can exchange information and find an optimal way to satisfy constraints. We believe that the formulations we have developed before can satisfy separability since our formulations are similar to those found in [34] for manufacturing scheduling. It was demonstrated that the separability condition does hold for those problems. In case the separability does not hold in general, we can still significantly benefit from the problem decomposition and subproblem groupings to improve our computations.

2.1.2 Modeling intra-organizational logistics

Before we can model intra-organizational logistics, we need to know its specific issues. Since suppliers (bases and other commanders) can only provide limited resources for any given period, the needs of the consumers (commanders) may not be fully satisfied. The goal of intra-organizational logistics is to reasonably allocate resources so that the profit of the whole organization is maximized. Depending on the following factors, the problem may be relatively easy or very complex:

- *Number of resource types*: For example, typical resource types are plane, ship, truck, fuel, ammunition, soldiers, airport, and so on.
- *Number of suppliers for each consumer*: If each consumer has only one supplier, the problem becomes easy. This is generally the case when a supplier is in charge of one geographical area. But with modern transportation means and networking, a supplier is no longer limited by its location. Therefore, the typical relationship between consumers and suppliers is a many-to-many relation.
- *Task properties*: A task can be the mission activities in a given period. Generally, a deadline is set on each task. The execution of a task needs a certain amount of resources. Sometimes, a task cannot be started unless all resources have been received. In other cases, lack of resources only delays the execution time or degrades the quality of a task. Similarly, extra resources may accelerate the execution of a task or achieve a better goal. A task may be viewed as a single step operation. Or it can consist of a sequence of stages. It is possible that some stages are critical. If a critical stage violates the deadline or cannot be continued for some reason, the whole task may be regarded as having failed. A consumer may only execute one task over a long time period or many tasks. Several tasks may be related, such as having a common goal. Related tasks have more constraints. For example, two tasks are required to begin at the same time or one before the other.
- *Resource properties*: Different resource types are not fully separated. Two different types of resources may have overlapping functionalities. For example, two kinds of planes can achieve the same objectives, only with different costs. We call this property resource exchangeability. Based on resource exchangeability, we can allocate alternative resources if critical resources are in demand. However, if alternative resources lead to higher costs, a trade-off exists between using these alternative resources and waiting for needed resources.
- *Uncertainty*: That the situation is changing over time increases the difficulty of managing intra-organizational logistics. For instance, tasks arrive dynamically because of

great variability of customer demands and resources may not be provided in time. Due to the insufficient resources or other uncertainties, like machine breakdown, tasks may not be finished before the deadline. There are so many uncertainties, how can we manage intra-organizational logistics efficiently?

Traditional models only consider subsets of these factors and deal with the problem in a centralized manner, which means all the information is collected at one place for analysis.

Our approach is different from existing approaches above in that we take into account a precise optimization model.[1] Through the use of Lagrangian Relaxation, we can decompose a resource allocation problem into subproblems, each of which will be solved by a specific agent. If the solutions for these subproblems are compatible with each other, we are done. Otherwise, these agents can exchange information with each other until a global optimal solution is found. Below, we provide various models for use depending on the complexity of the target planning problem.

Single resource type/single supplier/single job. In this case, we suppose that only one kind of resource exists in the system, and each consumer requests resources from a corresponding supplier to do a single job. We also fix the cost of using unit resource. Therefore, there is no need to consider the cost when we do job scheduling and the goal is to minimize job delay.

Since each consumer is related with only one job, we consider only jobs and suppliers in the remaining section. Let the number of jobs be N_c and the number of suppliers be N_s. For each job $i = 1, \ldots, N_c$, b_i represents the starting time, c_i represents the completion time, p_i represents the execution time, d_i represents the deadline for job i, s_i represents the supplier from which job i will get resources ($s_i \in [1, N_s]$), and r_i represents the needed resource number. Also, we use T_i to measure the delay of job $i(T_i \in [0, c_i - d_i])$. Because some jobs may be more important than others, it is not desirable to delay these jobs. To reflect this factor, we use w_i to represent the importance of each job. The higher the value of w_i is, the more important the job is. We can define the objective function as:

$$\text{minimize } \sum_{i=1}^{N_c} w_i T_i^2$$

At any given time t, the total granted resources from one supplier must be less than its capacity. We use $N_j (j \in [1, N_s])$ to represent the amount of resources that supplier j has. We suppose T is long enough to complete all the jobs. The resource capacity constraints can be expressed as:

$$\sum_{i=1}^{N_c} \delta_{ijt} \cdot r_i \leq N_j, \quad j = 1, \ldots, N_s, \quad t = 1, \ldots, T$$

$$\delta_{ijt} = \begin{cases} 1 & \text{If } s_i = j \wedge b_i \leq t \leq c_i \\ 0 & \text{Otherwise} \end{cases}$$

In addition, the following processing time constraints must be satisfied:

$$c_i - b_i + 1 = p_i, \quad i = 1, \ldots, N_c$$

Single resource type/single supplier/multiple jobs. In this case, each consumer can execute several jobs. We assume that at any time, each consumer can execute at most one job. Therefore, jobs belonging to one consumer cannot be overlapped. Here, we suppose that s_{ik} represents minimal switching time for executing job k after finishing job i if these two jobs are executed by the same consumer. For each job i, o_i represents the consumer who will do the job. Compared to the above case, the precedence constraints need to be considered:

$$\delta_{ik}(c_i + s_{ik} + 1 - b_k) + (1 - \delta_{ik})(c_k + s_{ki} + 1 - b_i) \leq 0,$$

where $i, k = 1, \ldots, N_c, i \neq k, o_i = o_k$;

$$\delta_{ik} = \begin{cases} 1 & \text{If job } k \text{ occurs after job } i \text{ has been finished} \\ 0 & \text{Otherwise} \end{cases}$$

Single resource type/multiple suppliers/multiple jobs. The drawback of the above models is that a consumer cannot execute a job until the corresponding supplier has enough resources. Through allowing a consumer to request resources from multiple suppliers, this model is more flexible. In most cases, this model can get better job scheduling than the above two models. However, this model is more complex. Though we can limit each consumer to only requesting resources from a specific set of suppliers, we assume each consumer can request resources from all suppliers. But the costs of using resources from different suppliers are different. Therefore, we need to consider the cost factor in the objective function, shown as the following:

$$\text{minimize } \sum_{i=1}^{N_c} \left(w_i T_i^2 + \sum_t \sum_{j=1}^{N_s} r_{ijt} c_{rj} \right),$$

[1] Existing approaches include [21, 33, 39, 48, 50, 51].

where r_{ijt} represents the number of resources that job i gets from supplier j at time t; $c_{rj} (j \in [1, N_s])$ is the cost of using unit resource from supplier j per unit time.

Also we modify the resource capacity constraints:

$$\sum_{i=1}^{N_c} r_{ijt} \le N_j, \quad j = 1, \ldots, N_s, \quad t = 1, \ldots, T$$

$$\sum_{j=1}^{N_s} r_{ijt} = \begin{cases} r_i & \text{If } b_i \le t \le c_i \\ 0 & \text{Otherwise} \end{cases}$$

Multiple resource types/multiple suppliers/multiple jobs. If we permit multiple jobs to be executed by one consumer, we need to consider the precedence constraints at the same time:

$$\delta_{ik}(c_i + s_{ik} + 1 - b_k) + (1 - \delta_{ik})(c_k + s_{ki} + 1 - b_i) \le 0,$$

where $i, k = 1, \ldots, N_c, i \ne k, o_i = o_k$;

$$\delta_{ik} = \begin{cases} 1 & \text{If job } k \text{ occurs after job } i \text{ has been finished} \\ 0 & \text{Otherwise} \end{cases}$$

The objective function and resource capacity constraints are the same as those in the fifth model.

In summary, we have deployed this approach using the various models above in a testbed that varies the above parameters in a manner reflecting real-world mission planning resource needs. We have achieved very good efficiency results from our testing and simulation. Our technique provides the ability to guarantee that the resource decisions made by the commander while satisfying the commander's need will also maximize the overall operational success in the theater without the need for centralized logistics optimization. In effect, we can naturally decompose and distribute the resource decisions for effective computation. This also allows us to provide the commander with informed knowledge concerning the expected impacts of such resource substitution decisions. For complete details of this model and experimental results, see Santos et al. [40, 41].

2.2 GTrans for mixed-initiative planning

To interface with its team of users, the MADGS system needed an interface that allowed the users to work together with the system to facilitate real-time operation and plan failures. Therefore, we used our theory of distributed goal satisfaction (goal transformation theory) and combined it with a graphical user interface (GTrans) for mixed-initiative team planning. To provide the automated planning portion, we incorporated the PRODIGY planner using a special agent (Prodigy/Agent) that interfaced to PRODIGY while also interfacing to the rest of the MADGS multiagent system ([2, 6, 7, 9, 17, 27, 28, 53]). First we discuss Prodigy/Agent followed by the GTrans user interface.

2.2.1 Prodigy/agent

To provide planning for MADGS, we decided to integrate a legacy planner instead of developing an agent-based planner from scratch. We chose to use PRODIGY [5, 46], which employs a state-space nonlinear planner and follows a means-ends analysis backward-chaining search procedure to reason about multiple goals and operators from its domain theory. We chose this system because it was designed as an architecture for testing classical theories of planning and learning and represents a stable platform within the planning community. Like most classical planners, it assumes complete knowledge and is deterministic. Such limitations represent a compromise traded for computational efficiency and are balanced in practice by the human-assisted replanning discussed below.

A PRODIGY domain theory is composed of a hierarchy of object classes and a suite of operators and inference rules that change the state of the objects. A planning problem is represented by an initial state (objects and propositions about the objects) and a set of goal expressions to achieve. Planning decisions consist of choosing a goal from a set of pending goals, choosing an operator (or inference rule) to achieve a particular goal, choosing a variable binding for a given operator, and deciding whether to commit to a possible plan ordering and to get a new planning state or to continue subgoaling for unachieved goals. Different choices give rise to alternative ways of exploring the search space.

However, as a legacy system, PRODIGY was not capable of directly interacting with the MADGS multi-agent system. To integrate PRODIGY into the multiagent system, we developed Prodigy/Agent. Prodigy/Agent [8, 13, 18, 19] (written in Allegro Common Lisp) "wraps" PRODIGY and allows it to behave as an independent agent. Prodigy/Agent communicates directly with PRODIGY via Lisp and uses KQML (Knowledge and Query Manipulation Language) [20] as its agent-communication language to interface with MADGS agents. A single copy of Prodigy/Agent can act as a general plan server that may be queried by any agent in the MADGS system. Multiple copies of Prodigy/Agent operate concurrently and coordinate their planning decisions with respect to resource limitations. The most important role for Prodigy/Agent in MADGS is in providing the underlying planning technology for the GTrans interface.

2.2.2 GTrans

To allow users to interact with MADGS, we developed the GTrans [6, 12, 52, 53] user interface. GTrans interacts directly with the Prodigy/Agent planner using mixed-initiative planning techniques. With GTrans, a human planner can focus on the goals, associated goal priorities, and resource to goal assignments (all of which can change over time) instead of low-level details. Rather than presenting planning as a search mechanism, we give the user a metaphor of planning as a goal manipulation problem [12]. The primary task for the user is to make decisions concerning goal change and management. By selecting goal changes, the user can reduce an initial goal to a slightly less demanding goal that partially achieves the state originally sought. Other selections can decompose a goal into a set of distributed subgoals, the achievement of which will satisfy the parent goal. PRODIGY provides background support to the user rather than making decisions regarding goal change itself.

From the commander's point of view, planning is achieved by manipulating the GTrans graphical user-interface to achieve objectives and project hypothetical situations. As such, it operationalizes an objectives-based planning model [26] and limits the detail thrust upon the human user because the focus of planning is change to the goal rather than the details needed to achieve them. Next, we discuss goal transformations and the GTrans user interface in more detail.

Goal transformations. In a dynamic environment, aspects that affect a plan and its execution may change at any point. Traditionally, this determines that a change to the plan be formulated that will allow the goal to be achieved when threatened. However, in many circumstances the goals themselves may need to change rather than the plan *per se* [11]. For example, it makes no sense to continue to pursue the goal of securing an airbase, if the battlespace has shifted to a distant location. At such a point, a robust planner must be able to alter the goal minimally to compensate. Otherwise, a correct plan to secure the old location will not be useful at execution time.

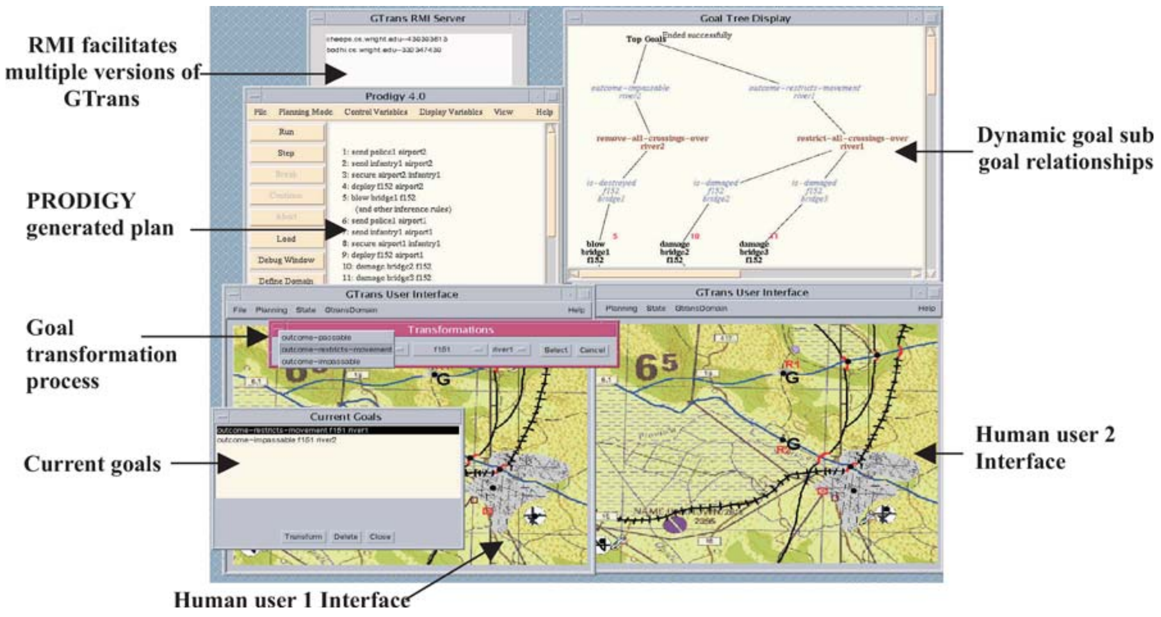
A *goal transformation* represents a goal shift or change. Conceptually it is a change of position for the goal along a set of dimensions defined by some abstraction hyperspace [11]. The hyperspace is associated with two hierarchies. First the theory requires a standard conceptual type-hierarchy within which instances are categorized. Such hierarchies arise in classical planning formalisms. They are used to organize arguments to goal predicates and to place constraints on operator variables. Goal transformation theory also requires a unique second hierarchy.

In normal circumstances the domain engineer creates arbitrary predicates when designing operator definitions. We require that these predicates be explicitly represented in a separate predicate abstraction hierarchy that allows goals to be designated along a varying level of specificity. For example consider the military domain. The domain-specific goal predicate *is-ineffective* takes an aggregate force unit as an argument (e.g., (is-ineffective enemy-brigade1)). This predicate may have two children in the goal hierarchy such as *is-isolated* and *is-destroyed*. The achievement of either will then achieve the more general goal [11]. Furthermore if the predicate *is-destroyed* had been chosen to achieve in-effective, the discovery of non-combatants in the battle area may necessitate a change to *is-isolated* in order to avoid unnecessary casualties. Note also that to defer the decision, the movement may be to the more general *is-ineffective* predicate. Then when the opportunity warrants and further information exists, the goal can be re-expressed. In any case, movement of goals along a dimension may be upward, downward or laterally to siblings.

Goal movement may also be performed by a change of arguments where the arguments exist as objects of or members of the standard object type-hierarchy. The goal represented as the type-generalized predicate (inside-truck Truck1 PACKAGE) is more general than the ground literal (inside-truck Truck1 PackageA). The former goal is to have some package inside a specific truck (thus existentially quantified), whereas the latter is to have a particular package inside the truck. Furthermore both of these are more specific than (inside-truck TRUCK PACKAGE). Yet movement is not fully ordered, because (inside-truck Truck1 PACKAGE) is neither more general or less general than (inside-truck TRUCK PackageA).

A further way goals can change is to modify an argument representing a value rather than an instance. For example the domain of chess may use the predicate outcome that takes an argument from the ordered set of values checkmate, draw, lose. Chess players often opt for a draw according to the game's progress. Thus to achieve the outcome of draw rather than checkmate represents a change of a player's goal given a deteriorating situation in the game.

GTrans user interface. To directly support the goal manipulation model used in MADGS, we implemented a mixed-initiative interface through which the user manipulates goals, the arguments to the goals, and other properties. The interface, written in Java, hides many of the planning algorithms and knowledge structures from the user and instead emphasizes the goal-manipulation process with a menu-driven and direct manipulation mechanism. GTrans presents a direct manipulation interface to the user that consists of a graphical map with drag and drop capability for objects superimposed upon the map surface. GTrans helps the user create and maintain a problem file that is internally represented as follows. A planning problem consists of an initial state (a set of objects and a set of relations between these objects) and a goal state (set of goals to achieve). The general sequence is (1) to



```
1: send police1 airport2
2: send infantry1 airport2
3: secure airport2 infantry1
4: deploy f152 airport2
5: blow bridge1 f152
   (and other inference rules)
6: send police1 airport1
7: send infantry1 airport1
8: secure airport1 infantry1
9: deploy f152 airport1
10: damage bridge2 f152
11: damage bridge3 f152
```

Fig. 2 The GTrans interface

create a planning problem, (2) invoke the underlying planner to generate a plan, and then until satisfied given planning feedback either (3a) change the goals or other aspects of the problem and request a plan or (3b) request a different plan for the same problem. Figure 2 shows the interface presented to the user.

We extended the GTrans user interface to allow multiple human planners to operate in teams. Each user has an independently executing copy of GTrans and a supporting component that allows plan manipulation on a common graphical map representation. The systems work independently of each other, yet they have the capacity to take initial conditions and goal descriptions from any of the other concurrent GTrans interfaces. GTrans uses sockets to communicate with Prodigy/Agent and with other executing copies of GTrans. The individual processes may be distributed on remote machines across a network. As such, we are able to fully integrate GTrans into the MADGS Carolina environment and provide mixed-initiative planning assistance to the human commander.

2.3 Carolina mobile agent server

For MADGS, the critical element for agent communications lies in the necessity of not having a *singular point of failure* in the system. This obviously means that we cannot afford to use centralized directories or look-up tables. Due to the network constraints we also cannot leave communications up to individual agents since this will effect the size of mobile agents and thereby bandwidth consumption. Another communications issue facing development of the MADGS system was location of agents and resources. Without centralized directories, look-up tables or the ability to farm communication concerns out to agent resources our research led to the quick determination that no available system answered the communication issues, resource tools or adaptability needed by our project. For this reason we embarked on the design and implementation of the Carolina agent server and mobile agent system. Unfortunately no generic agent system with sufficient basic communications and resource management abilities could be effectively utilized.

Systems identified made different assumptions and focused on a specific problem without offering some base functionality that all agent systems could generally use. It is hoped that one by-product of our work is to define the base functionality needed by all agent systems. This is not to say that all agent systems should or will be based on of our work given all systems do not work off general models.

The communication protocols we devised in Carolina attempt to insure several things:

1. The size of an agent will not increase significantly with increased interactions;
2. Each node will have a consistent view of the ‘world’;
3. The network load posed by communications can be minimized compared with alternate communication methods;
4. There is no central point of failure in the system; and,
5. All communications can be routed within a reasonable time frame.

These assurances can be made despite the volatile and intermittent nature of our network environment. The first assurance follows from the need for agents to only have an agent’s unique assigned name to maintain a communication link with that agent. The second assurance is possible because our system provides for information sharing between nodes that guarantees all nodes will know the exact location of all mobile resources (agents are resources) assuming no non-server resource movement for a time-period not to exceed some constant value τ. The third assurance is possible because our communication protocol negates the need for broadcast except for system-wide alerts that are hypothesized to be rare in any system. Since our protocol does not utilize a central look-up table or centrally located directories, there can be no single point of failure for communications, thus the fourth assurance. The last assurance stems from the second assurance. Since all nodes are guaranteed to know the location of all mobile-agents, it is possible to route ALL communications within a reasonable time frame.

The foundation of the MADGS system is the agent server named Carolina. Carolina has a three-tier architecture with several internal components that are described in this section. There are four main functions in Carolina:

1. Provide an agent execution environment;
2. Insure system integrity through role-based security techniques;
3. Allow access to system resources where appropriate; and,
4. Provide communication services that improve the overall system performance.

The first function is basic to all agent systems and needs no further mention. Carolina System Manager Component oversees the agent interactions to insure that only appropriate communications occur between different types of agents using cased-based reasoning techniques. Tied into this functionality is the access Carolina provides to system resources to requesting agents. These resources may only be accessed through proper channels, in other words through the proper agent since agents represent all resources. For example, an attempt to access a database directly instead of through the databases interface agent would meet with failure and cause retribution from Carolina as a hostile act. In this event, the System Manager will sever the agents thread and report the offending agents class and offense to a system user. The last function of Carolina currently revolves around communications. There are three things of interest in this area, server-to-server communications, agent migration, and agent-to-agent communications. Server-to-Server communications that is crucial to the operation of the MADGS system however only occurs twice, upon Server startup and shutdown. The primary function of the server-to-server communication is the location and interconnection the MADGS system, Carolina servers and the agents they host. Agent migrations in Carolina are handled as a form of communication though migrations are handled through a separate port than com-

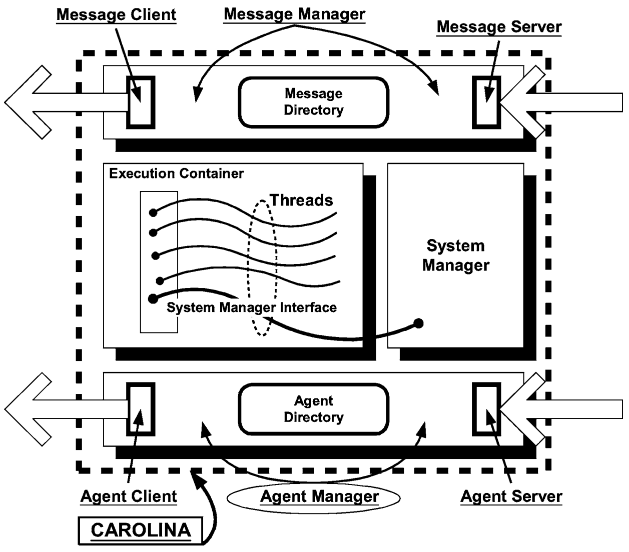


Fig. 3 Carolina architecture

munication messages. Since agents in the MADGS system are not allowed to directly communicate with another agent, Carolina must offer a service for local and remote communications.

The decision to not allow agents to directly communicate departs strikingly from the standard agent-based system protocols. The reason we take this unique stance is the limitations that occur when you allow an unlimited number of agents to freely migrate and communicate over a constantly changing network topology. Under such conditions the network traffic increases significantly as mobile-agents attempt to maintain the current location of other mobile-agents they are collaborating with. Given our network environment, bandwidth usage must be kept to a minimum; therefore management of this cannot be left to the individual agents. It becomes an issue of control and performance. Let us begin to examine the Carolina architecture (see Fig. 3).

Carolina receives agents through its AgentServer Port, which is controlled by the *AgentManager*. The *AgentManager* receives incoming agents and checks their intended destination (IP). If the intended destination is local, the *AgentManager* registers the agent in the AgentDirectory, deserializes it and passes it to the ExecutionContainer where the agent is provided with a thread for execution. If however, the agent's intended IP is not local, the *AgentManager* simply reroutes the serialized agent through the AgentClient Port after registering the transient agent in the AgentDirectory. The AgentDirectory is one of the key components of the Carolina communication scheme. It maintains data on all agents that the resident Carolina server has seen. Information stored in the AgentDirectory includes typical information including the agent's unique name assigned at creation, the agent's class, source IP, current (or last known) IP, and goal. Additionally, AgentDirectory stores a pointer to the messages stored in the Message Directory for individual agents.

Finally, since the MADGS system has been designed for use in a large-scale dynamic network architecture with massive numbers of mobile agents carrying out communications, point-to-point communications were not an option. The protocol we developed extends existing work [29]. Agents are not responsible for maintaining an address book of any kind. If communication is needed with a specific agent the agent only needs that agent's unique identification tag. Location of and routing of messages to agents in the network is performed as a joint effort between the servers and a *Communications Agent*. The server logs all agents entering or passing through the server in route to another node in the network. Carolina logs the agent name, unique identification tag, class, location or destination and time-stamp. In conjunction with this logging activity, *Communications Agents* use a random algorithm to canvas the network moving from server to server collecting the server's agent directory and compare it to their 'view' of the world. This agent then modifies (or cleans) the servers agent directory making additions, deletions and correction as needed.

2.4 AgentTool: Developing mobile agent in MADGS

To support MADGS, we created the *Multi-agent Systems Engineering* (MaSE) methodology and an agent development environment called *agentTool* [3, 14, 16]. The agentTool system is integrated into MADGS via the agentToolHandler agent. Using agentTool, developers follow MaSE, which guides them, step by step, through the analysis and design of complex, distributed, and dynamic multi-agent systems, such as MADGS. agentTool is a graphically based, interactive software engineering tool that fully supports MaSE. agentTool helps developers in specifying multi-agent organizations and then semi-automatically generating designs and correct, executable code. MaSE and agentTool are both independent of any particular agent architecture, programming language, or communication framework. Using agentTool, it is possible to generate implementations targeted at various frameworks without changing the design. To support the MADGS architecture, we have developed specific code generation modules to produce systems that work in the Carolina framework.

2.4.1 Multiagent systems engineering

The MaSE methodology is a specialization of more traditional software engineering methodologies. The general operation of MaSE follows the phases and steps shown below and uses the associated models. The goal of MaSE is to guide a system developer from an initial system specification to a multi-agent system implementation. This is accomplished by directing the developer through this set of inter-related system models.

Phases	**Models**
1. Analysis Phase	
a. Capturing Goals	Goal Hierarchy
b. Applying Use Cases	Use Cases, Sequence Diagrams
c. Refining Roles	Concurrent Tasks, Role Model
2. Design Phase	
a. Creating Agent Classes	Agent Class Diagrams
b. Constructing Conversations	Conversation Diagrams
c. Assembling Agent Classes	Agent Architecture Diagrams
d. System Design	Deployment Diagrams

Analysis phase. The goal of the MaSE analysis phase is to define a set of roles that can be used to achieve the system level goals. This process is captured in three steps: capturing goals, applying use cases, and refining roles.

- Capturing Goals. The first step is to capture the system goals by extracting them from the requirements, which is done by Identifying goals and Structuring goals. The purpose of the Identifying Goals is to derive the overall system goal and its subgoals. This is done by extracting scenarios from the requirements and then identifying scenarios goals. After the goals have been identified, the second step, Structuring Goals, categorizes and structures the goals into a goal tree, which results in a Goal Hierarchy Diagram that represents goals and goal/subgoal relationships.
- Applying Use Cases. In this step, goals are translated into use cases, which capture the previously identified scenarios with a detailed description and set of sequence diagrams. These use cases represent desired system behaviors and event sequences.
- Refining Roles. Refining Roles organizes roles into a Role Model [25], which describes the roles in the system and the communications between them (Fig. 4). Each role (denoted by boxes) is decomposed into a set of tasks (ovals), which are designed to achieve the goals for which the role is responsible. Communications between tasks are denoted by arrows pointing from the initiating task to the responding task. The behavior of these tasks is documented using finite state automata-base Concurrent Task Diagrams. Concurrent Task Diagram consist of a set of states and transitions that represent internal agent reasoning and communications. A special *move* action allows the analyst to specify agent mobility.

An example of a MaSE Concurrent Task Diagram is shown in Fig. 5, which shows the definition of the *CreateAgent* task from the Agent Generator role.

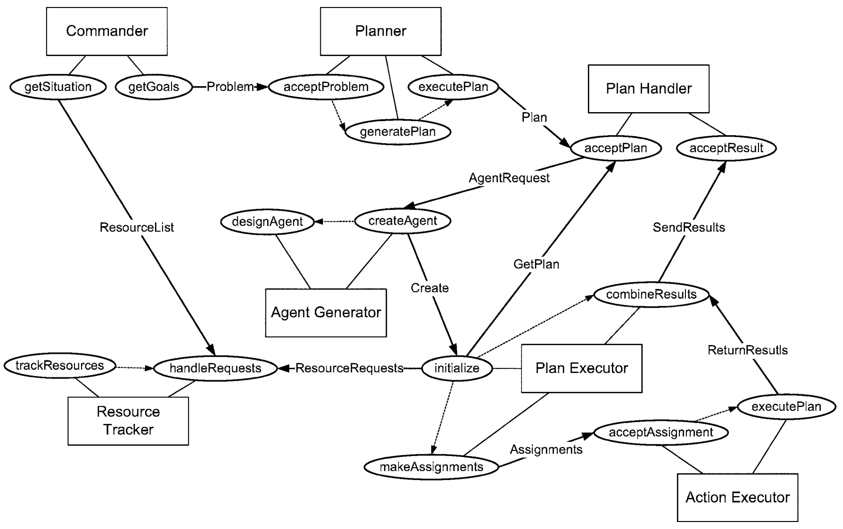


Fig. 4 MADGS role model

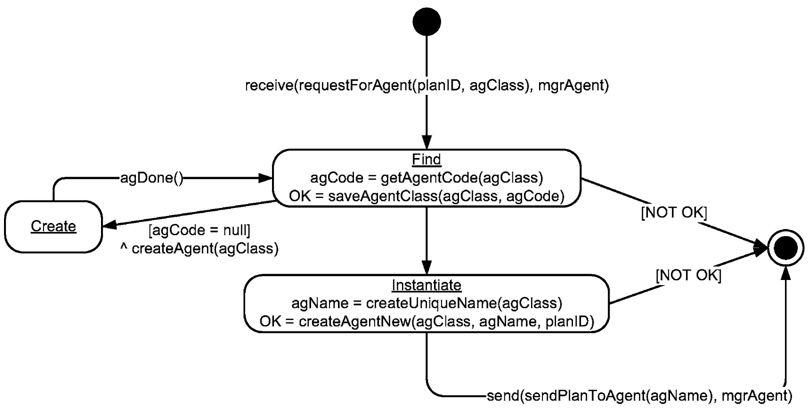

receive(requestForAgent(planID, agClass), mgrAgent)
Find
agCode = getAgentCode(agClass)
OK = saveAgentClass(agClass, agCode)
agDone()
Create
[agCode = null]
^ createAgent(agClass)
[NOT OK]
[NOT OK]
Instantiate
agName = createUniqueName(agClass)
OK = createAgentNew(agClass, agName, planID)
send(sendPlanToAgent(agName), mgrAgent)

Fig. 5 createAgent concurrent task diagram

The state transition syntax *trigger*(*args1*)[*guard*]/ *transmission*(*args2*) is interpreted to mean that if an event *trigger* is received with a number of arguments *args1* and the condition *guard* holds, then the message *transmission* is sent with the set of arguments *args2*. Actions may be performed in a state and are written as functions. Besides communicating with other agents, tasks can interact with the environment via reading percepts or performing operations that affect the environment. This interaction is typically captured by functions executed while in specific states.

Design phase. The purpose of the design phase is to take roles and tasks and to convert them into a form more amenable to implementation, namely agents and conversations. The MaSE design phase consists of four steps: designing agent classes, developing conversation, assembling agents and deploying the agents.

- *Construction of agent classes.* The first step in the design phase identifies agent classes and their conversations and then documents them in Agent Class Diagrams, as shown in Fig. 6, which depicts agent classes as boxes and the conversations between them as lines connecting the agent classes. The Agent Class Diagram that results from this step is similar to object-oriented class diagrams with two differences: (1) agent classes are defined by the roles instead of attributes and methods and (2) relations between agent classes are conversations. The *agentTool* and *PRODIGY* agent classes do not have assigned roles as they are legacy systems that work in conjunction with the *agentTool Handler* and *Prodigy Agent* agents to provide their functionality.
- *Constructing conversations.* Once the agent classes and the conversations are identified, the detailed conversation design is undertaken. Conversations model communications between two agent classes using a pair of finite state automata similar in form and function to concurrent tasks. Each task usually generates multiple conversations, as they require communication with more than one agent class. In the current version of agentTool, conversations may be verified to ensure that they do not deadlock, etc. This verification is straightforward as it is only done on single conversations; it does not check interactions between

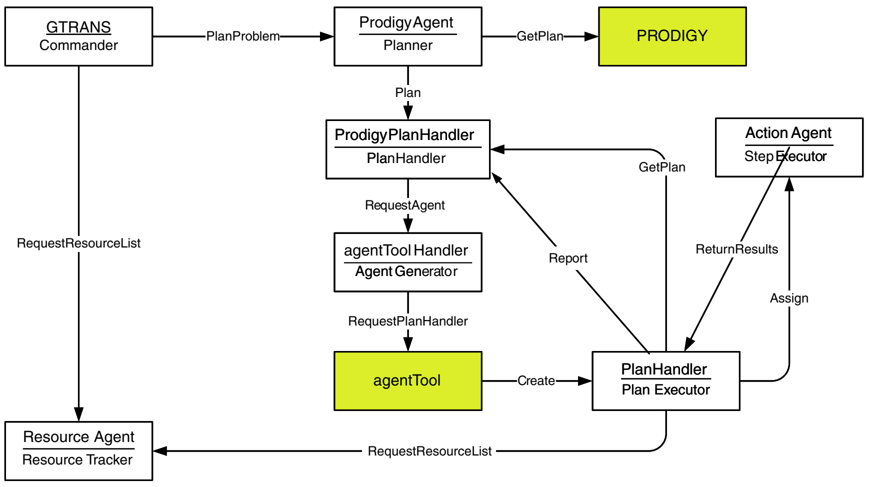


Fig. 6 MADGS agent class diagram

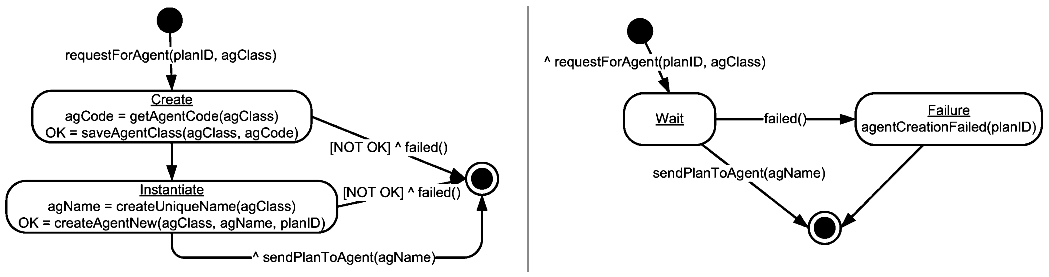

requestForAgent(planID, agClass)
Create
agCode = getAgentCode(agClass)
OK = saveAgentClass(agClass, agCode)
[NOT OK] ^ failed()
Instantiate
agName = createUniqueName(agClass)
OK = createAgentNew(agClass, agName, planID)
[NOT OK] ^ failed()
^ sendPlanToAgent(agName)
^ requestForAgent(planID, agClass)
Wait
failed()
Failure
agentCreationFailed(planID)
sendPlanToAgent(agName)

Fig. 7 RequestAgent conversation diagrams

conversations. Verification is done by converting the conversations into Promela code, which is verified using SPIN.

Both sides of the *RequestAgent* conversation are shown in Fig. 7. The initiator always begins the conversation by sending the first message. The syntax for Communication Class Diagrams is similar to that of Concurrent Task Diagrams except that conversations are binary exchanges between individual agents.

- *Assembling Agent Classes.* Assembling Agent Classes involves defining the agentsí internal architecture. MaSE does not assume any particular agent architecture and allows a wide variety of existing and new architectures to be used. The architecture is defined using components similar to those defined in UML.
- *System Design.* The final design step is to choose the actual configuration of the system, which consists of deciding the number and types of agents in the system and the platforms on which they should be deployed. These decisions are documented in a Deployment Diagram, which is similar to a UML Deployment Diagram, as shown in Fig. 8. The three dimensional boxes denote individual agents while the lines connecting them represent actual conversations. A dashed-line box encompasses agents that are located on the same physical platform.

2.4.2 agentTool

The goal of agentTool (Fig. 9) was to support and enforce the MaSE methodology and to supported automated agent synthesis. We approached automated agent synthesis along three major themes in agentTool: (1) automated transformation of analysis models into design models, (2) verification of analysis and design models prior to code generation, and (3) code generation for a variety of agent frameworks.

Analysis to design transformations. This section describes how agentTool transforms the analysis models into design models. The goal of these transformations is to take the analysis specification and transform it into a consistent design. The process is semi-automatic in that the designer initiates the transformation process and guides it along when help is required. The initial input to the transformations is the set of agent classes and the roles assigned to each agent class. The transformation system then generates the agent conversations as well as the internal agent component-based design based on the role model and tasks from the analysis phase. A detailed discussion of the automated analysis to design transformations, including a formal specification of the transformations, can be found in [45].

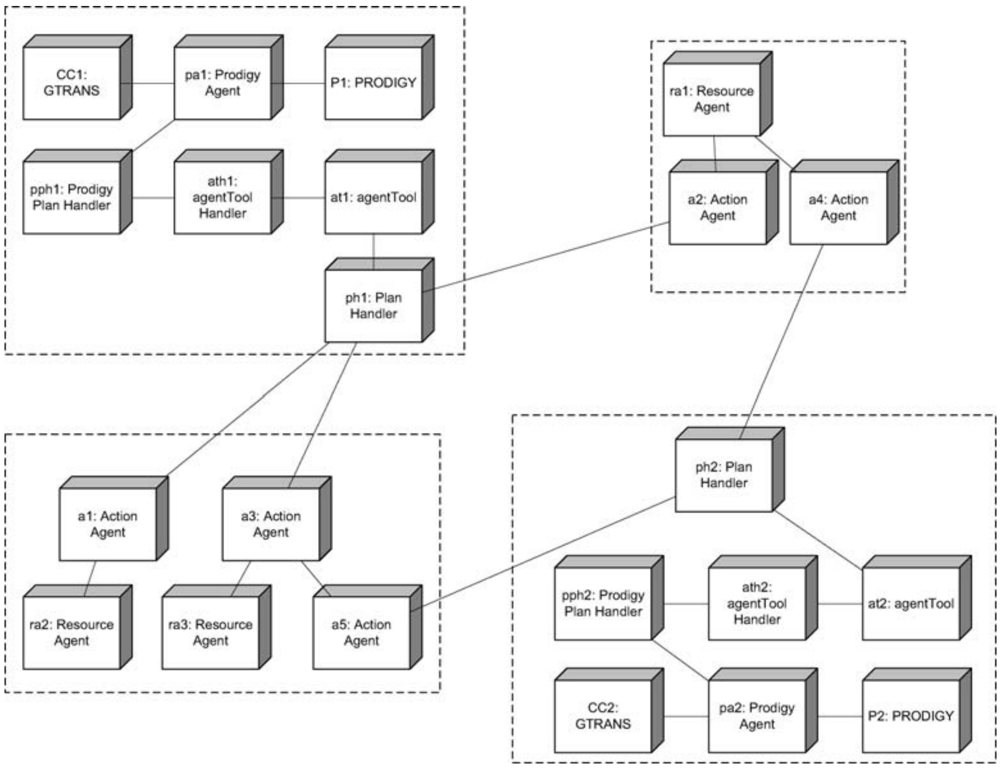
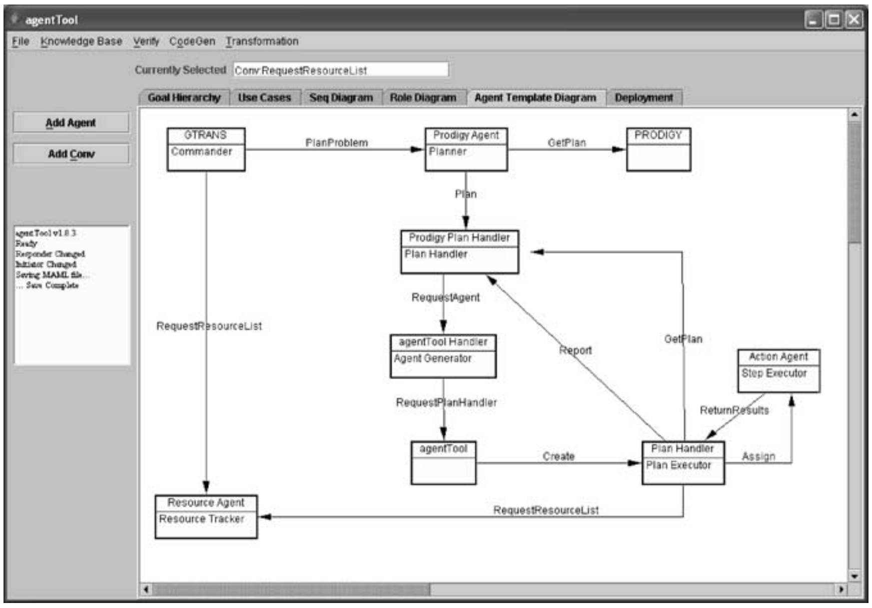


Fig. 8 MADGS deployment diagram

Fig. 9 agentTool


agentTool
File Knowledge Base Verify CodeGen Transformation
Currently Selected Conv:RequestResourceList
Goal Hierarchy Use Cases Seq Diagram Role Diagram Agent Template Diagram Deployment
Add Agent
Add Conv
GTRANS Commander
PlanProblem
Prodigy Agent Planner
GetPlan
PRODIGY
Plan
Prodigy Plan Handler Plan Handler
RequestAgent
agentTool Handler Agent Generator
RequestPlanHandler
agentTool
Create
Plan Handler Plan Executor
Report
GetPlan
Action Agent Step Executor
ReturnResults
Assign
RequestResourceList
Resource Agent Resource Tracker
RequestResourceList

```
agentTool v1.8.3
Ready
Responder Changed
Initiator Changed
Saving MAML file...
... Save Complete
```

A second set of transformations currently implemented in agentTool consists of transformations to add functionality required for mobility. In the analysis phase, mobility is specified using a *move* activity in the state of a concurrent task diagram. This *move* activity is copied directly into the associated component state diagram during the initial set of analysis-to-design transformation described above. During the mobility transformation, the existing design must be modified to coordinate the mobility requirements between all components in the agent design. According to the mobility design approach, the AgentComponent is responsible for coordinating the entire move and working with the external agent platform to save its current state and actually carry out the move. More detailed explanations of the design-to-design mobile agent transformations are contained in [42].

Automated verification. The second research area in multiagent system synthesis was the automatic verification of agent protocols. Since it is critical that distributed mission planning systems such as those envisioned by MADGS operate properly without extensive testing, efforts were focused on ensuring that the protocols used for agent communications were as reliable as possible before implementation. agentTool, via the Spin system [23], currently checks classic conversation centric errors including deadlocks, non-progress loops, livelock, infinite overtaking, unused messages, and mislabeled transitions. agentTool also has the capability to verify that Sequence Diagrams generated during the analysis phase can actually be generated by the system design [31, 32].

agentTool performs these verification tasks by generating Promela code [22] from the system design and passing that code to Spin. Spin then creates an analyzer to search the conversation state space, simulating all possible combination of messages in the conversation until either a deadlock condition occurs or the state space is exhausted. Conversations are considered deadlocked if they terminate in any state other than the end state. If a deadlock condition is detected, the analyzer writes a trace file that can be used to create a message sequence trace pinpointing the series of message events that led to the deadlock. Figure 10 shows the messages agentTool provides. All errors detected by agentTool are also displayed graphically by highlighting the state and/or transition that caused the error.

Code generation. The final focus of agentTool research was on the automated generation of code from design models. Currently, agentTool generates Java code that captures the structural aspects of the system as well as the communications protocols necessary to work within the Carolina framework. The AgentComponent, each component within the agents, and each conversation are generated as Java objects. To ensure the semantics of concurrent tasks are preserved, each component becomes a separate thread. The state machines that describe the components and the conversations are generated as the main function of each thread. The main functions have a single loop with a switch statement where each case captures a separate state. Figure 11 shows code generated for the main loop of the responder side of the *RequestAgent* conversation from Fig. 7. agentTool generated code has been used in a number of projects [15, 35] and was the basis for comparing the performance of mobile and static multiagent systems in [36].

```
********** DETAILED DEADLOCK INFORMATION **********

DEADLOCK CONDITION EXISTS IN THE FOLLOWING CONVERSATION:
Conversation Name = RequestAgent
Participant Name = Responder
Current State = Wait
State Transition = sendPlanToAgent

DEADLOCK CONDITION EXISTS IN THE FOLLOWING CONVERSATION:
Conversation Name = RequestAgent
Participant Name = Initiator
Current State = Instantiate
State Transition = null

********** TESTING COMPLETED **********
```

Fig. 10 Deadlock detection

```
Message m = new Message();
        int state = StartState;
        boolean notDone = true;
  .
  .
  .
   while (notDone)
   { switch (state)
     { case StartState :
         state = StartState_out;
         break;
       case StartState_out :
         m = new Message();
         m.performative = "requestForAgent";
         planID_agClass.planID = planID;
         planID_agClass.agClass = agClass;
         m.content = planID_agClass;
         sendMessage(m, output);
         state = Wait;
         break;
       case Wait :
         state = Wait_out;
         break;
       case Wait_out :
         m = readMessage(input);
         if (m.performative.equals("sendPlanToAgent"))
         { agName = m.content;
           notDone = false; }
         if (m.performative.equals("failed"))
           state = Failure;
         break;
       case Failure :
         parent.agentCreationFailed(planID);
         state = Failure_out;
         break;
       case Failure_out :
         notDone = false;
         break;
     }
   }
```

Fig. 11 Conversation code generation

agentToolHandler. In the current version of MADGS, system designers design and build MADGS mobile agents prior to deploying the MADGS system. The code for these agents is passed to the agentToolHandler who instantiates these agents upon request. In the future, to allow MADGS to evolve beyond its deployed configuration, we plan to allow the agentToolHandler to pass specifications for non-existent agents to the agentTool system where agentTool operators will design and create agents in real time from a set of existing components and communications protocols. As one might imagine, agentTool's verification and code generation capabilities are essential to this vision.

3 Putting it all together

To test our approach to distributed constraint satisfaction, we integrated our ideas discussed above into our prototype MADGS system. Specifically, we set out to demonstrate three capabilities:

- *Mixed-initiative planning and execution.* We achieve this through interactive goal transformation (GTrans) with the human planner and feedback from intelligent resource re-allocation during plan failures. We also provided support to the human user by intelligently re-organizing information that is made available to them.
- *Intelligent logistics assistance.* This is accomplished by assisting the human planner when resource requirements are not met for plan execution. The system intelligently and efficiently reallocates resources to provide alternatives to the human planner while helping mitigate potential conflicts that can arise from multiple missions and goals in the operational theater.
- *Mobile and multi-agent infrastructure deployment.* We demonstrate that the needs of the two items above can be satisfied in a systematic fashion through knowledge-based software engineering with the automatic generation and deployment of agent components. We focused on communications and mobility in order to efficiently deploy our prototype.

Our MADGS prototype was deployed on a mixture of hardware platforms. Our underlying Carolina agent environment was developed in Java and globally executed on Windows, Linux, and Solaris platforms with a mix of Pentium and Sparc CPUs. Our purpose in this mix was to demonstrate the ease of portability and mobility of agents using our approach for cross platform integration. As smaller mobile devices such as PDAs increase in power and capability, these results allow us to project that moving to smaller mobile systems will be readily realizable in the near future.

To illustrate the utility of the MADGS system to the commander, we have designed a simple empirical study: Two commanders are operating in the same theater with independent commands. Each commander is given a mission that potentially conflicts with the other primarily due to resource availability in the theater. We compose a number of problems where each commander is faced with a number of rivers, each spanned by one or more bridges. The goal is to make each river impassable by destroying the bridges across them. In order to accomplish this task, the commanders with the help of MADGS will assign F-15 tactical fighters from a pool of available aircraft placed at different locations. Each F-15 has the ability to destroy one bridge. However, each commander is only currently aware of the F-15s available near their immediate command. In this military interdiction scenario, commanders must schedule their strikes and will face shortages of F-15s that can only be resolved by communicating and negotiating with the other commander to best utilize all F-15s and achieve their respective missions. An example scenario with rivers and target bridges can be see in Fig. 12.

In this section, we first describe our MADGS interface and operations on one such problem scenario above. We then performed an empirical study to determine the overall efficiency and effectiveness of our replanning in the face of resource failure.

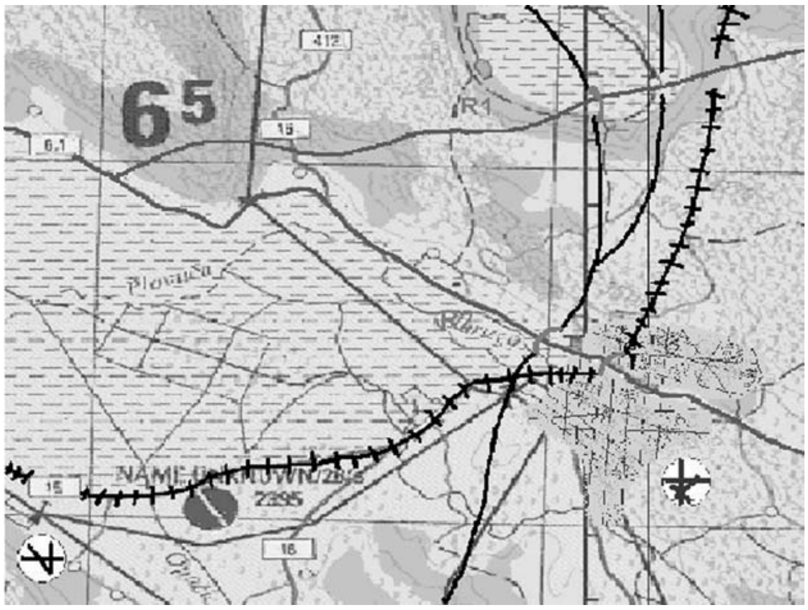
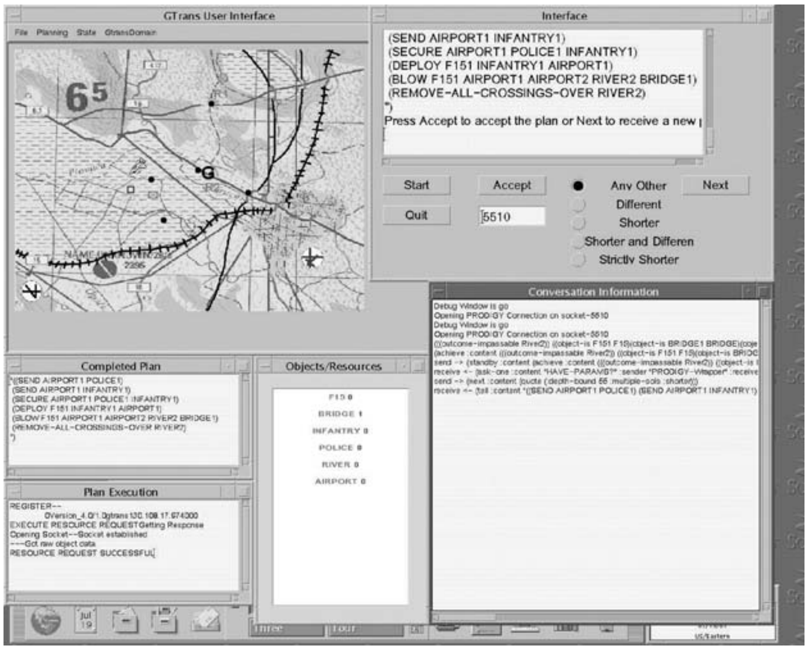


Fig. 12 Military interdiction scenario. Rivers have target bridges spanning them



```
(SEND AIRPORT1 INFANTRY1)
(SECURE AIRPORT1 POLICE1 INFANTRY1)
(DEPLOY F151 INFANTRY1 AIRPORT1)
(BLOW F151 AIRPORT1 AIRPORT2 RIVER2 BRIDGE1)
(REMOVE-ALL-CROSSINGS-OVER RIVER2)
")
Press Accept to accept the plan or Next to receive a new
```

5510

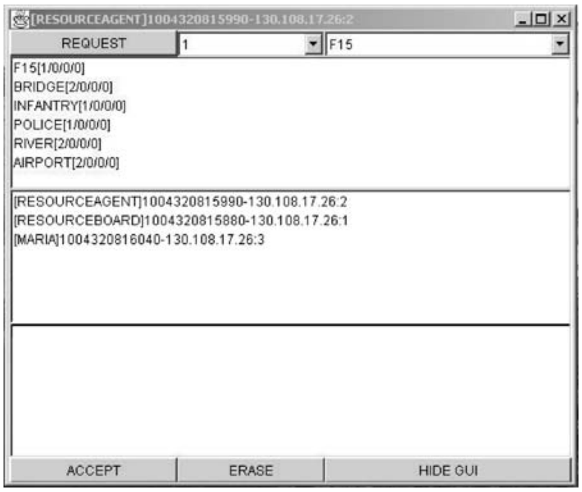
```
Debug Window is go
Opening PRODIGY Connection on socket-5510
Debug Window is go
Opening PRODIGY Connection on socket-5510
(((outcome-impassable River2)) ((object-is F151 F15)(object-is BRIDGE1 BRIDGE)(obje
(achieve :content (((outcome-impassable River2)) ((object-is F151 F15)(object-is BRIDC
send -> (standby :content (achieve :content (((outcome-impassable River2)) ((object-is I
receive <- (ask-one :content "HAVE-PARAMS?" :sender "PRODIGY-Wrapper" :receive
send -> (next :content (quote (:depth-bound 55 :multiple-sols :shorter)))
receive <- (tell :content "((SEND AIRPORT1 POLICE1) (SEND AIRPORT1 INFANTRY1)
```

```
"((SEND AIRPORT1 POLICE1)
 (SEND AIRPORT1 INFANTRY1)
 (SECURE AIRPORT1 POLICE1 INFANTRY1)
 (DEPLOY F151 INFANTRY1 AIRPORT1)
 (BLOW F151 AIRPORT1 AIRPORT2 RIVER2 BRIDGE1)
 (REMOVE-ALL-CROSSINGS-OVER RIVER2)
")
```

```
F15 0
BRIDGE 1
INFANTRY 0
POLICE 0
RIVER 0
AIRPORT 0
```

```
REGISTER--
        0Version_4.0/1.0gtrans 130.108.17.674000
EXECUTE RESOURCE REQUESTGetting Response
Opening Socket--Socket established
---Got raw object data
RESOURCE REQUEST SUCCESSFUL
```

Fig. 13 Plan execution commences after initial plan generated



```
F15[1/0/0/0]
BRIDGE[2/0/0/0]
INFANTRY[1/0/0/0]
POLICE[1/0/0/0]
RIVER[2/0/0/0]
AIRPORT[2/0/0/0]
```

```
[RESOURCEAGENT]1004320815990-130.108.17.26:2
[RESOURCEBOARD]1004320815880-130.108.17.26:1
[MARIA]1004320816040-130.108.17.26:3
```

Fig. 14 Resource allocation and negotiation

Figure 2 depicts our GTrans interface that is used by the human commander. For our prototype, we provide a GTrans interface to each user with the added capability of observing other human commanders and their activities to assist in coordination when necessary. This is depicted in the bottom right window in Fig. 2. As we can see, the mission goals (current goals) are identified and the commander interacts with PRODIGY to generate their initial plans with input from the latest information concerning available resources. Once the plan is generated (as seen in window labeled Prodigy 4.0), execution of the plan begins and resources are allocated to begin scheduling of the plan. Figure 13 shows the final confirmation of the plan by the commander together with the resource scheduling and execution process. The plan is then sent for execution with agentTool generating new agents in Carolina to carry out the plan execution. Figure 14 shows the DGS in action negotiating for resources as needed. Given multiple human commanders and missions, the resource requests are handled using our intra-organizational logistics models. Should a plan failure occur at any point, the commander is notified and either resource substitutions are recommended or replanning commences.

We now describe our empirical evaluations of our MADGS system.

3.1 Evaluation

Here, we consider two experiments to gauge the usefulness of MADGS using the scenarios we generated above. Our first experiment considers the usability of our GTrans interface, independent of the MADGS system. The second experiment studies the effectiveness of our overall MADGS approach to mixed-initiative replanning.

3.1.1 GTrans evaluation

An experiment was performed with human subjects to compare and contrast the models of search and goal manipulation [12, 52]. The first model is represented by users solving problems in an old interface used in the original PRODIGY planner [10] and the second model is represented by GTrans users. The experiment was designed to evaluate the differences of the two models under differing amount of task complexity using both expert and novices. This experiment uses 18 problems in the military domain as test problems. In these problems, insufficient resources exist with which to solve problems completely. Choices can be made, however, so that a solution is produced that achieves a *partial goal satisfaction* represented as a ratio of the subject's partial solution to the optimal partial solution.

Here we introduce a problem that illustrates the trade-offs between resource allocation decisions in the military planning domain. The "Bridges Problem" is simply to make rivers impassable by destroying all bridges across them. This task universally quantifies the variable <crossing> with the relation (enables-movement-over <crossing> river) and requires a separate air unit for each crossing in order to achieve the goal. We simplify the problem by assuming that an air resource can destroy one bridge and damage an arbitrary number of others. When a goal state is composed of conjunctive goals for multiple rivers, an interesting trade-off exists for the experimental subject. To maximize the goal satisfaction, an optimal user will allocate resources to rivers with fewer crossings [17].

The Bridges Problem is an instance of a class of problems with the following form

$$\forall x|goal(x)\forall y|subgoal(y,x)\exists r|$$
$$((resources(r) \wedge P(r,y)) \vee p'(y))$$

where $value(p'(y)) = \alpha(value(p(r,y)))$ and $0 \geq \alpha \geq 1$.

The problem is to maximize the value of the quantified expression above. That is for all goals, x, and for all subgoals of x, y, the problem is to maximize the value of the conjunction of disjunctions $p(r,y)$ or $p'(y)$ using some resources r where the value of the predicate p' is some fraction between 0 and 1 of the value of p. In the case of the Bridges Problem, the goals are to make impassable rivers x by the predicate *is-destroyed* applied to bridges y over the rivers or by the predicate *is-damaged* where a damaged bridge is considered 50% destroyed ($\alpha = 0.5$). Thus given enough resources and the goal to make a river impassable, it is enough to destroy all bridges across it. Without sufficient resources, a plan to destroy most bridges and damaging the rest results in maximally restricting the movement across it.

Figure 12 shows terrain with two rivers, the upper river having three bridges and the lower two. If four air units (resources) exist for use, the subject should allocate two resource units to the two-bridge river and two to the three-bridge one. By doing so the goal to make the first river impassable is achieved completely, whereas the second goal is more fully satisfied than if the reverse allocation was performed. Note that this determines a change to the second goal rather than the first. By transforming the goal (outcome-impassable river2) to (outcome-restricts-movement river2) the planner is able to achieve success.

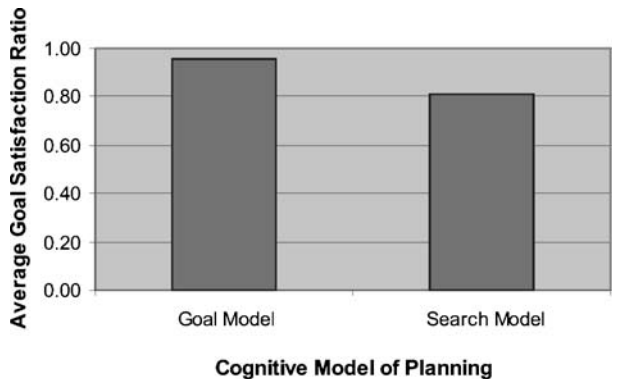


Fig. 15 Goal satisfaction as a function of cognitive model

GTrans provides a unique facility to transform or steer goals in such situations. When the subject receives feedback from the underlying Prodigy/Agent planner (either a successful plan that may not be acceptable by the subject, a failure to generate any plan, or an apparent problem due to the planner taking too long), the subject can asynchronously modify the goals and send them back to the planner for another round of planning. The subject does this by graphically manipulating the goals by selecting the "Change Goals" choice on the Planning pull-down menu.

The graph in Fig. 15 shows the mean of the goal satisfaction ratio under the goal manipulation model and the search model. When presented with the goal manipulation model, subjects achieved over 95% goal satisfaction on average. When presented with the search model, subjects achieved about 80% goal satisfaction on average.

Given that the cognitive model itself is an important factor as concluded in previous analysis, we next examine the possible relationships among three independent variables: planning model, problem complexity, and subject expertise. Figure 16 plots the average goal satisfaction ratio for each combination of the planning model and the problem complexity. As can be observed from the graph, when the goal manipulation model is presented to the user, the goal satisfaction

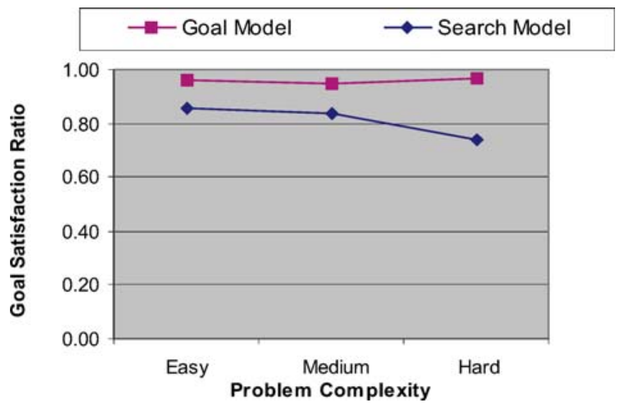


Fig. 16 Goal satisfaction as a function of problem complexity

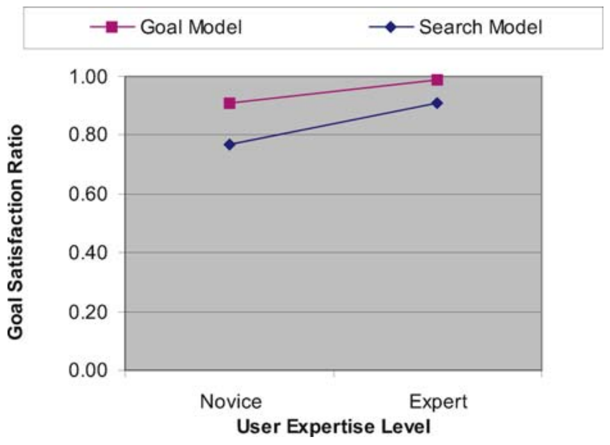


Fig. 17 Goal satisfaction as a function of expertise

ratio generally remains the same with increasing problem complexity; but when the search model is presented to the user, the goal satisfaction ratio decreases as the problem complexity increases. It is very likely that the effect of the planning model on the user performance depends on the problem complexity.

The next step of our analysis was to examine the possible interaction effects between the planning model and the user expertise level. Figure 17 shows the average goal satisfaction ratio for each combination of the planning model and the user expertise level. It is apparent that experts perform better than novices under both planning models. But the two plot lines representing each planning model are not parallel, indicating the possible interactions between the two factors.

3.1.2 Re-planning study

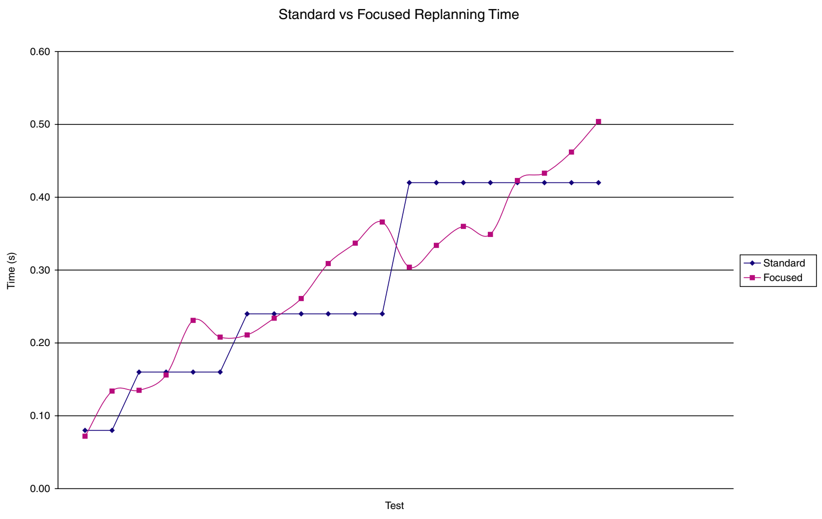
When most planning systems have to replan given changing environmental conditions such as resource constraints, the system replans completely. That is, if only a single goal fails, the system will still replan for all goals to generate a new solution. Instead, we have extended the Prodigy/Agent component of MADGS to identify the source of the failure and to replan for only that part of the plan that is affected by such failure. We call these two conditions standard replanning and focused replanning.

Our hypotheses is that MADGS will be more efficient in terms measured by the total number of search nodes (planning choice points) expanded and by the total planning time. The first dependent variable measures planning effort directly, because the fewer nodes expanded in the search tree, the fewer poor planning decisions are made. On the other hand, time is an indirect measure because factors other than that spent in planning may affect performance. For example on networked systems, network traffic has an effect.

Using our scenario template above, we generate a number of problems for this experiment. Each problem description contained $2n$ F-15s, where n is the number of rivers in the problem.

In order to test the replanning efficiency of each system (standard vs. focused), 20 tests were run. Each test came from one of four series, containing 2, 4, 6, or 8 rivers. For each series, the number of resource failures was varied from 1 to n. This means that for each test, two plans were computed by the underlying Prodigy/Agent planner within GTrans. The plans are the completely recomputed plan and the revised plan which solves the resource failure only. For example, Test 2.1 has two rivers, and one resource failure. This means that the planner assigns an F-15 to each bridge, and is then notified by Carolina that one of the F-15s assigned is no longer available. This causes the planner to replan for the missing aircraft. Similarly, Test 2.2 has two rivers, but 2 resource failures. The case in which no resource failures occur was not tested since both systems use the same strategy and perform identically in this case. Figures 18 and 19 depict the time spent on replanning with and without GTrans and the number

Fig. 18 Replanning time for standard (without GTrans) and focused (with GTrans)



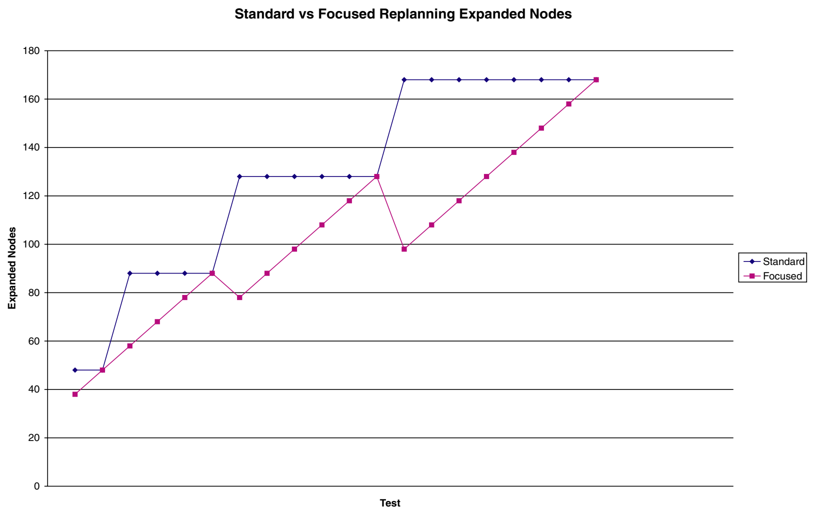


Fig. 19 Replanning nodes explored for standard and focused

of search nodes expanded during the replanning process, respectively.

For the small problems run with these tests, MADGS under the focused condition does not show a significant time savings over the standard condition. In many cases, Tests 2.2, 4.3, 4.4, 6.3–6.6, and 8.6–8.8, the focus condition requires more time. This can be attributed to two factors: The first is that for small problems like these, the computational overhead of identifying the removable goals is greater than the savings of replanning for a smaller problem. Note that with greater problem complexity, the performance measured by time begins to improve. Future testing should be run to show if this is true by evaluating the two systems on large, real-world problems where the replanning time would be significant. The other factor affecting the times is the planning environment itself. Each of our tests was run on our lab server. Due to lack of resources, the machine under test was being used by other people, causing times to fluctuate depending on the load. Future testing should be carried out on a separate machine in single user mode in order to isolate the tests from these kinds of problems.

Even with the time overhead issues, the focused condition did have significantly better direct performance over the standard condition when measured by the number of search nodes expanded. Indeed, at no time does the standard condition surpass the focused condition. At most it equals the performance. This is to be expected since the replanning done by the focused condition is only on a subset of the original problems.

4 Conclusion

We presented the MADGS framework for multi-commander dynamic mission planning and execution. We described the major elements of MADGS and illustrated the fundamental logistical and planning support that MADGS can provide to the battlefield commander through our case study. MADGS is currently capable of handling more complicated scenarios which include complex resource substitutions and intelligent allocation recommendations to the commander. Furthermore, in the face of imminent plan failure, MADGS also assists the commander through goal transformations in order to best achieve their given mission. The generation and deployment of agents via agentTool provides a dynamic, efficient, and robust environment that captures the changing nature of battlefield conditions.

We also detailed our prototype implementation of MADGS in order to provide a proof-of-concept behind our ideas as well as a better understanding of the complexities involved in such a system. The scenario we utilized in our prototype demonstrates our MADGS concept and the interactions between the major elements in MADGS. In particular, the scenario helps to highlight the assistance provided to the user (commander) and, more generally, the idea of replanning in the face of resource failures. Finally, another aim of this paper has been to provide a broad overview of the research conducted in carrying out the MADGS project.

One of the elements we intend to pursue next is to address the practical concerns of explaining the choices made (and rejected) by MADGS on recommendations to the commander. Such explanations are doubly critical during on-the-fly planning with partial execution. Hence, we must track the rationale for decisions in order to know when a decision has become inappropriate, due to the changes (either internal or external to the planning system). This will provide the needed transparency of understanding and context to the human commander with regards to MADGS' recommendations.

We must also realize that such complex mission planning will involve disparate types of units/entities; and simply providing a complete global explanation of the entire plan is often counter-productive and counter-intuitive. Most global information in the overall planning is often irrelevant at the lower levels. Thus, the appropriate context must be generated in the explanations with respect to the planning level of the commander(s) we are currently assisting.

The future of assisting the commander on and off the battlefield relies on a foundational understanding of the dynamics of the environment and the requirements of achieving various mission objectives. The MADGS project focuses on addressing the issues of multi-commander, multi-mission planning and execution. In particular, within a single theater of operation, multiple missions among multiple commanders which are potentially in conflict or in competition with one another.

Finally, in the MADGS project, we have also kept an eye toward the current technology trends such as portable mobile computing devices like PDAs, etc. and the potential capabilities that are needed to achieve a MADGS environment. We have envisioned that in the near future, with an understanding of the issues through MADGS, our military commanders, logistics officers, and intelligence analysts can be "armed" with PDAs that provide the critical information they need, when they need it, and also help them in their decision-making. Issues such as incomplete or uncertain information (fog of war) are naturally addressed through a MADGS framework. This paper has detailed the results of our MADGS project and also identified the next steps that should be pursued in reaching our ultimate vision.

Acknowledgments This project was supported in part by Air Force Office of Scientific Research Grant No. F49620-99-1-0244. We would also like to especially thank the reviewers for their comments which greatly improved this article.

References

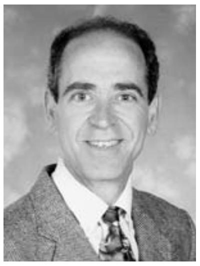
1. Barbuceanu M, Fox MS (1996) Coordinating multiple agents in the supply chain. In: Proceedings of the Fifth Workshops on Enabling Technology for Collaborative Enterprises(WET ICE'96), IEEE Computer Society Press, pp 134–141
2. Brown S, Cox MT (1999) Planning for information visualization in mixed-initiative systems. In: Cox MT (ed) Proceedings of the 1999 AAAI-99 Workshop on Mixed-Initiative Intelligence, AAAI Press, Menlo Park, CA, pp 2–10
3. Bryson J, Decker K, DeLoach SA, Huhns M, Wooldridge M (2001) Agent development tools. In: Intelligent Agents VII–Proceedings of the 7th International Workshop on Agent Theories, Architectures, and Languages (ATAL'2000), 2000 Springer Lecture Notes in AI, Springer Verlag, Berlin
4. Caldwell B (1995) Managing your inventory. Information WEEK 554:88
5. Carbonell JG, Blythe J, Etzioni O, Gil Y, Joseph R, Kahn D, Knoblock C, Minton S, Perez A, Reilly S, Veloso MM, Wang X (1992) Prodigy4.0: The manual and tutorial. Technical Report CMU-CS-92-150, Computer Science Department, Carnegie Mellon University
6. Cox MT (2000) A conflict of metaphors: modeling the planning process. In: Proceedings of 2000 Summer Computer Simulation Conference, 2000. The Society for Computer Simulation, San Diego, CA, pp 666–671,
7. Cox MT (2000) Interfaces for mixed-initiative planning. In: IUI'2000 Workshop on Using Plans in Intelligent User Interfaces, Cambridge, MA, MERL
8. Cox MT, Edwin G, Balasubramanian K, Elahi M (2001) Multiagent goal transformation and mixed-initiative planning using Prodigy/Agent. In: Proceedings of the 5th World Multiconference on Systemics, Cybernetics and Informatics, vol VII, pp 1–6
9. Cox MT, Kerkez B, Srinivas C, Edwin G, Archer W (2000) Toward agent-based mixed-initiative interfaces. In: Arabnia HR (ed) Proceedings of the 2000 International Conference on Artificial Intelligence, CSREA, vol. 1 Press, pp 309–315
10. Cox MT, Veloso MM (1997) Supporting combined human and machine planning: an interface for planning by analogical reasoning. In: Case-Based Reasoning Research and Development: Second International Conference on Case-Based Reasoning, pp 531–540
11. Cox MT, Veloso MM (1998) Goal transformations in continuous planning. In: Proceedings of the AAAI Fall Symposium on Distributed Continual Planning, AAAI Press
12. Cox MT, Zhang C (2005) Planning as mixed-initiative goal manipulation. In: Biundo S, Myers K, Rajan K (eds) Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling, AAAI Press, Menlo Park, CA, pp 2282–2291
13. Cox MT, Elahi M, Cleereman K (2003) A distributed planning approach using multiagent goal transformations. In: Proceedings of the 14th Midwest Artificial Intelligence and Cognitive Science Conference, pp 18–23
14. DeLoach SA (2001) Analysis and design using mase and agenttool. In: Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001), Miami University, Oxford, Ohio, March 31–April 1 2001
15. DeLoach SA, Matson ET, Li Y (2003) Exploiting agent oriented software engineering in the design of a cooperative robotics search and rescue system. The International Journal of Pattern Recognition and Artificial Intelligence
16. DeLoach SA, Wood M (2001) Developing multiagent systems with agenttool. In: Intelligent Agents VII - Proceedings of the 7th International Workshop on Agent Theories, Architectures, and Languages (ATAL'2000), 2000 Springer Lecture Notes in AI, Springer Verlag, Berlin
17. Edwin G, Cox MT (2001) Resource coordination in single agent and multiagent systems. In: Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence, IEEE Computer Society, Los Alamitos, CA, pp 18–24
18. Elahi MM (2003) A distributed planning approach using multiagent goal transformations. Master's thesis, Wright State University
19. Elahi MM, Cox MT (2003) User's manual for Prodigy/Agent, Ver 1.0. Technical Report WSU-CS-03-02, Dept. of Computer Science and Engineering, Wright State University
20. Finin T, McKay D, Fritzson R (1992) An overview of KQML: a knowledge query and manipulation language. Technical report, Computer Science Department, University of Maryland
21. Fox MS, Chionglo JF, Barbuceanu M (1993) The integrated supply chain management system Technical report, University of Toronto
22. Holzmann GJ (1991) Design and validation of computer protocols. Prentice Hall
23. Holzmann GJ (1997) The model checker SPIN. IEEE Trans Softw Engi 23(5):279–295
24. Kataoka N, Koizumi H, Simizu H (1997) Architecture of an autonomous distributed system and verification of implementation as a logistics information management system. In: Proceedings of the 3rd International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'97)
25. Kendall E (1988) Agent roles and role models: new abstractions for multiagent system analysis and design. In: Proceedings of the

International Workshop on Intelligent Agents in Information and Process Management, Bremen, Germany
26. Kent GA, Simons WE (1994) Objective-based planning. In: Davis PK (ed), New challenges for defense planning: rethinking how much is enough RAND, pp 59–71
27. Kerkez B, Cox MT (2000) Planning for the user interface: window characteristics. In: Proceedings of the 11th Midwest Artificial Intelligence and Cognitive Science Conference, AAAI Press, Menlo Park, MA, pp 79–84
28. Kerkez B, Cox MT, Srinivas C (2000) Planning for the user interface: Window content. In: Arabnia HR (ed), Proceedings of the 2000 International Conference on Artificial Intelligence CSREA Press, vol. 1, pp 345–351
29. Kramer KH, Minar N, Maes P (1999) Tutorial: mobile software agents for dynamic routing. http://www.media.mit.edu/ nelson/research/routes/sigmobile.ps
30. Kutanoglu E, Wu SD (1999) An auction-theoretic modeling of production scheduling to achieve distributed decision making. PhD thesis, Dept. of Industrial and Manufacturing Systems Engineering, Lehigh University
31. Lacey TH (2000) A formal methodology and technique for verifying communication protocols in a multi-agent environment. Afit/eng/00m-12, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base Ohio, USA
32. Lacey TH, DeLoach SA (2000) Automatic verification of multiagent conversations. In: Proceedings of the 11th Annual Midwest Artificial Intelligence and Cognitive Science Conference, Fayetteville, Arkansas
33. Liu J-S, Sycara KP (1997) Coordination of multiple agents for production management. Ann Oper Res 75:235–289
34. Luh PB, Hoitomt DJ (1993) Scheduling of manufacturing systems using the lagrangian relaxation technique. IEEE Trans Autom Contr 38:1066–1080
35. McDonald JT, Talbert ML, DeLoach SA (2000) Heterogeneous database integration using agent oriented information systems. In: Proceedings of the International Conference on Artificial Intelligence (IC-AI'2000), Monte Carlo Resort, Las Vegas, Nevada June 26–29 2000
36. O'Malley SA, Self AL, DeLoach SA (2000) Comparing performance of static versus mobile multiagent systems. In: National Aerospace and Electronics Conference (NAECON), Dayton, OH, October 10–12, 2000
37. QuÈinnec P, Padiou G (1993) Flight plan management in a distributed air traffic control system. In: First International Symposium on Autonomous Decentralized Systems (ISADS-93)
38. Saba GM, Santos E, Jr (2000) The multi-agent distributed goal satisfaction system. In: Proceedings of the International ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce, pp 389–394
39. Sadeh NM, Hildum DW, Kjenstad D, Tseng A (1999) Mascot: an agent-based architecture for coordinated mixed-initiative supply chain planning and scheduling. In: Proc. 3rd Int'l. Conf. on Autonomous Agents (Agents'99) Workshop on Agent-Based Decision Support for Managing the Internet-Enabled Supply Chain, Seattle, WA
40. Santos E, Jr, Zhang F, Luh PB (2001) Multi-agent logistics management. In: Proceedings of the International Conference on Internet Computing (IC '2001), pp 240–246
41. Santos E, Jr, Zhang F, Luh PB (2003) Intra-organizational logistics management through multi-agent systems. Electr Comm Res 3:337–364
42. Self A (2001) Design and specification of dynamic, mobile, and reconfigurable multiagent systems. Afit/eng/01m-11, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base Ohio, USA
43. Shen W, Norrie DH (1998) An agent-based approach for manufacturing enterprise integration and supply chain management. In: Jacucci G (ed), Globalization of manufacturing in the digital communications era of the 21st century: innovation, agility, and the virtual enterprise Kluwer Academic Publishers, pp 579–590
44. Shi L, Chen C-H, Ycesan E (1999) Simultaneous simulation experiments and nested partition for discrete resource allocation in supply chain management. In: The 1999 Winter Simulation Conference (WSC'99)
45. Sparkman CH (2001) Transforming analysis models into design models for the multiagent systems engineering methodology. Afit/eng/01m-21, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base Ohio, USA
46. Veloso MM, Carbonell JG, Perez A, Borrajo D, Fink E, Blythe J (1995) Integrating planning and learning: The PRODIGY architecture. J Theor Exper Artif Intell 7(1):81–120
47. Wagner T, Garvey A, Lesser VR (1997) Design-to-criteria scheduling: Managing complexity through goal-directed satisficing. In: Proceedings of the AAAI Workshop on Building Resource-Bounded Reasoning Systems
48. Walsh WE, Wellman MP (1998) A market protocol for decentralized task allocation. In: Proceedings of the Third International Conference on Multi-Agent Systems, Paris, France, pp 325–332
49. Wellman MP (1996) Market-oriented programming: Some early lessons. In: Clearwater S (ed) Market-based control: a paradigm for distributed resource allocation, chapter 4 World Scientific
50. Yung SK, Yang CC (1999) A new approach to solve supply chain management problem by integrating multi-agent technology and constraint network. In: Proceedings of the 32nd Hawaii International Conference on System Sciences(HICSS-1999), Maui, Hawaii
51. Zeng DD, Sycara K (1999) Dynamic supply chain structuring for electronic commerce among agents. In: Klusch M, (ed) Intelligent information agents, Springer
52. Zhang C (2002) Cognitive models for mixed-initiative planning. Master's thesis, Wright State University
53. Zhang C, Cox MT, Immaneni T (2002) GTrans version 2.1 user manual and reference. Technical Report WSU-CS-02-02, Dept. of Computer Science and Engineering, Wright State University

Eugene Santos, Jr. received the B.S. degree in mathematics and Computer science and the M.S. degree in mathematics (specializing in numerical analysis) from Youngstown State University, Youngstown, OH, in 1985 and 1986, respectively, and the Sc.M. and Ph.D. degrees in computer science from Brown University, Providence, RI, in 1988 and 1992, respectively.

He is currently a Professor of Engineering at the Thayer School of Engineering, Dartmouth College, Hanover, NH, and Director of the Distributed Information and Intelligence Analysis Group (DI^{2}AG). Previously, he was faculty at the Air Force Institute of Technology, Wright-Patterson AFB and the University of Connecticut, Storrs, CT. He has over 130 refereed technical publications and specializes in modern statistical and probabilistic methods with applications to intelligent systems, multi-agent systems, uncertain reasoning, planning and optimization, and decision science. Most recently, he has pioneered new research on user and adversarial behavioral modeling. He is an Associate Editor for the IEEE Transactions on Systems, Man, and Cybernetics: Part B and the International Journal of Image and Graphics.

Scott DeLoach is currently an Associate Professor in the Department of Computing and Information Sciences at Kansas State University. His current research interests include autonomous cooperative robotics, adaptive multiagent systems, and agent-oriented software engineering. Prior to coming to Kansas State, Dr. DeLoach spent 20 years in the US Air Force, with his last assignment being as an Assistant Professor of Computer Science and Engineering at the Air Force Institute of Technology. Dr. DeLoach received his BS in Computer Engineering from Iowa State University in 1982 and his MS and PhD in Computer Engineering from the Air Force Institute of Technology in 1987 and 1996.



Michael T. Cox is a senior scientist in the Intelligent Distributing Computing Department of BBN Technologies, Cambridge, MA. Previous to this position, Dr. Cox was an assistant professor in the Department of Computer Science & Engineering at Wright State University, Dayton, Ohio, where he was the director of Wright State's Collaboration and Cognition Laboratory. He received his Ph.D. in Computer Science from the Georgia Institute of Technology, Atlanta, in 1996 and his undergraduate from the same in 1986. From 1996 to 1998, he was a postdoctoral fellow in the Computer Science Department at Carnegie Mellon University in Pittsburgh working on the PRODIGY project. His research interests include case-based reasoning, collaborative mixed-initiative planning, intelligent agents, understanding (situation assessment), introspection, and learning. More specifically, he is interested in how goals interact with and influence these broader cognitive processes. His approach to research follows both artificial intelligence and cognitive science directions.