

AFRL-IF-RS-TR-2004-266
Final Technical Report
October 2004



SCALABLE REAL-TIME NEGOTIATION TOOLKIT

University of Massachusetts at Amherst

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. H354

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-266 has been reviewed and is approved for publication

APPROVED: /s/

DANIEL E. DASKIEWICH
Project Engineer

FOR THE DIRECTOR: /s/

JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE			<i>Form Approved</i> <i>OMB No. 074-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE OCTOBER 2004	3. REPORT TYPE AND DATES COVERED Final Jun 99 – Dec 03	
4. TITLE AND SUBTITLE SCALABLE REAL-TIME NEGOTIATION TOOLKIT			5. FUNDING NUMBERS C - F30602-99-2-0525 PE - 62301E PR - H354 TA - 01 WU - 01	
6. AUTHOR(S) Victor R. Lesser				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Massachusetts at Amherst Department of Computer Science 140 Governors Drive CMPSCI Room 100 Amherst Massachusetts 01003			8. PERFORMING ORGANIZATION REPORT NUMBER N/A	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency AFRL/ITB 3701 North Fairfax Drive 26 Electronic Parkway Arlington Virginia 22203-1714 Rome New York 13441-4514			10. SPONSORING / MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2004-266	
11. SUPPLEMENTARY NOTES AFRL Project Engineer: Daniel E. Daskiewich/ITB/(315) 330-7731/ Daniel.Daskiewich@rl.af.mil				
12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.				12b. DISTRIBUTION CODE
13. ABSTRACT (Maximum 200 Words) The major technical progress of this effort centered on three different interrelated activities, each of which were critical to the successful building of the software infrastructure necessary to implement an adaptive distributed sensor network. These activities involved the development of a distributed soft, real-time heuristic resource allocation protocol, the development of a domain-independent soft, real time agent architecture, and finally the development of techniques for building and evolving an agent organization. Additionally, the distributed resource allocation protocol led to a new approach to solving distributed constraint satisfaction and optimization problems based on a mediation-based negotiation (partial centralization) with overlapping context and extended views along critical paths for search and communication efficiency. The distributed algorithms developed using this approach, called APO and optAPO have better performance than the best known complete and optimal distributed algorithms - AWC and ADOPT.				
14. SUBJECT TERMS Distributed Resource Allocation, Distributed Constraint Satisfaction, Distributed Constraint Optimization, Adaptive Distributed Sensor Networks, Agent Coordination, Agent Organizations, Soft Real-Time Agent Architecture				15. NUMBER OF PAGES 123
				16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

TABEL OF CONTENTS

1 Summary	1
2 Introduction	1
3 Methods, Assumptions, and Procedures	2
3.1 Organizational Design	2
3.2 Agent Architecture	8
3.2.1 Soft Real-Time Control	8
3.2.2 Task Analysis, Environmental Modeling and Simulation	10
3.2.3 Scheduling	12
3.3 Resource Allocation	13
3.3.1 Tracking as Resource Allocation	13
3.3.2 Scalable, Periodic, Anytime Mediation Protocol	14
3.3.3 Stage 1	15
3.3.4 Stage 2	16
4 Results and Discussion	18
5 Conclusions	19
6 Technology Transfer	19
7 Professional Personnel	20
Bibliography	21
Appendix A	25
Appendix B	38
Appendix C	47
Appendix D	56
Appendix E	101
Appendix F	110

List of Figures

1. High-level architecture	3
2. Overview of agent's organizational hierarchy, with some information flows represented	4
3. Abstraction of messages and reasoning used for target detection by sensor agents	5
4. Track manager queries the manager of the new sector	8
5. The soft real-time control architecture	9
6. An abbreviated view of the sensor initialization TÆMS task	11
7. Utility of taking a single, coordinated measurement from a set of sensors	13
8. Stage 2 of the SPAM protocol	16
9. Example of a common contention for resources	17
10. A solution derived by SPAM to the problem in Figure 9	18

1. Summary

The major technical progress of this effort centered on three different interrelated activities, each of which were critical to the successful building of the software infrastructure necessary to implement an adaptive distributed sensor network. These activities involved the development of a distributed soft, real-time heuristic resource allocation protocol, the development of a domain-independent soft, real-time agent architecture, and finally the development of techniques for building and evolving an agent organization. Additionally, the distributed resource allocation protocol led to a new approach to solving distributed constraint satisfaction and optimization problems based on a mediation-based negotiation (partial centralization) with overlapping context and extended views along critical paths for search and communication efficiency. The distributed algorithms developed using this approach, called APO and optAPO have better performance than the best known complete and optimal distributed algorithms – AWC and ADOPT.

2. Introduction

The initial objectives of this project were to develop: 1) a domain-independent toolkit for distributed negotiations for task and resource allocation in demanding dynamic environments involving hard and soft real-time constraints and large numbers of agents that may have complex resource interactions and 2) analytic and empirical tools to estimate the performance metrics to determine the degree to which overall goals are met and understand the effectiveness of different negotiation protocols under a variety of operating conditions. Early in the project these high-level objectives were re-oriented towards solving the Autonomous Negotiating Teams (ANTs) Electronic Warfare (EW) sensor network challenge problem. However, we tried as much as possible to satisfy the initial project objectives in developing our solutions to the EW challenge problem.

The negotiation toolkit that we developed can be thought of as a wrapper around a conventional agent problem-solving architecture that allows an agent to establish commitments with other agents to perform tasks (or use resources) that the agent cannot accomplish locally. This can occur because of the lack of appropriate resources or expertise, or due to overload of an agent's resources as a result of tasks already committed to. The focus of negotiation is not only on locating agents to accomplish a task, but also on when and how they will accomplish the task in terms of resources used and the quality of the result produced. Negotiation may also result in agents needing to re-evaluate or relax their current objective functions and constraints in order to achieve the overall goals successfully.

Before discussing the technical work in detail, let us first discuss at a high level the EW challenge problem. The sensor network hardware configuration consists of sensor platforms that have three scanning regions, each with a 120-degree arc encircling the sensor. Only one of these regions can be used to perform measurements at a time. The communication medium uses a low-speed, unreliable, radio-frequency (RF) system over eight separate channels. Messages cannot be both transmitted and received simultaneously regardless of channel assignment, and no two agents can transmit on a single channel at the same time without causing interference. The sensor platforms are capable of locally hosting one or more processes, which share a common CPU (in this case a commodity PC and signal processing hardware). The goal of this application is to track one or more targets that are moving through the sensor environment (in this case model railroad trains traveling on railroad tracks whose pattern is unknown). The radar sensor measurements consist of only amplitude and frequency values, so no one sensor has the ability to precisely determine the location of a target by itself. The sensors must therefore be organized and coordinated in a manner that permits their measurements to be used for triangulation. The need to triangulate a target's position requires frequent, closely coordinated actions amongst the agents, ideally three or more sensors performing their measurements at the same time. In order to produce an accurate

track, the sensors must therefore minimize the amount of time between measurements during triangulation, and maximize the number of triangulated positions. Ignoring resources, an optimal tracking solution would have all agents capable of tracking the target taking measurements at the same precise time as frequently as possible. Restrictive communication and computation, however, limits our ability to coordinate and implement such an aggressive strategy. Low communication bandwidth hinders complex coordination and negotiation, limited processor power prevents exhaustive planning and scheduling, and restricted sensor usage creates a trade-off between discovering new targets and tracking existing ones.

3. Methods, Assumptions, and Procedures

The key ideas that were used in developing distributed task allocation protocols that can operate effectively in large and complex environments are the following. First, there is no one best protocol for all situations; rather, what is needed is a family of protocols having different end-to-end completion times, meta-level information needs, and resource requirements. The approach to making the negotiation process operate in real time involves not only controlling the number of negotiation steps that are needed for arriving at a suitable decision, but also how much computation is required for each step. A family of protocols has been developed based on seeing this family as one basic protocol with many parameters that can be adjusted dynamically. Second, wherever possible, organizational and situational knowledge (learned or developed) should be exploited about the best agents with whom to negotiate, and what type of protocol should be used in this situation. Third, the negotiation can go on with respect to different granularities of resources. Basic to the approach is that the protocol works at an abstract level of resource allocation and that it does not have to resolve all conflicts at this abstract level in order to generate an effective resource allocation plan. Because of the power of the soft real-time agent architecture that has been constructed to support negotiation, many resource conflicts that are unresolved at this more abstract level can be solved through local modification of an agent schedule to accomplish, in a partially degraded way, the activities that have resource conflicts. In this way, the protocol can take available time into account when deciding how much negotiation is possible in the current context. The negotiation protocol also exploits meta-level information about flexibility in other agents to make decisions among a small group of agents about how to best handle an over-constrained resource problem so that it does not propagate through the network. As part of this work on negotiation in an organizational context, the University of Massachusetts (UMASS) team focused on the issues of how dynamic organizations can be established as part of a higher-level negotiation process. There are two issues that were addressed. One was how to construct an initial organization with limited initial information, and the other was how to evolve the organization based on analysis of its more global performance characteristics. Additionally, in the later stages of the effort, a top-down knowledge-based approach was developed for defining an appropriate organization given the environment and resource characterizations of the overall problem.

In order to understand the details of our approach, let us consider how they were implemented in the EW challenge problem. First, we discuss the organization design used as a context for dynamic resource allocation. Next, we discuss the Soft, Real-Time Agent-architecture (SRTA), and finally, the Scalable, Periodic, Anytime Mediation (SPAM) distributed resource allocation protocol.

3.1 Organizational Design

The notion of “organizational design” is used in many different fields, and generally refers to how entities in a society act and relate with one another. This is true of multi-agent systems, where the organizational design of a system can include a description of what types of agents exist in the environment, what roles they take on, and how they interact with one another. The objectives of a particular design will depend on the desired solution characteristics, so for different problems one might specify organizations that aim toward scalability, reliability, speed, or efficiency, among other things.

The organizational design used in this solution primarily attempts to address the scalability problem by concentrating knowledge within the agents that are most likely to need it and by imposing limits on how far certain classes of information propagate. As will be seen below, this is done at the expense of reaction speed, because by limiting the scope any single agent has, one necessarily increases the required overhead when the agent's task moves outside that scope.

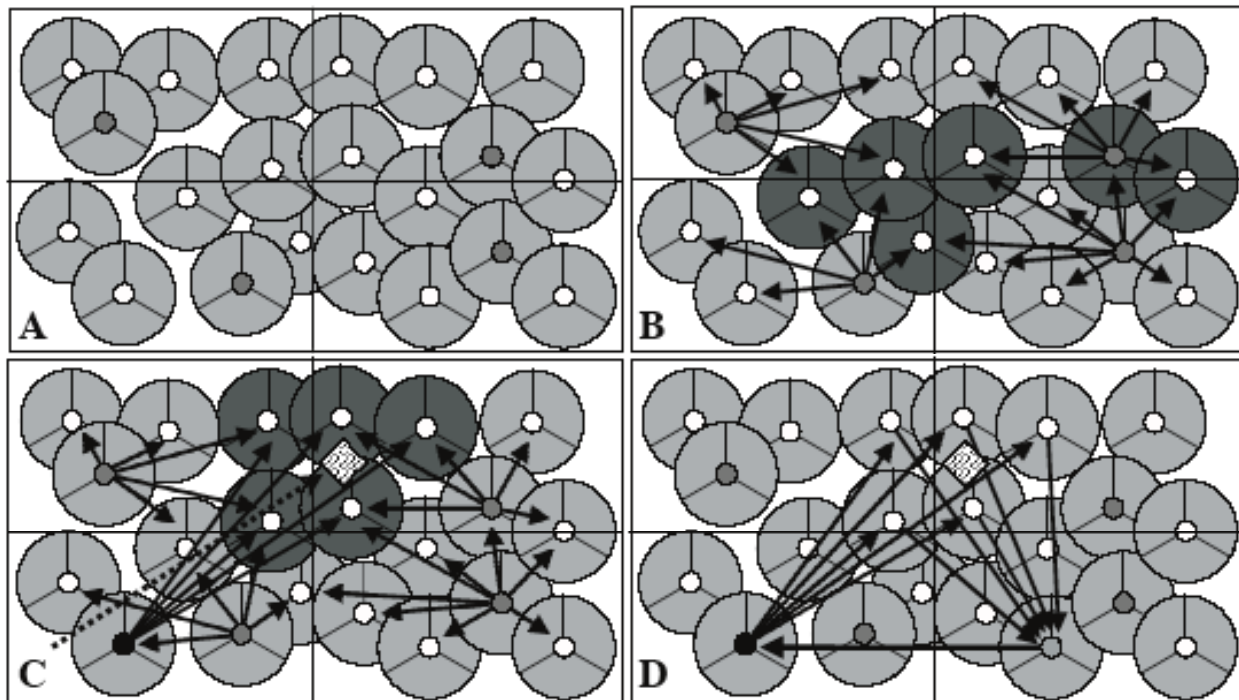


Figure 1: High-level architecture. A: sectorization of the environment; B: distribution of the scan schedule; C: communicating over tracking measurements, and D: fusion of tracking data.

In the EW challenge problem, the environment itself is organizationally partitioned into a series of sectors, each a non-overlapping, rectangular portion of the available area, shown in Figure 1.A. The purpose of this division, as will be shown below, is to exploit locality and limit the interactions needed between sensors. In Figure 1.A, sensors are represented as divided circles, where each 120-degree arc represents a direction the node can sense in. As agents come online, they must first determine which sectors they can affect. Because the environment itself is bounded, this can be done trivially by providing each agent the height and width of the sectors. The agents can then use this information, along with their known position and sensor radius, to determine which sectors they are capable of scanning in. We use this technique to dynamically adapt the agent population for scanning and tracking activities to better partition and focus the flow of information.

Within a given sector, agents may work concurrently on one or more of several high-level goals: managing a sector, tracking a target, producing sensor data, and processing sensor data. The organizational hierarchy is abstractly represented in Figure 2. The organizational leader of each sector is a single sector manager, which serves as the locus of activity for that sector. This manager generates and distributes plans (to the sensor data producers) needed to scan for new targets, stores and provides local sensor information as part of a directory service, and assigns track managers. The sector managers act as hubs within a nearly decomposable hierarchical organization, by directly specifying scanning activities, and then selecting agents to oversee tracking activities. They also concentrate non-local information, facilitating the transfer of that knowledge to interested parties. Individual track managers initially obtain their information from their originating sector manager, but will also interact directly, though less frequently, with other sector and track managers, and thus do not follow a fixed chain of command or

operate solely within their parent sector as one might see in a fully-decomposable organization. Track managers will also form commitments with one or more agents to gather sensor data, but this relationship is on a voluntary basis, and that gathering agent's behavior is ultimately determined locally.

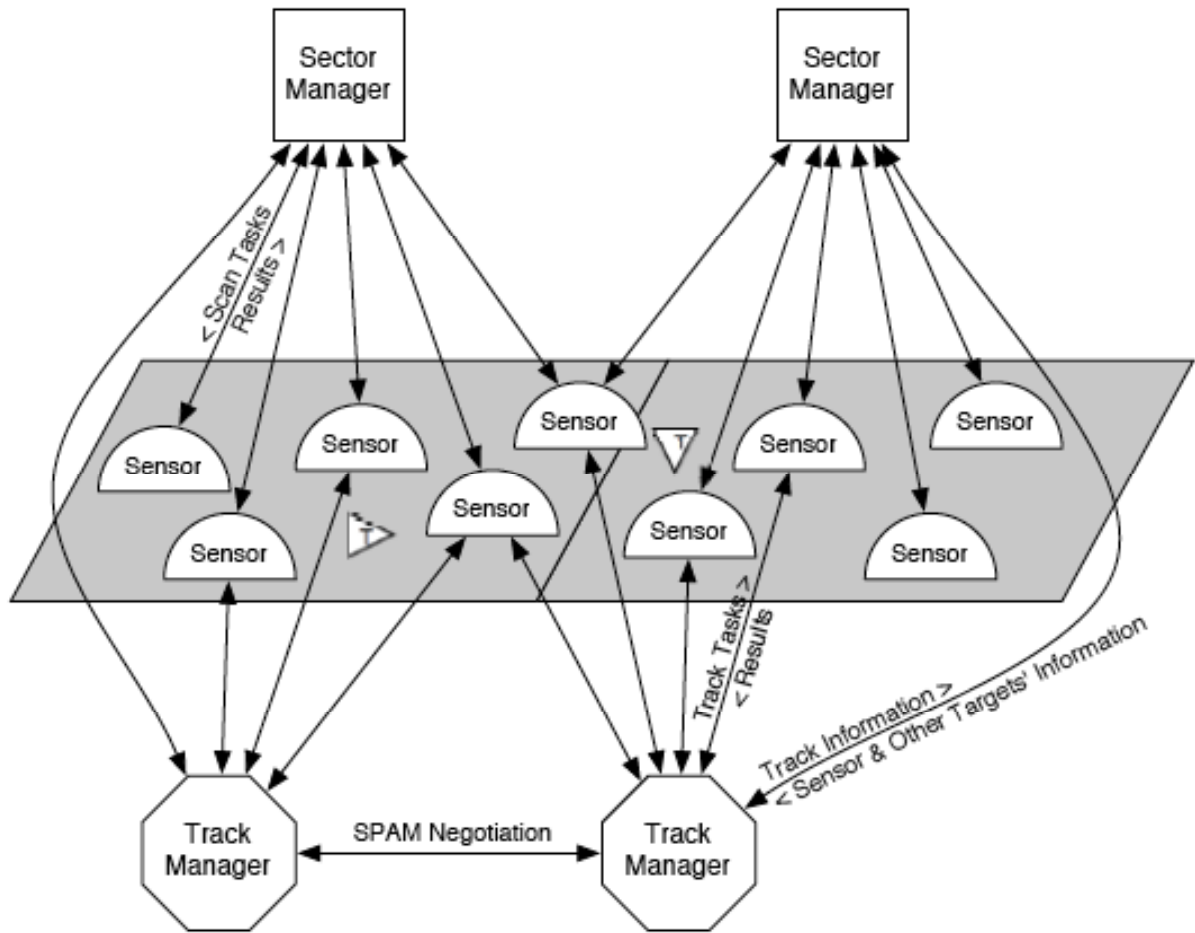


Figure 2: Overview of the agent's organizational hierarchy, with some information flows represented.

Because much of the information being communicated is contained within sectors, the size and shape of the sector has a tangible effect on the system's performance. If the sector is too large, and contains many sensors, then the communication channel used by the sector manager may become saturated. If the sector is too small, then track managers may spend excessive effort sending and receiving information to different sector managers as its target moves through the environment.

We found empirically that a reasonable sector in the EW challenge problem would contain 8 sensors, but would still function adequately with as many as 10 or as few as 5. The physical dimensions of such a sector depend on the density of the sensors, and in different environments one would need to take into account sensor range, communication medium characteristics and maximum target speed. Further information on partitioning agent populations, including a more sophisticated technique that utilizes heterogeneous regions, can be found in work by Sims in 2003.

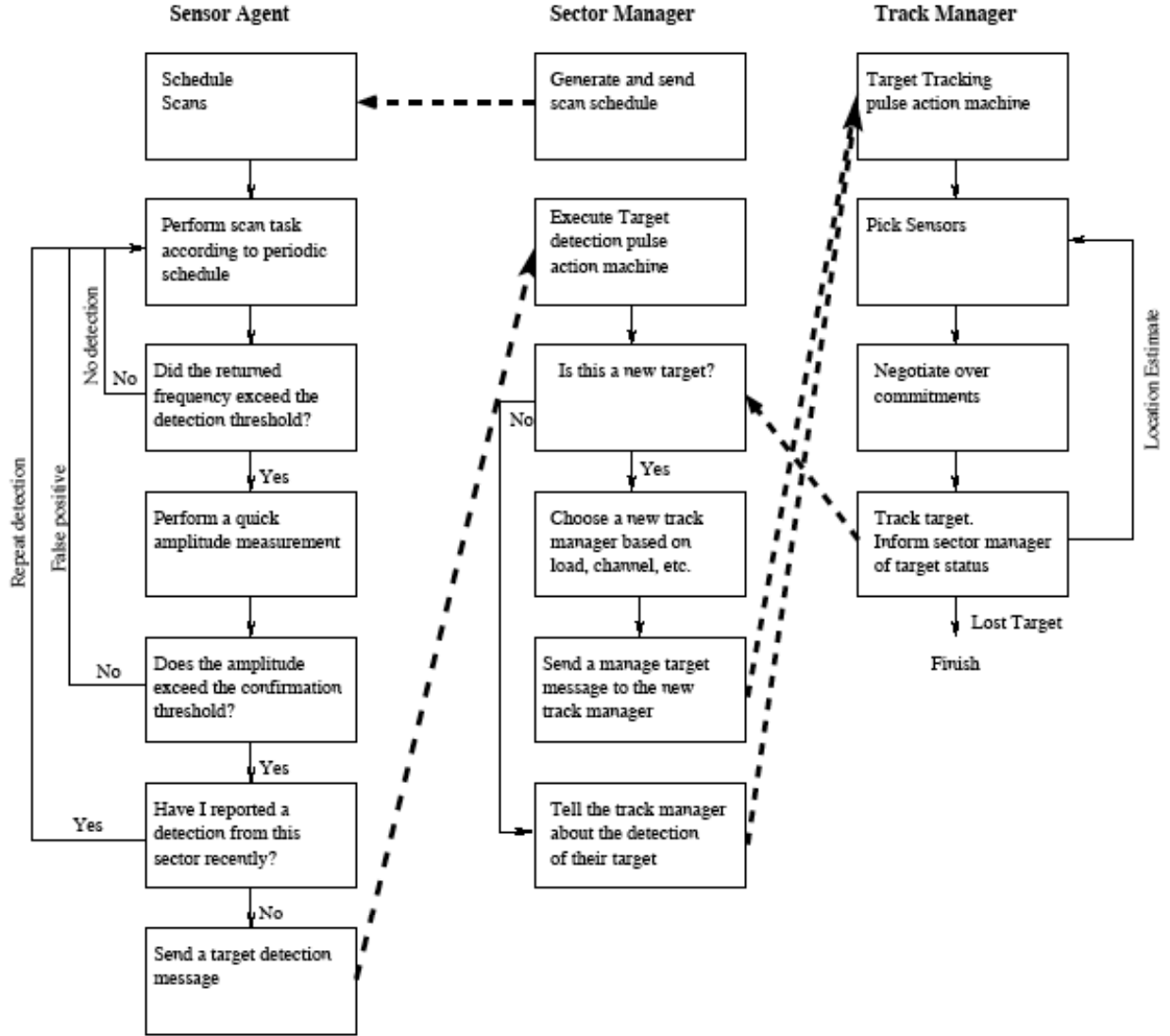


Figure 3: An abstraction of the messages and reasoning used for target detection by sensor agents, sector and track managers.

To see how the organization works in practice, consider a scenario starting with agents determining what sectors they can affect, and which agents are serving as the managers for those sectors. Ideally, the sector managerial duty would be delegated and discovered dynamically at runtime, but due to the lack of a true broadcast capability in the RF communication medium, we statically define and disburse this information a priori¹. In Figure 1, these sector managers are represented with shaded inner circles. Once an agent recognizes its manager(s), it sends each a description of its capabilities. This includes such things as the position, orientation, and range of the agent's sensor. The manager then has the task of using this data to organize the scanning schedule for its sector. The goal of the scan schedule is to use the sensors available to it to perform inexpensive, fast sensor sweeps of the area, in an effort to discover new targets. The manager formulates a schedule indicating where and when each sensor should scan, and communicates with agents over their respective responsibilities in that schedule (see Figure 1.B). The manager does not strictly assign these tasks. The agents have autonomy to decide locally what action gets performed when.

¹ A limited broadcast capability does exist, which can reach all sensors listening on a single channel. It is not possible in this architecture to broadcast a single message to agents that are using different channels.

This is important because sensors can potentially scan in multiple sectors, thus there is the possibility that an agent may receive multiple, conflicting requests for commitments from different sector managers. The agent's autonomy and associated local controller permit the agent itself to be responsible for detecting and resolving these conflicts. If one receives conflicting requests for commitments, it can elect to delay or decommit as needed. Shaded sensors in Figure 1 show agents receiving multiple scan schedule commitments.

Once the scan is in progress, individual sensors report any positive detections to the sector manager which assigned them the scanning task, which can then spawn a new track manager as shown in Figure 3. Internally, the sector manager maintains a list of all local agents that currently perform the role of track manager, and location estimates for the targets they are tracking. These location estimates are used to determine the likelihood of the positive detection being a new target, or one already being tracked. If the target is new, the manager uses a range of criteria to select one of the agents in its sector to be the track manager for that target. Not all potential track managers are equally qualified, and an uninformed choice can lead to very poor tracking behavior if the agent is overloaded or shares communication bandwidth with other garrulous agents. Therefore, in making this selection, the manager considers the agents' estimated load, communication channel assignment, geographic location and activity history. Ideally, it will select an agent that has minimal channel overlap, is not currently tracking a target, but has tracked one previously. This will minimize the potential for communication collisions, which occur if two agents on the same channel attempt to send data at the same time, but maximize the potential amount of cached organizational data the agent can reuse. As we have seen previously, this notion of limited communication is an important motivating factor and recurring theme in this architecture that contributes to the organizational structure, role selection, protocol design and the frequency and verbosity of communication actions.

The assigned track manager (shown in Figure 1.C with a blackened inner circle) is responsible for organizing the tracking of its assigned target. To do this, it first discovers sensors capable of detecting the target, and then communicates with members of that group to gather the necessary data. Discovery is done using the directory service provided by the sector managers. One or more queries are made asking for sensors that can scan in the area the target is predicted to occupy. The track manager must then determine when the scans should be performed, considering such things as the desired track fidelity and time needed to perform the measurement, and coordinate with the discovered agents to disseminate this goal (see Figure 1.C). As with scanning, conflicts can arise between the new task and existing commitments at the sensor, which the agent must resolve locally.

The data gathered from individual sensors is collected by an agent responsible for fusing the measurements into a location estimate and extending the computed track (see Figure 1.D). In a general sense, this data fusion agent could be any agent in the population able to communicate efficiently with both the data sources and the ultimate destination of the tracking data. However, the data fusion process for this application is fairly lightweight, and thus does not benefit from distribution for load balancing purposes. In addition, transferring the fused data results introduces an unnecessary delay while it is being communicated to the track manager. For this reason, in this work, the data fusion and track manager roles are always performed by the same agent.

When the track manager receives raw measurement data, it verifies the quality and association of the measurement. Quality can be measured by several means, and in this work, the signal-to-noise ratio is used. If a measurement is returned with significantly high signal strength, it is considered for potential fusion. The association of a measurement really involves two different things: temporal and target association.

Temporal association attempts to match measurements taken from various sensors based on the time the measurements were taken. As previously mentioned, the accuracy of triangulating a moving target is in part dependent on the relative temporal coordination of the measurements. If a track manager were to

fuse measurements that were taken over a wide range of time, the resulting estimation could be quite poor. There are several reasons why a measurement may be returned that cannot be temporally associated with measurements from other sensors. One reason is delays and loss introduced by the communications. For example, if the commitment request from the track manager never gets to the sensor, no measurement will be taken. Another reason is unresolved resource contention within the individual sensors. If, for example, one of the sensors decides to delay the start of a measurement in order to deconflict its internal schedule, it will be harder to match the resulting measurement with data from other sensors.

Within our system, temporal association maintained by queuing measurements for a finite period and matching them with one another based on their actual measurement time. We make the assumption that the computers controlling the sensors are time synchronized using a Network Time Protocol (NTP) like mechanism.

Target association, as the name implies, tries to match measurements with targets. Consider the case of two targets, T1 and T2, which are following one another through the sensor field. Now imagine that T1 moves out of the viewable area of one of the sensor heads being used to measure it and T2 moves into this head's view. If a measurement is taken by that head before the T1's track manager changes its allocation, the resulting amplitude will be of the wrong target. When this measurement is fused with the track of T1, it will appear to be closer to T2 than before. Over time, if this pattern continues, the targets will be indistinguishable from one another. In fact, T1 could end up being ignored and both track managers could try to track T2.

To prevent this from happening, track managers check to ensure that each measurement has a higher likelihood of belonging to their target than others that may be near to it. To do this, we exploit the view of the sector manager. Because sector managers are periodically given estimated target locations for all of the targets within their sector, it is easy for them to determine when two or more track managers are likely to have a target association problem. When the sector manager detects this possibility, it informs the appropriate track managers by sending them the locations of the target that are close to them. This allows the track managers to discard any measurement that was likely to have been collected by a sensor viewing another target. If, for some reason, two track managers do fuse their target into one, the sector manager can also recognize this fact and inform one of the managers to stop tracking. This alleviates the unnecessary resource contention created by having two track managers fight over the same resources to track the same target.

If the data values returned are of high enough quality, and the agent determines those measurements were taken from the correct target, then they are used to triangulate what the position of the target was at that time. This data point is then added to the track, which itself is used as a predictive tool when determining where the target is likely to be in the future. At this point the track manager must again decide which agents are needed and where they should scan, and the sequence of activities is repeated.

Partitioning the environment reduces the amount of information and processing that agents must perform for several different tasks. For example, generating a coherent scan schedule for a group of sensors is simplified by only taking into account a tractable number of them. Similarly, when a new target is detected as a result of a scan, that information can be sent to only the appropriate sector manager, which can determine directly if it is a new or existing target based on local information. Sectors also facilitate gathering data about the sensors themselves, as track managers need only perform a single query to the appropriate sector manager to discover all the sensors available within that region (see Figure 4). In fact, the partitioning makes nearly every aspect of this solution scalable to arbitrary numbers, with the exception of the tracking allocation, which has its own solution to this problem, as shown later. As a side effect, partitioning does reduce the system's reactivity, because an extra step may be required to fetch information that is not available locally. We cope with this problem wherever possible by caching such data to avoid redundant queries, and by assigning new roles whenever possible to agents that have served that same role in the past, to take advantage of that cached data.

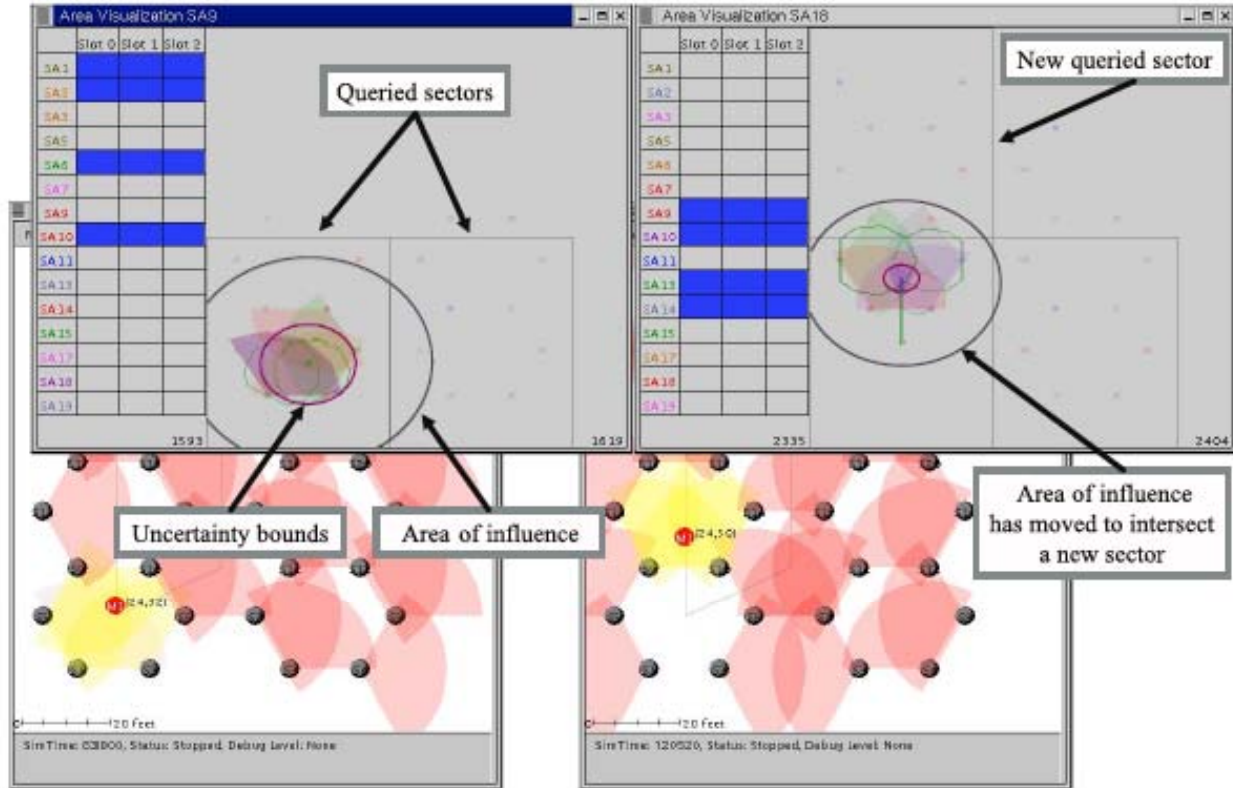


Figure 4: As the target moves to another sector, the track manager queries the manager of the new sector.

Although not required in the scenarios we present here, it is interesting to note the applicability of this organization to situations where agents have an additional limitation or attenuation of communication capability based on the geographic distance separating the participants. In this case, this partitioned organization could serve as the basis of an ad-hoc network, where messages are routed from one sector to the next, using the organizational structure as a guide, until they reach their destination. This further emphasizes the notion that “local” communication is more efficient, and the locality of information should be exploited by the organization to take advantage of it.

In this section we have shown how the organization plays a critical role in ensuring that information flowing within the sensor network is managed to both minimize the delay and expensive of communication and to maximize its availability for effective decision making. The next section describes some of the mechanisms that go into forming the agent-level control of the sensors.

3.2 Agent Architecture

3.2.1 Soft Real-Time Control

The Soft Real-Time Control Architecture (SRTA), the agent control engine used by this solution, provides several key features to the agents within our sensor network:

1. The ability to quickly generate plans and schedules for goals that are appropriate for the available resources and applicable constraints, such as deadlines and earliest start times.
2. The ability to merge new goals with existing ones, and multiplex their solution schedules.

3. The ability to use explicit representations of uncertainty and efficiently handle deviations in expected plan behavior that arise out of variations in resource usage patterns and unexpected action characteristics.

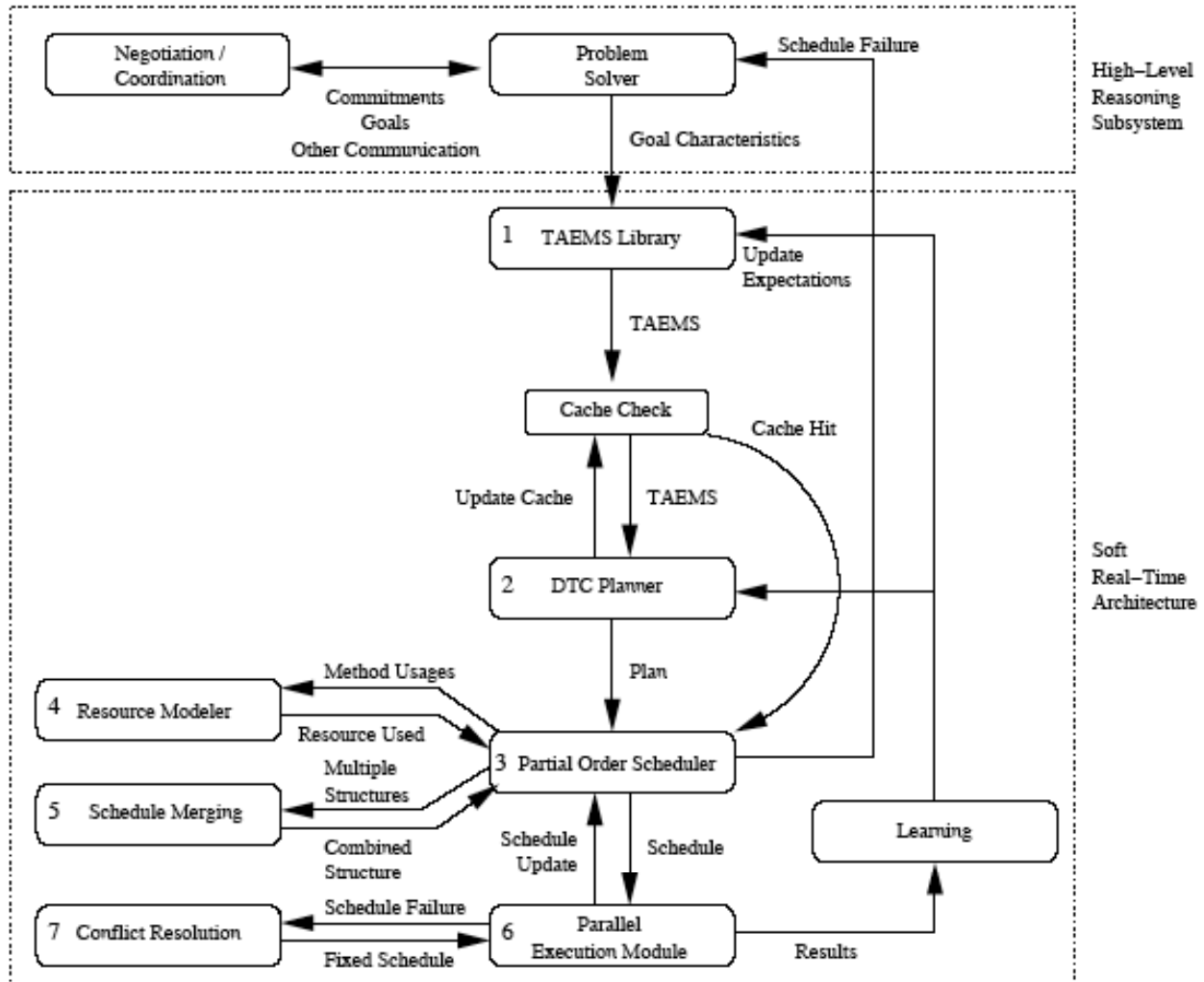


Figure 5: The soft real-time control architecture.

Abstractly, SRTA (shown in Figure 5) operates as a single functional unit within an agent, which itself is running on a conventional (i.e., not real-time) operating system. The SRTA controller is designed to be used in a layered architecture, occupying a position below the high-level reasoning component in an agent. In this role, it accepts new goals, reports the results of the activities used to satisfy those goals, and also serves as a knowledge source about the potential ability to schedule future activities by answering “what-if” style queries. The components that comprise SRTA assume a majority of the responsibility needed to satisfy goals, which allows the high-level reasoning system to focus on goal selection, determining goal objectives and other potentially domain-dependent issues. For example, agents may elect to coordinate using abstractions of their activities or resource allocations, which are then locally translated into a precise schedule. SRTA can then use these schedules to both enforce the semantics of the commitments which were generated, and automatically attempt to resolve conflicts that were not addressed through coordination. In the next section, TÆMS, the modeling language used to describe goals to SRTA, is explained. Following that, we will show how the components of the SRTA architecture work together to provide the agents with the ability to handle multiple, concurrently executing goals.

3.2.2 The Task Analysis, Environmental Modeling and Simulation

The Task Analysis, Environmental Modeling and Simulation (TÆMS) language is used to quantitatively describe the alternative ways a goal can be achieved. A TÆMS task structure is essentially an annotated task decomposition tree. The highest-level nodes in the tree, called task groups, represent goals that an agent may try to achieve. For example, the goal of the structure shown in Figure 6 is **Setup-Hardware**. Below a task group there will be a set of tasks and methods that describe how that task group may be performed, including sequencing information over subtasks, data flow relationships and mandatory versus optional tasks. Tasks represent sub-goals, which can be further decomposed in the same manner. **Setup-Hardware**, for instance, can be performed by completing **Startup**, **Init**, and **Obtain-Background-Noise**. Methods, on the other hand, are terminal, and represent the primitive actions an agent can perform. Methods are quantitatively described, in terms of their expected quality, cost and duration. **Activate-Sector_0**, then, would be described with its expected duration and quality, allowing the scheduling and planning processes to reason about the effects of selecting this method for execution. The Quality Accumulation Function (QAF) below a task describes how the quality of its subtasks is combined to calculate the task's overall quality. For example, the `q_min` QAF below **Init** specifies that the quality of **Init** will be the minimum quality of its subtasks — so all the subtasks must be successfully performed for the **Init** task to succeed. Interactions between methods, tasks, and affected resources are also quantitatively described.

The curved lines in Figure 6 represent resource interactions, describing, for instance, the produces and consumes effects that method **Set-Sample-Size** has on the resource **SensLock**, and how the level of **SensLock** can limit the performance of the method. TÆMS structures are used by our agents to describe how particular goals may be achieved. Rather than hard coding, for by instance, the task of initializing the sensor, we encode the various steps in a TÆMS structure similar to that shown in Figure 6. This simplifies the process of evaluating the alternative pathways by allowing the designer to work at a higher level of abstraction, rather than be distracted by how it can be implemented in code. More importantly, it also provides a complete, quantitative view that can be reasoned about by planning, scheduling and execution processes. A given task structure begins its existence when it is created, read in from a library (Figure 5-1), or dynamically instantiated from a template at runtime. Planning elements are involved both in the generation of the structure, and then in the selection of the most appropriate sequence of methods from that structure which should be performed to achieve the goal given the currently available resources. This sequence is then used by a scheduling process to determine the correct order of execution, with respect to such things as precedence constraints and resource usage. Finally, this schedule will be used by an execution process to perform the specified actions, the results of which are written back to the original task structure. The schedules produced by individual TÆMS structures are the building blocks for an agent's overall schedule of execution. A valid schedule completely describing an agent's activities will allow it to correctly reason about and act upon the deadlines and constraints that it will encounter, for example a resource restriction. Typically, however, schedules are only used to describe lower-level activity. In the EW challenge domain, this encompasses sensor initialization, scanning and tracking activity, data fusion and the like.

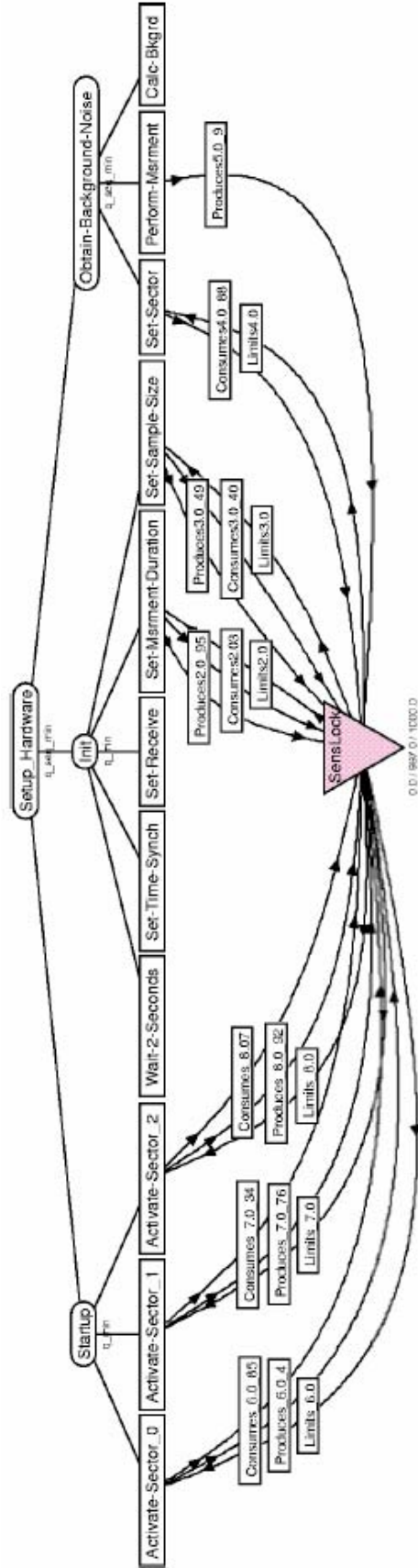


Figure 6: An abbreviated view of the sensor initialization TAMS task

3.2.3 Scheduling

In the SRTA architecture, we have attempted to make the scheduling and planning process incremental and compartmentalized. New goals can be added piecemeal to the execution schedule, without the need to re-plan all the agent's activities, and exceptions can be typically handled through changes to only a small subset of the schedule. Figure 5 shows the organization of SRTA.

In this architecture, goals can arrive at any time, in response to environmental change, local planning, or because of requests from other agents. The goal is used by the problem-solving component to generate a TÆMS task structure, which quantitatively describes the alternative ways that the goal may be achieved. The TÆMS structure can be generated in a variety of ways; in our case we use a TÆMS "template" library, which we use to dynamically instantiate and characterize structures to meet current conditions. Other options include generating the structure directly in code, or making use of an approximate base structure and then employing learning techniques to refine it over time. SRTA uses the Design-To-Criteria (DTC) component to generate linear plans solving the goal described in the TÆMS structure (Figure 5-2). It employs a battery of techniques to efficiently discover and reason about the various activity schedules that can address that goal. The ability to make trade-offs while respecting commitments is particularly important, as DTC attempts to select the quantitatively "best" plan that meets the specified requirements. DTC uses criteria such as potential deadlines, minimum quality, external commitments, and soft and hard action interrelationships to select an appropriate sequence of activities.

The resulting plan is used to build a partially ordered schedule, which uses structural details of the TÆMS structure to determine precedence constraints and search for actions which can be performed in parallel (Figure 5-3). The Partial Order Scheduler (POS) also provides the ability to quickly shift execution order at any point in time instead of performing costly re-planning. In a real-time environment, schedule adjustments are more frequent; by not imposing unnecessary ordering constraints on our agent's schedule the agent has a better chance of achieving the time, cost and quality criteria of its goal. A pair of specialized components is used to assist the POS during this final scheduling phase. The first, a resource-modeling component, is used to ensure that resource constraints are respected (Figure 5-4). A schedule-merging module then allows the partial order scheduler to incorporate the actions derived from the new goal with existing schedules (Figure 5-5). Our notion of "parallel" in this architecture includes activities that run concurrently in parallel, as in a multiple processor environment, and those that run virtually in parallel, as in a time-slicing, multi-processing operating system. If we view the sensor as a specialized, separate processor, our task structures contain both types of methods. For instance, a sensor measurement action can take place concurrently with actions on the primary processor. Unifying these notions simplifies the scheduling process, and can be represented appropriately using TÆMS. Once the schedule has been created, an execution module is responsible for initiating the various actions in the schedule (Figure 5-6). It also keeps track of execution performance and the state of actions' preconditions, potentially re-invoking the partial order scheduler when failed expectations require it. Using the ordering constraints described in the schedule, the execution component can directly determine which methods can be run concurrently. By overlapping their execution, we reduce the total execution time, which effectively increases the agents overall work capacity. The gain in execution time, and resulting flexibility, is used to address resource availability, in addition to improving the likelihood the scheduler can accommodate real-time changes without breaking deadline constraints. If this is unsuccessful, a conflict resolution module is used to reason about mutually exclusive tasks and commitments, determining the best way to handle conflicts (Figure 5-7). Repairs can be accomplished in a variety of ways, for instance, by relaxing constraints such as the goal completion criteria or delaying its deadline, completing a substitute goal with different characteristics, or decommitting from a lower-priority goal or the goal causing the failure.

The execution characteristics of the SRTA architecture as a whole depend largely on the frequency and complexity of the goals it is asked to plan and schedule. On average, we observe cycle times of between

50 and 100 milliseconds on 400 MHz x86-based systems, although this can jump to a half-second or more if a particularly complex situation arises. A cycle represents a single pass of the SRTA engine analyzing the current goals and executing methods. Because the system runs on a conventional operating system (Linux in this case), competing external processes may add an additional level of performance uncertainty.

3.3 Resource Allocation

Here we describe the SPAM resource allocation protocol that provides this system with the ability to handle resource conflicts by elevating the decision making to the track managers.

3.3.1 Tracking as Resource Allocation

Modeling the target-tracking domain as a resource allocation problem is fairly straightforward. Each of the targets in the environment can be considered a task, which is assigned to a track manager. The sensors are the resources and the job of the track managers is to obtain enough sensing time from the correct sensors to track their targets. There are a number of characteristics about this particular resource allocation problem that make it challenging. The first is that, although this problem can be solved crudely using constraint satisfaction techniques, it lends itself most naturally to being solved using some type of optimization. Optimization is a good fit for two reasons. The first is that the value of a sensor platform to a track manager is directly correlated with the distance and relative angle of its most appropriate sensor head to the manager's target. Increasing the angle or distance decreases the accuracy and the resulting value of a measurement. In addition, increasing the number of sensors involved in taking coordinated measurements improves the accuracy of the resulting location estimation (see Figure 7).

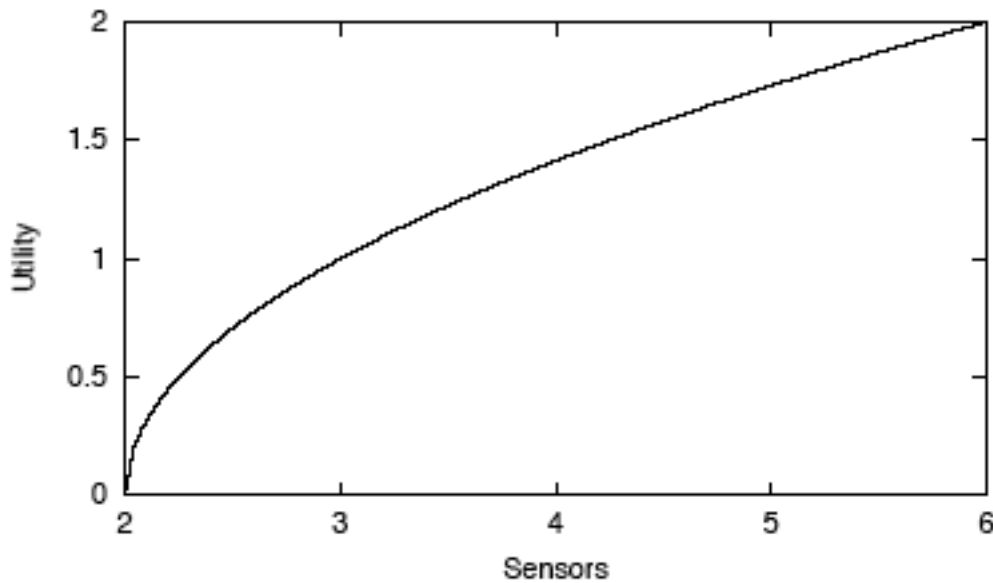


Figure 7: Utility of taking a single, coordinated measurement from a set of sensors.

The goal of the track managers, therefore, is not simply to come up with a conflict free resource assignment, but to derive a solution that *maximizes* the number and the value of the sensors used to track the targets. The second major difficulty is the need to temporally coordinate the actions of the sensors. Coordinating the resources in this way adds another dimension to the problem and increases its difficulty considerably. The biggest problem with temporal coordination is that time is a continuous value. There are essentially an infinite number of possible combinations to consider when trying to deconflict and

coordinate the sensor schedules. As mentioned in the previous section, the method used in this system to reduce the number of possible combinations is to use a finite planning horizon on a slot-based schedule. By using temporally coordinated slots, the track managers lose the ability to perform fine-grained scheduling, but the overall complexity of the resource allocation is decreased. So, in fact, the track managers not only have to maximize the number and value of the sensors, but they also have to maximize the amount of time those sensors dedicate to taking measurements for their target. If we say that M_s^i is the set of good sensors measurements (can see the target) leading to the positional estimate in a single slot s for a task i , and $Util(M_s^i)$ is defined as the utility function in Figure 7 then the utility function for that task during a specific period is:

$$U_i(a_i) = \sum_{s=1}^k Util(M_s^i)$$

Fortunately, the need for coordination actually allows us to consider a much smaller subset of the possible allocations for a given task. In fact, track managers within our system use a simplified set of *objective levels* defined by their utility functions to assign resources to their targets. Each objective level is expressed as a cross product $D_m \times D_s$ denoting the number for sensors desired for a number of slots in the planning horizon. For example, a track manager may wish to have three sensors for two slots, which is denoted 3×2 . Although the number of slots in a period is variable, for this domain we typically set it to match the number of sensor heads on each platform, which is three. In order to prevent certain targets from being ignored in order to improve the quality of another target's estimate, track managers are penalized for not triangulating their target during a full period.

The third characteristic of interest is the dynamic nature of the problem. As the targets move through the environment, moving in and out of the range of the sensors, the underlying resource allocation problem changes in structure. This drives the need to constantly monitor, re-evaluate, and reallocate the sensors used to measure the positions of the targets. It also means that any method used to allocate the resources has to be responsive to changes that occur in the middle of the allocation process.

The last major challenge is that the allocation technique needs to be resource aware. The communications infrastructure is RF-based, which in this case makes it very slow and unreliable. These properties make it essential to not only limit the amount of communications, but to be aware of and adapt to changes in the overall ability to communicate. For example, if it is taking a long time to get messages to a particular track manager, it probably makes sense to avoid creating conflicts with it. Also, if a particular allocation is only useful for a very short period of time, it may not make sense to engage in a complex reallocation process when a simpler one may meet the basic need to track.

3.3.2 The SPAM Protocol

This section describes the Scalable, Periodic, Anytime Mediation (SPAM) protocol, which uses cooperative mediation to solve a dynamic, distributed optimization problem. The SPAM algorithm is built around the principle of “good enough, fast enough.” As such, the protocol is actually divided into two major stages. Stage 1 of the protocol uses local information to derive and bind temporary solutions that are seldom free of conflict and are often based on inaccurate, incomplete information. Stage 2 of the protocol solves conflicts and distributes resources by initiating a *cooperative mediation* session. During a mediation session, one of the track managers takes on the role of mediator. As the mediator, it gathers information from track managers that are in conflict, computes and recommends possible solutions to the problem, and then announces a final solution. The SPAM protocol is activated under two conditions. The first condition occurs when the resources needed to track a target change due to a target’s movement. These types of changes often alter the structure of the underlying optimization problem. As such,

whenever they occur, managers instantiate the first stage of the protocol to quickly modify their current solution or to create an initial solution in the case of a new target assignment. Once stage 1 completes, a manager can choose to begin a mediation session if they determine that the benefits outweigh the costs.

The second case occurs when a manager detects a conflict within one of the resources it is using. This type of conflict is caused when another manager is unable to mediate a session, is able to mediate but due to latency has not yet begun it, or is unaware that the resource is already being used. In each of these cases, the manager detecting the conflict has the option to immediately activate stage 2 and mediate a session to repair it. A distributed locking mechanism prevents more than one manager from concurrently mediating if latency was the cause of the conflict detection.

3.3.3 Stage 1

Stage 1 of SPAM serves three primary functions. The first function is to attempt to find a solution within the context of the information that the protocol has when it starts up. Like the Asynchronous Weak Commitment (AWC) protocol, each of the agents tries to find an assignment that is consistent with its potentially incomplete or inconsistent *agent_view*. However, because this protocol attempts to maximize the social utility, each of the agents tries to maximize their local utility without causing new constraint violations. If this can be done, then no further mediation is necessary, and the protocol terminates at the end of stage 1.

We should mention that a trade-off exists between communication overhead and utility, due to the initial selections of the objective level in stage 1. If each of the managers chooses to use every available resource (sensors able to see their target), the possibility for contention over resources greatly increases in the environment, thereby causing the execution of stage 2 to occur more frequently. However, if the agents decide to start with a lower objective level (and correspondingly less utility), the social utility may suffer unnecessarily.

Stage 1 has what we refer to as a concession rate. The concession rate defines what percentage of the local solution quality a track manager is willing to concede to find a violation-free solution in an attempt to avoid a potentially expensive stage 2-mediation session. So, as the manager's utility drops, the amount they are willing to concede drops as well. This causes managers to mediate (stage 2) more frequently in critically constrained tracking environments.

The second function of stage 1 is to ensure some utility is obtained while waiting for stage 2 to complete. Since these temporarily applied solutions are only applied when a completely conflict-free assignment is not possible, unresolved conflict are left to the individual sensor agents to handle. As mentioned in the previous section, sensor agents can use one of a number of techniques, including slot boundary shifting, less expensive measurement types, or task rotation, in order to resolve such conflicts. To the track manager, whether or not they get a measurement from a conflicted sensor is probabilistically random.

Temporarily applied solutions do not use the concession rate. In fact, because of environmental changes and the probabilistic nature of getting measurements from conflicted sensors, managers always use their maximum possible objective level (within the bounds of the number of sensors that can see the target). The reason for this is rather subtle, but important. Let's say that a new resource was added to the possible resources that could be used by manager T1. Let's also say that another manager, T2, who has more than enough available resources for itself, was using that entire resource. If T1 starts mediation at that lower level, it can never obtain its highest level as a result of the session, even though a solution exists where T2 just gives up the entire conflicted resource.

The third purpose of stage 1 relates to the anytime characteristics of the protocol. Because a solution is always derived and applied during stage 1, managers don't necessarily have to enter stage 2. They can stop the process at the end of stage 1 and accept the results that they have achieved. This is often done if a target's movement causes the resource needs to change faster than the expected time it would take to

complete stage 2. The expected time to complete stage 2 is computed based on both previous experience and the current estimated channel loads for the track managers that would be in the mediation session.

3.3.4 Stage 2

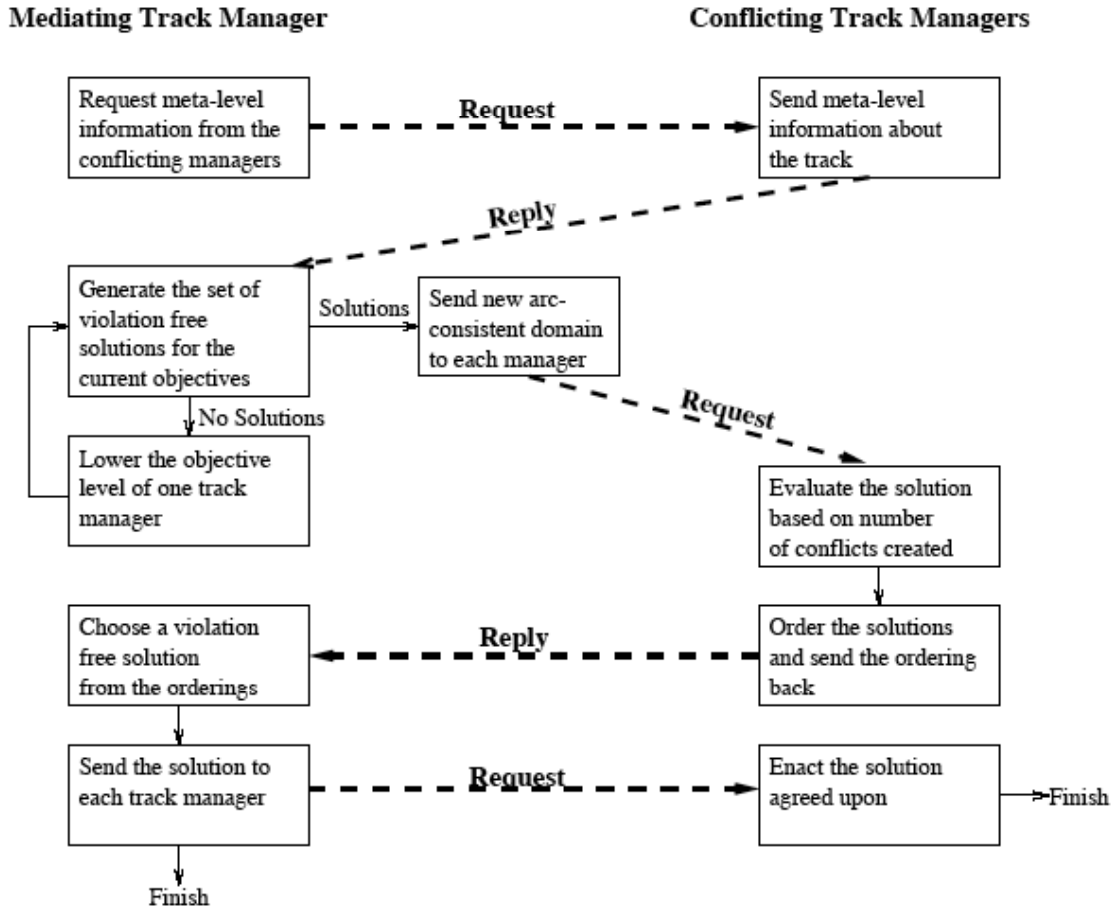


Figure 8: Stage 2 of the SPAM protocol.

Stage 2 is the heart of the SPAM protocol (See Figure 8). Stage 2 attempts to resolve all of the local conflicts a track manager has by elevating the problem to the track managers that are using the desired resources. To do this, the originating track manager takes the role of the mediator (note that multiple sessions can occur in parallel in the environment). As the mediator, it becomes responsible for gathering all of the information needed to generate alternative solutions, creating solutions that may involve changes to the objective levels of the managers involved, and finally choosing a solution to apply to the problem. These solutions are generated without full global information and may lead to newly introduced non-local conflict. If this occurs, other track managers can begin sessions, which propagate the conflict even further.

The best way to explain the operation of stage 2 is through an example. Consider Figure 9, which depicts a commonly encountered form of contention. Here, track manager T2 has just been assigned a target. The target is located between two existing targets that are being tracked by track managers T1 and T3. This creates contention for sensors S3, S4, S5, and S6.

Following the protocol for the example in Figure 9, track manager T2, as the originator of the conflict, takes on the role of mediator. It begins the mediation session by requesting information from each of the track managers involved in the resource conflict. Upon receiving the request, each track manager replies

with their current objective level, the number of sensors which can see their target, the names of the sensors that are in direct conflict with the mediator, and any additional conflicts that the manager has. To continue our example, T2 sends a request for information to T1 and T3. T1 and T3 both return that they have four sensors that can track their targets, the list of sensors that are in direct conflict (i.e., T1(S3, S4), T3(S5,S6)) their objective level (4×3 for both of them) and that they have no additional conflicts outside of the immediate one being considered.

As seen in Figure 8, T2 enters a loop that attempts to generate solutions followed by lowering the objective level of one of the track managers if none exist. A heuristic method, designed to balance the resources, is used to choose the track manager to lower. Namely, the track manager is chosen that has both the highest objective level and is unable to support it without using resources from the set of sensors being mediated over. Whenever two or more managers have the same highest objective level, we choose to lower the objective level of the manager with the least amount of external conflict. By doing this, it is our belief that track managers with more external conflict will maintain higher objective levels, which provides them with leverage in resolving subsequent conflicts that may occur as a result of propagation.

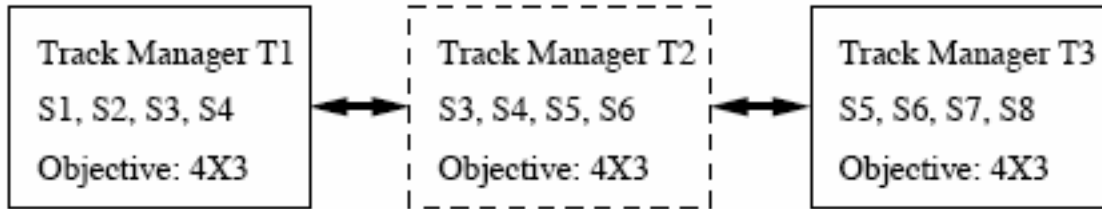


Figure 9: Example of a common contention for resources. Track manager T2 has just been assigned a target and contention is created for sensors S3, S4, S5 and S6.

Note that although this has similarities to the techniques used in Partial Constraint Satisfaction (PCSP)s, this differs in that the problem changes as the objective levels are changed. PCSP techniques choose to satisfy some subset of the constraints; this technique changes the constraints themselves until the problem is satisfiable.

The solution generation loop is terminated under one of two conditions. First, if given the current objective levels for each of the track managers the mediator is able to generate at least one satisfying assignment, then the session enters the solution evaluation phase. Second, the session ends if the mediator cannot generate a satisfying assignment and it cannot drop the objective level of one of the track managers in the session. Under these conditions, the session is terminated and the mediator lowers its own objective level to the lowest possible level, conceding that it cannot find a satisfying assignment, and binds a solution, which minimizes the number of conflicts.

Continuing our example, T2 first lowers the objective level of T1 (choosing T1 at random because they all have equal external conflict). No satisfying assignments are possible under the new set of objective levels, so the loop continues. It continues, in fact, until each of the track managers has an objective level of 3×2 at which time T2 is able generate a set of 216 satisfying assignments to the problem.

During solution evaluation, the mediator proposes the set of solutions to each track manager by sending them a list of sensor assignments that occur in at least one solution. Because each of these assignments is part of a solution, each the lists have the important property that they are arc-consistent with the lists being transmitted to the other managers in the session. In addition, the mediating track manager is guaranteed to have a conflict-free sensor assignment whenever it successfully concludes a session.

Upon receiving their list, each of the track manager rates the assignments based on their local *agent_view* and their internal utility functions. Continuing our example, T2 sends a list of assignments to T1, a list of assignments to itself, and a list of assignments to T3. In our system, track managers rank the assignments

based on a utility function that includes the amount of conflict that will be introduced by taking the assignment and on the desirability of the sensors. This is similar to the min-conflict heuristic and is an integral part of the hill-climbing nature of the algorithm.

Once the mediator has the ratings from the track managers, it chooses a particular solution to apply to the problem. This is done using a dynamic priority method based on the number of constraints each of the managers has external to the mediation, a form of meta-level information. The basic notion is similar to the priority order changes in AWC: try to find the task that is most heavily constrained and elevate it in the orders. Our impression is that this helps stem the propagation because it leaves the most constrained tasks with the best choices. This allows those managers to maintain violation-free solutions if they exist in the alternatives presented to them.

	Slot 1	Slot 2	Slot 3
S1	T1	T1	T1
S2	T1	T1	T1
S3	T1/T2	T1/T2	T1/T2
S4	T1/T2	T1/T2	T1/T2
S5	T3/T2	T3/T2	T3/T2
S6	T3/T2	T3/T2	T3/T2
S7	T3	T3	T3
S8	T3	T3	T3



	Slot 1	Slot 2	Slot 3
S1		T1	T1
S2		T1	T1
S3	T2		T1
S4	T2	T1	T2
S5	T2		T2
S6	T3	T3	T2
S7	T3	T3	
S8	T3	T3	

Figure 10: A solution derived by SPAM to the problem in Figure 9. The table on the left is before track manager T2 mediates over T1 and T3. The table on the right is the result of stage 2.

In our example, T2 collects the ordering from T1, T2, and T3. T3 is given first choice. By its ordering, it ranked alternative 0 the highest. This restricts the choice for T2 to alternatives 0, 1, 2, and 3. T2 ranked 0 highest from this set of alternatives, restricting T1's choice to its 0th, 1st, and 2nd alternatives. It turns out that T1 likes its 0th solution the best so the final solution is composed of T3's alternative 0, T2's alternative 0, and T1's alternative 0.

The last phase of the protocol is the solution implementation phase. Here, the mediator simply informs each of the track managers of its final choice. Each of the track managers then implements the final solution. At this point, each of the track managers is free to propagate and mediate if it chooses. Figure 10 shows the configuration of the sensors before and after T2 completes stage 2.

4. Results and Discussion

The following is a list of the major accomplishments on the project:

- Development of SPAM heuristic resource allocation protocol
 - Showed importance of mediation-based negotiation (partial centralization) with overlapping context and extended views along critical paths for search and communication efficiency
- Development of APO distributed constraint algorithm and optAPO distributed optimization algorithm

based on SPAM concepts

- o Better performance than best known complete and optimal algorithms – AWC and ADOPT
- Development of SRTA soft real-time architecture
 - o Demonstrated that a sophisticated domain-independent agent architecture that operates in soft real-time could be built
- Demonstrated importance of organizational structuring for distributed resource allocation
 - o Showed how using negotiation organization could be dynamically constructed and efficiently modified as the environment changed
 - o Showed how an organization could be designed top-down
 - o Showed how the performance of a reasonably complex organization could be empirically analyzed and modeled analytically

Each of these topics is covered in detail in the collection of papers in the appendices (the complete list of supported publications is included in the bibliography).

5. Conclusions

We have described our solution to a real-time distributed tracking problem. The agents in the environment are first organized by partitioning them into sectors, reducing the level of potential interaction between agents. Within each sector, agents dynamically specialize to address scanning, tracking, or other goals, which are instantiated as task structures for use by the SRTA control architecture. These elements exist to support resource allocation, which is directly effected through the use of the SPAM negotiation protocol. The agent problem solving component first discovers and generates commitments for sensors to use for gathering data, then determines if conflicts exist with that allocation, finally using arbitration and relaxation strategies to resolve such conflicts. We have empirically tested and evaluated these techniques in both the Radsim simulation environment and using a hardware-based system.

Despite the fact that many of the details of our solution were designed for the distributed sensor net problem, much of the higher-level architecture is quite general, and applicable to different problems. SRTA, for instance, uses the domain-independent TÆMS language as its basis, which can and has been used successfully in a variety of domains. The SPAM negotiation protocol can be used to solve new distributed, interdependent resource allocation problems by implementing a suitable objective function. SPAM's technique of allowing conflicts to exist and be resolved by local control concurrent with a more complete allocation search can be used in nearly any environment where the participants are tolerant of such uncertainty. Our organizational structure as a whole is quite specific, but individual aspects such as partitioning, task migration and local control are general and applicable to a variety of different distributed architectures.

6. Technology Transfer

The technology developed in the project has had significant impact on how the following military and non-military applications have been approached.

- Rockwell Collins, Inc. is considering SRTA architecture components and SPAM/APO for DARPA HURT.
- The SPAM/APO is being evaluated by Boeing for air space deconfliction as part of their network-centric operations deconfliction thrust.
- Honeywell Laboratories has licensed the TÆMS/SRTA technologies for research use in First-Responder Applications.

- Real-time Tornado Tracking using Network of Phase-Arrayed Radars as part of NSF ERC at University of Massachusetts—scheduled for initial deployment in Oklahoma in 2006.

7. Professional Personnel Associated With This Effort

Professor Victor Lesser (PI), Professor Abhijit Deshmukh, Professor Krithi Ramamritham, Professor Janis Terpenney, Professor Yang Xiang, Dr. Daniel Corkill, Dr. Zhaotong Lian, Dr. Regis Vincent, Mr. Bryan Horling.

Bibliography

- Horling, Bryan; Mailler, Roger; and Lesser, Victor (2004). "Farm: A Scalable Environment for Multi-Agent Development and Evaluation." In *Advances in Software Engineering for Multi-Agent Systems*, pp. 220-237.
- Horling, Bryan; Mailler, Roger; and Lesser, Victor (2004). "A Case Study of Organizational Effects in a Distributed Sensor Network." To appear as extended abstract in *Proceedings of Third International Joint Conference on Autonomous Agents and MultiAgent Systems*. Full version available as University of Massachusetts Computer Science Technical Report #04-03.
- Lesser, V.; Decker, K.; Wagner, T.; Carver, N.; Garvey, A.; Horling, B.; Neiman, D.; Podorozhny, R.; NagendraPrasad, M.; Raja, A.; Vincent, R.; Xuan, P.; Zhang, X.Q. (2004). "Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework." To appear in *Autonomous Agents and Multi-Agent Systems*. Vol 9(1): 87–143. (This is a revised version of the paper that appeared in *Proceedings 1st International Conference on Autonomous Agents and Multi-Agent Systems*, 2002.)
- Mailler, Roger; and Lesser, Victor (2004). "Solving Distributed Constraint Optimization Problems Using Cooperative Mediation." To appear in *Proceedings of Third International Joint Conference on Autonomous Agents and MultiAgent Systems*.
- Mailler, Roger; and Lesser, Victor (2004). "Using Cooperative Mediation to Solve Distributed Constraint Satisfaction Problems." To appear in *Proceedings of Third International Joint Conference on Autonomous Agents and MultiAgent Systems*.
- Egyed, Alexander; Horling, Bryan; Becker, Raphen; and Balzer, Robert (2003). "Visualization and Debugging Tools." Distributed Sensor Networks: A multiagent perspective, (Lesser, V.; Ortiz, C.; Tambe, M., eds.), pp. 33-41.
- Horling, Bryan; Mailler, Roger; and Lesser, Victor (2003) "Farm: A Scalable Environment for Multi-Agent Development and Evaluation." In *Proceedings of the 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2003)*, pp. 171-177.
- Horling, Bryan; Mailler, Roger; Shen, Jiaying; Vincent, Regis; and Lesser, Victor (2003). "Using Autonomy, Organizational Design and Negotiation in a Distributed Sensor Network." Distributed Sensor Networks: A multiagent perspective, (Lesser, V.; Ortiz, C.; Tambe, M., eds.), pp. 139-183.
- Horling, Bryan; Mailler, Roger; Sims, Mark; and Lesser, Victor (2003). "Using and Maintaining Organization in a Large-Scale Distributed Sensor Network." In *Proceedings of the Workshop on Autonomy, Delegation, and Control (AAMAS03)*.
- Lesser, V.; Ortiz, C.; Tambe, M. (2003). Distributed Sensor Networks: A Multiagent Perspective. In Series: Multiagent Systems, Artificial Societies, and Simulated Organizations, Volume 9.
- Mailler, Roger and Lesser, Victor (2003). "Cooperative Negotiation for Optimized Distributed Resource Allocation in Soft Real-Time." UMass Computer Science Technical Report #2003-07.
- Mailler, Roger; and Lesser, Victor (2003). "A Mediation-Based Protocol for Distributed Constraint Satisfaction." In *The Fourth International Workshop on Distributed Constraint Reasoning*, pp. 49-58.
- Mailler, Roger; Lesser, Victor; and Horling, Bryan (2003). "Cooperative Negotiation for Soft Real-Time Distributed Resource Allocation." In *Proceedings of Second International Joint Conference on Autonomous Agents and MultiAgent Systems*, pp. 576-583.
- Mailler, Roger; Horling, Bryan; Lesser, Victor; Vincent, Regis (2003). "The Control, Coordination, and Organizational Design of a Distributed Sensor Network." Submitted to *IEEE Transactions on Systems, Man, and Cybernetics: Part C*. (Under review)
- Sims, Mark; Goldman, Claudia; and Lesser, Victor (2003). "Self-Organization through Bottom-up Coalition Formation." In *Proceedings of Second International Joint Conference on Autonomous Agents and MultiAgent Systems*, pp. 867-874.
- Wang, Guandong; Zhang, Weixiong; Mailler, Roger; and Lesser, Victor (2003). "Analysis of Negotiation Protocols by Distributed Search." Distributed Sensor Networks: A multiagent

- perspective, (Lesser, V.; Ortiz, C.; Tambe, M., eds.), pp. 339-361.
- Xiang, Yang; Lesser, Victor (2003). "On the Role of Multiply Sectioned Bayesian Networks for Cooperative Multiagent Systems." *IEEE Systems, Man, and Cybernetics (Part A)*, Volume 33, Number 4, pp. 489-501.
- Xuan, Ping; Lesser, Victor (2003). "Using Agent Commitments as Planning Contexts." In *International Journal on Cooperative Information Systems*. (Under review)
- Zhang, X.Q.; Lesser, V.R.; Wagner, T. (2003). "A Two-Level Negotiation Framework for Complex Negotiations." In *Proceedings of the 2003 IEEE/WIC International Conference on Intelligent Agent Technology*, pp. 311-317.
- Zhang, XiaoQin; Lesser, Victor; Podorozhny, Rodion. (2003). "Multi-Dimensional, MultiStep Negotiation for Task Allocation in a Cooperative System." To appear in *Autonomous Agents and MultiAgent Systems*.
- Zhang, XiaoQin; Lesser, Victor; Wagner, Thomas. (2003). "Integrative Negotiation in Complex Organizational Agent Systems." *Proceedings of the 2003 IEEE/WIC International Conference on Intelligent Agent Technology (IAT 2003)*, pp. 140-146.
- Zhang, X.Q.; Lesser, V.; and Abdallah, S. (2003). "Efficient Ordering and Parameterization of Multi-Linked Negotiation," (Extended abstract). *Proceedings 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 1170-1171. (Full version available as University of Massachusetts Computer Science Technical Report #02-42.)
- Zhang, X.Q.; Lesser, V.; and Wagner, T. (2003). "A Multi-Leveled Negotiation Framework," (Extended abstract). *Proceedings 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 1172-1173. (Full version available as University of Massachusetts Computer Science Technical Report #02-44.)
- Horling, Bryan; Lesser, Victor; Vincent, Regis and Wagner, Thomas. (2002). "The Soft Real-Time Agent Control Architecture." *Proceedings of the AAAI/KDD/UAI-2002 Joint Workshop on Real-Time Decision Support and Diagnosis Systems*. (Also available as UMass Computer Science Tech Report 02-14.)
- Lesser, V.; Decker, K.; Wagner, T.; Carver, N.; Garvey, A.; Horling, B.; Neiman, D.; Podorozhny, R.; NagendraPrasad, M.; Raja, A.; Vincent, R.; Xuan, P.; Zhang, X.Q. (2002). "Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework," (Plenary Lecture/Extended Abstract). *Proceedings 1st International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 1-2. (Full version available as University of Massachusetts/Amherst Computer Science Technical Report 02-03.)
- Raja, Anita; Lesser, Victor (2002). "Meta-Level Control in Multi-Agent Systems." *Proceedings of AAAI/KDD/UAI-2002 Joint Workshop on Real-Time Decision Support and Diagnosis Systems*, Volume WS-02-15, pp. 47-53.
- Xuan, Ping; Lesser, Victor (2002). "Multi-Agent Policies: From Centralized Ones to Decentralized Ones." *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems*, Part 3, pp. 1098-1105.
- Zhang, Xiaoqin; Lesser, Victor (2002). "Multi-Linked Negotiation in Multi-Agent Systems." *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems*, pp. 1207-1214.
- Zhang, Xiaoqin; Lesser, Victor; and Wagner, Tom (2002). "Integrative Negotiation in Complex Organizational Agent Systems." In University of Massachusetts Computer Science Tech Report #02-43.
- Zhang, Xiaoqin; Lesser, Victor; Wagner, Tom (2002). "Integrative Negotiation in Complex Organizational Agent Systems." *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems*, pp. 503-504. (Full version available as University of

Massachusetts Computer Science Technical Report #02-43.)

- Horling, Bryan, Benyo, Brett, and Lesser, Victor (2001). "Using Self-Diagnosis to Adapt Organizational Structures." *Proceedings of the 5th International Conference on Autonomous Agents*, pp. 529–536.
- Horling, Bryan, Vincent, Regis, Mailler, Roger, Shen, Jiaying, Becker, Raphen, Rawlins, Kyle and Lesser, Victor (2001). "Distributed Sensor Network for Real Time Tracking." *Proceedings of the 5th International Conference on Autonomous Agents*, pp. 417–424.
- Mailler, Roger; Vincent, Regis; Lesser, Victor; Shen, Jiaying; Middlekoop, Tim (2001). "Soft Real-Time, Cooperative Negotiation for Distributed Resource Allocation." *AAAI Fall Symposium on Negotiation Methods for Autonomous Cooperative Systems*, AAAI Press, Technical Report FS-01-03, pp. 63–69.
- Raja, A.; Lesser, V. (2001). "Real-Time Meta-Level Control in Multi-Agent Systems." *Proceedings of Multi-Agent Systems and Applications - ACAI 2001 & EASSS 2001 Student Sessions*. (Also in *Proceedings of Adaptability and Embodiment Using Multi-Agent Systems: AEMAS 2001 Workshop*.)
- Raja, A.; Wagner, T.; Lesser, V. (2001). "Reasoning about Uncertainty in Agent Control." *Proceedings of the 5th International Conference on Information Systems, Analysis, and Synthesis, Computer Science and Engineering*. Part 1, Volume VII, pp. 156–161.
- Raja, Anita; Lesser, Victor (2001). "Towards Bounded-Rationality in Multi-Agent Systems: A Reinforcement Learning Based Approach." In University of Massachusetts Computer Science Technical Report #01-34.
- Vincent, Regis; Horling, Bryan; Lesser, Victor; Wagner, Thomas (2001). "Implementing Soft Real-Time Agent Control." *Proceedings of the 5th International Conference on Autonomous Agents*, pp. 355–362.
- Vincent, Regis; Horling, Bryan; and Lesser, Victor (2001). "An Agent Infrastructure to Build and Evaluate Multi-Agent Systems: The Java Agent Framework and Multi-Agent System Simulator." *Lecture Notes in Artificial Intelligence: Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, Volume 1887.
- Xuan, P.; Lesser, V.; Zilberstein, S. (2001). "Communication Decisions in Multi-agent Cooperation: Model and Experiments." *Proceedings of the 5th International Conference on Autonomous Agents*, pp. 616–623.
- Zhang, X.Q.; Lesser, V.; Podorozhny, R. (2001). "New Results on Cooperative, MultiStep Negotiation over a Multi-Dimensional Utility Function." *AAAI Fall Symposium on Negotiation Methods for Autonomous Cooperative Systems*, AAAI Press, Technical Report FS-01-03, pp. 1–10.
- Zhang, X.Q.; Lesser, V.; Wagner, T. (2001). "A Proposed Approach to Sophisticated Negotiation." *AAAI Fall Symposium on Negotiation Methods for Autonomous Cooperative Systems*, AAAI Press, Technical Report FS-01-03, pp. 96–105.
- Horling, Bryan; Benyo, Brett; Lesser, Victor (2000). "Using Self-Diagnosis to Adapt Organizational Structures," (Extended abstract). *Proceedings of 4th International Conference on MultiAgent Systems*, pp. 397–398.
- Raja, Anita; Lesser, Victor; Wagner, Thomas (2000). "Toward Robust Agent Control in Open Environments." *Proceedings of 5th International Conference of Autonomous Agents*, pp. 84–91.
- Raja, Anita; Wagner, Thomas; Lesser, Victor (2000). "Reasoning about Uncertainty in Design-to-Criteria Scheduling." *Proceedings of AAAI 2000 Spring Symposium on Real-Time Autonomous Systems*, pp. 76–83.
- Wagner, Thomas; Lesser, Victor (2000). "Design-to-Criteria Scheduling: Real-Time Agent Control." *Proceedings of AAAI 2000 Spring Symposium on Real-Time Autonomous Systems*, pp. 89–96.
- Wagner, Thomas; Benyo, Brett; Lesser, Victor; Xuan, Ping (2000). "Investigating Interactions Between Agent Conversations and Agent Control Components." *Issues in Agent Communication, Lecture Notes in Artificial Intelligence*, pp. 314–331.

- Xiang, Yang; Lesser, Victor (2000). "Justifying Multiply Sectioned Baysean Networks." *Proceedings of the 4th International Conference on Multi-Agent Systems*, pp. 349-356.
- Xuan, Ping; Lesser, Victor; Zilberstein, Shlomo (2000). "Communication in Multi-agent Markov Decision Processes," (Poster session). *Proceedings of the 4th International Conference on MultiAgent Systems*, pp. 467-468.
- Horling, Bryan; Benyo, Brett; Lesser, Victor (1999). "Using Self-Diagnosis to Adapt Organizational Structures." In UMass Computer Science Tech Report 1999-64.

FINAL TECHNICAL REPORT
Agreement No. F30602-99-2-0525
“Scalable Real-Time Negotiation Toolkit”

APPENDIX A

A Cooperative Mediation-Based Protocol for Dynamic, Distributed Resource Allocation

Roger Mailler and Victor Lesser

Abstract—In this article, we present a cooperative mediation-based protocol that solves a distributed resource allocation problem while conforming to soft real-time constraints in a dynamic environment. Two central principles are used in this protocol that allow it to operate in constantly changing conditions. First, we frame the allocation problem as an optimization problem, similar to a Partial Constraint Satisfaction Problem (PCSP), and use relaxation techniques to derive conflict (constraint violation) free solutions. Second, by using overlapping mediation sessions to conduct the search, we are able to prune large parts of the search space by using a form of arc-consistency. This allows the protocol to both quickly identify situations when the problem is over-constrained and to determine the appropriate repair. From the global perspective, the protocol has a hill climbing behavior and because it was designed to work in dynamic environments, is an approximate one. We describe the domain which inspired the creation of this protocol, as well as discuss experimental results.

I. INTRODUCTION

Resource allocation is a classic problem that has been studied for years by multi-agent systems researchers [1], [2]. The reason for this is that resource allocation shares a number of characteristics that are common to a wide range of multi-agent domains. For example, resource allocation requires search and is often too complex and time consuming to perform in a centralized manner when the environmental characteristics are both distributed and dynamic. In fact, in environments where search is being conducted and the costs associated with continuously centralizing a lot of information are impractical, distributed techniques become imperative.

Cooperative, iterative search (negotiation), has been viewed as a viable technique for handling complex searches of this type that include multi-linked interacting subproblems [1]. Unfortunately, a common drawback to this technique is that it prevents the agents from making informed decisions about the effects of changing their local allocation without actually doing it. Because of the length of time required for this technique to converge on a solution, researchers have often abandoned the optimization portion of resource allocation, instead modeling them as distributed constraint satisfaction problems [3], [4], in order to provide reasonable solution speed.

In this work, we extend the traditional formulation of the resource allocation problem in two ways. First, we introduce soft real-time deadlines on the protocol's behavior. These deadlines require the protocol to adapt to the remaining available time, which is estimated dynamically as a result of

emerging environmental conditions. Second, we reformulate the resource allocation task as an optimization problem, and like the Distributed Partial Constraint Satisfaction Problem (PCSP) [5]–[7], use constraint relaxation techniques to find a conflict-free solution while maximizing the social utility of the agents.

In this article, we present a distributed, mediation-based protocol that takes advantage of the cooperative nature of the agents in the environment to maximize social utility. By mediation-based, we are referring to the ability of each of the agents to act in a mediator capacity when resource conflicts are recognized. As a mediator, an agent gains a localized, partial view of the global allocation problem and makes suggestions to the allocations for each of the agents involved in the *mediation session*. This allows the mediator to identify over-constrained subproblems and make suggestions to eliminate such conditions. In addition, the mediator can perform a localized arc-consistency check, which potentially allows large parts of the search space to be eliminated without having to go through an number of trial and error steps. Together with the fact that regions of mediation overlap, the agents rapidly converge on solutions that are in most cases good enough and fast enough. Overall, the protocol has many characteristics in common with distributed breakout [8], particularly its distributed hill-climbing nature and the ability to exploit parallelism by having multiple mediated sessions occur simultaneously.

In the remaining sections of this article, we introduce the distributed monitoring and tracking application which motivated the development of our protocol. Next, we describe the Scalable Protocol for Anytime Mediation (SPAM), a distributed, cooperative mediation-based protocol that was developed and has been tested on actual sensor hardware. In section IV, we will introduce Farm, a distributed simulation environment used to test SPAM, and present and discuss the results of testing SPAM within that simulator. The last section of the article will present conclusions for this work.

II. DOMAIN

The resource allocation problem that motivated this work requires an efficient allocation of distributed sensing resources to the task of tracking targets in an environment. In this problem, multiple sensor platforms are distributed with varying orientations in a real-time environment [9]. Each platform has three distinct radar-based sensors, each with a 120 degree viewable arc, which are capable of taking amplitude (measuring distance from the platform) and/or frequency (measuring

the relative velocity of the target) measurements. In order to track a target, and therefore obtain utility, at least three of the sensor platforms must take a coordinated measurement of the target, which are then fused to triangulate the target's position. Increasing the number, frequency, and/or relative synchronization of the measurements yields better overall quality in estimating the target's location and provides a higher quality solution. The sensor platforms are restricted to only taking measurements from one sensor head at a time with each measurement taking about 500 milliseconds. These key restrictions form the basis of the resource allocation problem.

Each of the sensor platforms is controlled by a single agent which may take one or more organizational roles, in addition to managing its local sensor resources. Each of the agents in the system maintains a high degree of local autonomy, being able to make trade-off decisions about competing tasks using the SRTA agent architecture [10].

One notable role that an agent may take on is that of track manager. As a track manager, the agent becomes responsible for determining which sensor platforms and which sensor heads are needed both now and in the future for tracking a single target. Track managers also act to fuse the measurements taken from the individual sensor platforms into a single location. Because of this, track managers are the focal point of any activities that take place as part of resolving resource contention.

Dynamics are introduced into the problem as a result of target movement. During the course of a run, targets continuously enter and leave the viewable area of different sensors, which then require track managers to continuously evaluate and revise their resource requirements. This, in turn, changes the underlying structure of the actual allocation problem. In addition, these dynamics drive the need for real-time problem solving, because a particular problem structure only holds for a limited amount of time.

Resource contention is introduced when more than one target enters the viewable range of the same sensor platform. Because of the time it takes to perform a measurement, combined with the fact that each sensor can take only one measurement at a time, track managers must come to an agreement over how to share sensor resources, without causing any targets to be lost. This local agreement can have profound global implications. For example, what if as part of its local agreement, a track manager relinquishes control of a sensor platform and takes another instead? This may introduce contention with another track manager already using that sensor, who may then have to request alternate sensor resources to make up for the new deficiency.

A. The Resource Allocation Problem

Generally speaking, we say that a resource allocation problem is the problem of assigning a (usually limited) number of resources to a set of tasks. Each of the tasks may have different resource requirements, and may have the potential for varying utility depending on which resources they use. The goal is to maximize the global utility of the assignment, choosing the right options for the tasks and assigning the

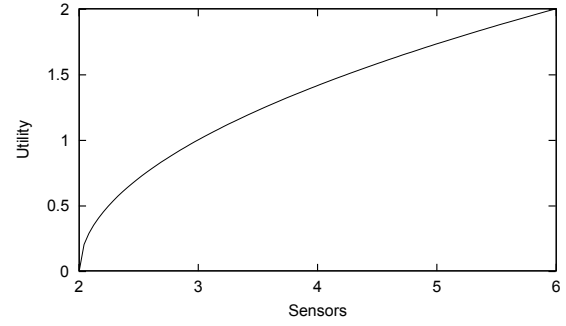


Fig. 1. Utility of taking a single, coordinated measurement from a set of sensors.

correct resources to them. More formally, a resource allocation problem is comprised of:

- a set of tasks, $T = \{t_1, \dots, t_n\}$
- a set of resources $R = \{r_{1,1}, \dots, r_{j,k}\}$ where j is the number of resources and k is the planning horizon for the resource.
- a set of utility functions each associated with one of the tasks $U = \{U_1, \dots, U_n | U_i : 2^R \mapsto \mathbb{R}\}$

The goal of the problem is to come up with an allocation $A = \{a_1, \dots, a_n | a_i \in 2^R\}$ such that the following conditions are met:

- 1) $\sum_{i=1}^n U_i(a_i)$ is maximized.
- 2) $\bigcap_{i=1}^n a_i = \emptyset$.

The notation 2^R is used to indicate the power-set of the resources. Because the resource requirements may change over time, or a particular pattern of resource usage may be needed to obtain utility for a task, resources are broken down on both the resource and time dimensions, hence the need for a planning horizon. Increasing the number of resources or the planning horizon can have a significant effect on the overall complexity of the allocation problem, which is known to be NP-complete.

The first condition above basically makes this problem an optimization problem and can be viewed as a soft constraint on the solution. The second condition is a hard constraint since we know that a single resource cannot be applied to two tasks simultaneously. As we will discuss later, we may not always strictly adhere to the second condition using high level distributed search. In fact, we rely on the agents within the system to always ensure this condition is satisfied during times when the SPAM protocol has not.

According to this problem formulation, each task has its own utility function and the utility of assigning a set resources to a task is strictly dependent on that individual function. In fact, in dynamic domains, these function may change over time which alters the underlying relationships between tasks. What this also means is that due to the sharing of resources, increasing the utility of a particular task may not increase the global utility. We make no assumptions in this paper about task independence.

The distributed version of the resource allocation problem, which is the focus of this paper, has each task assigned to a single agent. However, in general an agent may take on more

than one task.

B. Tracking as Resource Allocation

Modeling the target tracking domain as a resource allocation problem is fairly straightforward. Each of the targets in the environment can be considered a task, which is assigned to a track manager. The sensors are the resources and the job of the track managers is to obtain enough sensing time from the correct sensors to track their targets.

At any given time, each of the targets is within the viewable range of some subset of the sensors. That means that as the targets move from the viewable range of some sensors to others, the utility function associated with each of the tasks change. In addition, tracking involves coordinating measurements from three or more sensors which are then fused together to form an estimated position of the target. Increasing the number of sensors improves the quality of an estimate by the function given in Figure 1 which is based on the RMS minimization method used to triangulate the targets. Increasing the frequency of the triangulation yields a linear increase in the overall quality of the track i.e. two measurements during a given period is twice as good as one.

Because targets are often in the viewable range of a sensor for an extended period of time, planning within our system is periodic. This simply means that the sensors continuously repeat their assigned schedules until a change is made. We often refer to the planning horizon (corresponds to k) as a period and an individual element within the period as a slot.

If we say that M_s^i is the set of good sensors measurements (can see the target) leading to the positional estimate in a single slot s for a task i , then the utility function for that task during a specific period is:

$$U_i(a_i) = \sum_{s=1}^k Util(M_s^i)$$

which basically says that the utility of a task for a specific period is the sum of the slot utilities for the slots within the period.

The special nature of the utility functions in the tracking domain actually allow us to consider a much smaller subset of the possible allocations for a given task. In fact, track managers within our system use a simplified set of objective levels defined by their utility functions to assign resources to their targets. Each objective level is expressed as a cross product $D_m \times D_s$ denoting the number of resources from their acceptable set, desired for a number of slots in planning horizon. For example, a track manager may wish to have three sensors for two slots, which is denoted 3×2 . Although the number of slots in a period is variable, for this domain, we typically set it to match the number of sensor heads on each platform, which is three.

There are essentially two benefits to using this abstract approach to resource scheduling. First, this representation vastly reduces the search space by discretizing time into slots and aligning the slots between the agents (the agents are time synchronized using Network Time Protocol (NTP)). It is easy for a manager to know that its measurements are coordinated

if it has scheduled all of them during slot 1 and it knows that each of the sensors executes methods in slot 1 at about the same time. Second, by working abstractly, managers leave the details of the actual implementation of the period-based schedule to the agents themselves. This leaves the individual agents quite a bit of flexibility in how they internally manage competing tasks.

Note that if a target is ignored (i.e. not being triangulated at all during a full period), we penalize ourselves by subtracting two from the social utility. This penalty approximates the expected gain the agents would obtain by starving one of the track managers which makes the allocations a bit more “fair”.

III. PROTOCOL

The Scalable Protocol for Anytime Mediation (SPAM) is built around the principle of good enough, fast enough. As such, the protocol is actually divided into two major stages. As the protocol transitions from the first stage to the second, the agent acting as the track manager gains more context information and is, therefore, able to improve the quality of its overall decision. In addition, to allow stage 2 time to complete, without losing all quality in the interim, stage 1 of the protocol always ensures that at least some solution has been obtained. So, at any time after the completion of stage 1, the track manager can choose to stop the protocol and is ensured to have a solution, albeit not necessarily a good one.

The SPAM protocol is activated whenever a local change in the resources is needed or if a manager detects a change in the level of contention within one of the resources it's using. Detecting a change in the resource needs is done by monitoring the location of the target as it moves within the sensor field. Track managers constantly evaluate each of the sensors based first, on the ability of the sensor to see their target and second, on the expected value of the raw data returned by the sensor. This can be most easily understood as a change to the utility function used for the track. Whenever a target moves out of the view of one of the sensors currently being used to track it, into the view of a new sensor, or if a sensor currently not being utilized becomes more valuable than a sensor being used, the manager starts a SPAM session.

Whenever SPAM is activated for this reason, managers set their objective level to the highest possible value which ensures the hill-climbing nature of the algorithm. To understand this point, consider the following example:

Let's say that a new resource were added to the possible resources that could be used by manager T1 to track its target. Let's also say that another manager, T2, who has more than enough available resources to itself, was using that resource. If T1 starts SPAM at a low objective level, it may find a solution that is conflict-free, but will never realize that it could have gotten a better solution where T2 just gives up the entire conflicted resource. From a PCSP perspective this just means that whenever the structure of a CSP changes, the PCSP algorithm should reset its initial bound and attempt to satisfy all of the constraints before beginning relaxation.

The second reason for starting a SPAM session is when a manager recognizes a change in the level of contention within

one of the sensors currently being used to track its target. This differs considerably from a resource change. Here the manager is recognizing that there is a change in the utility being obtained or could potentially be obtained for its target. The utility function itself has not been changed, only the interactions between it and some other function within another manager. Track managers detect these changes by monitoring the resource schedules of the sensors. To facilitate this process, sensors inform the managers that are utilizing them about the state of their resource schedule whenever a change occurs. It is certainly conceivable and, in fact likely, for two managers to detect changes in contention at the same time.

There are actually two separate cases here. When a manager recognizes a previously unknown conflict (i.e. a new restriction), it is most likely caused by another manager choosing an assignment that uses the resources because it either didn't know it was being used, or was forced to as a result of a mediation. In other words, this case most often occurs when there is a multi-linked problem within the environment. When managers recognize this case, they do not change their objective level before starting SPAM. The reason for this is easy to understand after considering an example. Let's say you have three managers, T1, T2, and T3. T1 has a conflict with T2, and T2 is sharing resources with T3, but is not in conflict. As a result of a mediation between T1 and T2 their conflict is solved, but it creates a conflict between T2 and T3. When T2 recognizes this problem, if it reset its objective level, the problem becomes harder to solve because it may reintroduce conflict with T1 as well as increasing the conflict with T3.

The other type of change occurs when there is a relaxation of contention on a resource. Again, managers recognize this type of change by monitoring the resource schedules of the sensors they are using. Whenever a manager realizes that it can improve its local utility (increase its objective level) without creating new conflicts, it sets its objective level to that new increased value, and starts SPAM. As you will see, when SPAM executes, it will make a simple local change to its assignment to take advantage of the additional resources because it can find a local solution that is conflict free.

A. Stage 1

Stage 1 of SPAM (see figure 2) serves three primary functions. The first is to find a suitable solution within the context of the information that the protocol has when it starts up. Like the Asynchronous Weak Commitment (AWC) protocol [11], each of the agents tries to find a resource assignment that is based solely on their incomplete or inconsistent view of the current resource schedules from the sensors. Resource assignments derived at this level must meet two criteria. First, they must find at the objective level that was set at the startup of SPAM. This ensures the continued hill-climbing nature of the search. Second, the resource assignment cannot create a conflict with another track manager. This criteria ensures that the overall effect of the assignment is not globally negative. If an assignment is found that adheres to these restrictions, then no further work is needed and the protocol terminates at the end of stage 1.

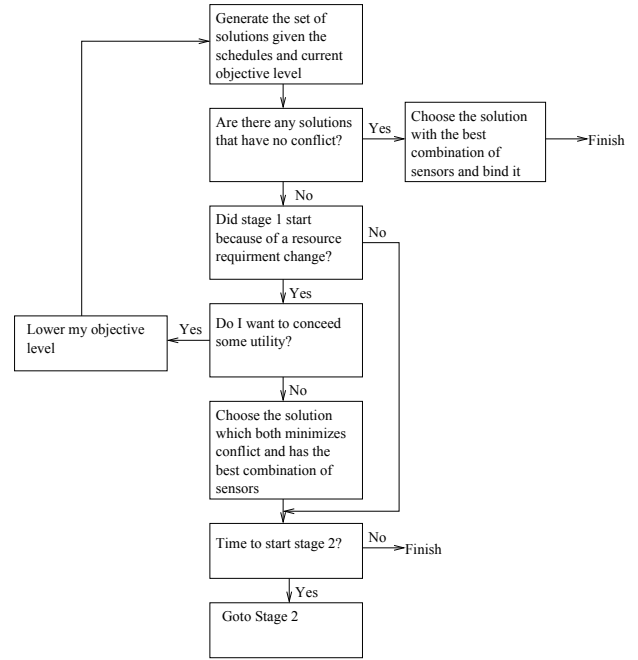


Fig. 2. Stage 1 of the SPAM protocol.

We should mention that a trade-off exists between communication overhead/solution speed and utility based on the selection of the objective level that was set at the startup of the protocol. If each of the managers chooses to use every available resource (sensors able to see their target), the possibility for contention over resources greatly increases in the environment, thereby causing the execution of stage 2 to occur more frequently. However, if the agents decide to start with at a lower objective level (and correspondingly less utility), the social utility may suffer unnecessarily.

To take advantage of this trade-off, stage 1 was designed with a feature, called *utility concessioning*. The key idea behind utility concessioning is that often, small changes in a manager's local utility can both remove all of the conflicts on its resource assignment (thus improving global utility) and prevent it from having to wait for a mediation session to finish. In SPAM, we have a parameter, called the concession rate, which defines how much of the local solution quality a track manager is willing to concede to find a violation-free solution, in an attempt to avoid the potentially expensive stage 2. The rate is defined as a percentage of the manager's current utility, so as the manager's utility drops, the amount they are willing to concede drops as well. That means that in critically constrained tracking environments, the managers attempt to mediate more frequently preventing unnecessary loss of utility.

To demonstrate the effects of the concession rate on the SPAM protocol, we conducted a series of tests that varied the concession rate and the number of targets within a fixed 20 sensor environment. So, as the number of targets increases, the resource contention over the sensors increases as well. Each data point represents the average over 50 runs where both the targets and sensors are placed in the environment at a fixed, random location then the SPAM protocol is run until the agents reach a solution. A total of 4400 test runs were conducted to collect this data.

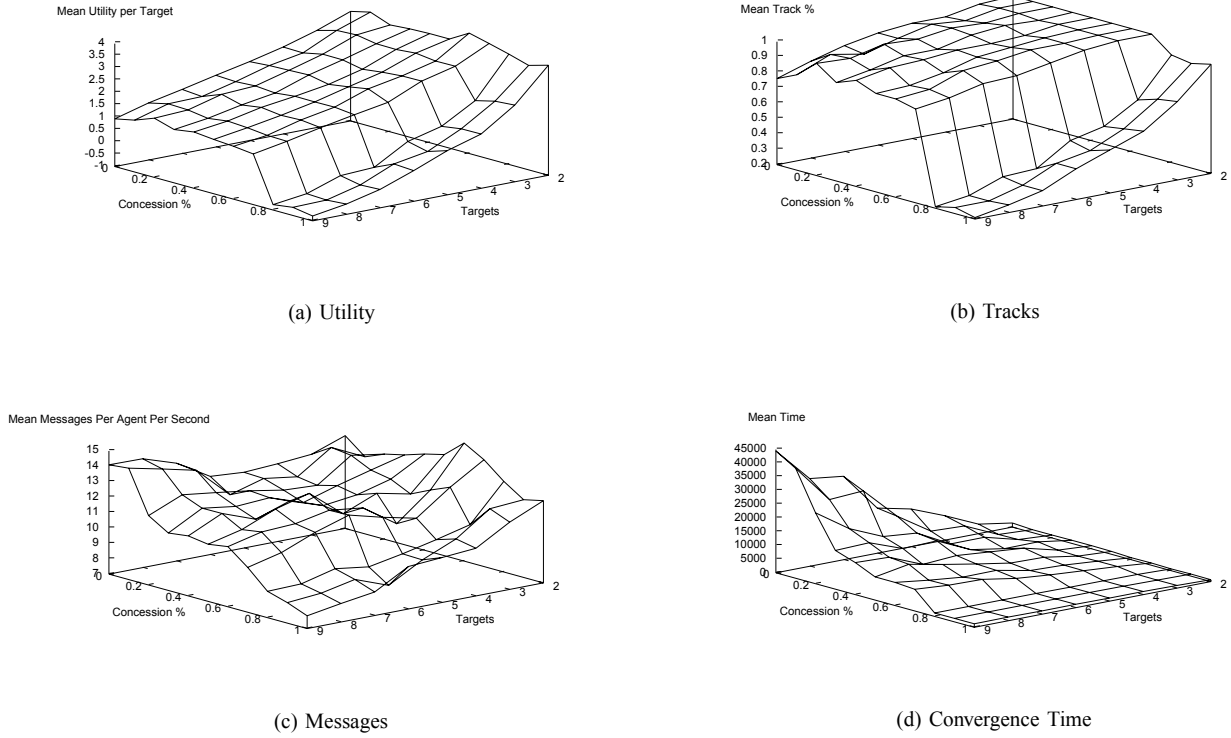


Fig. 3. Results of experimentation with the utility concessioning used in Stage 1 of the SPAM protocol.

The results of these experiments can be seen in figure 3. As you can see from the graphs, the local concession rate has a profound effect on the overall systems utility, the number of targets being tracked, the convergence time, and the number of messages. For most scenarios a high concession rate leads to relatively high utility while saving vast amounts of communication and computation. For example, for 9 targets, graph 3(a) shows that the utility is not dramatically effected by increasing the concession rate, but according to graph 3(d), increasing the rate considerably improves the convergence time for the protocol. The effects of losing local utility become apparent at higher concession rates though. The dramatic drop-off that occurs at a rate of 0.8 is caused by agents conceding all of their local utility in order to become conflict free. Essentially, the agents begin to ignore their targets by conceding all of their local utility in order to avoid having to mediate.

The second function of stage 1 is to ensure that some degree of utility is obtained as soon as possible whenever the protocol is started due to a resource requirement change. This solution, although not conflict free, has the ability to obtain utility while the manager tries to get a better solution by going into stage 2. Conflicts that are unresolved during this period of time are left to the individual sensor agents to handle. Sensor agents can use one of a number of techniques, including slot boundary shifting, less expensive measurement types, or task rotation, in order to solve such conflicts. To the track manager, whether or not they get a measurement from a conflicted sensors is probabilistically random.

The third function of stage 1 is to provide the protocol with anytime characteristics. Because a solution is always derived

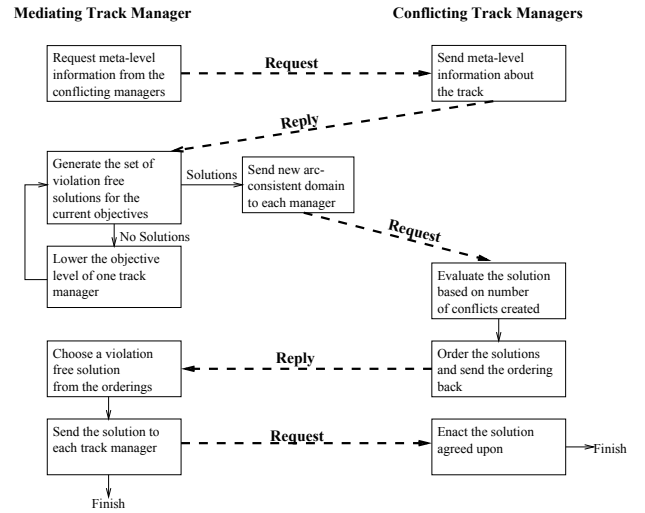


Fig. 4. Stage 2 of the SPAM protocol.

and applied during stage 1, managers don't necessarily have to enter stage 2. They can stop the process at the end of stage 1 and accept the results that they have achieved. This is often done if a target's movement causes the resource needs to change faster than the expected time it would take to complete stage 2. The expected time to complete stage 2 is computed based on both previous experience and the current estimated communication speeds for the track managers that would be in the mediation session.

B. Stage 2

Stage 2 is the heart of the SPAM protocol (See figure 4). Stage 2 attempts to resolve resource contention by elevating the discussion to the track managers that are in direct conflict. To do this, one of the track managers takes the role of the mediator for the local conflict (note that multiple mediation sessions can occur in parallel in the environment). As the mediator, it becomes responsible for gathering all of the information needed to generate alternative solutions, generating these solutions which may involve changes to the objective levels of the managers involved, and finally choosing a solution to apply to the problem. Because these solutions are generated without full global information, however, the final solution may lead to newly introduced non-local conflict. If this occurs, another track managers can choose to take over the role of mediator in order to correct these newly introduced conflicts if they have the time. So, what started out as a new target or resource requirement change, may lead to a number of mediation sessions propagating across the problem landscape.

Looking at this from a more formal perspective. If the set of resources that are usable for a single task t_i is defined as

$$R(t_i) = \{r_{u,v} | r_{u,v} \in R \wedge \exists a (U_i(a \cup r_{u,v}) > U_i(a))\}$$

then the set of acceptable resource assignments for a single task t_i is

$$D(t_i) = \{a | a \in 2^{R(t_i)} \wedge U_i(a) > 0\}$$

and the neighbor tasks to a mediator m are

$$N_m = \{t_i | t_i \in T \wedge R(t_m) \cap R(t_i) \neq \emptyset\}$$

then the problem that a mediating manager m is working on is

- a set of tasks, $T_m = \{t_m \cup N_m\}$
- a set of resources $R_m = \{r_{u,v} | r_{u,v} \in (\bigcup_{t_i \in N_m} R(t_i)) \cap R(t_m)\}$
- a set of utility functions $\hat{U} = \{\hat{U}_i | t_i \in T_m\}$

The goal of this subproblem is the same as the goal of the global problem. The notation $\hat{U}_i$ is used to indicate an approximation function to the actual U_i for each of the managers. Also note that $R_m \subseteq \bigcup_{t_i \in N_m} R(t_i)$. What this means is that the view of the mediating manager is limited to only the constraints that arise from the sharing of a resource with the mediator. These conditions, when combined together, indicate that the estimated utility of a solution to the subproblem is always either equal to or an over-approximation to the actual utility obtained socially. This is simply a by-product of performing a localized search. The mediator never knows if the assignments it proposes at a given utility value will cause conflict outside of its view, which is why we allow the managers to propagate. You should also note that the set T_m may not strictly include every one of the mediator's neighbors. Some track managers may not be using be a resource from $R(t_m)$ even though that resource belongs to their $R(t_i)$ and therefore cannot be seen by the mediator (i.e. the mediator is unaware of their relationship).

The best way to explain how stage 2 operates is through an example. Consider figure 5. This figure depicts a commonly

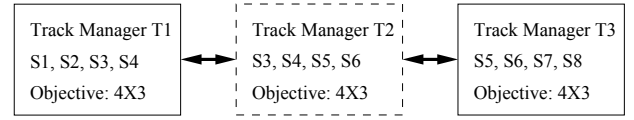


Fig. 5. Example of a common contention for resources. Track manager T2 has just been assigned a target and contention is created for sensors S3, S4, S5 and S6.

encountered form of contention. Here, track manager T2 has just been assigned a target. The target is located between two existing targets that are being tracked by track managers T1 and T3. This creates contention for sensors S3, S4, S5, and S6.

Following the protocol for the example in figure 5, track manager T2, as the originator of the conflict, takes on the role of mediator. It begins the solution generation phase by requesting meta-level information from all of the track managers that are involved in the resource conflict. The information that is returned includes the current objective level that the track manager is using, the number of sensors which could possibly track the target, the names of the sensors that are in direct conflict with the mediator, and any additional conflicts that the manager has. To continue our example, T2 sends a request for information to T1 and T3. T1 and T3 both return that they have 4 sensors that can track their targets, the list of sensors that are in direct conflict (i.e. $T1(S3, S4)$, $T3(S5, S6)$) their objective level (4×3 for both of them) and that they have no additional conflicts outside of the immediate one being considered. Note that sensors S1, S2, S7, and S8 are not in direct conflict and therefore are not mentioned by T1 and T3.

Using this information, T2 is able to generate $D(t_i)$ for each of the tasks in the set T_m for the objective levels that are passed in as part of the meta-level information (see section III-D). With the full set of $D(t_i)$'s, it's fairly easy to generate all possible satisfying assignments $\mathbf{A}$ with each element being a particular $A_m = \{a_i | t_i \in T_m \wedge a_i \in D(t_i)\}$ s.t. the condition $\bigcap_{a_i \in A_m} a_i = \emptyset$ is met.

As you can see in figure 4, T2 enters a loop that involves attempting to generate these sets followed by lowering one of the track manager's objective level if $\mathbf{A} = \emptyset$ given the current objective levels of each of the track managers. One of the principle questions that we are currently investigating is how to choose the track manager that gets its objective level lowered when $\mathbf{A}$ is empty. Right now, this is done by choosing the track manager with the highest current objective level, which cannot support its demands with resources outside of the set R_m and lowering them. This has the overall effect of balancing the objective levels of the track managers involved in the session. Whenever two or more managers have the same highest objective level, we choose to lower the objective level of the manager with the least amount of external conflict. By doing this, it is our belief, that track managers with more external conflict will maintain higher objective levels, which leaves them more leverage to use in subsequent sessions as a result of propagation.

You should note that although this has similarities to the techniques used in PCSPs, this differs in that the actual CSP problem changes as the objective levels are changed.

PCSP techniques, such as [5]–[7] choose a subset of the constraints to satisfy, we actually change the structure of the constraints, removing them by lowering the objective levels, until the problem becomes satisfiable. We also differ from the Distributed Constraint Optimization (DCOP) [12], [13] work in that although DCOPs have a utility function over the possible assignments to a problem, methods for solving them do not change the underlying CSP to ensure satisfiability.

The solution generation loop is terminated under one of two conditions. First, if given the current objective levels for each of the track managers, the set $\mathbf{A} \neq \emptyset$, the session enters the solution evaluation phase. Second, we cannot find a track manager to lower without $D(t_i) = \emptyset$ and $\mathbf{A} = \emptyset$. Under this condition, the session is terminated and the mediator takes a partial solution at the lowest objective level that minimizes the resulting conflict, conceding that it cannot find a full solution.

Continuing our example, T2 first lowers the objective level of T1 (choosing T1 at random because they all have equal external conflict). No full solutions are possible under the new set of objective levels, so the loop continues. It continues, in fact, until each of the track managers has an objective level of 3×2 at which time T2 is able to generate a set of 216 (the number of elements in $\mathbf{A}$) solutions to the problem.

During the solution evaluation phase, the mediator sends each of the track managers a set:

$$d_i = \{a | a \in D(t_i) \wedge \exists A_m \in \mathbf{A} (a \in A_m)\}$$

What should be clear is that each of the d_i is arc-consistent for every constraint between elements in the set R_m . What that means is that for the mediator's resources, all constraints are satisfied.

The purpose of this message is actually two-fold. The first purpose is to obtain information about the effect of imposing a particular solution. The second purpose is to obtain a lock from the conflicting manager. This lock prevents the manager from changing its value while it is in a session which allows multiple sessions to occur simultaneously in the environment. If the manager is already locked, it informs the mediator who simply drops them from the session. This, of course, means that the overall session may not end with an entirely conflict-free solution, but in most cases allows the mediator to correct some of the conflicts while it waits for the lock to clear.

Each of the managers that remains in the session, using its set d_i and a revised objective level, determines which, if any, of the solutions are satisfiable given the local *agent_view* and which is best given the actual U_i . In our example, T2 sends 24 alternatives to T1, 24 alternatives to itself, and 24 alternatives to T3. T1 is only sent 24 alternatives because, only 24 of its elements from the set $D(t_1)$ exist in the set $\mathbf{A}$. This means that most of the elements from $D(t_1)$ do not appear in d_1 because they were not consistent with at least one combination of elements from $D(t_2)$ and $D(t_3)$.

In our current implementation, each of the track managers orders alternatives from best to worst based on the number of new conflicts that will be introduced and the desirability of the particular resources present in the alternative. This has a min-conflict heuristic [14] like a vor and is an integral part of the hill-climbing nature of the algorithm. Currently, we are

	Slot 1	Slot 2	Slot 3
S1	T1	T1	T1
S2	T1	T1	T1
S3	T1/T2	T1/T2	T1/T2
S4	T1/T2	T1/T2	T1/T2
S5	T3/T2	T3/T2	T3/T2
S6	T3/T2	T3/T2	T3/T2
S7	T3	T3	T3
S8	T3	T3	T3

	Slot 1	Slot 2	Slot 3
S1		T1	T1
S2		T1	T1
S3	T2		T1
S4	T2	T1	T2
S5	T2		T2
S6	T3	T3	T2
S7	T3	T3	
S8	T3	T3	

Fig. 6. A solution derived by SPAM to the problem in Figure 5. The table on the left is before track manager T2 mediates with T1 and T3. Notice that a number of slots have two or more tasks scheduled. The table on the right is the result of stage 2 which is conflict-free.

looking at a number of alternative techniques for providing local preference information to the mediator including simply returning utility values for each solution and assigning solutions to a finite set of equivalence classes.

Once the mediator has the orderings from the track managers, it chooses a particular A_m to apply to the problem. This is done using a dynamic priority method based on the number of constraints each of the managers has external to the mediation, a form of meta-level information. The basic notion is similar to the priority order changes in AWC [11]; try to find the task which is most heavily constrained and elevate it in the orders. Our impression is that this helps stem the propagation because it leaves the most constrained tasks with the best choices. This allows those managers to maintain violation free solutions if they exist in the alternatives presented to them. Let's say that the priority ordering for the tasks is $(t_h, t_{h-1}, \dots, t_0)$. The mediator iteratively prunes the set $\mathbf{A}$ by creating a set $\mathbf{A}_{t_h} = \{A_m | A_m \in \mathbf{A} \wedge \forall A_i \in \mathbf{A} (priority_h(a_u \in A_m) \geq priority_h(a_v \in A_i))\}$. This newly created list is pruned in the same way for each of the managers until $|\mathbf{A}| = 1$.

In our example, T2 collects the ordering from T1, T2, and T3. T3 is given first choice. By its ordering it ranked alternative 0 the highest. This restricts the choice for T2 to alternatives 0, 1, 2, and 3. T2 ranked 0 highest from this set of alternatives, restricting T1's choice to its 0th, 1st, and 2nd alternatives. It turns out that T1 likes its 0th solution the best so the final solution is composed of T3's alternative 0, T2's alternative 0, and T1's alternative 0.

The last phase of the protocol is the solution implementation phase. Here, the mediator simply informs each of the track managers of its final choice. Each of the track managers then implements the final solution. At this point, each of the track managers is free to propagate and mediate if it chooses.

Figure 6 shows the starting and ending state of the resource schedules for the example problem. The columns represent the slots within the periodic schedules of the sensors. The rows represent the sensors. Notice that before T2 mediates, sensor S4 has two managers, T1 and T2, scheduled during every slot. After the mediation ends, all of the conflict has been removed and each manager obtains a 3×2 configuration with T1 alternating the use of S3 and S4 in slot 2 and 3.

C. Oscillation

Because the SPAM protocol operates in a local manner, a condition known as oscillation can occur. Say that, from our previous example, track manager T1 originated a mediation with track manager T2. In addition assume that T2 had previously resolved a conflict with manager T3, that terminated with neither T2 or T3 having unresolved conflict. Now when T1 mediates with T2, T1 in the end gets a locally unconflicted solution, but in order for that to occur, T2 conflicts with T3. It is possible that when T2 propagates, that the original conflict between T1 and T2 is reintroduced, leading to an oscillation.

There are actually a number of ways to prevent this from happening when the problem being worked on is static. For example, in both [11], [15], the authors use global prioritization, static in one, dynamic in the other, to prevent loops in the constraint network, and also maintain *nogood* lists to ensure a complete search.

We explored a method in which each track manager maintains a history of the sensor schedules that were being mediated over whenever a negotiation terminated. By doing this, managers were able determine if they have previously been in a state which caused them to propagate in the past. To stop the oscillation, the propagating manager lowered its objective level to force itself to explore different areas of the solution space. It should be noted that in certain cases oscillation was incorrectly detected using this technique, which resulted in having the track manager unnecessarily lower its objective level.

This technique is similar to that applied in [3], where a *nogood* is annotated with the state of the agent storing it. Unfortunately, none of these techniques work well when complex interrelationships exist and are dynamically changing. Because the problem changes continuously, previously explored parts of the search space need to be constantly revisited to ensure that an invalid solution has not recently become valid. Currently, we allow the agents to enter potential oscillation, maintaining no prior state other than objective levels from session to session and rely on the environment to break oscillations through the movement of the targets, asynchrony of the communications, timeouts, etc.

D. Generating Solutions

Generating the set **A** for the domain described earlier involves taking the information that was provided through communications with the conflicting track managers and assuming that the sensors that are in the set $\bigcup_{t_i \in N_m} D(t_i) - R(t_m)$ are freely available. In addition, because the track manager that is generating full solutions only knows about the sensors which are in direct conflict, it only creates and poses solutions for those sensors. That means that $\forall_a a \in d_i \rightarrow a \in R_m$. The formula below illustrates the basic mechanism that task manager's use to generate task alternatives. Here, k is the number of slots that are available in the planning horizon, D_s is the number of slots that are desired based on the objective level for the track manager, $|R(t_i)|$ is the number of sensors available to track the target (those that can see it), D_m is the number of sensors desired in the objective function, and

$C_i = |R(t_i) \cap R(t_m)|$ is the number of sensors under direct consideration because they are conflicting.

$$|D(t_i)| = \binom{k}{D_s} \left(\sum_{u=\max(0, D_m - |R(t_i)| + C_i)}^{\min(C_i, D_m)} \binom{C_i}{u} \right)^{D_s}$$

As can be seen by this formula, every combination of slots that meets the objective level is created, and for each of the slots, every combination of the conflicted sensors is generated such that the track manager has the capability of meeting its objective level using the sensors that are available to it. For instance, let's say that a track manager has four sensors S1, S2, S3, and S4 available to it. The track manager has a current objective level of 3×2 and sensors S2 and S3 are under conflict. The generation process would create the 3 combinations of slot possibilities and then for each possible slot, it would generate the combination of sensors such that three sensors could be obtained. The only possible sensor combinations in this scenario would be that the track manager gets either S2 or S3 (assuming that the manager will take the other two available sensors) or it gets S2 and S3 (assuming it only takes one of the other two). Therefore, a total of 27 possible solutions would be generated.

It is interesting to note that we use this same formula for alternative solutions in stage 1 of the protocol. This special case generation is actually done by simply setting $C_i = |R(t_i)|$. In this case, the formula above reduces to:

$$|D(t_i)| = \binom{k}{D_s} \left(\binom{C_i}{D_m} \right)^{D_s}$$

We can also generate partial solutions when there are a number of pre-existing constraints on the use of certain slot/sensor combinations. Simply by calculating the number of available sensors for each of the slots, and using this as a basis for determining which slots can still be used, we can reduce the number of possible solutions considerably.

Using the ability to impose constraints on the alternatives generated for a given track manager allows us to generate full solutions for the track managers in stage 2. By recursively going through the track managers using the results from earlier track managers as constraints for lower precedence ones, we can do a full search of the localized subproblem.

This can view this as a tree-based search where the top level of the tree is the set of alternatives for one track manager. Each of the nodes at this level may or may not have a number of children which are the alternatives available to the second track manager and so on. Only branches of the tree that have a depth equal to one less than the number of track managers are added to the set **A**. If there are no branches that meet this criteria, then the problem is considered over-constrained.

E. Handling Dynamics

By far one of the most interesting characteristics of the SPAM protocol is its ability to operate in environments that are highly dynamic. The SPAM protocol employs a number of techniques to deal with the effects of environmental dynamics both from a global perspective and a local perspective. One

of the most useful techniques that SPAM employs is the localization of mediation sessions. By limiting the context of the problem solving to only its immediate neighbors, agents can rapidly generate solutions to considerably smaller problems than would be faced by centralizing the entire problem, computing a solution, and later redistributing an answer. This technique alone would be no better than a one look-ahead greedy method however, if it weren't for the use of overlapping context in the problem solving and the ability for managers to propagate the conflicts. Globally, this leads to a great deal of parallelism in the search although it may lead to suboptimal solutions.

Within an individual session, SPAM handles dynamics by having both a multi-stage and multi-step mediation process. By breaking apart the protocol into 2 stages, SPAM can stop processing after stage 1 if it either predicts that it will or actually does run out of time during stage 2. In addition, within stage 1, an agent can concede some of its local utility in order to avoid engaging in time consuming mediation sessions and try to find solutions that only require localized changes to the resource schedules.

The mediation session itself is broken into several distinct phases. Mediators can place deadlines on each of these phases and at any time can drop another agent for the session or terminate it all together. Although not currently implemented, it is easy to see that a scheduler could be used to set these deadlines based on the expected duration of a resource need, the expected communications delay with individual agents, etc. In fact, mediators can even place deadlines on their internal searches. The algorithm used by the agents to generate solutions can be terminated at any time and will return the set of the solutions generated up to that point.

Lastly, the mediation itself is limited to the sensors that the mediator wishes to use. That means that track managers within the session are only given schedules for the sensors that are desired by the mediator and have considerable flexibility in the actual implementation of their local solutions. For example, let's say that a mediator T1 concludes a session with another manager, T2 which involves a single sensor S1. The solution T1 has generated has T2 only using S1 during the third slot of its schedule. T2 is free to implement any local solution, as long as it doesn't use S1 during its first or second slot. In fact, if T2's target moves outside of the view of S1 during the session, it can decide not to use S1 at all.

IV. TESTING

SPAM was implemented and successfully tested in the environment described in section II. However, due to the variability created by using actual hardware, properly testing SPAM was problematic. Thus, to more systematically and rigorously evaluate the SPAM protocol, we implemented a model of the domain in a simulation environment called Farm [16]. Farm is a component-based, distributed simulation environment written in Java where individual components have responsibility for particular encapsulated aspects of the simulation. For example, they may consist of agent clusters, visualization or analysis tools, environmental or scenario drivers, or provide some

other utility or autonomous functionality. These components or agent clusters may be distributed across multiple servers to exploit parallelism, avoid memory bottlenecks, or utilize local resources.

The actual model used for testing SPAM has both sensor and track manager agents. Each of the sensor agents represents a single sensor which was placed in a fixed location within the world. These sensor agents are very simple, and only maintain a local schedule, which is not actually performed in any tangible way. A fixed number of targets is introduced into the world, and one track manager per target is created to manage the resources needed to track that target. The targets can move through the environment with random trajectories that have a random, bounded speed. As the simulation progresses, the simulator continuously updates the position of the targets, and for each target calculates the set of sensors that are able to track it. The track managers can obtain their candidate sensor lists from the simulation environment and follow the SPAM protocol to allocate resources.

We ran two test series, one to test the effectiveness of our approach and the other to test its scalability.

A. Effectiveness

For the first test series, we wanted to determine the effectiveness and runtime characteristics of the protocol given different levels of resource contention. In this test series, we randomly placed 20 sensors within the environment and between 2 and 9 concurrent targets. Each of the targets maintained a static location throughout the run to allow the protocol to reach quiescence for the sake of measuring the convergence time.

For comparison purposes, we also implemented functions to compute solutions that:

- 1) Would be obtained by greedy agents.
- 2) Have the optimal utility.
- 3) Track the optimal number of targets.

Greedy agents each request all of the available (can see their target) resources to track their targets. These requests may potentially overriding each other in the sensors' schedules leading to poor performance in areas of high contention.

The optimal utility algorithm computes the maximal set of objective levels that is satisfiable in the environment. This is done by having the algorithm perform a complete search of the space of allowable objective levels, where each one is checked for satisfiability using a modified version of the complete search algorithm presented in section III-D. To make the search go faster, we prevent it from checking satisfiability on solutions that have utilities less than the best already obtained (i.e. Branch and Bound [5]), and do a simple arc-consistency check (using the pigeon hole principle) to prune obviously over-constrained problems.

The algorithm used for finding the optimal number of tracks determines the largest number of targets that can be tracked given the available resources. For clarity, a target is considered tracked if one coordinated triangulation occurs from three or more sensors during a given period. To obtain the optimal number of tracks, a search similar to the optimal utility is done. In this search, the only objective levels that need to be

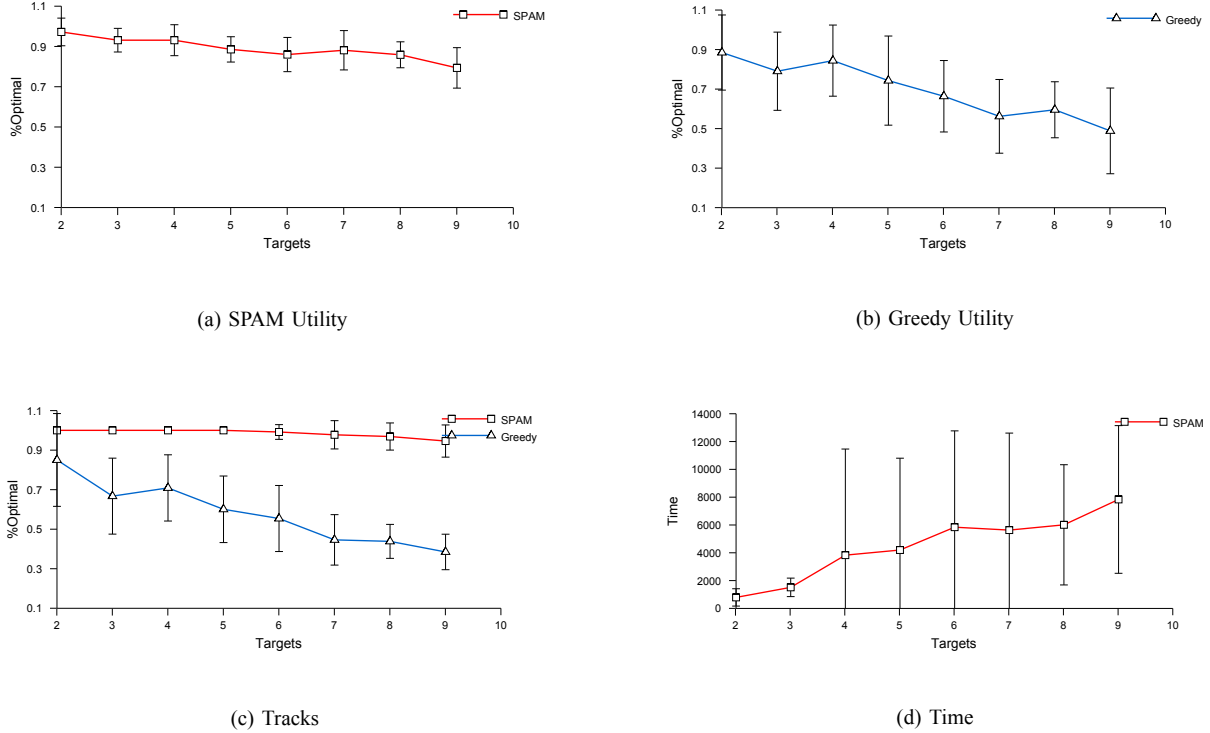


Fig. 7. Results of 20 sensor and varying target experiments comparing Greedy, SPAM and optimal allocations.

checked are either a minimal tracking (i.e. 3×1) or no tracking at all (0×0) making this search very fast.

We compared greedy and SPAM based on their achieved utility and the number of targets they tracked as a percentage of the optimal values over 20 test runs. A total of 180 tests were conducted for this series.

Figures 7(a), 7(b), 7(c), and 7(d) summarize the results of this series. As you can see from the graphs, SPAM does quite well when compared to both greedy and optimal. For the greedy method, the problem begins to become over-constrained at around 4 targets. SPAM provide reasonably good results (over 80% optimal for utility) for all of the configurations tested. Two things in particular are interesting about these results. First, for tracking targets, SPAM performs nearly 100% optimal. This is caused by the fact that SPAM is trying to optimize the balance of resources so that as many targets can be tracked as possible. Figure 7(d) shows another interesting result. As the problem gets harder SPAM has a linear increase in the time it takes to converge. This is very promising, considering the allocation problem is known to be NP-complete. Unfortunately, we have not yet implemented other solutions, which could be used to compare this running time. It should also be noted that the optimal solution took between a few seconds (for two targets) to several days (for nine targets) to compute.

Something we were not able to show in the graphs is that there are cases when the greedy algorithm obtained higher utility than SPAM, but was ignoring a large number of the targets in order to achieve it. We think that this may be caused by not penalizing enough for ignoring targets. It is not clear what that penalty should be, and initially seems to be strongly

domain dependent.

Lastly, there was at least one case where SPAM entered an oscillation. The utility obtained during the oscillation varied only slightly and the number of unresolved global conflicts fluctuated back and forth from 2 to 3. As mentioned earlier, this is a result of the localization of the search and in a dynamic environment probably would have been eliminated due to the targets' motion.

B. Scalability

For the second simulation series we wanted to investigate the scalability of the protocol given a fixed level of contention and fixed sensor field density. In these experiments a fixed ratio of 2.5 sensors per target were used while varying the number of agents, n , from 100 to 800. This ratio was chosen because it represents a fairly over-constrained problem since each track manager needs three sensors to track its target. The field density was fixed at 4 sensors per point which ensured overlap of the resources desired by the agents. The width and height of the environment were calculated as follows:

$$width = \sqrt{\frac{s\pi r^2}{4}}$$

where s is the number of sensors and r is the sensors viewable radius (20 feet for these sensors). So, for 700 agents we would have 500 sensors in an environment of $396ft \times 396ft$ with 200 targets which all move with a uniformly random speed between 0 and 2 feet per second. Each of the 20 simulation runs lasted three minutes and were on a different sensor field layout. So, the values reported here are an average

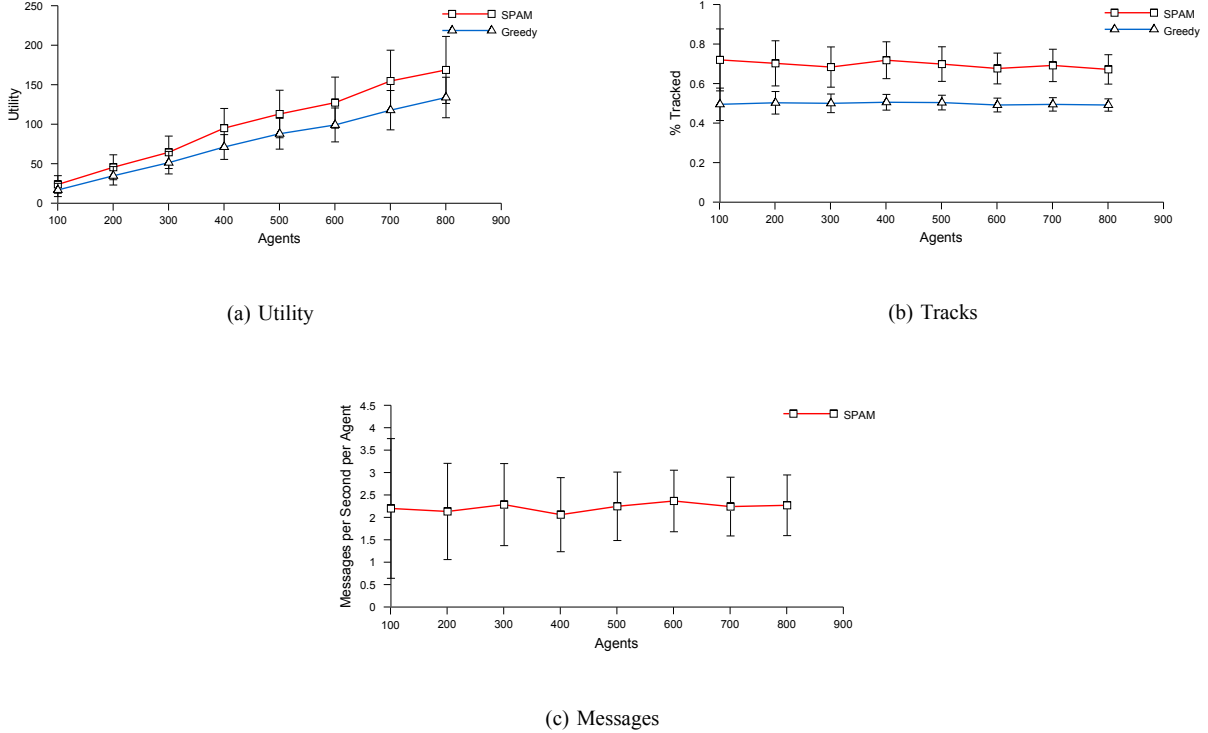


Fig. 8. Results of scale experiments conducted with a field density of 4 sensors per point and resource ratio of 2.5 sensors per target.

over 1 hour of runtime. For comparison purposes, we also ran the greedy algorithm once again.

One very important thing to note is that the greedy algorithm is part of the simulation and therefore is given access to the global state and is not penalized for computation time or communication delays. This means that it computes a solution to a static problem at each instant of time. The targets are stopped while it computes to ensure that the problem state does not change before it determines its answer. Overall, this means that the results returned by the greedy algorithm overestimates the utility that greedy *agents* would obtain.

SPAM, on the other hand, must explicitly communicate to gain information, is explicitly charged for computation time, works with incomplete and inaccurate information due to the targets continuous motion, as is not given credit for its solution until it is actually implemented in the sensor agents. Overall, the utility values calculated for SPAM are a very accurate representation of the actual values that would be obtained in real-time environments.

Figures 8(a), 8(b), 8(c) show the results for this series. As can be seen, as the number of agents increases linearly, so does the utility for SPAM and the greedy algorithm, which is not entirely surprising. Notice though, that even with the large advantage that the greedy algorithm is given, SPAM consistently outperforms it.

The two other interesting results from these experiments are the percentage of targets tracked and the number of messages being used by the agents. As the number of targets increase, the percentage of targets being effectively tracked remains almost constant and the number of messages being communicated by each agent per second remains constant as

well. This would suggest that the methods being used by SPAM to break apart the multi-linking of interdependencies between the track manager agents is actually very effective. Independent analysis of the SPAM protocol presented in [17] verifies these findings.

V. CONCLUSION

In this article, we described a distributed, cooperative mediation-based protocol which was built to solve resource allocation problems in a soft real-time environment. The protocol exploits the fact that agents within the environment are both cooperative and autonomous, and employs a number of techniques to operate in highly dynamic environments. Included in these techniques are mapping the resource allocation problem into an optimization problem, applying arc-consistency techniques to quickly prune the search space, breaking the protocol into multiple stages and phases to allow it to make time/quality trade-offs appropriate for current conditions, and minimizing the effects of long chains of interdependencies by localizing the scope of individual mediations.

As it turns out, the core ideas used in SPAM, particularly cooperative mediation, work quite well for solving static distributed problems, including distributed constraint satisfaction (DCSP) and distributed constraint optimization (DCOP). Our current work has focused on exploiting the power of this general technique for solving problems within these areas. As such, we have developed a complete algorithm, called *asynchronous partial overlay* (APO) [18], for DCSPs and an optimal algorithm, called *optimal asynchronous partial overlay* (OptAPO) [19], for DCOPs that are based on the

concept of cooperative mediation. These algorithms are, to the best of our knowledge, the fastest known methods for solving problems of these types.

Unfortunately, even though these algorithms are the fastest known, they still cannot operate in dynamic environments as they are unable to cope with rapidly changing conditions. This fact necessitates the existence of algorithms and techniques that perform both good enough and fast enough, like SPAM. The results of this work are encouraging, and although we consider the problems associated distributed resource allocation in dynamic environments to be an open research question, we feel that SPAM is a step in the right direction.

ACKNOWLEDGMENTS

The effort represented in this paper has been sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-99-2-0525 and the National Science Foundation under grant number IIS-9812755. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), Air Force Research Laboratory, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The authors would like to thank Tim Middlekoop, Jiaying Shen, and Regis Vincent for helping during the initial protocol development. We would also like to thank Bryan Horling for developing the Farm simulation environment used extensively for testing.

REFERENCES

- [1] S. E. Conry, K. Kuwabara, V. R. Lesser, and R. A. Meyer, "Multistage negotiation for distributed constraint satisfaction," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, no. 6, Nov. 1991.
- [2] R. G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Transactions on Computers*, vol. 29, no. 12, pp. 1104–1113, 1980.
- [3] P. J. Modi, H. Jung, M. Tambe, W.-M. Shen, and S. Kulkarni, "Dynamic distributed resource allocation: A distributed constraint satisfaction approach," in *Pre-proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001)*, J.-J. Meyer and M. Tambe, Eds., 2001, pp. 181–193.
- [4] M. Yokoo, *Distributed Constraint Satisfaction*, ser. Springer Series on Agent Technology. Springer, 1998.
- [5] E. C. Freuder and R. J. Wallace, "Partial constraint satisfaction," *Artificial Intelligence*, vol. 58, no. 1–3, pp. 21–70, 1992.
- [6] K. Hirayama and M. Yokoo, "Distributed partial constraint satisfaction problem," in *Principles and Practice of Constraint Programming (CP-97)*, ser. Lecture Notes in Computer Science, G. Smolka, Ed., vol. 1330. Springer-Verlag, 1997, pp. 222–236.
- [7] —, "An approach to overconstrained distributed constraint satisfaction problems: Distributed hierarchical constraint satisfaction," in *International Conference on Multi-Agent Systems (ICMAS)*, 2000.
- [8] M. Yokoo and K. Hirayama, "Distributed breakout algorithm for solving distributed constraint satisfaction problems," in *International Conference on Multi-Agent Systems (ICMAS)*, 1996.
- [9] B. Horling, R. Vincent, R. Mailler, J. Shen, R. Becker, K. Rawlins, and V. Lesser, "Distributed Sensor Network for Real Time Tracking," in *Proceedings of the 5th International Conference on Autonomous Agents*. Montreal: ACM Press, June 2001, pp. 417–424. [Online]. Available: <http://mas.cs.umass.edu/paper/199>

- [10] R. Vincent, B. Horling, V. Lesser, and T. Wagner, "Implementing Soft Real-Time Agent Control," in *Proceedings of the 5th International Conference on Autonomous Agents*. Montreal: ACM Press, June 2001, pp. 355–362. [Online]. Available: <http://mas.cs.umass.edu/paper/198>
- [11] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara, "The distributed constraint satisfaction problem: Formalization and algorithms," *Knowledge and Data Engineering*, vol. 10, no. 5, pp. 673–685, 1998.
- [12] M. Yokoo and E. H. Durfee, "Distributed constraint optimization as a formal model of partially adversarial cooperation," University of Michigan, Ann Arbor, MI 48109, Tech. Rep. CSE-TR-101-91, 1991.
- [13] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo, "An asynchronous complete method for distributed constraint optimization," in *Proceedings of the Second International Joint Conference on Autonomous Agent and Multiagent Systems (AAMAS-03)*, Melbourne, Australia, July 2003.
- [14] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird, "Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems," *Artificial Intelligence*, vol. 58, no. 1–3, pp. 161–205, 1992.
- [15] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara, "Distributed constraint satisfaction for formalizing distributed problem solving," in *International Conference on Distributed Computing Systems*, 1992, pp. 614–621.
- [16] B. Horling, R. Mailler, and V. Lesser, "Farm: A Scalable Environment for Multi-Agent Development and Evaluation," *Proceedings of the 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2003)*, pp. 171–177, May 2003. [Online]. Available: <http://mas.cs.umass.edu/paper/243>
- [17] G. Wang, W. Zhang, R. Mailler, and V. Lesser, *Analysis of Negotiation Protocols by Distributed Search*. Kluwer Academic Publishers, 2003, pp. 339–361. [Online]. Available: <http://mas.cs.umass.edu/paper/249>
- [18] R. Mailler and V. Lesser, "A Mediation Based Protocol for Distributed Constraint Satisfaction," *The Fourth International Workshop on Distributed Constraint Reasoning*, pp. 49–58, August 2003. [Online]. Available: <http://mas.cs.umass.edu/paper/250>
- [19] —, "Solving Distributed Constraint Optimization Problems Using Cooperative Mediation," *Submitted to AAMAS 2004*, 2004. [Online]. Available: <http://mas.cs.umass.edu/paper/355>



Roger Mailler is a PhD candidate at the University of Massachusetts Multi-Agent Systems (MAS) Lab. He received a BS with Honors in computer science from the State University of New York at Stony Brook in 1999. His main research interests are distributed problem solving, multi-agent systems organizational design, and machine learning.



Victor Lesser received his Ph.D. from Stanford University in 1972 and has been a professor of computer science at the University of Massachusetts at Amherst since 1977. He is a founding fellow of AAAI, and the founding president of the International Foundation for Multi-Agent Systems. His major research focus is on the control and organization of complex AI systems. He has been working in the field of Multi-Agent Systems for over 25 years. Prior to coming to the University of Massachusetts, he was a research scientist at Carnegie-Mellon University

where he was the systems architect for the Hearsay-II speech understanding system, which was the first blackboard system developed.

FINAL TECHNICAL REPORT
Agreement No. F30602-99-2-0525
“Scalable Real-Time Negotiation Toolkit”

APPENDIX B

Solving Distributed Constraint Optimization Problems Using Cooperative Mediation*

Roger Mailler[†] and Victor Lesser
University of Massachusetts
Department of Computer Science
Amherst, MA 01003
{mailler, lesser}@cs.umass.edu

Abstract

Distributed Constraint Optimization Problems (DCOP) have, for a long time, been considered an important research area for multi-agent systems because a vast number of real-world situations can be modeled by them. The goal of many of the researchers interested in DCOP has been to find ways to solve them efficiently using fully distributed algorithms which are often based on existing centralized techniques. In this paper, we present an optimal, distributed algorithm called optimal asynchronous partial overlay (OptAPO) for solving DCOPs that is based on a partial centralization technique called cooperative mediation. The key ideas used by this algorithm are that agents, when acting as a mediator, centralize relevant portions of the DCOP, that these centralized subproblems overlap, and that agents increase the size of their subproblems as the problem solving unfolds. We present empirical evidence that shows that OptAPO performs better than other known, optimal DCOP techniques.

1. Introduction

For a number of years now, researchers in distributed problem solving have struggled with the question of how to find an optimal assignment to a set of variables spread over a number of agents which have interdependencies. Out of this work, a number of formulations have arisen to describe these problems includ-

ing the distributed partial constraint satisfaction problem (DPCSP)[2], distributed valued constraint satisfaction problem [5], distributed hierarchical constraint satisfaction problems [3], and the distributed constraint optimization problem (DCOP) [10].

A number of powerful distributed algorithms have been developed that solve these problems either optimally, or close to optimally. For example, Distributed Depth-first Branch and Bound (DDBB) and Distributed Iterative Deepening (DID) [10], Anchor and Ascend [6], Synchronous Branch and Bound (SBB) and Iterative Distributed Breakout (IDB) [2], Distributed Greedy Repair [5], and the Asynchronous Distributed Optimization (Adopt) algorithm [8]. Each of these algorithms has, at their core, two common threads. First, their basic design originated directly from an associated centralized algorithm and second, they maintain total separation of the agents' knowledge during the problem solving process.

In this paper, we present a cooperative, mediation-based DCOP protocol, called Optimal Asynchronous Partial Overlay (OptAPO), that allows the agents to extend and overlap the context that they use for making their local decisions as the problem solving unfolds. When an agent acts as a mediator, it computes a solution to a portion of the overall problem and recommends value changes to the agents involved in the *mediation session*. This algorithm, like its DCSP variant APO [7], allows for rapid, distributed, asynchronous problem solving without the explosive communications overhead normally associated with current distributed algorithms. OptAPO represents a new methodology that simultaneously exploits the speed of centralized techniques and the ability of distributed problem solving to identify problem substructure. In the graph coloring domain, this algorithm performs better, both in terms of communication and computation, than the Adopt algorithm which is currently the fastest known complete DCOP technique.

In the rest of this paper, we present a formalization of the DCOP problem. We then present the OptAPO algo-

* Effort sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-99-2-0525 and F30602-03-C-0010. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), Air Force Research Laboratory, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

[†] The first author is a student

```

procedure initialize
   $d_i \leftarrow \text{random } d \in D_i$ ;
   $F_i^* \leftarrow 0$ ;
   $p_i \leftarrow \text{sizeof}(\text{neighbors})$ ;
   $m_i \leftarrow \text{active}$ ;
   $mediate \leftarrow \text{none}$ ;
  add  $x_i$  to the good_list;
  send (init,  $(x_i, p_i, d_i, m_i, D_i, C_i, \text{path}_{i,j})$ ) to neighbors;
  initList  $\leftarrow$  neighbors;
end initialize;

when received (init,  $(x_j, p_j, d_j, m_j, D_j, C_j, [\text{path}_{j,i}])$ ) do
  Add  $(x_j, p_j, d_j, m_j, D_j, \text{constraints}_j, \text{path})$  to agent_view;
  if  $x_j$  is a neighbor of some  $x_k \in \text{good\_list}$  do
    add  $x_j$  to the good_list;
    add all  $x_l \in \text{agent\_view}$  and  $x_l \notin \text{good\_list}$ 
    that can now be connected to the good_list;
     $p_i \leftarrow \text{sizeof}(\text{good\_list})$ ;
  end if;
  if  $x_j \notin \text{initList}$  do
    send (init,  $(x_i, p_i, d_i, m_i, D_i, C_i)$ ) to  $x_j$ ;
  else
    remove  $x_j$  from initList;
  check\_agent\_view;
end do;

```

Figure 1. The OptAPO procedures for initialization and linking.

rithm and discuss the issues of soundness and optimality as well as presenting an example of the execution on a simple problem. Next, we present the results of extensive testing that compares OptAPO with Adopt within the commonly used graph coloring domain. Lastly, we discuss some of our conclusions.

2. Distributed Constraint Optimization Problem

A Constraint Optimization Problem (COP) consists of the following:

- a set of n variables $V = \{x_1, \dots, x_n\}$.
- discrete, finite domains for each of the variables $D = \{D_1, \dots, D_n\}$.
- a set of cost functions $f = \{f_1, \dots, f_m\}$ where each $f_i(x_{i,1}, \dots, x_{i,j})$ is function $f_i : D_{i,1} \times \dots \times D_{i,j} \rightarrow N \cup \infty$.

The problem is to find an assignment $A^* = \{d_1, \dots, d_n | d_i \in D_i\}$ such that the global cost, called F , is minimized. Although the algorithm presented will work for any associative, commutative, monotonic aggregation function defined over a totally ordered set of values, with min and max elements, in this paper, F is defined as follows

$$F(A) = \sum_{i=1}^m f_i(A)$$

```

when received (value?,  $(x_j, p_j, d_j, m_j)$ ) do
  update agent_view with  $(x_j, p_j, d_j, m_j)$ ;
  check\_agent\_view;
end do;

procedure check\_agent\_view
  if initList  $\neq \emptyset$  or mediate  $\neq \text{none}$  do
    return;
   $m'_i \leftarrow \text{none}$ ;
  if  $F_i > F_i^*$  and  $\exists_j (f_j < f_j^* \wedge p_j < p_i)$  do
     $m'_i \leftarrow \text{active}$ ;
  else if  $F_i > F_i^*$ 
     $m'_i \leftarrow \text{passive}$ ;
  if  $m'_i \neq \text{none}$  and  $\neg \exists_j (p_j > p_i \wedge m_j == \text{active})$ 
    if  $\exists (d'_i \in D_i) (d'_i \cup \text{agent\_view} \text{ causes } F_i == F_i^*)$ 
      and changes are with lower priority neighbors
       $d_i \leftarrow d'_i$ ;
       $m_i \leftarrow \text{none}$ ;
      send (value?,  $(x_i, p_i, d_i, m_i)$ ) to all  $x_j \in \text{agent\_view}$ ;
    else
      do mediate( $m'_i$ );
    else if  $m_i \neq m'_i$ 
       $m_i \leftarrow m'_i$ ;
      send (value?,  $(x_i, p_i, d_i, m_i)$ ) to all  $x_j \in \text{agent\_view}$ ;
    else if  $m_i == \text{none}$ 
      for  $\forall x_j \in \text{agent\_view} \wedge x_j \notin \text{good\_list}$  do
        for  $\forall x_k$  on the path to  $x_j \wedge x_k \notin \text{agent\_view}$  do
          send (init,  $(x_i, p_i, d_i, m_i, D_i, \text{constraints}_i)$ ) to  $x_k$ ;
          add  $x_k$  to initList;
        end do;
      end do;
    end if;
  end check\_agent\_view;

```

Figure 2. The procedures for doing local resolution, updating the *agent_view* and the *good_list*.

In the distributed version of this problem, DCOP, each agent is assigned one or more variables along with the functions on their variables. Overall, COP and DCOP have both been shown to be NP-complete, making some form of search a necessity.

In this paper, we restrict ourselves to the case where each agent is assigned a single variable and is given knowledge of its functional relationship with other neighboring variables. Since each agent is assigned a single variable, we will refer to the agent by the name of the variable it manages. Also, we restrict ourselves to considering only binary functions which are of the form $f_i(x_{i1}, x_{i2})$. It should become fairly apparent that our approach can be extended to handle cases where one or both of these restrictions are removed.

Throughout the paper, we use the term *neighbors* to refer to agents that appear within a single cost function or *functional relation*. The graph formed by representing the agents as nodes and the functional relationships as edges is called the *relationship graph*.

```

procedure mediate( $m'_i$ )
  preferences  $\leftarrow \emptyset$ ;
  counter  $\leftarrow 0$ ;
  for each  $x_j \in \text{good\_list}$  do
    send (evaluate?, ( $x_i, p_i, m'_i$ )) to  $x_j$ ;
    counter ++;
  end do;
  mediate  $\leftarrow m'_i$ ;
   $m_i \leftarrow m'_i$ ;
end mediate;

when receive (wait!, ( $x_j, p_j$ )) do
  counter --;
  if counter == 0 do choose_solution;
end do;

when receive (evaluate!, ( $x_j, p_j, \text{labeled } D_j$ )) do
  record ( $x_j, \text{labeled } D_j$ ) in preferences;
  counter --;
  if counter == 0 do choose_solution;
end do;

```

Figure 3. The procedures for mediating in OptAPO.

3. Optimal APO

3.1. The Algorithm

Figures 1, 2, 3, 4, and 5 present the OptAPO algorithm. The algorithm works by constructing a *goodList* and maintaining a structure called the *agent.view*. The *agent.view* holds the names and some information about the values, domains and functional relationships of agents in the environment that are linked to the owner of the *agent.view*. The *goodList* holds the names of the agents that the owner has identified either a direct or indirect relationship to through one or more functional relations in the relationship graph.

In order to facilitate the problem solving process, each agent has a dynamic priority that is based on the size of their *goodList*. Priorities are used by the agents to decide the order in which one or more agents mediate when they have a need. Priority ordering is important for two reasons. First, priorities ensure that the agents with the most knowledge gets the make the decisions. This improves the efficiency of the algorithm by decreasing the effects of myopic decision making. Second, priorities improve the effectiveness of the mediation process. Because lower priority agents expect higher priority agents to mediate, they are less likely to be involved in a session when the mediation request is sent.

3.1.1. Initialization (Figure 1) On startup, the agents are provided with the value (they pick it randomly if one isn't assigned) and the functions on their variable. Initialization proceeds by having each of the agents send out an "init" message to each of its neigh-

```

procedure choose_solution
  select a solution  $s$  using a Branch and Bound search that:
    1. minimizes the cost for the agents in the goodList
    2. minimizes the cost for the agents not in the session;
   $F'_i \leftarrow F_i + \text{current cost for agents not in the session}$ ;
   $F'_s \leftarrow F_s + \text{cost for agents not in the session}$ ;
  if mediate == active and  $F'_s \leq F'_i$  do
     $d_i \leftarrow d'_i$ ;
  end if;
  for each  $x_j \in \text{agent.view}$  do
    if  $x_j \in \text{preferences}$  do
      if  $d'_j \in s$  violates an  $x_k$  and  $x_k \notin \text{agent.view}$  do
        send (init, ( $x_i, p_i, d_i, m_i, D_i, C_i, \text{path}_{i,k}$ )) to  $x_k$ ;
        add  $x_k$  to initList;
      end if;
      if mediate == active and  $F'_s \leq F'_i$  do
        send (accept!, ( $d'_j, x_i, p_i, d_i$ )) to  $x_j$ ;
        update agent.view for  $x_j$ 
      else if mediate == active and  $F'_s > F'_i$  do
        send (accept!, ( $d_j, x_i, p_i, d_i$ )) to  $x_j$ ;
      end if;
    else if mediate == active do
      send (value?, ( $x_i, p_i, d_i, m_i$ )) to  $x_j$ ;
    end if;
  end do;
  mediate  $\leftarrow$  none;
  check_agent.view;
end choose_solution;

```

Figure 4. The procedure for choosing a solution.

bors. This initialization message includes the variable's name (x_i), priority(p_i), current value(d_i), domain(D_i), and functional relationships(C_i). Also included in this message are the variable m_i , which indicates the agents current desire to mediate, and a list of agents that lie between i and j in the relationship graph. The purpose of both of these pieces of information will be described below. The array *initList* records the names of the agents that initialization messages have been sent to, the reason for which will become immediately apparent.

When an agent receives an initialization message (either during the initialization or through a later link request), it records the information in its *agent.view* and adds the variable to the *goodList* if it can. An agent is only added to the *goodList* if it is connected to another agent already in the list through a functional relationship. This ensures that the graph created by the agents in the *goodList* always remains connected. The *initList* is then checked to see if this message is a link request or a response to a link request. If an agent is in the *initList*, it means that this message is a response, and the agent is simply removed from the list. If the agent is not in the *initList* then it means this is a request, so a response "init" is generated and sent.

It is important to note that, at times, the agents in the *goodList* are a subset of the agents contained in the *agent.view*. This is done to maintain the integrity of the

```

when received (evaluate?,  $(x_j, p_j, m_j)$ ) do
  update agent_view with  $(x_j, p_j, m_j)$ ;
  if (mediate  $\neq$  none or  $\exists_k (p_k > p_j \wedge m_k == \text{active})$ )
    and  $m_j == \text{active}$  do
      send (wait!,  $(x_i, p_i)$ );
    else
      if mediate  $\neq$  active do
        mediate  $\leftarrow m_j$ ;
        label each  $d \in D_i$  with the names of the agents
          and associated costs incurred by setting  $d_i \leftarrow d$ ;
        send (evaluate!,  $(x_i, p_i, \text{labeled } D_i)$ );
      end if;
    end do;

when received (accept!,  $(d, x_j, p_j, d_j, m_j)$ ) do
   $d_i \leftarrow d$ ;
  mediate  $\leftarrow$  false;
  send (value?,  $(x_i, p_i, d_i, m_i)$ ) to all  $x_j$  in agent_view;
  update agent_view with  $(x_j, p_j, d_j, m_j)$ ;
  check_agent_view;
end do;

```

Figure 5. Procedures for receiving a session.

good_list and allow links to be bidirectional. To understand this point, consider the case when a single agent has repeatedly mediated and has extended its local subproblem down a long path in the relationship graph. As it does so, it links with agents that may have a very limited view and therefore are unaware of their indirect connection to the mediator. In order for the link to be bidirectional, the receiver of the link request has to store the name of the requester, but cannot add the them to their *good_list* until a path can be identified.

In order to ensure the optimality of the algorithm, each of the agents does not terminate until all of the agents in its *agent_view* appears in its *good_list*. During periods of inactivity, an agent tries to balance these two lists by linking to the anyone that they were told are on direct path between themselves and a disconnected agent. This is where the path information provided as part of the initialization message comes into play. This process ensures that the following property is enforced at the termination of the algorithm:

Property 1 Upon termination, $\forall_{i,j} j \in \text{good_list}_i \leftrightarrow i \in \text{good_list}_j$

3.1.2. Checking the agent view (Figure 2) After the agents receive all of the initialization messages, they execute the *check_agent_view* procedure. In this procedure, the current *agent_view* (assigned, known variable values) is used to calculate the current cost, F_i , of the relationship subgraph formed by the agents within the *good_list*. If, during this check, an agent finds that F_i is greater than the optimal value of the subsystem, called F_i^* , and has not been told by a higher priority agent that

they want to mediate, it assumes the role of the mediator. This check ensure that the following property is obtained at the termination of the algorithm:

Property 2 Upon termination, $\forall_i F_i^* == F_i$

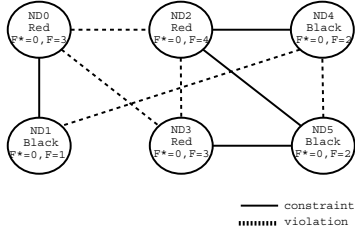
Agents within the system are able to tell when a higher priority agent wants to mediate because of the m_i flag mentioned in the previously section. Whenever an agent checks its *agent_view* it recomputes the value of this flag which indicates its desire to mediate in the future if given the opportunity. This information is shared with each of the agents in its *agent_view* whenever the value changes. The overall effect of the m_i flag is similar to the two-phase commit commonly seen in database systems and ensures that the protocol remains live-lock and deadlock free.

As the mediator, an agent first attempts to rectify the suboptimal condition by changing its own variable. This simple technique prevents sessions from occurring unnecessarily, when it works, which stabilizes the system and saves messages and time. If the mediator finds a value that makes $F_i == F_i^*$ and it finds that the functional relationships being improved by the change are shared with only lower priority agents, it makes the change and sends out a “value?” message to the agents in its *agent_view*. A “value?” message is similar to an “init” message, in that it contains information about the priority, current value, etc. of a variable. If cannot find such a value, it starts either an *active* or *passive* mediation session.

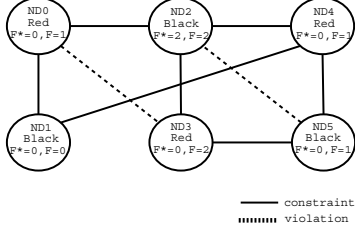
Agents decide between an active or passive mediation based on whether or not one of the suboptimal functional relations in their *good_list* has an agent in it that is lower priority. If one of the agents is lower priority, the mediation will be active, otherwise, it will be passive.

There are two main difference between active and passive mediation. First, during an active mediation, the receiving agent sets its *mediate* flag. This flag prevents it from starting or engaging in another active mediation until it is cleared. This causes a region of stability in the agent system which allows to mediation session to have the full effect but, reduces parallelism because it prevents other agents from mediating. The second difference is really based on the intent. The intent of passive mediation is to verify and understand the results that higher priority agents have obtained without interfering in their actions or changing their current solution. This both increases the parallelism of the problem solving and allows agents to gain more context (extend their views) without causing system instability.

3.1.3. Mediation (Figures 3, 4, and 5) The most complex and certainly most interesting part of the protocol is the mediation session. As was previously mentioned in this section, an agent decides to mediate if it finds that $F_i > F_i^*$ and is not expecting a session request from a higher priority agent. The mediation session



(a) Startup

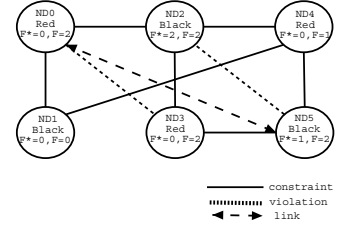


(b) After ND2 Mediates

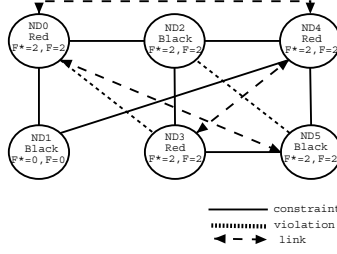
Figure 6. The startup and first step of OptAPO solving an example 2-coloring problem.

starts when the mediator sending out “evaluate?” messages to each of the agents in its *good_list*. The purpose of this message is two-fold. First, it informs the receiving agent that a mediation is about to begin and, as mentioned earlier, if the mediation is active, tries to obtain a lock from that agent. The second purpose of the message is to obtain information from the agent about the effects of making them change their local value. This is a key point. By obtaining this information, the mediator gains information about variables and relationships outside of its local view without having to directly and immediately link with those agents. When an agent receives a mediation request, it will respond with either a “wait!” or “evaluate!” message. Agents respond with a “wait” message whenever the request is for an active session and the agent is either currently involved in another active session or is expecting a request for an active session from an agent that is higher priority than the requester. This allows for a great deal of parallelism because all passive requests are responded to and active requests are only turned away when absolutely needed. Whenever it can, the agent evaluates each of its domain elements and labels them with the names of the agents that would have a shared functional relation with cost $f_i > f_i^*$ along with that cost if it were asked to take that value.¹ This information is returned in an “evaluate!” message. It should be

¹ In the graph coloring domain, the labeled domain can never exceed $O(|d_i| + n)$.



(a) After ND5 Mediates



(b) Termination

Figure 7. The second step and final step of OptAPO solving an example 2-coloring problem.

noted that, although in this implementation, the agents need not return all of the names if for security reasons they wish not to. This effects the optimality of the algorithm because it prevents agents from gaining enough context, but does provides some degree of autonomy and privacy to the agents.

When the mediator has received a response from all of the agents that it has sent a request to, it chooses a solution. Agents that sent a “wait!” message are dropped from the mediation, but the mediator attempts to fix whatever problems it can based on the information it receives from the agents in the session. The mediator conducts a Branch and Bound search [1] which, as a primary criteria, minimizes the cost of the subproblem in the *good_list* and as a secondary criteria minimizes the costs for agents outside of the session. The results of minimizing the primary criteria, being the optimal value for the subproblem in the *good_list* is saved as F_i^* .

The agents employ two special tactics during this search. First, the values are ordered so that the first branch of the search is the current solution. This usually causes the bound to become very close to F_i^* after it is explored taking advantage of previous work that has been done on the problem. This has the overall effect of improving the search speed as was reported in [9]. The second tactic terminates the search early whenever the bound is equal to the current F_i^* and the cost for agents outside of the mediation is 0. This method works because the current F_i^* is always an upper bound on the

actual F^* and F_i^* only increases on successive search due to the monotonic nature of the number of variables in the search and the aggregation function. The effort conducted during the search can be reduced considerably using this method.

After finishing the local search, the mediator links (sends “init” messages) with any agent that is not in its *agent_view* and has been forced to have an increased cost as a result of the solution it found. This step expands the agent’s *good_list* so that the next time it mediates, it does not repeat the same mistake. It then computes the overall effect of making the changes based on its new larger perspective. If the overall effect is negative, the mediator ignores the newly computed solution and reverts to the current solution, effectively preventing itself from making a locally optimal decision that has obviously bad global consequences. The mediator concludes the session by sending “accept!” messages (if the session was active) to the agents in the session, who, in turn, adopt the proposed answer and “value?” messages to any agent that responded with a “wait” message.

3.2. An Example

Consider the 2-coloring problem in figure 6(a), which was chosen to illustrate the algorithms features without being overly complicated. In this problem there are 6 variables, each assigned to one agent, and 9 functional relations between the variables. Each of the functional relations is a ‘not equals’ and has a cost of 1 for being violated. The goal of the agents is to minimize the global cost. In other words, this is a MaxSAT problem instance.

On startup, the problem is in the state in figure 6(a). The current cost of the system is 5, because 5 of the ‘not equals’ relationships are violated (indicated by the dotted lines). Each of the agent has an internal optimal sub-system value, $F_i^* = 0$, since none of them have mediated and therefore computed the actual value.

Following the protocol, the agents send out “init” messages to each of their neighbors. So, for example in this problem, ND0 send out “init” messages to ND1, ND2, and ND3. As the “init” messages are received, the agents add each of their neighbors to their *good_list* because they have a direct path through a shared relation.

Once all of the “init” messages are received, the agents check their *agent_view*. Several of the agent are able to detect the non-optimal state of the problem by computing the value of F_i . For example, ND2 computes an $F_i = 4$ and sets its m_i flag to **active**. In fact, being the highest priority agent in the system, it has a priority of 5, it elects to take over as the mediator and begins an active session with ND0, ND3, ND4, and ND5.

As the mediator, ND2 first checks to see if it can correct the suboptimality by making a local change, which it cannot. It sends “evaluate?” messages to ND0, ND3,

ND4, and ND5. Each of these agents, upon receiving the message, checks to see if they are expecting a mediation from a higher priority agent, which they are not, and then sets their *mediate* flag to **active**. They label each of their domain elements and reply with the following information using “evaluate!” messages:

- ND0 - Black conflicts with ND1; Red conflicts with ND2 and ND3
- ND3 - Black conflicts with ND5; Red conflicts with ND0 and ND2
- ND4 - Black conflicts with ND1 and ND5; Red conflicts with ND2
- ND5 - Black conflicts with ND4; Red conflicts with ND2 and ND3

ND2 conducts a Branch and Bound search and finds that $F_i^* = 2$ for its *good_list* as well as finding a solution which has 0 conflicts external to the mediation. It tells ND2 and ND4 to change their value to Black, leaving the system in the state in figure 6(b).

All of the agents check their *agent_view* again. This time, ND5 decides to mediate and being unable to correct the difference in its F_i and F_i^* , it sends “evaluate?” messages to ND2, ND3, and ND4. It receives the following information from those agents in the returned “evaluate!” messages:

- ND2 - Black conflicts with ND5; Red conflicts with ND0, ND3, and ND4
- ND3 - Black conflicts with ND2 and ND5; Red conflicts with ND0
- ND4 - Black conflicts with ND1, ND2, and ND5; Red causes no conflicts

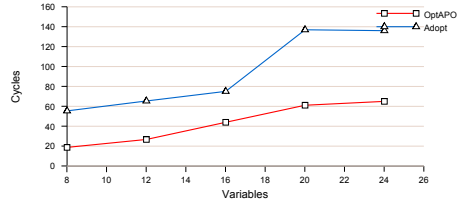
ND5 also conducts an internal search, and finds a solution with an $F_i^* = 1$, but has one conflict outside of the session unresolved. The solution it found, in fact, is identical to the current assignments. It links with ND0, having been unable to fix the conflict between ND0 and ND3 which leaves the problem in the state in figure 7(a).

After several more steps, the algorithm terminates in the state in figure 7(b).

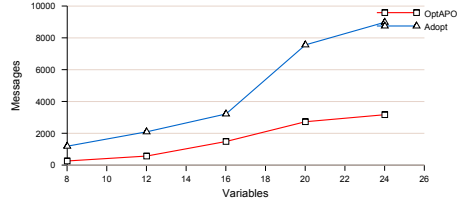
3.3. Soundness and Optimality

The soundness and optimality proofs for OptAPO are quite lengthy and due to space limitations cannot be presented here. The key ideas of the proof are as follows:

1. Let’s assume that the algorithm terminated in a sub-optimal state. That means that at least one f_i has a value that is higher than it should be.
2. Since we know by Property 2 that at the very least each of the agents within f_i must have $F_i^* = F_i$, each of them must have a subproblem in their *good_list* that justifies the value at f_i otherwise they would have continued processing, contradicting the assumption.



(a) Cycles



(b) Messages

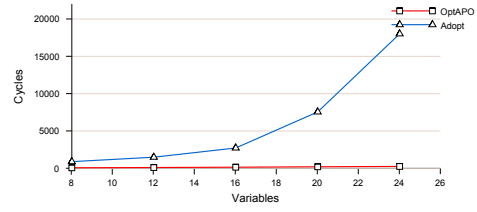
Figure 8. Comparison of OptAPO and Adopt for various graph sizes at $m = 2.0n$.

3. There cannot be another f_j in their *good_lists* that should have a reduced value instead of f_i .
 - (a) Assume that some other f_j should have had its value increased instead of f_i .
 - (b) In order to make a difference, from a global optimality perspective, f_j must be part of multiple overlapping subproblems that each justify an increased value and will be improved by increasing f_j .
 - (c) Since, by Property 1, we know that the agents within f_j have the agents in f_i within their *good_lists* and, in fact, have the agents for any other increased f_k that would be fixed by increasing f_j , they must have a $F^* < F$ because they couldn't have found a subproblem that justifies the entire increased cost.
 - (d) This means they will continue processing, contradicting the termination assumption.

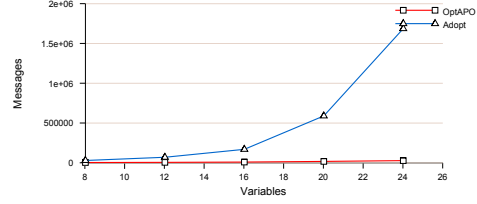
In the worse case, each of the agents centralizes the entire problem in order to terminate, making the overall complexity of this algorithm exponential. The space complexity of the algorithm is polynomial in the number of variables.

4. Evaluation

To evaluate the OptAPO algorithm, we ran two series of tests. In the first series of tests, we compared the number of *cycles* and messages used by OptAPO against the



(a) Cycles

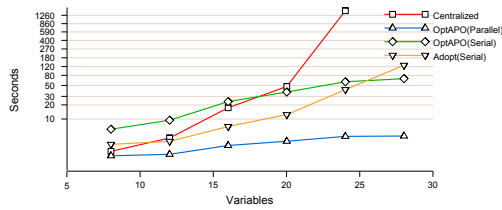


(b) Messages

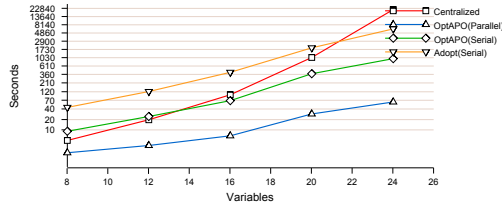
Figure 9. OptAPO and Adopt on various problem sizes at $m = 3.0n$.

currently fastest known DCOP algorithm, Adopt [8] in the commonly used graph coloring domain. We created random graph coloring instances with $m = 2.0n$ (under-constrained) and $m = 3.0n$ (over-constrained). We generated 100 instances at $n=8,12,16,20$, and 24 for a total of 1000 individual graphs. During a cycle, all of the incoming messages are delivered and processed, and outgoing messages are queued for delivery at the beginning of the next cycle. For fairness, we used the same graphs to test both algorithms. The results of this series can be seen in figures 8(a) 8(b), 9(a), and 9(b).

In the second set of tests, we compared the actual run times of the two algorithms against each other. For the Adopt implementation, we downloaded the simulator and algorithm available on-line. We implemented the OptAPO algorithm in the Farm simulation environment described in [4]. This environment was designed to simulate the run time behavior of a distributed algorithm by only allowing a finite amount of CPU time to the agents within a given cycle. We are therefore able to report the serial time, the total time it took the simulation to finish, and the parallel time, the execution time if the n agent were on n dedicated machines. Both algorithms were run on a single, dedicated 2.8 GHz Pentium 4 with 512MB of RAM on a 25 instance subset of the graphs from the first series of tests. To show that OptAPO's performance was not simple an artifact of the Branch and Bound search internal to the agents, we ran that algorithm on the graph instances as well. The results from this test can be seen in figures 10(a) and 10(b)



(a) $m = 2.0n$



(b) $m = 3.0n$

Figure 10. Comparison of runtime of OptAPO, Adopt, and centralized Branch and Bound.

which are logarithm scale.

As you can see from the figure, OptAPO outperforms Adopt for both cycles and messages on both under and over constrained problems. However, when comparing the run time results, we can see that Adopt runs slightly faster on small under-constrained problems. This may, in fact, be caused by the overhead associated with the Farm simulator. As the number of agents increases, the overhead tends to be less dominant and the OptAPO serial time is comparable to Adopt's (we extended the test series to 28 variables to show that OptAPO outperforms Adopt at higher values of n for under-constrained problem). What you should also notice is that although OptAPO uses the Branch and Bound search internally, it actually runs faster than when this search is conducted in a centralized fashion. This is by no means meant to show that a distributed algorithm operates faster than a centralized one, but shows that the runtime characteristics of the algorithm are not simply a by-product of the centralized search. In fact, the improvements in search time over the centralized search are most likely caused by the combination of the value ordering heuristic and early search termination method talked about in section 3.1.3.

5. Conclusions

In this paper, we presented a new method for solving DCOPs called *Optimal Asynchronous Partial Overlay* (OptAPO). The key features of this technique are that agents mediate over conflicts, the context they use to make local decisions overlaps with that of other agents,

and as the problem solving unfolds, the agents gain more context information along the critical paths within the problem to improve their decisions. We have shown that the OptAPO algorithm is both sound and optimal and that it performs better than the Adopt algorithm on most MaxSAT graph coloring problems.

Acknowledgments

The authors would like to thank Pragnesh Jay Modi for allowing us to use his implementation of the Adopt algorithm and Bryan Horling for developing the Farm simulation environment.

References

- [1] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1–3):21–70, 1992.
- [2] K. Hirayama and M. Yokoo. Distributed partial constraint satisfaction problem. In G. Smolka, editor, *Principles and Practice of Constraint Programming (CP-97)*, volume 1330 of *Lecture Notes in Computer Science*, pages 222–236. Springer-Verlag, 1997.
- [3] K. Hirayama and M. Yokoo. An approach to overconstrained distributed constraint satisfaction problems: Distributed hierarchical constraint satisfaction. In *International Conference on Multi-Agent Systems (ICMAS)*, 2000.
- [4] B. Horling, R. Mailler, and V. Lesser. Farm: A Scalable Environment for Multi-Agent Development and Evaluation. *Proceedings of the 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2003)*, pages 171–177, May 2003.
- [5] M. Lemaitre and G. Verfaillie. An incomplete method for solving distributed valued constraint satisfaction problems. In *Proceedings of the AAAI Workshop on Constraints and Agents*, 1997.
- [6] J. Liu and K. P. Sycara. Exploiting problem structure for distributed constraint optimization. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, 1995.
- [7] R. Mailler and V. Lesser. A Mediation Based Protocol for Distributed Constraint Satisfaction. *The Fourth International Workshop on Distributed Constraint Reasoning*, pages 49–58, August 2003.
- [8] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *Proceedings of the Second International Joint Conference on Autonomous Agent and Multiagent Systems (AAMAS-03)*, Melbourne, Australia, July 2003.
- [9] R. Wallace. Enhancements of branch and bound methods for the maximal constraint satisfaction problem. In M. Jampel, E. Freuder, and M. Maher, editors, *OCS'95: Workshop on Over-Constrained Systems at CP'95*, Cassis, Marseilles, 18 1995.
- [10] M. Yokoo and E. H. Durfee. Distributed constraint optimization as a formal model of partially adversarial cooperation. Technical Report CSE-TR-101-91, University of Michigan, Ann Arbor, MI 48109, 1991.

FINAL TECHNICAL REPORT
Agreement No. F30602-99-2-0525
“Scalable Real-Time Negotiation Toolkit”

APPENDIX C

Using Cooperative Mediation to Solve Distributed Constraint Satisfaction Problems*

Roger Mailler[†] and Victor Lesser
University of Massachusetts
Department of Computer Science
Amherst, MA 01003
{mailler, lesser}@cs.umass.edu

Abstract

Distributed Constraint Satisfaction (DCSP) has long been considered an important area of research for multi-agent systems. This is partly due to the fact that many real-world problems can be represented as constraint satisfaction and partly because real-world problems often present themselves in a distributed form. In this paper, we present a complete, distributed algorithm called asynchronous partial overlay (APO) for solving DCSPs that is based on a cooperative mediation process. The primary ideas behind this algorithm are that agents, when acting as a mediator, centralize small, relevant portions of the DCSP, that these centralized subproblems overlap, and that agents increase the size of their subproblems along critical paths within the DCSP as the problem solving unfolds. We present empirical evidence that shows that APO performs better than other known, complete DCSP techniques.

1. Introduction

Distributed constraint satisfaction has become a classic formulation that is used to describe a number of distributed problems including distributed re-

source allocation [2, 10], distributed scheduling [11], and distributed interpretation [7]. It's no wonder that a vast amount of effort and research has gone into creating algorithms, such as distributed breakout (DBO) [13], asynchronous backtracking (ABT) [12], and asynchronous weak-commitment (AWC) [14], for solving these problems.

Unfortunately, a common drawback to each of these techniques is that they prevent the agents from making informed local decisions about the effects of changing their local variable value without actually doing it. For example, in AWC, agents have to try a value and wait for another agent to tell them that it will not work through a *nogood* message. Because of this, agents never learn why another agent or set of agents is unable to accept the value, they only learn that their value in combination with other values doesn't work.

In this paper, we present a *cooperative mediation* based DCSP protocol, called Asynchronous Partial Overlay (APO), that allows the agents to extend and overlap the context that they use for making their local decisions. When an agent acts as a mediator, it computes a solution to a portion of the overall problem and recommends value changes to the agents involved in the *mediation session*. This technique allows for rapid, distributed, asynchronous problem solving without the explosive communications overhead normally associated with current distributed algorithms. APO represents a new methodology that lies somewhere between centralized and distributed problem solving which exploit the best characteristics of both. In the graph coloring domain, this algorithm performs better, both in terms of communication and computation, than the AWC algorithm. This is particularly true for problems that lie near or to the right of the phase transition.

In the rest of this paper, we present a formalization of the DCSP problem. We then present the APO algorithm and discuss the issues of soundness and complete-

* The effort represented in this paper has been sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-99-2-0525. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), Air Force Research Laboratory, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

[†] The first author is a student.

```

procedure initialize
   $d_i \leftarrow \text{random } d \in D_i$ ;
   $p_i \leftarrow \text{sizeof}(\text{neighbors})$ ;
   $m_i \leftarrow \text{true}$ ;
   $\text{mediate} \leftarrow \text{false}$ ;
  add  $x_i$  to the good_list;
  send (init,  $(x_i, p_i, d_i, m_i, D_i, C_i)$ ) to neighbors;
   $\text{initList} \leftarrow \text{neighbors}$ ;
end initialize;

when received (init,  $(x_j, p_j, d_j, m_j, D_j, C_j)$ ) do
  Add  $(x_j, p_j, d_j, m_j, D_j, C_j)$  to agent_view;
  if  $x_j$  is a neighbor of some  $x_k \in \text{good\_list}$  do
    add  $x_j$  to the good_list;
    add all  $x_l \in \text{agent\_view} \wedge x_l \notin \text{good\_list}$ 
      that can now be connected to the good_list;
     $p_i \leftarrow \text{sizeof}(\text{good\_list})$ ;
  end if;
  if  $x_j \notin \text{initList}$  do
    send (init,  $(x_i, p_i, d_i, m_i, D_i, C_i)$ ) to  $x_j$ ;
  else
    remove  $x_j$  from initList;
  check\_agent\_view;
end do;

```

Figure 1. The APO procedures for initialization and linking.

ness as well as presenting an example of the execution on a simple problem. Next, we present the results of extensive testing that compares APO with AWC within the commonly used graph coloring domain. Lastly, we discuss some of our conclusions and future directions.

2. Distributed Constraint Satisfaction

A Constraint Satisfaction Problem (CSP) consists of the following:

- a set of n variables $V = \{x_1, \dots, x_n\}$.
- discrete, finite domains for each of the variables $D = \{D_1, \dots, D_n\}$.
- a set of constraints $R = \{R_1, \dots, R_m\}$ where each $R_i(x_{i1}, \dots, x_{ij})$ is a predicate on the Cartesian product $D_{k1} \times \dots \times D_{kj}$ that returns true iff the value assignments of the variables satisfies the constraint.

The problem is to find an assignment $A = \{d_1, \dots, d_n | d_i \in D_i\}$ such that each of the constraints in R is satisfied. Overall, CSP has been shown to be NP-complete, making some form of search a necessity.

In the distributed case, using variable-based decomposition, each agent is assigned one or more variables along with the constraints on their variables. The goal of each agent, from a local perspective, is to ensure that each of the constraints on its variables are satisfied. It should be fairly clear that for each of the agents,

```

when received (ok?,  $(x_j, p_j, d_j, m_j)$ ) do
  update agent_view with  $(x_j, p_j, d_j, m_j)$ ;
  check\_agent\_view;
end do;

procedure check\_agent\_view
  if  $\text{initList} \neq \emptyset$  or  $\text{mediate} \neq \text{false}$  do
    return;
   $m'_i \leftarrow \text{hasConflict}(x_i)$ ;
  if  $m'_i$  and  $\neg \exists_j (p_j > p_i \wedge m_j = \text{true})$ 
    if  $\exists (d'_i \in D_i) (d_i \cup \text{agent\_view} \text{ does not conflict})$ 
      and conflicts are with lower priority neighbors
       $d_i \leftarrow d'_i$ ;
      send (ok?,  $(x_i, p_i, d_i, m_i)$ ) to all  $x_j \in \text{agent\_view}$ ;
    else
      do mediate;
    else if  $m_i \neq m'_i$ 
       $m_i \leftarrow m'_i$ ;
      send (ok?,  $(x_i, p_i, d_i, m_i)$ ) to all  $x_j \in \text{agent\_view}$ ;
    end if;
  end check\_agent\_view;

```

Figure 2. The procedures for doing local resolution, updating the *agent_view* and the *good_list*.

achieving this goal is not independent of the goals of the other agents in the system. In fact, in all but the simplest cases, the goals of the agents are strongly inter-related. For example, in order for one agent to satisfy its local constraints, another agent, potentially not directly related through a constraint, may have to change the value of its variable.

In this paper, for the sake of clarity, we restrict ourselves to the case where each agent is assigned a single variable and is given knowledge of the constraints on that variable. Since each agent is assigned a single variable, we will refer to the agent by the name of the variable it manages. Also, we restrict ourselves to considering only binary constraints which are of the form $R_i(x_{i1}, x_{i2})$. It is fairly easy to extend our approach to handle the case where one or both of these restrictions are removed.

3. Asynchronous Partial Overlay

3.1. Key Ideas

The key ideas behind the creation of the APO algorithm are

- Using mediation, agents can solve subproblems of the DCSP through local search.
- These local subproblems can and should overlap to allow for more rapid convergence of the problem solving.
- Agents should, over time, increase the size of the subproblem they work on along critical paths

```

procedure mediate
  preferences  $\leftarrow \emptyset$ ;
  counter  $\leftarrow 0$ ;
  for each  $x_j \in \text{good\_list}$  do
    send (evaluate?,  $(x_i, p_i)$ ) to  $x_j$ ;
    counter++;
  end do;
  mediate  $\leftarrow \text{true}$ ;
end mediate;

when receive (wait!,  $(x_j, p_j)$ ) do
  counter--;
  if counter == 0 do choose_solution;
end do;

when receive (evaluate!,  $(x_j, \text{labeled } D_j)$ ) do
  record  $(x_j, \text{labeled } D_j)$  in preferences;
  counter--;
  if counter == 0 do choose_solution;
end do;

```

Figure 3. The procedures for mediating an APO session.

within the CSP. This activity increases the overlap with other agents and ensures the completeness of the search.

3.2. The Algorithm

Figures 1, 2, 3, 4, and 5 present the basic APO algorithm. The algorithm works by constructing a *good_list* and maintaining a structure called the *agent_view*. The *agent_view* holds the names and some information about the values, domains, and constraints of agents that are linked to the owner of the *agent_view*. The *good_list* holds the names of the agents that the owner has identified a path to through the constraint graph.

In order to facilitate the problem solving process, each agent has a dynamic priority that is based on the size of their *good_list*. Priorities are used by the agents to decide who mediates a session when a conflicts arises. Priority ordering is important for two reasons. First, priorities ensure that the agents with the most knowledge gets to make the decisions. This improves the efficiency of the algorithm by decreasing the effects of myopic decision making. Second, priorities improve the effectiveness of the mediation process. Because lower priority agents expect higher priority agents to mediate, they are less likely to be involved in a session when the mediation request is sent.

3.2.1. Initialization (Figure 1) On startup, the agents are provided with the value (they pick it randomly if one isn't assigned) and the constraints on their variable. Initialization proceeds by having each of the agents send out an "init" message to each of its neighbors. This initialization message includes the variable's

```

procedure choose_solution
  select a solution  $s$  using a Branch and Bound search that:
  1. satisfies the constraints between agents in the good_list
  2. minimizes the violations for agents outside of the session
  if  $\neg \exists s$  that satisfies the constraints do
    broadcast no solution;
  for each  $x_j \in \text{agent\_view}$  do
    if  $x_j \in \text{preferences}$  do
      if  $d'_j \in s$  violates an  $x_k$  and  $x_k \notin \text{agent\_view}$  do
        send (init,  $(x_i, p_i, d_i, m_i, D_i, C_i)$ ) to  $x_k$ ;
        add  $x_k$  to initList;
      end if;
      send (accept!,  $(d'_j, x_i, p_i, d_i, m_i)$ ) to  $x_j$ ;
      update agent_view for  $x_j$ 
    else
      send (ok?,  $(x_i, p_i, d_i, m_i)$ ) to  $x_j$ ;
    end if;
  end do;
  mediate  $\leftarrow \text{false}$ ;
  check_agent_view;
end choose_solution;

```

Figure 4. The procedure for choosing a solution during an APO mediation.

name (x_i), priority (p_i), current value (d_i), the agent's desire to mediate (m_i), domain (D_i), and constraints (C_i). The array *initList* records the names of the agents that initialization messages have been sent to, the reason for which will become immediately apparent.

When an agent receives an initialization message (either during the initialization or through a later link request), it records the information in its *agent_view* and adds the variable to the *good_list* if it can. An agent is only added to the *good_list* if it is connected to another agent already in the list through a constraint. This ensures that the graph created by the agents in the *good_list* always remains connected. The *initList* is then checked to see if this message is a link request or a response to a link request. If an agent is in the *initList*, it means that this message is a response, and the agent is simply removed from the list. If the agent is not in the *initList* then it means this is a request, so a response "init" is generated and sent.

It is important to note that the agents contained in the *good_list* are a subset of the agents contained in the *agent_view*. This is done to maintain the integrity of the *good_list* and allow links to be bidirectional. To understand this point, consider the case when a single agent has repeatedly mediated and has extended its local subproblem down a long path in the constraint graph. As it does so, it links with agents that may have a very limited view and therefore are unaware of their indirect connection to the mediator. In order for the link to be bidirectional, the receiver of the link request has to store the name of the requester, but cannot add them to their *good_list* until a path can be

```

when received (evaluate?,  $(x_j, p_j)$ ) do
   $m_j \leftarrow \text{true}$ ;
  if  $\text{mediate} == \text{true}$  or  $\exists_k (p_k > p_j \wedge m_k == \text{true})$  do
    send (wait!,  $(x_i, p_i)$ );
  else
     $\text{mediate} \leftarrow \text{true}$ ;
    label each  $d \in D_i$  with the names of the agents
      that would be violated by setting  $d_i \leftarrow d$ ;
    send (evaluate!,  $(x_i, p_i, \text{labeled } D_i)$ );
  end if;
end do;

when received (accept!,  $(d, x_j, p_j, d_j, m_j)$ ) do
   $d_i \leftarrow d$ ;
   $\text{mediate} \leftarrow \text{false}$ ;
  send (ok?,  $(x_i, p_i, d_i, m_i)$ ) to all  $x_j$  in  $\text{agent\_view}$ ;
  update  $\text{agent\_view}$  with  $(x_j, p_j, d_j, m_j)$ ;
  check_agent_view;
end do;

```

Figure 5. Procedures for receiving an APO session.

identified.

3.2.2. Checking the agent view (Figure 2) After the agents receive all of the initialization messages they are expecting, they execute the check_agent_view procedure. In this procedure, the current *agent_view* (assigned, known variable values) is checked to identify conflicts between an agent and its neighbors. If, during this check, an agent finds a conflict and has not been told by a higher priority agent that they want to mediate, it assumes the role of the mediator.

An agent can tell when a higher priority agent wants to mediate because of the m_i flag mentioned in the previous section. Whenever an agent checks its *agent_view* it recomputes the value of this flag based on whether or not it has existing conflicts with its neighbors. When this flag is set to **true** it indicates that the agent wishes to mediate if it is given the opportunity. This mechanism acts like a two-phase commit protocol, commonly seen in database systems, and ensures that the protocol is live-lock and deadlock free.

As the mediator, an agent first attempts to rectify the conflict(s) by changing its own variable. This simple, but effective technique prevents sessions from occurring unnecessarily, when it works, which stabilizes the system and saves message and time. If the mediator finds a value that removes the conflict, it makes the change and sends out an “ok?” message to the agents in its *agent_view*. If cannot find a non-conflicting value, it starts a mediation session. An “ok?” message is similar to an “init” message, in that it contains information about the priority, current value, etc. of a variable.

3.2.3. Mediation (Figures 3, 4, and 5) The most complex and certainly most interesting part of the protocol is the mediation. As was previously mentioned in this section, an agent decides to mediate if it is in conflict with one of its neighbors and is not expecting a session request from a higher priority agent. The mediation starts with the mediator sending out “evaluate?” messages to each of the agents in its *good_list*. The purpose of this message is two-fold. First, it informs the receiving agent that a mediation is about to begin and tries to obtain a lock from that agent. This lock, referred to as *mediate* in the figures, prevents the agent from engaging in two sessions simultaneously or from doing a local value change during the course of a session. The second purpose of the message is to obtain information from the agent about the effects of making them change their local value. This is a key point. By obtaining this information, the mediator gains information about variables and constraints outside of its local view without having to directly and immediately link with those agents.

When an agent receives a mediation request, it will respond with either a “wait!” or “evaluate!” message. The “wait” message indicates to the requester that the agent is currently involved in a session or is expecting a request from an agent of higher priority than the requester. If the agent is available, it labels each of its domain elements with the names of the agents that it would be in conflict with if it were asked to take that value.¹ This information is returned in the “evaluate!” message. It should be noted that, although in this implementation, the agents need not return all of the names if for security reasons they wish not to. This effects the completeness of the algorithm, because the completeness relies on one or more of the agents eventually centralizing the entire problem, but does provide some degree of autonomy and privacy to the agents.

When the mediator has received either a “wait!” or “evaluate!” message from all of the agents that it has sent a request to, it chooses a solution. Agents that sent a “wait!” message are dropped from the mediation, but the mediator attempts to fix whatever problems it can based on the information it receives from the agents in the session. Currently, solutions are generated using a Branch and Bound search [5] where all of the constraints must be satisfied and the number of outside conflicts is minimized (like the min-conflict heuristic [9]). If no satisfying assignments are found, the agent announces that the problem is unsatisfiable and the algorithm terminates. Once the solution is cho-

¹ In the graph coloring domain, the labeled domain can never exceed $O(|d_i| + n)$.

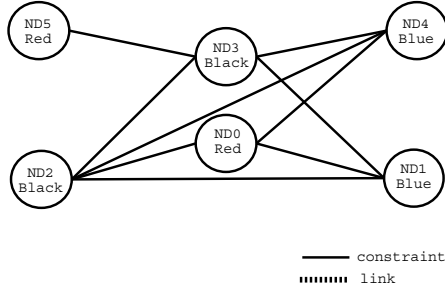


Figure 6. An example of a 3-coloring problem with 6 nodes and 9 edges.

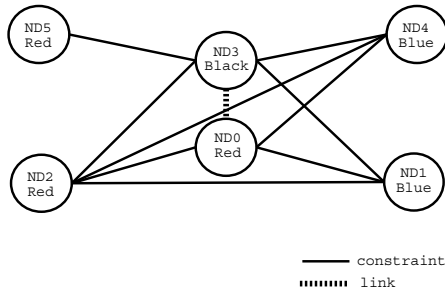


Figure 7. The state of the sample problem after ND3 leads the first mediation.

sen, “accept!” messages are sent to the agents in the session, who, in turn, adopt the proposed answer.

3.3. An Example

Consider the 3-coloring problem presented in figure 6. In this problem, there are 6 agents, each with a variable and 9 edges or constraints between them. The constraint between ND2 and ND3 is in violation because both agents have the color Black assigned to their variables. Following the algorithm, upon startup each agent adds itself to its *good_list* and sends an “init” message to its neighbors. Upon receiving these messages, the agents add each of their neighbors to their *good_list* because they are able to identify a shared constraint with themselves.

Once the startup has been completed, each of the agents checks its *agent_view*. ND3 and ND2 find that they are in conflict. ND2, being the lower priority of the two, waits for ND3 to start a mediation. ND3, knowing it is higher priority, first checks to see if it can resolve the conflict by changing its value, which in this case, it cannot. ND3 starts a session that involves ND1, ND2, ND4, and ND5. It sends each of them an “eval-

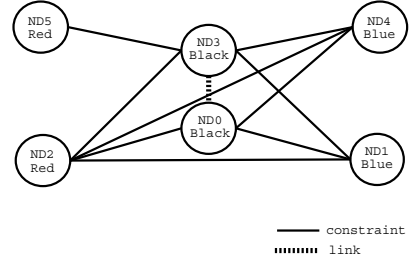


Figure 8. The final solution after ND2 leads the second mediation.

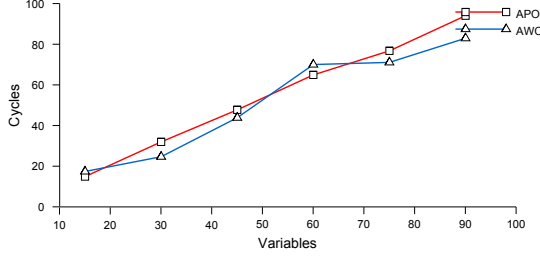
uate?” message.

When each of the agents in the mediation receives the “evaluate?” message, they label their domain elements with the names of the variables that they would have be in conflict with as a result of adopting that value. They each send ND3 an “evaluate!” message with this information. The following are the labeled domains for each of the agents

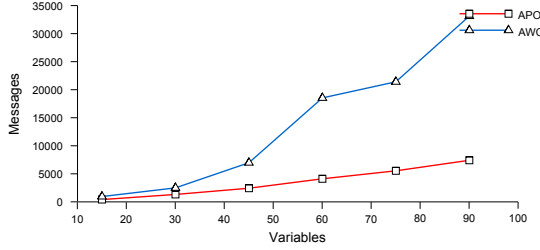
- ND1 - Black conflicts with ND2 and ND3; Red conflicts with ND0; Blue causes no conflicts
- ND2 - Black conflicts with ND3; Red conflicts with ND0; Blue conflict with ND1 and ND4
- ND4 - Black conflicts with ND2 and ND3; Red conflicts with ND0; Blue causes no conflicts
- ND5 - Black conflicts with ND3; Red causes no conflicts; Blue causes no conflicts

Once all of the responses are received, the mediator, ND3, conducts a branch and bound search that attempts to find a satisfying assignment to the problem that minimizes the amount of conflict that would be created outside of the mediation. If it cannot find at least one satisfying assignment, it broadcasts that a solution cannot be found. In the example, it chooses to have ND2 change its color to Red, introducing a new conflict between ND2 and ND0. As the last part of the mediation, ND3 links with ND0, leaving the problem in the state shown in figure 7. Note that when this happens, ND3 adds ND0 to its *good_list* and vice versa.

ND1, ND2, ND4 and ND5 inform the agents in their *agent_view* of their new values, then check for conflicts. This time, ND2 and ND0 notice that their values are in conflict. ND2, being higher priority, becomes the mediator and mediates a session with ND0, ND1, ND3, and ND4. Following the protocol, ND2 sends out the “evaluate?” messages and the receiving agents label and respond. The following are the labeled domains that are returned:



(a) Cycles



(b) Messages

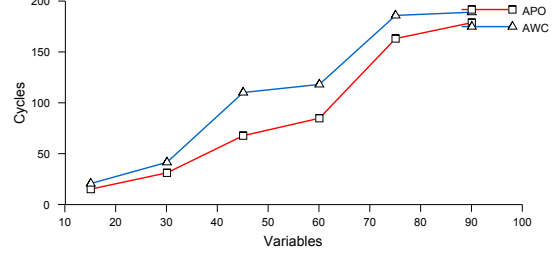
Figure 9. Comparison of the number of cycles and messages needed to solve satisfiable, low-density 3-coloring problems of various size by AWC and APO.

- ND0 - Black causes no conflicts; Red conflicts with ND2; Blue conflicts with ND1 and ND4
- ND1 - Black conflicts with ND3; Red conflicts with ND2; Blue causes no conflicts
- ND3 - Black causes no conflicts; Red conflicts with ND2 and ND5; Blue conflicts with ND1 and ND4
- ND4 - Black conflicts with ND3; Red conflicts with ND2; Blue causes no conflicts

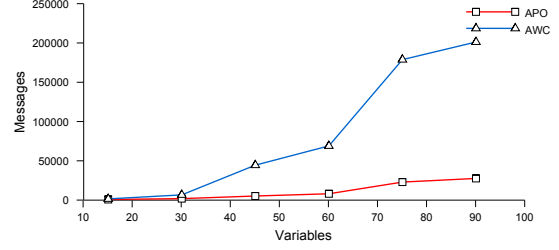
ND2, after receiving these messages, conducts its search and finds two solutions that solve its subproblem. It chooses to change the color of ND0 to Black and the problem is solved (see figure 8).

3.4. Soundness and Completeness

The proofs of APO's soundness and completeness are quite lengthy, so for simplicity, we refer the reader to [8] for their full details. Below are the main ideas that are used in them. In the actual proofs, we assume that all communications are reliable, meaning that if a message is sent from x_i to x_j that x_j will receive the



(a) Cycles



(b) Messages

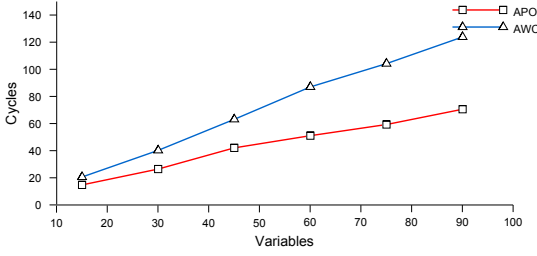
Figure 10. Comparison of the number of cycles and messages needed to solve satisfiable, medium-density 3-coloring problems of various size by AWC and APO.

message in a finite amount of time. We also assume that if x_i sends a message m_1 to x_j before sending another message m_2 to x_j , that m_1 will be received before m_2 .

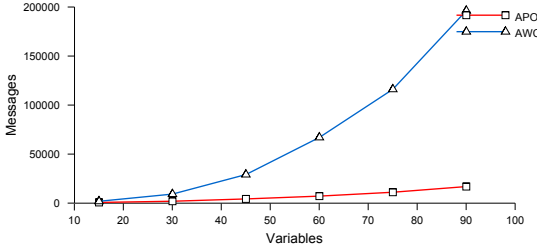
- If at anytime an agent identifies a constraint subgraph that is not satisfiable, it announces that the problem cannot be solved. Half of the soundness.
- If a constraint violation exists, someone will try to fix it. The protocol is dead-lock free. The other half of the soundness.
- Eventually, in the worst case, one or more of the agents will centralize the entire problem and will derive a solution, or report that no solution exists. This ensures completeness.

4. Evaluation

To test the APO algorithm, we implemented the AWC and APO algorithms and conducted experiments in the distributed 3-coloring domain. The particular AWC algorithm we implemented can be found in [14]



(a) Cycles

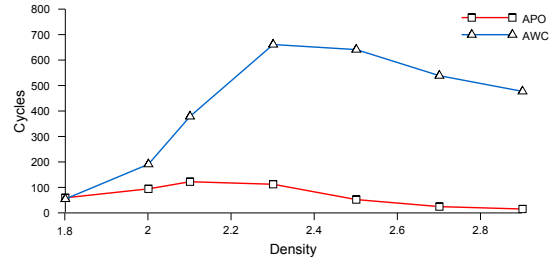


(b) Messages

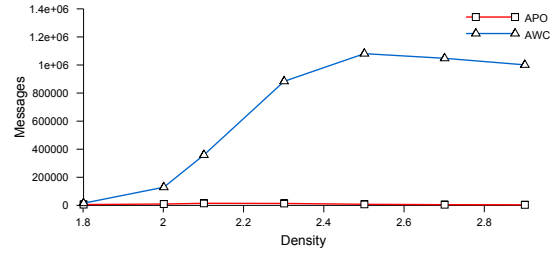
Figure 11. Comparison of the number of cycles and messages needed to solve satisfiable, high-density 3-coloring problems of various size by AWC and APO.

which includes the resolvent *nogood* learning mechanism described in [6]. The distributed 3-coloring problem is a 3-coloring problem with n variables and m binary constraints distributed amongst the agents. We conducted two sets of experiments. In the first set of experiments, we created solvable graph instances with $m = 2.0n$ (*low-density*), $m = 2.3n$ (*medium-density*), and $m = 2.7n$ (*high-density*) according to the method presented in [9]. Although, it was reported in [1] that $m = 2.7n$ was within the critical region for 3-colorability, they were using reduced graphs for their analysis. In [3], the critical region was identified as being approximately $m = 2.3n$ and therefore was included in our tests. We generated 10 random graph for $n = 15, 30, 45, 60, 75, 90$ and for each instance generated 10 initial variable assignments. For each combination of n and m , we ran 100 trials making a total of 1800 trials. The results from this experiment can be seen in figures 9(a) through 11(b).

In the second set of experiments, we created completely random 60 node graphs of various density from $1.8n$ to $2.9n$. This was done to test the completeness



(a) Cycles



(b) Messages

Figure 12. Number of cycles and messages needed to solve completely random 60 variable problems of various density using AWC and APO.

of the algorithms and to verify the correctness of their implementations. For each density value, we generated 200 random graphs each with a single set of initial values. In total, 1400 graphs were generated and tested. We stopped the execution of the algorithms at 1000 cycles for the sake of time. The results of these experiments are shown in figures 12(a) and 12(b).

To evaluate the relative strengths and weakness of each of the approaches, we measured the number of *cycles* and the number of messages used during the course of solving each of the problems. During a cycle, incoming messages are delivered, the agent is allowed to process the information, and any messages that were created during the processing are added to the outgoing queue to be delivered at the beginning of the next cycle. The actual execution time given to one agent during a cycle varies according to the amount of work needed to process all of the incoming messages. The random seeds used to create each graph instance and variable instantiation were saved and used by each of the algorithms for fairness.

When looking at the results for satisfiable graph instances, you can see that on low-density satisfiable

graphs AWC and APO perform almost identically in terms of cycles to completion. As the density of the graph increases however, the difference become more apparent. APO begins to scale more efficiently than AWC. This can be attributed to the ability of APO to rapidly identify strong interdependencies between variables and to derive solutions to them using a centralized search of the partial subproblem. We should mention that the results of the testing on AWC obtained from these experiments agree with those reported in [6] verifying the correctness of our implementation.

On random instances, APO significantly outperforms AWC on all but the simplest of problems (see figure 12(a)). The most direct cause of this is the performance of AWC's poor performance on unsatisfiable problem instances as previously reported in [4].

The most profound difference in the algorithms can be seen in the number of messages used by them to solve problems. In all cases, APO outperform AWC by a significant amount. There are two primary causes for this. First, the mediation process creates regions of stability in the agent environment. So, unlike AWC, APO is able to avoid thrashing behavior that is caused by the asynchrony of operating in a distributed environment. Second, because APO uses partial centralization to solve problems, it avoids having to use a large number of messages to discover implied constraints through trial and error.

5. Conclusions

In this paper, we presented a new method for solving DCSPs called the *Asynchronous Partial Overlay* algorithm. The key features of this technique are that agents mediate over conflicts, the context they use to make local decisions overlaps with that of other agents, and as the problem solving unfolds, the agents gain more context information along the critical paths of the constraint graph to improve their decisions. We have shown that the APO algorithm is both sound and complete and that it performs as well as, if not better than the AWC algorithm on graph coloring problems of various size and difficulty.

References

- [1] P. Cheeseman, B. Kanefsky, and W. Taylor. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 331–337, 1991.
- [2] S. E. Conry, K. Kuwabara, V. R. Lesser, and R. A. Meyer. Multistage negotiation for distributed constraint satisfaction. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6), Nov. 1991.
- [3] J. Culberson and I. Gent. Frozen development in graph coloring. *Theoretical Computer Science*, 265(1–2):227–264, 2001.
- [4] C. Fernandez, R. Bejar, B. Krishnamachari, C. Gomes, and B. Selman. *Distributed Sensor Networks: A Multiagent Perspective*, chapter Communication and Computation in Distributed CSP Algorithms, pages 299–317. Kluwer Academic Publishers, 2003.
- [5] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1–3):21–70, 1992.
- [6] K. Hirayama and M. Yokoo. The effect of nogood learning in distributed constraint satisfaction. In *The 20th International Conference on Distributed Computing Systems (ICDCS)*, pages 169–177, 2000.
- [7] V. R. Lesser and D. D. Corkill. The distributed vehicle monitoring testbed. *AI Magazine*, 4(3):63–109, Fall 1983.
- [8] R. Mailler and V. Lesser. A Mediation Based Protocol for Distributed Constraint Satisfaction. *The Fourth International Workshop on Distributed Constraint Reasoning*, pages 49–58, August 2003.
- [9] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1–3):161–205, 1992.
- [10] P. J. Modi, H. Jung, M. Tambe, W.-M. Shen, and S. Kulkarni. Dynamic distributed resource allocation: A distributed constraint satisfaction approach. In J.-J. Meyer and M. Tambe, editors, *Pre-proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001)*, pages 181–193, 2001.
- [11] K. Sycara, S. Roth, N. Sadeh, and M. Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1446–1461, November/December 1991.
- [12] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *International Conference on Distributed Computing Systems*, pages 614–621, 1992.
- [13] M. Yokoo and K. Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *International Conference on Multi-Agent Systems (ICMAS)*, 1996.
- [14] M. Yokoo and K. Hirayama. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, 3(2):198–212, 2000.

FINAL TECHNICAL REPORT
Agreement No. F30602-99-2-0525
“Scalable Real-Time Negotiation Toolkit”

APPENDIX D

The Control, Coordination, and Organizational Design of a Distributed Sensor Network*

Roger Mailler[†], Bryan Horling[‡], Victor Lesser[§], and Régis Vincent

December 12, 2003

Abstract

In this paper we describe our solution to a real-time distributed tracking problem. The system works not by finding an optimal solution, but through a satisficing search for an allocation that is “good enough” to meet the specified resource requirements, which can then be revised over time if needed. The agents in the environment are first organized by partitioning them into sectors, reducing the level of potential interaction between agents. Within each sector, agents dynamically adopt various roles depending on the current state of the environment. These roles form goals which then are instantiated as task structures for use by the SRTA control architecture. These elements exist to support resource allocation, which is directly effected through the use of the SPAM mediation protocol. The agent problem solving component first discovers and generates commitments for sensors to use for gathering data, then determines if conflicts

*Effort sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory Air Force Materiel Command, USAF, under agreements number F30602-99-2-0525 and DOD DABT63-99-1-0004. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. This material is also based upon work supported by the National Science Foundation under Grant No. IIS-9812755. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), Air Force Research Laboratory or the U.S. Government.

[†]R. Mailler is with the University of Massachusetts.

[‡]B. Horling is with the University of Massachusetts.

[§]V. Lesser is with the University of Massachusetts.

[¶]R. Vincent is with SRI International.

exist with that allocation, finally using arbitration and relaxation strategies to resolve such conflicts. We have empirically tested and evaluated these techniques in both the Radsim simulation environment and using a hardware-based system.

1 Introduction

Recently, distributed sensors networks have begun to receive a great deal of attention from researchers in a variety of disciplines such as electrical engineering, computer networking [1, 2], and multi-agent systems [3]. Classically, sensors and their associated networks have received the most attention from engineers and researchers studying the properties and fabrication of the sensors themselves. These sensors have most commonly been controlled and monitored using high-powered, expensive centralized facilities which rely on dedicated underlying networks to transmit control information and receive raw or partial processed data from the sensors. In addition, most existing sensor networks are special purpose and entirely independent of one another when they could be use to fulfill multiple purposes. For example, weather monitoring and air traffic control radar systems have classically been isolated from one another even though they essentially use the same type of sensors.

With the increasing availability of a range of network technologies, researchers have begun to investigate the idea of using low-cost, off-the-shelf computers to monitor and control large numbers of shared sensors in a distributed way. There are a number of benefits to migrating control and monitoring away from a small number of centralized facilities. Among these are fault tolerance, localization of computation and communication within the networks for scalability, and rapid adaptability of shared sensing resources to changing environmental conditions.

For example, envision a system in which a number of chemical sensors are deployed throughout a region. Each of these sensors may have some limited ability to sense chemical signatures from the environment. From day to day, these sensors receive tasks of varying importance from scientists performing environmental studies such as monitoring the effects of greenhouse gases, acid rain, heavy metals, etc. At the same time, in the background, these same sensors monitor for more insidious chemical signatures, such as chemical weapons.

Now imagine that one of these sensors detects a poisonous gas. Immediately,

it would autonomously switch its focus and processing power to specifically monitor the concentrations of that specific gas. It could begin to coordinate with neighboring chemical and weather tracking sensors to verify the detection, hone in on a specific chemical by using sensors with other capabilities, as well as monitor and predict the chemical's spread. The computer controlling the sensor could enlist the assistance of other computers which could then disseminate the information to the proper authorities, begin the evacuation of effected areas, etc.

In this article, we describe a distributed sensor network for discovering and tracking targets that has a number of characteristics in common with the example chemical detection sensor network described above. In particular, each of the sensors within our system has multiple, competing goals of various importance, high-level goals can only be accomplished by coordinating the actions of several sensors, and the control needs to be both distributed and autonomous in order to operate in real-time.

Our approach to managing this network consists of three major architectural and behavioral contributions. First, each sensor platform is controlled by a single agent which exists as part of a larger, heterogeneous organizational structure. This structure helps bound and focus the computation necessary to solve the distributed tracking problem by associating individual agents with one or more roles within that organization, taking responsibility for different parts of the overall goal. Second, individual agents are sophisticated, autonomous problem solvers. Each incorporates a domain independent soft real-time control architecture (SRTA) which is used to model and control the activities of the agent, and a domain specific problem solving component which reasons about and reacts to the surrounding environment. Finally, a negotiation mechanism and protocol (SPAM) is employed to allocate sensor resources and resolve conflicts. Agents in the organization responsible for tracking use this protocol to ensure sufficient quantity and qualities of data are obtained for all targets to be tracked, if at all possible.

Each of these technologies plays an important role solving the sensor-to-target allocation problem. The SPAM protocol [4] lies at the heart of this process, enabling agents to request sensors, and then dynamically detect and resolve conflicts when they arise by using distributed negotiation. Where SPAM resolves conflict between agents, the SRTA architecture [5] resolves conflicts that

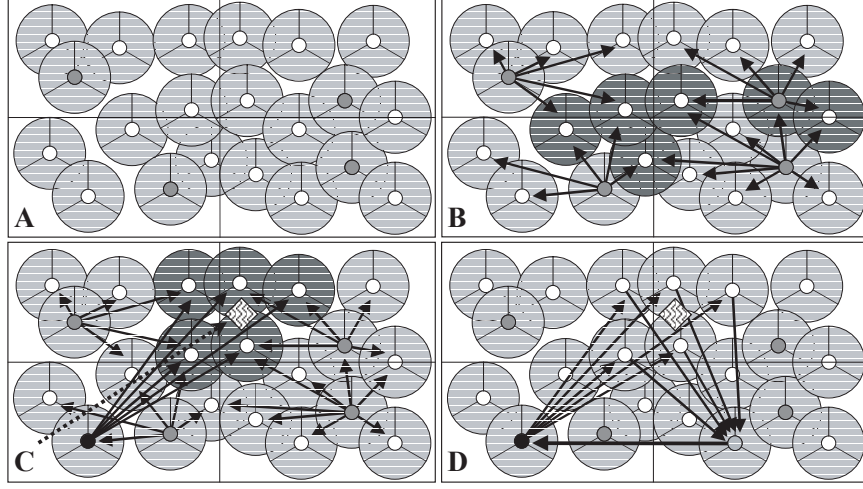


Figure 1: High-level architecture. A: sectorization of the environment, B: distribution of the scan schedule, C: communicating over tracking measurements, and D: fusion of tracking data.

exist within an agent. By modeling tasks and commitments and using several scheduling techniques to manage local activities, SRTA allows allocations to accrue quality even when there are unresolved conflicts. The organizational design both acts to limit the distance over which information must be propagated and concentrates the information needed for the various organizational roles to operate efficiently. This has the important effect of reducing communication effort and facilitate the allocation process. All of these technologies must operate in real-time to be effective in this distributed sensor network environment.

In the remainder of this article, we will give an overview of the sensor network used in this work, describe its organization, the agent control mechanisms, and the resource allocation protocol. Section 6 discusses results and experiences from implementing the system both in simulation and on actual hardware. The article will conclude with some final thoughts and discussion of future work.

2 Overview

The sensor platforms used in our network each incorporate three Doppler radar-based sensor heads (Zemany and Gaughan in [3]) that have a range of approx-



Figure 2: The sensor platforms have three Doppler radar sensor heads and are controlled by a BASIC stamp micro-controller.

imately 20 feet and a 120 degree viewable arc (see figure 2). The platform is controlled by a BASIC stamp micro-controller which may only collect data from one of these sensor heads at any time. Each measurement takes about 400 milliseconds and returns an average amplitude (relative distance) or frequency shift (relative velocity) over the collection period. Sensor platforms can communicate with one another over an 8-channel, radio-frequency (RF) system. But, due to the close proximity of the receiver and transmitter within the platforms, they are unable to simultaneously transmit and receive messages regardless of channel assignment. In addition, because this is RF, multiple agents cannot transmit on a single channel at the same time without causing interference. Each of the platforms is connected to a computer which is capable of locally hosting one or more processes.

Several key items should be extracted from this architectural style description of the sensor platforms. The first is that no single platform can effectively track a target by itself. Tracking, in this sense, should be interpreted as being able to estimate a target's position with some degree of certainty and collecting these estimates over time to model and predict the targets future position. Because the sensors are only capable of returning relative distance, single platforms cannot localize a target to region smaller than a 120 degree arc at some distance based on a noisy measurement. So, in order to improve its estimate it must enlist the aid of other sensor platforms to triangulate the target's position. If these additional measurement occur in a temporally coordinated fashion the quality of the triangulation improves, particularly when the target is moving.

The need for explicit coordination prevents us from using methods like those presented in [6] which only required one sensor to track a target and migrated the task through the field as the target moved.

The second key feature of these platforms is that their communication is unreliable, has limited bandwidth, and limited range. The effect of this is that information and control must be distributed in order to be reactive to environmental changes. For instance, let's say that we were to employ a single agent to manage all of the coordination needed to track the several targets. In order for this agent to assign sensors to each of the targets it would need to know, at very least, the targets' locations. Since the targets are moving, these locations would need to be updated frequently enough for the agent to continue allocating the correct resources to track. In addition, every time a change is needed to the allocation, this agent would need to communicate with each of the affected sensor platforms. It should be clear that in a limited communications environment, this is likely to be infeasible for anything but a small number of targets. Distributing the work across a number of agents prevents bottle necks by parallelizing the communications and control.

A high-level view of the solution described in this article can be seen in Figure 1. Each sensor is controlled by a single agent, and the organizational design divides these sensor agents into location-based sectors. Each sector has a *sector manager*, a role in the organization which has several responsibilities associated with information flow and activity within the sector. Among these responsibilities is the dissemination of a scan schedule to each of the sensors in its sector, which specifies the rate and frequency which should be used to scan for new targets. An example schedule might be to have the sensors on the edge of the sector scanning frequently for incoming targets and the sensors in the interior of the sector lying dormant until a target is detected. The scan schedule is used by each sensor to create a description of the scanning task, which is in turn used by the SRTA architecture to schedule local activities.

When a new target is detected, the sector manager selects a *track manager*, a different organizational role responsible for tracking that target as it moves through the environment. This entails estimating future location and heading, gathering available sensor information, requesting and coordinating over the sensors, and fusing the data they produce. After determining which sensors can see its target, the track manager requests commitments from the them.

Upon receipt of such a commitment, like ‘repeatedly take measurements using head 1’, sensors takes on a *data collection* role. Like the scan schedule, these commitments form the basis of task descriptions used by SRTA to schedule local activities. If conflicting commitments are received by a sensor, which implies that the agent has been asked to perform multiple concurrent data collection roles, SRTA will attempt to satisfy all requests as best possible. This provides a window of marginal quality where SPAM can detect the conflict, and then negotiate with the competing agent to find an equitable long-term solution. As the data is gathered, it is transmitted back to the track manager for incorporation into the target’s track.

The process of collecting measurements, fusing the data into position estimates, modeling the targets predicted path based on these estimates, and allocating new sensor resources continues until the target is lost or moves outside the sensor field. In either case, the track manager revokes all commitments from the sensors and informs the sector manager that it is no longer tracking its target. This allows the sensors to be freed up to potentially re-acquire the target, and allows the sector manager to determine that the detection must be assigned to a new track manager.

In the next section, we will discuss how the environment is partitioned into sectors and how roles are assigned to the agents within them.

3 Organizational Design

The notion of “organizational design” is used in many different fields, and generally refers to how entities in a society act and relate with one another. This is true of multi-agent systems, where the organizational design of a system can include a description of what types of agents exist in the environment, what roles they take on, and how they interact with one another. The objectives of a particular design will depend on the desired solution characteristics, so for different problems one might specify organizations which aim toward scalability, reliability, speed, or efficiency, among other things.

The organizational design used in this solution primarily attempts to address the scalability problem by concentrating knowledge within the agents that are most likely to need it and by imposing limits on how far certain classes of information propagate. As will be seen below, this is done at the expense of

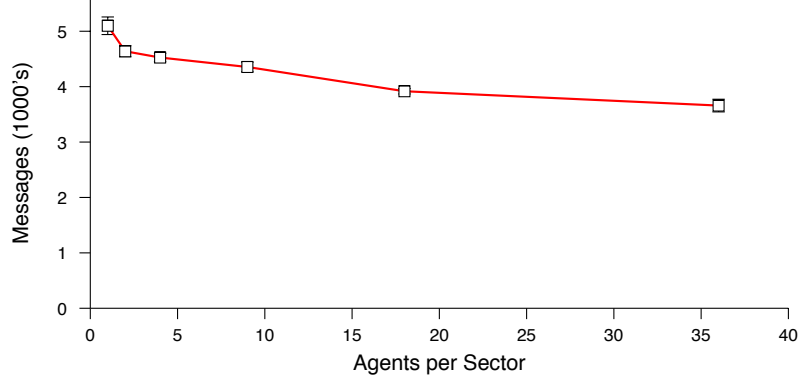


Figure 3: Total number of messages for 36 agents tracking 4 targets given various sector sizes.

reaction speed, because by limiting the scope any single agent has, one necessarily increases the required overhead when the agent’s task moves outside that scope.

The environment itself is organizationally partitioned into a series of sectors, each a non-overlapping, rectangular portion of the available area, shown in Figure 1A. The purpose of this division, as will be shown below, is to exploit locality and limit the interactions needed between sensors. This is an important element of our attempt to make the solution scalable; these same pressures lead to the similar approaches taken in [7,8]. In figure 1A, sensors are represented as divided circles, where each 120 degree arc represents a direction the node can sense in. As agents come online, they must first determine which sectors they can affect. Because the environment itself is bounded, this can be trivially done by providing each agent the height and width of the sectors. The agents can then use this information, along with their known position and sensor radius, to determine which sectors they are capable of scanning in. We use this technique to dynamically adapt the agent population for scanning and tracking activities to better partition and focus the flow of information.

Within a given sector, agents may work concurrently on one or more of several high level goals: managing a sector, tracking a target, producing sensor data, and processing sensor data. The organizational hierarchy is abstractly represented in Figure 5. The organizational leader of each sector is a single sec-

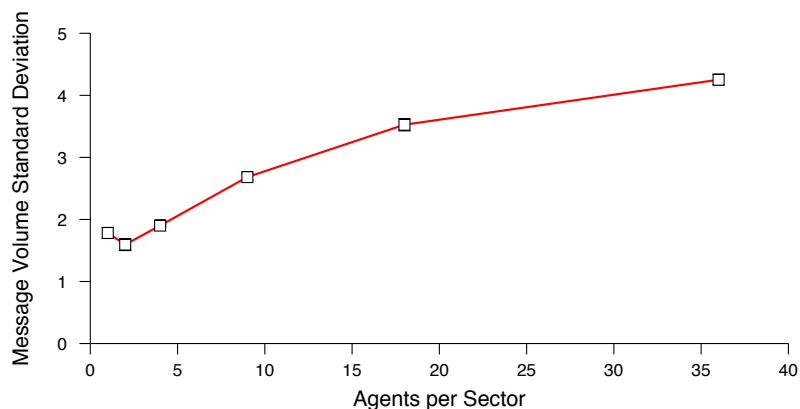


Figure 4: Communication load disparity among agents within each sector.

tor manager, which serves as the locus of activity for that sector. This manager generates and distribute plans (to the sensor data producers) needed to scan for new targets, stores and provides local sensor information as part of a directory service, and assigns track managers. The sector managers act as hubs within a nearly-decomposable hierarchical organization, by directly specifying scanning activities, and then selecting agents to oversee tracking activities. They also concentrate nonlocal information, facilitating the transfer of that knowledge to interested parties. Individual track managers initially obtain their information from their originating sector manager, but will also interact directly, though less frequently, with other sector and track managers, and thus do not follow a fixed chain of command or operate solely within their parent sector as one might see in a fully-decomposable organization. Track managers will also form commitments with one or more agents to gather sensor data, but this relationship is on a voluntary basis, and that gathering agent's behavior is ultimately determined locally. Because much of the information being communicated is contained within sectors, the size and shape of the sector has a tangible effect on the system's performance. If the sector is too large, and contains many sensors, then the communication channel used by the sector manager may become saturated. If the sector is too small, then track managers may spend excessive effort sending and receiving information to different sector managers as its target moves through the environment.

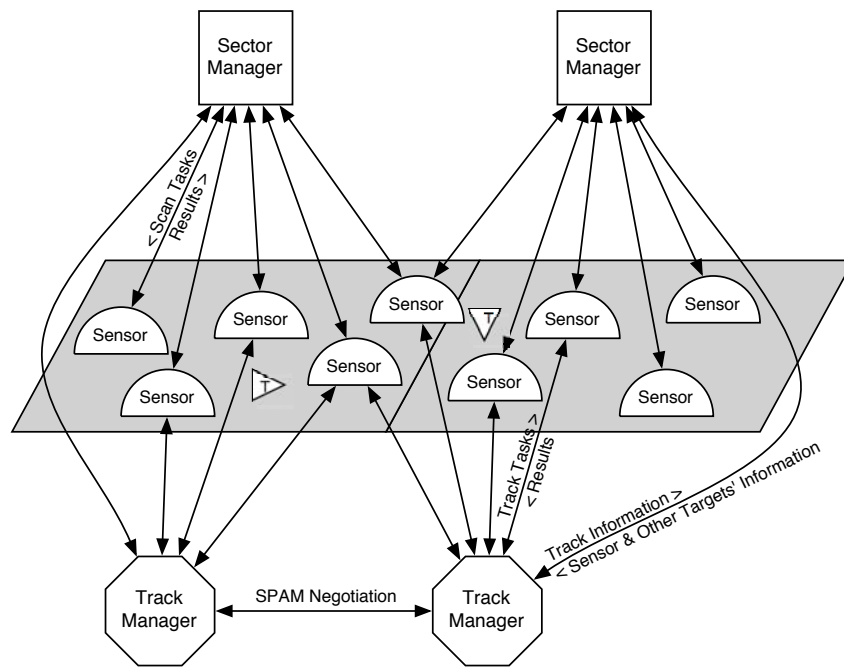


Figure 5: Overview of the agent's organizational hierarchy, with some information flows represented.

To study the effects of sector size on communication cost, we conducted an empirical study. In these experiments, a group of 36 sensors were organized into between 1 and 36 equal-sized sectors. Four targets were introduced into the environment and the number of messages were measured for a fixed duration simulation. These simulations were conducted using a perfect communications model (i.e. channel congestion is not accounted for). As you can see from Figure 3, sector size has a fairly strong effect on the number of messages being sent. As you increase the number of agents within a sector, the number of messages goes down. What is noteworthy about this effect is that it diminishes rapidly as you increase the sector size. This suggests that there is a point at which the benefits of increasing the sector size are outweighed by the cost of centralizing too much information and the resulting communication within the sector manager. Figure 4 verifies this intuition. This figure shows that as you put more agents within a single sector, the standard deviation of the message volume increases. The increase is caused by having a greater portion of the organization load concentrated within fewer agents.

We have found empirically that a reasonable sector would contain 8 sensors, but would still function adequately with as many as 10 or as few as 5. The physical dimensions of such a sector depend on the density of the sensors, and in different environments one would need to take into account sensor range, communication medium characteristics and maximum target speed. Further information on partitioning agent populations, including a more sophisticated technique which utilizes heterogeneous regions, can be found in [9].

To see how the organization works in practice, consider a scenario starting with agents determining what sectors they can affect, and which agents are serving as the managers for those sectors. Ideally, the sector managerial duty would be delegated and discovered dynamically at runtime, but due to the lack of a true broadcast capability in the RF communication medium, we statically define and disburse this information a priori¹. In Figure 1, these sector managers are represented with shaded inner circles. Once an agent recognizes its manager(s), it sends each a description of its capabilities. This includes such things as the position, orientation, and range of the agent's sensor. The manager then has the

¹A limited broadcast capability does exist, which can reach all sensors listening on a single channel. It is not possible in this architecture to broadcast a single message to agents which are using different channels.

task of using this data to organize the scanning schedule for its sector. The goal of the scan schedule is to use the sensors available to it to perform inexpensive, fast sensor sweeps of the area, in an effort to discover new targets. The manager formulates a schedule indicating where and when each sensor should scan, and communicates with each agent over their respective responsibilities in that schedule (see Figure 1B). The manager does not strictly assign these tasks - the agents have autonomy to locally decide what action gets performed when. This is important because sensors can potentially scan in multiple sectors, thus there is the possibility that an agent may receive multiple, conflicting requests for commitments from different sector managers. The agent's autonomy and associated local controller permit the agent itself to be responsible for detecting and resolving these conflicts. If one receives conflicting requests for commitments, it can elect to delay or decommit as needed. Shaded sensors in the previous figure show agents receiving multiple scan schedule commitments.

Once the scan is in progress, individual sensors report any positive detections to the sector manager which assigned them the scanning task, which can then spawn a new track manager as shown in Figure 6. Internally, the sector manager maintains a list of all local agents that currently perform the role of track manager, and location estimates for the targets they are tracking. These location estimates are used to determine the likelihood of the positive detection being a new target, or one already being tracked. If the target is new, the manager uses a range of criteria to select one of the agents in its sector to be the track manager for that target. Not all potential track managers are equally qualified, and an uninformed choice can lead to very poor tracking behavior if the agent is overloaded or shares communication bandwidth with other garrulous agents. Therefore, in making this selection, the manager considers the agents' estimated load, communication channel assignment, geographic location and activity history. Ideally, it will select an agent which has minimal channel overlap, is not currently tracking a target, but has tracked one previously. This will minimize the potential for communication collisions, which occur if two agents on the same channel attempt to send data at the same time, but maximize the potential amount of cached organizational data the agent can be reuse. As we have seen previously, this notion of limited communication is an important motivating factor and recurring theme in this architecture which contributes to the organizational structure, role selection, protocol design and the frequency

and verbosity of communication actions.

The assigned track manager (shown in figure 1C with a blackened inner circle) is responsible for organizing the tracking of its assigned target. To do this, it first discovers sensors capable of detecting the target, and then communicates with members of that group to gather the necessary data. Discovery is done using the directory service provided by the sector managers. One or more queries are made asking for sensors which can scan in the area the target is predicted to occupy. The track manager must then determine when the scans should be performed, considering such things as the desired track fidelity and time needed to perform the measurement, and coordinate with the discovered agents to disseminate this goal (see Figure 1C). As with scanning, conflicts can arise between the new task and existing commitments at the sensor, which the agent must resolve locally.

The data gathered from individual sensors is collected by an agent responsible for fusing the measurements into a location estimate and extending the computed track (see Figure 1D). In a general sense, this data fusion agent could be any agent in the population able to communicate efficiently with both the data sources and the ultimate destination of the tracking data. However, the data fusion process for this application (see Vargas et al. in [3]) is fairly lightweight, and thus does not benefit from distribution for load balancing purposes. In addition, transferring the fused data results introduces an unnecessary delay while it is being communicated to the track manager. For this reason, in this work, the data fusion and track manager roles are always performed by the same agent.

When raw measurement data is received by the track manager, it verifies the quality and association of the measurement. Quality can be measured by several means and in this work, the signal-to-noise ratio is used. If a measurement is returned with a significantly high signal strength, it is considered for potential fusion. The association of a measurement really involves two different things: temporal and target association.

Temporal association attempts to match measurements taken from various sensors based on the time the measurements were taken. As previously mentioned, the accuracy of triangulating a moving target is in part dependent on the relative temporal coordination of the measurements. If a track manager were to fuse measurements that were taken over a wide range of time, the resulting

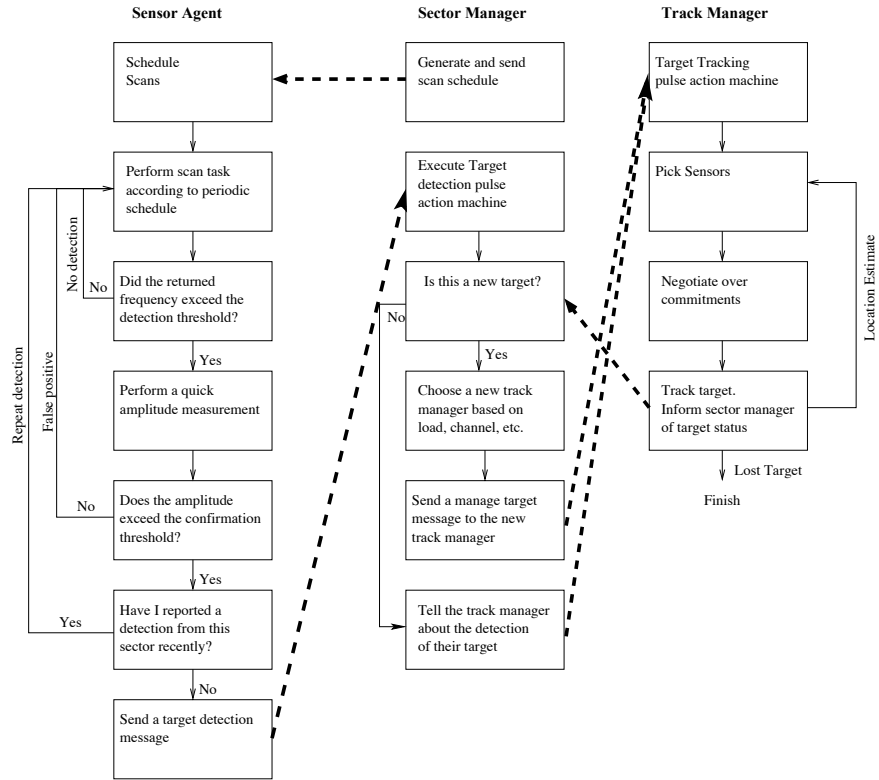


Figure 6: An abstraction of the messages and reasoning used for target detection by sensor agents, sector and track managers.

estimation could be quite poor. There are several reasons why a measurement may be returned that cannot be temporally associated with measurements from other sensors. One reason is delays and loss introduced by the communications. For example, if the commitment request from the track manager never gets to the sensor, no measurement will be taken. Another reason is unresolved resource contention within the individual sensors. If, for example, one of the sensors decides to delay the start of a measurement in order to deconflict its internal schedule, it will be harder to match the resulting measurement with data from other sensors.

Within our system, temporal association maintained by queuing measurements for a finite period and matching them with one another based on their actual measurement time. We make the assumption that the computers controlling the sensors are time synchronized using a Network Time Protocol (NTP) like mechanism.

Target association, as the name implies, tries to match measurements with targets. Consider the case of two targets, T1 and T2, that are following one another through the sensor field. Now imagine that T1 moves out of the viewable area of one of the sensor heads being used to measure it and T2 moves into its this head's view. If a measurement is taken by that head before the T1's track manager changes its allocation, the resulting amplitude will be of the wrong target. When this measurement is fused with the track of T1, it will appear to be closer to T2 than before. Over time, if this pattern continues, the targets will be indistinguishable from one another. In fact, T1 could end up being ignored and both track managers could try to track T2.

To prevent this from happening, track managers check to ensure that each measurement has a higher likelihood of belong to their target than others that may be near to it. To do this, we exploit the view of the sector manager. Because sector managers are periodically given estimated target locations for all of the targets within their sector, it is easy for them to determine when two or more track managers are likely to have a target association problem. When the sector manager detects this possibility, it informs the appropriate track managers by sending them the locations of the target that are close to them. This allows the track managers to discard any measurement that was likely to have been collected by a sensor viewing another target. If, for some reason, two track managers do fuse their target into one, the sector manager can also recognize

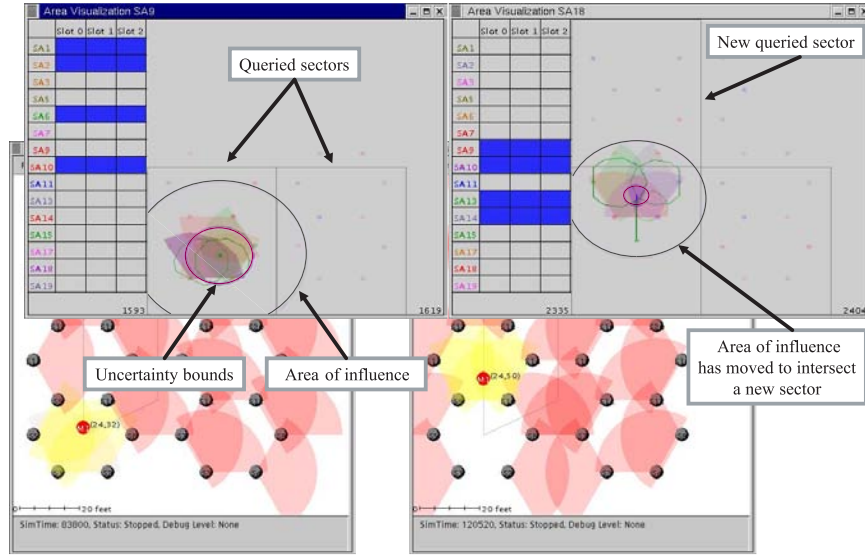


Figure 7: As the target moves to another sector, the track manager queries the manager of the new sector.

this fact and inform one of the managers to stop tracking. This alleviates the unnecessary resource contention created by having two track managers fight over the same resources to track the same target.

If the data values returned are of high enough quality, and the agent determines those measurements were taken from the correct target, then they are used to triangulate what the position of the target was at that time. This data point is then added to the track, which itself is used as a predictive tool when determining where the target is likely to be in the future. At this point the track manager must again decide which agents are needed and where they should scan, and the sequence of activities is repeated.

Partitioning the environment reduces the amount of information and processing that agents must perform for several different tasks. For example, generating a coherent scan schedule for a group of sensors is simplified by only taking into account a tractable number of them. Similarly, when a new target is detected as a result of a scan, that information can be sent to only the appropriate sector manager, which can determine directly if it is a new or existing target based on local information. Sectors also facilitate gathering data about the sensors themselves, as track managers need only perform a single query to

the appropriate sector manager to discover all the sensors available in within that region (see figure 7). In fact, the partitioning makes nearly every aspect of this solution scalable to arbitrary numbers, with the exception of the tracking allocation, which has its own solution to this problem, as shown later. As a side effect, partitioning does reduce the system’s reactivity, because an extra step may be required to fetch information that is not available locally. We cope with this problem wherever possible by caching such data to avoid redundant queries, and by assigning new roles whenever possible to agents which have served that same role in the past, to take advantage of that cached data.

Although not required in the scenarios presented in this article, it is interesting to note the applicability of this organization to situations where agents have an additional limitation or attenuation of communication capability based on the geographic distance separating the participants. In this case, this partitioned organization could serve as the basis of an ad-hoc network, where messages are routed from one sector to the next, using the organizational structure as a guide, until they reach their destination. This further emphasizes the notion that “local” communication is more efficient, and the locality of information should be exploited by the organization to take advantage of it.

In this section we have shown how the organization plays a critical role in ensuring that information flowing within the sensor network is managed to both minimize the delay and expensive of communication and to maximize its availability for effective decision making. The next section describes some of the mechanisms that go into forming the agent level control of the sensors.

4 Agent Architecture

4.1 Soft Real-Time Control

The Soft Real-Time Control Architecture (SRTA), the agent control engine used by this solution, provides several key features to the agents within our sensor network:

1. The ability to quickly generate plans and schedules for goals that are appropriate for the available resources and applicable constraints, such as deadlines and earliest start times.

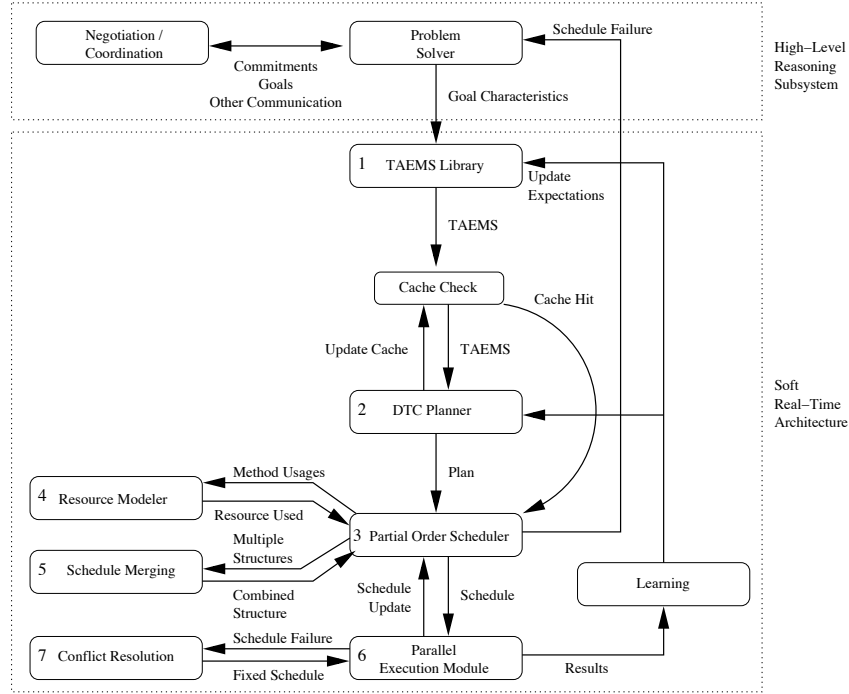


Figure 8: The soft real-time control architecture.

2. The ability to merge new goals with existing ones, and multiplex their solution schedules.
3. The ability to use explicit representations of uncertainty and efficiently handle deviations in expected plan behavior that arise out of variations in resource usage patterns and unexpected action characteristics.

Abstractly, SRTA (shown in Figure 8) operates as a single functional unit within an agent, which itself is running on a conventional (i.e. not real-time) operating system. The SRTA controller is designed to be used in a layered architecture, occupying a position below the high-level reasoning component in an agent. In this role, it accepts new goals, report the results of the activities used to satisfy those goals, and also serve as a knowledge source about the potential ability to schedule future activities by answering what-if style queries.

The components that comprise SRTA assume a majority of the responsibility needed to satisfy goals, which allows the high-level reasoning system to focus on goal selection, determining goal objectives and other potentially domain-

dependent issues. For example, agents may elect to coordinate using abstractions of their activities or resource allocations which are then locally translated into a precise schedule [4]. SRTA can then use these schedules to both enforce the semantics of the commitments which were generated, and automatically attempt to resolve conflicts that were not addressed through coordination.

In the rest of this section, TÆMS, the modeling language used to describe goals to SRTA, is explained. Following that, we will show how the components of the SRTA architecture work together to provide the agents with the ability to handle multiple, concurrently executing goals. Lastly, we will describe a specialized component, called the Periodic Task Controller (PTC). The PTC provides SPAM with an abstract view of the current resource schedule which it uses as a basis for mediation.

4.2 TÆMS

TÆMS, the Task Analysis, Environmental Modeling and Simulation language, is used to quantitatively describe the alternative ways a goal can be achieved [10]. A TÆMS task structure is essentially an annotated task decomposition tree. The highest level nodes in the tree, called task groups, represent goals that an agent may try to achieve. For example, the goal of the structure shown in Figure 9 is **Setup-Hardware**. Below a task group there will be a set of tasks and methods which describe how that task group may be performed, including sequencing information over subtasks, data flow relationships and mandatory versus optional tasks. Tasks represent sub-goals, which can be further decomposed in the same manner. **Setup-Hardware**, for instance, can be performed by completing **Startup**, **Init**, and **Obtain-Background-Noise**.

Methods, on the other hand, are terminal, and represent the primitive actions an agent can perform. Methods are quantitatively described, in terms of their expected quality, cost and duration. **Activate-Sector_0**, then, would be described with its expected duration and quality, allowing the scheduling and planning processes to reason about the effects of selecting this method for execution. The quality accumulation function (QAF) below a task describes how the quality of its subtasks is combined to calculate the task's quality. For example, the **q_min** QAF below **Init** specifies that the quality of **Init** will be the minimum quality of its subtasks - so all the subtasks must be successfully performed for the **Init** task to succeed. Interactions between methods, tasks,

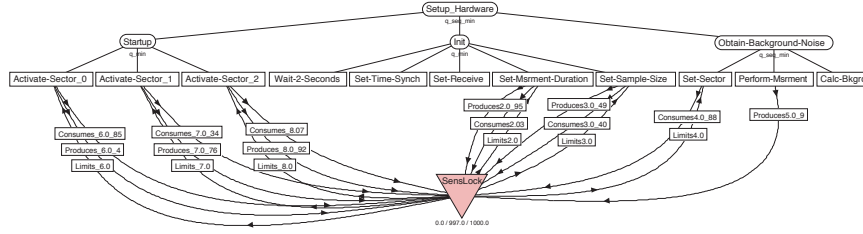


Figure 9: An abbreviated view of the sensor initialization TÆMS task structure.

and affected resources are also quantitatively described. The curved lines in Figure 9 represent resource interactions, describing, for instance, the produces and consumes effects method **Set-Sample-Size** has on the resource **SensLock**, and how the level of **SensLock** can limit the performance of the method.

TÆMS structures are used by our agents to describe how particular goals may be achieved. Rather than hard coding, for instance, the task of initializing the sensor, we encode the various steps in a TÆMS structure similar to that shown in Figure 9. This simplifies the process of evaluating the alternative pathways by allowing the designer to work at a higher level of abstraction, rather than be distracted by how it can be implemented in code. More importantly, it also provides a complete, quantitative view that can be reasoned about by planning, scheduling and execution processes. A given task structure begins its existence when it is created, read in from a library (Figure 8-1), or dynamically instantiated from a template at runtime. Planning elements are involved both in the generation of the structure, and then in the selection of the most appropriate sequence of methods from that structure which should be performed to achieve the goal given the currently available resources. This sequence is then used by a scheduling process to determine the correct order of execution, with respect to such things as precedence constraints and resource usage. Finally, this schedule will be used by an execution process to perform the specified actions, the results of which are written back to the original task structure.

The schedules produced by individual TÆMS structures are the building blocks for an agent's overall schedule of execution. A valid schedule completely describing an agent's activities will allow it to correctly reason about and act upon the deadlines and constraints that it will encounter, for example a resource restriction. Typically, however, schedules are only used to describe lower-level activity. In this domain, this encompasses sensor initialization, scanning and

tracking activity, data fusion and the like.

4.3 Scheduling

In the SRTA architecture, we have attempted to make the scheduling and planning process incremental and compartmentalized. New goals can be added piecemeal to the execution schedule, without the need to re-plan all the agent's activities, and exceptions can be typically handled through changes to only a small subset of the schedule. Figure 8 shows the organization of SRTA. In this architecture, goals can arrive at any time, in response to environmental change, local planning, or because of requests from another agents. The goal is used by the problem solving component to generate a TÆMS task structure, which quantitatively describes the alternative ways that goal may be achieved. The TÆMS structure can be generated in a variety of ways; in our case we use a TÆMS "template" library, which we use to dynamically instantiate and characterize structures to meet current conditions. Other options include generating the structure directly in code, or making use of an approximate base structure and then employing learning techniques to refine it over time.

SRTA uses the Design-To-Criteria component [11] to generate linear plans solving the goal described in the TÆMS structure (Figure 8-2). It employs a battery of techniques to efficiently discover and reason about the various activity schedules which can address that goal. The ability to make trade-offs while respecting commitments is particularly important, as DTC attempts to select the quantitatively "best" plan which meets the specified requirements. DTC uses criteria such as potential deadlines, minimum quality, external commitments, and soft and hard action interrelationships to select an appropriate sequence of activities.

The resulting plan is used to build a partially ordered schedule, which uses structural details of the TÆMS structure to determine precedence constraints and search for actions which can be performed in parallel (Figure 8-3). The partial order scheduler (POS) also provides the ability to quickly shift methods' execution order at any point in time instead of performing costly re-planning [5]. In a real-time environment, schedule adjustments are more frequent; by not imposing unnecessary ordering constraints on our agent's schedule the agent has a better chance of achieving the time, cost and quality criteria of its goal. A pair of specialized components are used to assist the POS during this final

scheduling phase. The first, a resource modeling component, is used to ensure that resource constraints are respected (Figure 8-4). A schedule merging module then allows the partial order scheduler to incorporate the actions derived from the new goal with existing schedules (Figure 8-5).

Our notion of “parallel” in this architecture includes activities which run concurrently in parallel, as in a multiple processor environment, and those which run virtually in parallel, as in a time-slicing, multi-processing operating system. If we view the sensor as a specialized, separate processor, our task structures contain both types of methods. For instance, a sensor measurement action can take place concurrently with actions on the primary processor. Unifying these notions simplifies the scheduling process, and can be represented appropriately using TÆMS.

Once the schedule has been created, an execution module is responsible for initiating the various actions in the schedule (Figure 8-6). It also keeps track of execution performance and the state of actions’ preconditions, potentially re-invoking the partial order scheduler when failed expectations require it. Using the ordering constraints described in the schedule, the execution component can directly determine which methods can be run concurrently. By overlapping their execution, we reduce the total execution time, which effectively increases the agents overall work capacity. The gain in execution time, and resulting flexibility, is used to address resource availability, in addition to improving the likelihood the scheduler can accommodate real-time changes without breaking deadline constraints.

If this is unsuccessful, a conflict resolution module is used to reason about mutually-exclusive tasks and commitments, determining the best way to handle conflicts (Figure 8-7). Repairs can be accomplished in a variety of ways, for instance, by relaxing constraints such as the goal completion criteria or delaying its deadline, completing a substitute goal with different characteristics, or decommitting from a lower priority goal or the goal causing the failure.

The execution characteristics of the SRTA architecture as a whole depend largely on the frequency and complexity of the goals it is asked to plan and schedule. On average, we observe cycle times of between 50 and 100 milliseconds on 400 MHz x86-based systems, although this can jump to a half-second or more if a particularly complex situation arises. A cycle represents a single pass of the SRTA engine analyzing the current goals and executing methods. Because the

system runs on a conventional operating system (Linux in this case), competing external processes may add an additional level of performance uncertainty.

4.4 Periodic Tasks

In addition to the general purpose scheduling outlined above, the agents also incorporate a more specialized periodic scheduling mechanism. In order to reduce communication, commitments between agents can be expressed as tasks which are to be performed periodically and indefinitely until notified otherwise. To reduce computation, these periodic tasks are arranged and scheduled according to a slot-based scheme, where a given task will be assigned to run in one or more slots in a repeating cycle. For example, in this architecture a cycle consists of three slots, each of which has a length of one second, for a total cycle time of three seconds. Then, elements of a commitment might specify that a scan or track measurement should be performed indefinitely on sensor head 1 in slot 2. A subsequent message would indicate when to stop performing the task. Using this technique, a pair of messages can cause a large amount of data to be gathered, and the required local scheduling overhead is reduced by abstracting the continuous timeline into discrete independent portions.

Like any scheduling process, conflicts can arise when two tasks are to be performed in the same slot. Different resolution techniques are used to cope with this situation. Individual tasks have priorities associated with them - for example, tracking tasks are more important than scanning tasks, and as mentioned previously, some tracking tasks may be more important than others. These priorities can be used to give preference to certain tasks. If a task is preempted, it may either be suspended for the lifetime of the higher priority task, or shifted to a free time slot if one is available. Alternately, if two tasks have the same priority, the scheduler will attempt to divide the slot between them, so that the first will run during one cycle, the second during the subsequent cycle, then the first, and so on. A third technique, not used in this domain, could also attempt to perform both tasks in the same slot if sufficient time, or a shorter duration alternative, exists to do so.

In this section, we described the SRTA architecture and showed how it is used within our system to provide the agents with autonomous reasoning capabilities. These capabilities are exploited to both reduce communications, by allowing agents to transmit control information in an abstract manner, and to provide

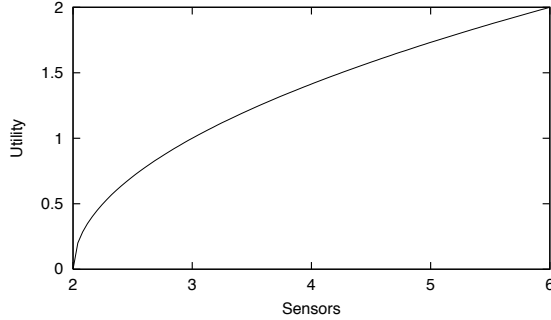


Figure 10: Utility of taking a single, coordinated measurement from a set of sensors.

tolerance to rapid environmental changes, by allowing agent to re-plan and re-schedule reactively based on local information. The next section of this article describes the SPAM resource allocation protocol which provides this system with the ability to handle resource conflicts by elevating the decision making to the track managers.

5 Resource Allocation

5.1 Tracking as Resource Allocation

Modeling the target tracking domain as a resource allocation problem is fairly straightforward. Each of the targets in the environment can be considered a task, which is assigned to a track manager. The sensors are the resources and the job of the track managers is to obtain enough sensing time from the correct sensors to track their targets.

There are a number of characteristics about this particular resource allocation problem that make it challenging. The first is that, although this problem can be solved crudely using constraint satisfaction techniques, it lends itself most naturally to being solved using some type of optimization. Optimization is a good fit for two reasons. The first is that the value of a sensor platform to a track manager is directly correlated with the distance and relative angle of its most appropriate sensor head to the manager's target. Increasing the angle or distance decreases the accuracy and the resulting value of a measurement. In addition, increasing the number of sensors involved in taking coordinated

measurements improves the accuracy of the resulting location estimation (see figure 10). The goal of the track managers, therefore, is not simply to come up with a conflict free resource assignment, but to derive a solution that *maximizes* the number and the value of the sensors used to track the targets.

The second major difficulty is the need to temporally coordinate the actions of the sensors. Coordinating the resources in this way adds another dimension to the problem and increases its difficulty considerably. The biggest problem with temporal coordination is that time is a continuous value. There are essentially an infinite number of possible combinations to consider when trying to deconflict and coordinate the sensor schedules. As mentioned in the previous section, the method used in this system to reduce the number of possible combinations is to use a finite planning horizon on a slot based schedule. By using temporally coordinated slots, the track managers lose the ability to perform fine-grained scheduling, but the overall complexity of the resource allocation is decreased. So, in fact, the track managers not only have to maximize the number and value of the sensors, but have to maximize the amount of time those sensors are dedicated to taking measurements for their target.

If we say that M_s^i is the set of good sensors measurements (can see the target) leading to the positional estimate in a single slot s for a task i , and $Util(M_s^i)$ is defined as the utility function in Figure 10 then the utility function for that task during a specific period is:

$$U_i(a_i) = \sum_{s=1}^k Util(M_s^i)$$

Fortunately, the need for coordination actually allow us to consider a much smaller subset of the possible allocations for a given task. In fact, track managers within our system use a simplified set of *objective levels* defined by their utility functions to assign resources to their targets. Each objective level is expressed as a cross product $D_m \times D_s$ denoting the number for sensors, desired for a number of slots in the planning horizon. For example, a track manager may wish to have three sensors for two slots, which is denoted 3×2 . Although the number of slots in a period is variable, for this domain, we typically set it to match the number of sensor heads on each platform, which is three. In order to prevent certain targets from being ignored in order to improve the quality of another targets estimate, track managers are penalized for not triangulating their target during a full period.

The third characteristic of interest is the dynamic nature of the problem. As the targets move through the environment, moving in and out of the range of the sensors, the underlying resource allocation problem changes in structure. This drives the need to constantly monitor, re-evaluate, and reallocate the sensors used to measure the positions of the targets. It also means that any method used to allocate the resources has to be responsive to changes that occur in the middle of the allocation process.

The last major challenge is that the allocation technique needs to be resource aware. As mentioned in Section 2, the communications infrastructure is RF-based, which in this case makes it is very slow and unreliable. These properties make it essential to not only limit the amount of communications, but to be aware of and adapt to changes in the overall ability to communicate. For example, if it is taking a long time to get messages to a particular track manager, it probably makes sense to avoid creating conflicts with it. Also, if a particular allocation is only useful for a very short period of time, it may not make sense to engage in a complex reallocation process when a simpler one may meet the basic need to track.

5.2 The SPAM Protocol

Scalable, Periodic, Anytime Mediation (SPAM) algorithm is built around the principle of “good enough, fast enough”. As such, the protocol is actually divided into two major stages. Stage 1 of the protocol uses local information to derive and bind temporary solutions that are seldom free of conflict and are often based on inaccurate, incomplete information. Stage 2 of the protocol solves conflicts and distributes resources by initiating a *cooperative mediation* session. During a mediation session, one of the track managers takes on the role of mediator. As the mediator, it gathers information from track managers that are in conflict, computes and recommends possible solutions to the problem, and then announces a final solution.

The SPAM protocol is activated under two conditions. The first condition is occurs when the resources needed to track a target change due to a target’s movement. These types of changes often alter the structure of the underlying optimization problem. As such, whenever they occur, managers instantiate the first stage of the protocol to quickly modify their current solution or to create an initial solution in the case of a new target assignment. Once stage 1 completes,

a manager can choose to begin a mediation session if they determine that the benefits outweigh the costs.

The second case occurs when a manager detects a conflict within one of the resources its using. This type of conflict is caused when another manager is unable to mediate a session, is able to mediate but due to latency has not yet begun it, or is unaware that the resource is already being used. In each of these cases, the manager detecting the conflict has the option to immediately activate stage 2 and mediate a session to repair it. A distributed locking mechanism prevents more than one manager from concurrently mediating if latency was the cause of the conflict detection.

5.3 Stage 1

Stage 1 of SPAM serves three primary functions. The first function is to attempt to find a solution within the context of the information that the protocol has when it starts up. Like the Asynchronous Weak Commitment (AWC) protocol [12], each of the agents tries to find an assignment that is consistent with its potentially incomplete or inconsistent *agent_view*. However, because this protocol attempts to maximize the social utility, each of the agents tries to maximize their local utility without causing new constraint violations. If this can be done, then no further mediation is necessary, and the protocol terminates at the end of stage 1.

We should mention that a trade-off exists between communication overhead and utility, due to the initial selections of the objective level in stage 1. If each of the managers chooses to use every available resource (sensors able to see their target), the possibility for contention over resources greatly increases in the environment, thereby causing the execution of stage 2 to occur more frequently. However, if the agents decide to start with at a lower objective level (and correspondingly less utility), the social utility may suffer unnecessarily.

Stage 1 has what we refer to as a concession rate. The concession rate defines what percentage of the local solution quality a track manager is willing to concede to find a violation-free solution in an attempt to avoid a potentially expensive stage 2 mediation session. So, as the manager's utility drops, the amount they are willing to concede drops as well. This causes managers to mediate (Stage 2) more frequently in critically constrained tracking environments.

The second function of stage 1 is to ensure some utility is obtained while

waiting for stage 2 to complete. Since these temporarily applied solutions are only applied when a completely conflict-free assignment is not possible, unresolved conflict are left to the individual sensor agents to handle. As mentioned in the previous section, sensor agents can use one of a number of techniques, including slot boundary shifting, less expensive measurement types, or task rotation, in order to resolve such conflicts. To the track manager, whether or not they get a measurement from a conflicted sensors is probabilistically random.

Temporarily applied solutions do not use the concession rate. In fact, because of environmental changes and the probabilistic nature of getting measurements from conflicted sensors, managers always use their maximum possible objective level (within the bounds of the number of sensors that can see the target). The reason for this is rather subtle, but important. Let's say that a new resource were added to the possible resources that could be used by manager T1. Let's also say that another manager, T2, who has more than enough available resources to itself, were using that entire resource. If T1 starts a mediation at that lower level, it can never obtain its highest level as a result of the session, even though a solution exists where T2 just gives up the entire conflicted resource.

The third purpose of stage 1 relates to the anytime characteristics of the protocol. Because a solution is always derived and applied during stage 1, managers don't necessarily have to enter stage 2. They can stop the process at the end of stage 1 and accept the results that they have achieved. This is often done if a target's movement causes the resource needs to change faster than the expected time it would take to complete stage 2. The expected time to complete stage 2 is computed based on both previous experience and the current estimated channel loads for the track managers that would be in the mediation session..

5.4 Stage 2

Stage 2 is the heart of the SPAM protocol (See Figure 11). Stage 2 attempts to resolve all of the local conflicts a track manager has by elevating the problem to the track managers that are using the desired resources. To do this, the originating track manager takes the role of the mediator (note that multiple sessions can occur in parallel in the environment). As the mediator, it becomes responsible for gathering all of the information needed to generate alternative

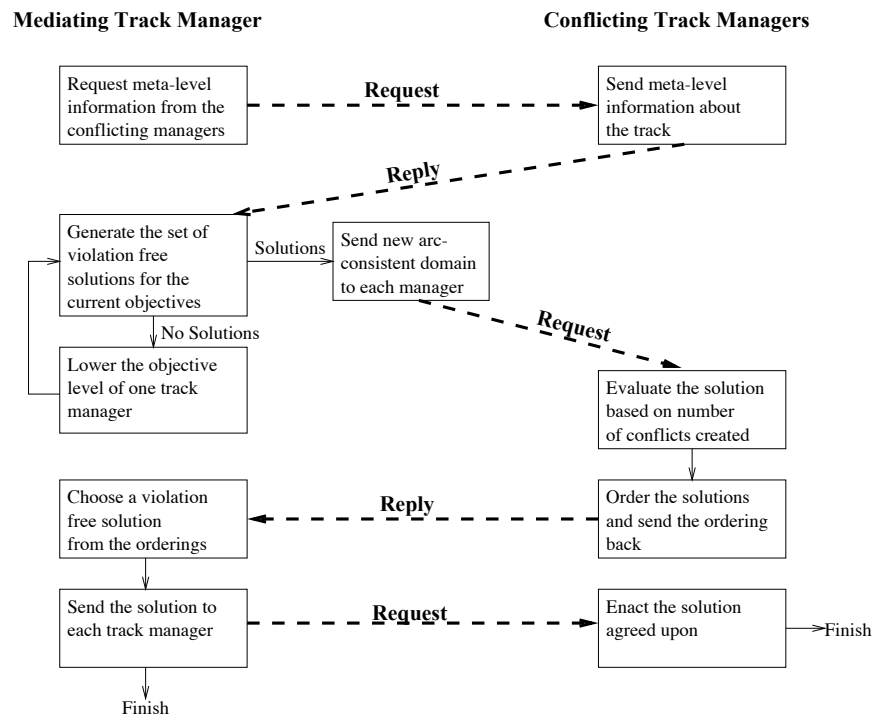


Figure 11: Stage 2 of the SPAM protocol.

solutions, creating solutions which may involve changes to the objective levels of the managers involved, and finally choosing a solution to apply to the problem. These solutions are generated without full global information and may lead to newly introduced non-local conflict. If this occurs, other track managers can begin sessions which propagates the conflict even further.

The best way to explain the operation of stage 2 is through an example. Consider Figure 12, which depicts a commonly encountered form of contention. Here, track manager T2 has just been assigned a target. The target is located between two existing targets that are being tracked by track managers T1 and T3. This creates contention for sensors S3, S4, S5, and S6.

Following the protocol for the example in Figure 12, track manager T2, as the originator of the conflict, takes on the role of mediator. It begins the mediation session by requesting information from each of the track managers involved in the resource conflict. Upon receiving the request, each track managers replies with their current objective level, the number of sensors which can see their target, the names of the sensors that are in direct conflict with the mediator, and any additional conflicts that the manager has. To continue our example, T2 sends a request for information to T1 and T3. T1 and T3 both return that they have 4 sensors that can track their targets, the list of sensors that are in direct conflict (i.e $T1(S_3, S_4)$, $T3(S_5, S_6)$) their objective level (4×3 for both of them) and that they have no additional conflicts outside of the immediate one being considered.

As seen in Figure 11, T2 enters a loop that attempts to generate solutions followed by lowering the objective level of one of the track managers if none exist. A heuristic method, designed to balance the resources, is used to choose the track manager to lower. Namely, the track manager is chosen that has both the highest objective level and is unable to support it without using resources from the set of sensors being mediated over. Whenever two or more managers have the same highest objective level, we choose to lower the objective level of the manager with the least amount of external conflict. By doing this, it is our belief that track managers with more external conflict will maintain higher objective levels, which provides them with leverage in resolving subsequent conflicts that may be occur as a result of propagation.

You should note that although this has similarities to the techniques used in Partial Constraint Satisfaction (PCSP)s, this differs in that the problem changes

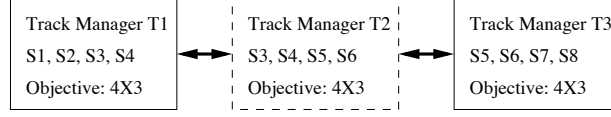


Figure 12: Example of a common contention for resources. Track manager T2 has just been assigned a target and contention is created for sensors S3, S4, S5 and S6.

as the objective levels are changed. PCSP techniques, such as [13] choose to satisfy some subset of the constraints, this technique changes the constraints themselves until the problem is satisfiable.

The solution generation loop is terminated under one of two conditions. First, if given the current objective levels for each of the track managers, the mediator is able to generate at least one satisfying assignment, the session enters the solution evaluation phase. Second, the session ends if the mediator cannot generate a satisfying assignment and it cannot drop the objective level of one of the track managers in the session. Under these conditions, the session is terminated and the mediator lowers its own objective level to the lowest possible level, conceding that it cannot find a satisfying assignment, and binds a solution which minimizes the number of conflicts.

Continuing our example, T2 first lowers the objective level of T1 (choosing T1 at random because they all have equal external conflict). No satisfying assignments are possible under the new set of objective levels, so the loop continues. It continues, in fact, until each of the track managers has an objective level of 3×2 at which time T2 is able generate a set of 216 satisfying assignments to the problem. More information on the solution generation process can be found in [4].

During solution evaluation, the mediator proposes the set of solutions to each track managers by sending them a list of sensor assignments that occur in at least one solution. Because each of these assignments is part of a solution, each the lists have the important property that they are arc-consistent with the lists being transmitted to the other managers in the session. In addition, the mediating track manager is guaranteed to have a conflict-free sensor assignment whenever it successfully concludes a session.

Upon receiving their list, each of track managers rates the assignments based

on their local *agent_view* and their internal utility functions. Continuing our example, T2 sends a list of assignments to T1, a list of assignments to itself, and a list of assignments to T3. In our system, track managers rank the assignments based a utility function that includes the amount of conflict that will be introduced by taking the assignment and on the desirability of the sensors. This is similar to the min-conflict heuristic [14] and is an integral part of the hill-climbing nature of the algorithm.

Once the mediator has the ratings from the track managers, it chooses a particular solution to apply to the problem. This is done using a dynamic priority method based on the number of constraints each of the managers has external to the mediation, a form of meta-level information. The basic notion is similar to the priority order changes in AWC [12]; try to find the task which is most heavily constrained and elevate it in the orders. Our impression is that this helps stem the propagation because it leaves the most constrained tasks with the best choices. This allows those managers to maintain violation-free solutions if they exist in the alternatives presented to them.

In our example, T2 collects the ordering from T1, T2, and T3. T3 is given first choice. By its ordering, it ranked alternative 0 the highest. This restricts the choice for T2 to alternatives 0, 1, 2, and 3. T2 ranked 0 highest from this set of alternatives, restricting T1's choice to its 0th, 1st, and 2nd alternatives. It turns out that T1 likes its 0th solution the best so the final solution is composed of T3's alternative 0, T2's alternative 0, and T1's alternative 0.

The last phase of the protocol is the solution implementation phase. Here, the mediator simply informs each of the track managers of its final choice. Each of the track managers then implements the final solution. At this point, each of the track managers is free to propagate and mediate if it chooses. Figure 13 shows the configuration of the sensors before and after T2 completes stage 2.

This section described the Scalable, Periodic, Anytime Mediation (SPAM) protocol. This protocol uses cooperative mediation to solve a dynamic, distributed optimization problem. The next section, presents results from experimentation that was done in simulation and later verified with hardware.

	Slot 1	Slot 2	Slot 3
S1	T1	T1	T1
S2	T1	T1	T1
S3	T1/T2	T1/T2	T1/T2
S4	T1/T2	T1/T2	T1/T2
S5	T3/T2	T3/T2	T3/T2
S6	T3/T2	T3/T2	T3/T2
S7	T3	T3	T3
S8	T3	T3	T3

→

	Slot 1	Slot 2	Slot 3
S1		T1	T1
S2		T1	T1
S3	T2		T1
S4	T2	T1	T2
S5	T2		T2
S6	T3	T3	T2
S7	T3	T3	
S8	T3	T3	

Figure 13: A solution derived by SPAM to the problem in Figure 12. The table on the left is before track manager T2 mediates over T1 and T3. The table on the right is the result of stage 2.

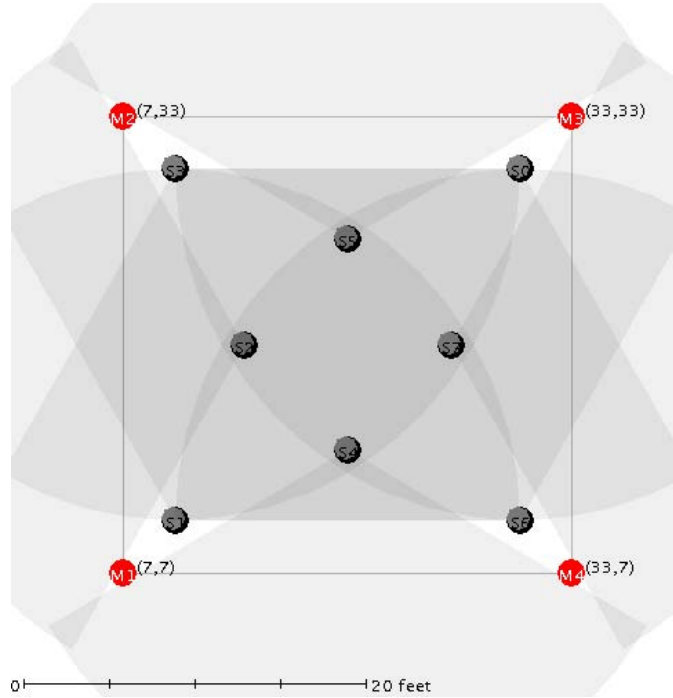


Figure 14: Radsim environment with 8 sensor nodes (S1-S8) and 4 targets (M1-M4).

6 Results

6.1 Resource Allocation

A distributed sensor network environment lends itself to many different types of metrics. The most obvious is the accuracy of the tracking process, as determining accurate positions for the various targets is arguably the point of the system. In a conventional, unified system this would be the case, however, because the tracking component of this solution was a black box developed by a third party, we instead focus our attention on the number and relative coordination of the measurements being taken for each of the targets.

To test the effectiveness of SRTA and SPAM, we used a simulation tool, called Radsim (see Lawton in [3]), that very closely models the target tracking environment and sensor hardware. Radsim simulates the behavior of one or more targets that have defined paths through the environment. Agents connect to Radsim and use a standard API to send control signals to their respective sensor platforms. Radsim, in turn, uses a model of the platforms' behavior to return, in the case of taking a measurement, a hypothetical value for amplitude and frequency based on the current locations of the targets.

We conducted 400 experimental runs, each 3 minutes long, using the environment in Figure 14 consisting of 8 sensors and between 1 and 4 mobile targets. The paths of the targets and the layout of the sensors were chosen to maximize contention with minimal possibility of target ambiguity, thus creating a need for proper sensor allocation to effectively track. We analyzed three resource management techniques, comparing them based on their average periodic utility (see section 5.1) during the course of the scenario.

The first and simplest allocation process, single commitment, specifies that each sensor can work on only one commitment at a time. New commitments accepted by a sensor override any existing ones. In the second set of experiments, agents use SRTA to work on multiple commitments, but only local conflict resolution strategies are employed. In this case, the periodic task controller is responsible for managing conflicted commitments as best it can. The third strategy uses the SPAM + SRTA allocation style which represents the full solution presented earlier in this article. Commitments are generated, and existing initial conflicts are resolved locally using SRTA. SPAM then attempts to resolve these conflicts through more intelligent allocation of the sensor resources.

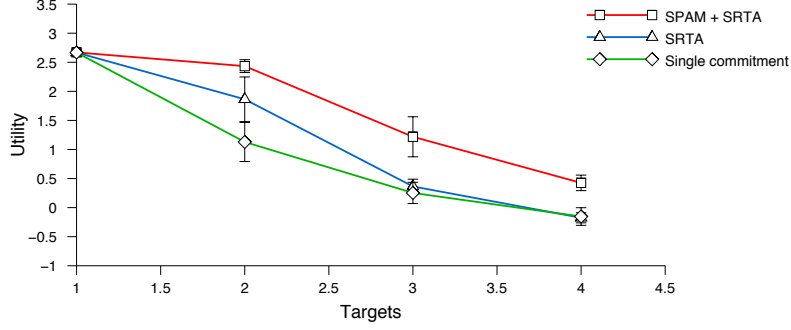


Figure 15: Average utility for a period for three allocation techniques under various conditions.

As seen in Figure 15, when there is only one target, all three of the methods do equally well, because there is no contention for the sensors. The score of about 2.6 stems from the fact that the target, over the course of its movement, enters regions where sensor coverage is limited to just 2 sensors. During these periods, the score becomes -1, lowering the overall average for the track.

As the number of targets increase, resource contention becomes more significant and the differences between the methods becomes clear. For two targets, the single commitment strategy essentially ignores one of the targets whenever contention for the sensors occurs. The SRTA-only strategy has better performance than the single commitment strategy because, although it does not produce a conflict free solution, the local agents are able to occasionally get coordinated measurements. Because this happens essentially as a random event, however, this strategy results in a relatively high standard deviation among scenarios. The SPAM + SRTA strategy clearly does better on average and also has a very small standard deviation. This indicates that in most instances, SPAM is able to resolve the conflict and maintain coordination between the sensors.

As the number of targets increases above two, the differences become more apparent. By four targets (which is highly over constrained), the SRTA only method does no better than single commitment. This is caused by the effects of random coordination of measurements from the sensors causing one or more of the targets to be ignored for most of the scenario. SPAM + SRTA also degrades but the average stays very close to 1 indicating that most of the time, all of the

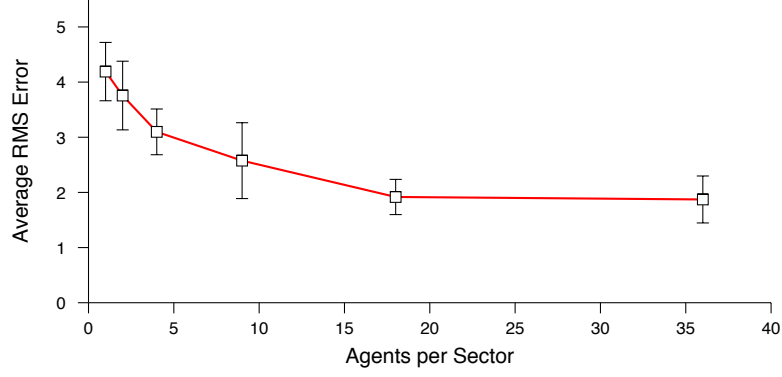


Figure 16: The effects of sector size on tracking quality with ideal communications.

targets are being tracked to some degree.

6.2 Effects of Sector Size

We also conducted a number tests to understand the effect sector sizes on tracking accuracy. In these experiments, a group of 36 sensors were organized into between 1 and 36 equal-sized sectors. Four targets were introduced into the environment and the average tracking accuracy was measured for fixed duration of time. These tests were conducted using ideal communication conditions, i.e. no message loss or congestion effects. As you can see in Figure 16, as you increase the number of agents within a single sector, the RMS error and its variance goes down. As can be seen in Figure 17 this effect can be partially attributed to an increase in the number of sensor measurements being received by the track managers and partially caused by the a reduced delay in receiving sensor information from the sector manager as the target moves through the environment. Again, these results do not take into account the effects of message congestion. If these tests had been conducted using the RF-based system, the effects of centralizing too much information and processing would have become apparent as the number of agents per sector increases. As seen in figure 4, this can cause excessive load in particular agents.

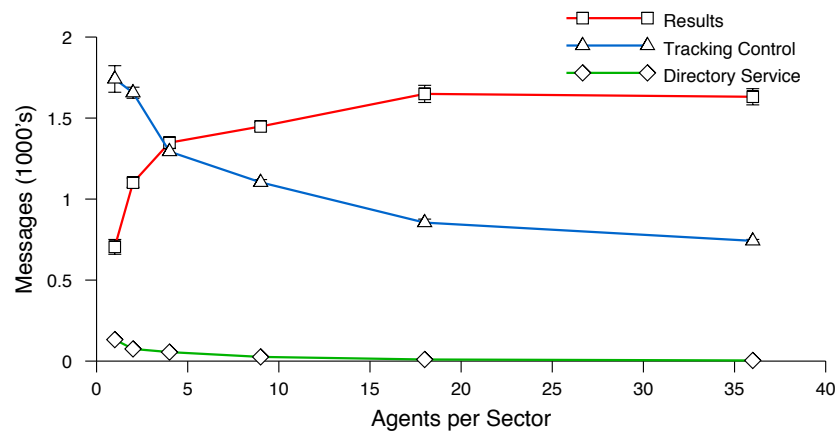


Figure 17: The effects of sector size on various messages being communicated.



Figure 18: Researchers work on a demonstration involving 36 sensors and 3 mobile targets.

6.3 Hardware Evaluation

A fair amount of testing was also done during several hardware-based evaluations (see Figure 18). In these evaluations, sensors were placed in an environment which had a fixed number of model trains, acting as the targets, moving through it. The goal of the evaluation was to track the trains with the highest possible quality while minimizing the amount of communication being used. The results of these tests were mixed. Although our system proved to be very robust, it had difficulty maintaining cohesive tracks for long periods of time. There are a number of factors which caused this to occur. At the lowest level, the raw sensor data proved to be quite poor due to the effects of multi-pathing, poor calibration, and equipment failure. These low-level difficulties caused false target detections and large amounts of uncertainty in estimating the targets' locations and velocities. Compounding the problem is the closed loop nature of target tracking. Since location estimates are used to select sensors, poor estimates lead to improper selection which potentially reduces the quality of the next estimate due to unnecessary increases in resource contention and target association problems. In the end, the track managers lost their targets which were later re-detected and tracked again. Post-processing of the data collected during these experiments showed the agents performed the correct actions given the projections they were provided by the underlying tracking components. This supports the claim that the results derived in simulation are representative of the behavior we would expect with components that provided more reliable data.

7 Conclusions

In this paper we have described our solution to a real-time distributed tracking problem. The agents in the environment are first organized by partitioning them into sectors, reducing the level of potential interaction between agents. Within each sector, agents dynamically specialize to address scanning, tracking, or other goals, which are instantiated as task structures for use by the SRTA control architecture. These elements exist to support resource allocation, which is directly effected through the use of the SPAM negotiation protocol. The agent problem solving component first discovers and generates commitments for sensors to use for gathering data, then determines if conflicts exist with that allocation, finally using arbitration and relaxation strategies to resolve such

conflicts. We have empirically tested and evaluated these techniques in both the Radsim simulation environment and using a hardware-based system.

Despite the fact that many of the details of our solution were designed for the distributed sensor net problem, much of the higher-level architecture is quite general, and applicable to different problems. SRTA, for instance, uses the domain-independent TÆMSlanguage as its basis, which can and has been used successfully in a variety of domains. The SPAM negotiation protocol can be used to solve new distributed, interdependent resource allocation problems by implementing a suitable objective function. SPAM's technique of allowing conflicts to exist and be resolved by local control concurrent with a more complete allocation search can be used in nearly any environment where the participants are tolerant of such uncertainty. Our organizational structure as a whole is quite specific, but individual aspects such as partitioning, task migration and local control are general and applicable to a variety of different distributed architectures.

Our solution as described covers many different aspects of the distributed sensor interpretation problem including uncertainty, target association, and resource allocation. Despite this, there remain parts of the domain which we have not yet explored that seem to offer additional possibility for intellectual contribution. For example, while the sensors in this environment had several different modes of operation with different execution characteristics, in practice it rarely if ever made sense to use anything but the cheapest, fastest measurement type. In addition, the sensor population as a whole was discriminated only by location and orientation, and not by differing capabilities or qualities. If the sensors were more heterogeneous, or if they offered a range of useful modes of operation, it would offer the opportunity for a richer reasoning process. Agents would need to determine not only which sensors to use, but which modes they should operate in, and which functionalities should be exploited, and to trade off these choices against the additional costs they would likely incur. Related to sensor heterogeneity, the ability discriminate among targets also presents a new dimension in which to reason. If targets were identifiable, and correlated with either known characteristics or expected routes, this information would allow agents the possibility of a more effective tracking procedure by exploiting such knowledge. As mentioned earlier, the notion of distance-attenuated communication could also create an interesting environment, requiring agents to more

directly reason about the consequences of long-distance agent relationships.

Acknowledgment

We would like to acknowledge the following people for their important contributions to this research. Jiaying Shen and Kyle Rawlins for implementing the Periodic Task Controller (PTC), Tom Wagner for supporting our scheduling efforts, Tim Middelkoop for evaluating hardware experiments, Raphen Becker for implementing track visualization tools, and Kenneth Parker, Alex Meng and Harry Hogenkamp for their time and expertise preparing for hardware demonstrations. We would also like to thank Stephen Fitzpatrick for his photo of the hardware evaluation.

References

- [1] C.-Y. Wan, A. Campbell, and L. Krishnamurthy, “Psfq: A reliable transport protocol for wireless sensor networks,” in *Proceedings of the First Workshop on Sensor Networks and Application (WSNA)*, Atlanta, GA, September 2002.
- [2] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, “A two-tier data dissemination model for large-scale wireless sensor networks,” in *Proceedings of ACM MobiCOM*, Los Angeles, CA, 2002.
- [3] V. R. Lesser, C. L. Ortiz, and M. Tambe, Eds., *Distributed Sensor Networks: A Multiagent Perspective*. Kluwer Academic Publishers, 2003.
- [4] R. Mailler, V. Lesser, and B. Horling, “Cooperative Negotiation for Soft Real-Time Distributed Resource Allocation,” in *Proceedings of Second International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2003)*. Melbourne, AUS: ACM Press, July 2003, pp. 576–583. [Online]. Available: <http://mas.cs.umass.edu/paper/241>
- [5] B. Horling, V. Lesser, R. Vincent, and T. Wagner, “The Soft Real-Time Agent Control Architecture,” *Proceedings of the AAAI/KDD/UAI-2002 Joint Workshop on Real-Time Decision Support and Diagnosis Systems*, July 2002. [Online]. Available: <http://mas.cs.umass.edu/paper/221>

- [6] F. Zhao, J. Shin, and J. Reich, "Information-driven dynamic sensor collaboration for tracking applications," *IEEE Signal Processing Magazine*, 2002.
- [7] D. Li, K. Wong, Y. Hu, and A. Sayeed, "Detection, classification, tracking of targets in micro-sensor networks," *IEEE Signal Processing Magazine*, pp. 17–29, March 2002.
- [8] R. Brooks, P. Ramanathan, , and A. Sayeed, "Distributed target classification and tracking in sensor networks," *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1163–1171, 2003.
- [9] M. Sims, C. Goldman, and V. Lesser, "Self-Organization through Bottom-up Coalition Formation," in *Proceedings of Second International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2003)*, vol. AAMAS03. Melbourne, AUS: ACM Press, July 2003, pp. 867–874. [Online]. Available: <http://mas.cs.umass.edu/paper/238>
- [10] K. S. Decker and V. R. Lesser, "Quantitative modeling of complex environments," *International Journal of Intelligent Systems in Accounting, Finance, and Management*, vol. 2, no. 4, pp. 215–234, Dec. 1993, special issue on "Mathematical and Computational Models of Organizations: Models and Characteristics of Agent Behavior".
- [11] T. Wagner, A. Garvey, and V. Lesser, "Criteria-Directed Heuristic Task Scheduling," *International Journal of Approximate Reasoning, Special Issue on Scheduling*, vol. 19, no. 1-2, pp. 91–118, 1998, a version also available as UMASS CS TR-97-59.
- [12] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara, "The distributed constraint satisfaction problem: Formalization and algorithms," *Knowledge and Data Engineering*, vol. 10, no. 5, pp. 673–685, 1998.
- [13] E. C. Freuder and R. J. Wallace, "Partial constraint satisfaction," *Artificial Intelligence*, vol. 58, no. 1–3, pp. 21–70, 1992.
- [14] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird, "Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems," *Artificial Intelligence*, vol. 58, no. 1-3, pp. 161–205, 1992.

A Figure Captions

Figure 1: High-level architecture. A: sectorization of the environment, B: distribution of the scan schedule, C: communicating over tracking measurements, and D: fusion of tracking data.

Figure 2: The sensor platforms have three Doppler radar sensor heads and are controlled by a BASIC stamp micro-controller.

Figure 3: Total number of messages for 36 agents tracking 4 targets given various sector sizes.

Figure 4: Communication load disparity among agents within each sector.

Figure 5: Overview of the agent’s organizational hierarchy, with some information flows represented.

Figure 6: An abstraction of the messages and reasoning used for target detection by sensor agents, sector and track managers.

Figure 7: As the target moves to another sector, the track manager queries the manager of the new sector.

Figure 8: The soft real-time control architecture.

Figure 9: An abbreviated view of the sensor initialization TÆMS task structure.

Figure 10: Utility of taking a single, coordinated measurement from a set of sensors.

Figure 11: Stage 2 of the SPAM protocol.

Figure 12: Example of a common contention for resources. Track manager T2 has just been assigned a target and contention is created for sensors S3, S4, S5 and S6.

Figure 13: A solution derived by SPAM to the problem in Figure 12. The table on the left is before track manager T2 mediates over T1 and T3. The table on the right is the result of stage 2.

Figure 14: Radsim environment with 8 sensor nodes (S1-S8) and 4 targets (M1-M4).

Figure 15: Average utility for a period for three allocation techniques under various conditions.

Figure 16: The effects of sector size on tracking quality with ideal communications.

Figure 17: The effects of sector size on various messages being communicated.

Figure 18: Researchers work on a demonstration involving 36 sensors and 3 mobile targets.

B Notes

B.1 Manuscript received.....

B.2 Author affiliations

- Roger Mailler, Department of Computer Science, University of Massachusetts, 140 Governors Drive, Amherst, MA 01003.
- Bryan Horling, Department of Computer Science, University of Massachusetts, 140 Governors Drive, Amherst, MA 01003.
- Victor Lesser, Department of Computer Science, University of Massachusetts, 140 Governors Drive, Amherst, MA 01003.
- Régis Vincent, SRI International, Room EJ274, 333 Ravenswood Avenue, Menlo Park, CA 94025-3493.

B.3 Sponsorship

Effort sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory Air Force Materiel Command, USAF, under agreements number F30602-99-2-0525 and DOD DABT63-99-1-0004. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. This material is also based upon work supported by the National Science Foundation under Grant No. IIS-9812755. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), Air Force Research Laboratory or the U.S. Government.

B.4 Footnotes

Footnote 1: A limited broadcast capability does exist, which can reach all sensors listening on a single channel. It is not possible in this architecture to broadcast a single message to agents which are using different channels.

FINAL TECHNICAL REPORT
Agreement No. F30602-99-2-0525
“Scalable Real-Time Negotiation Toolkit”

APPENDIX E

Self-Organization through Bottom-up Coalition Formation*

Mark Sims
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003 USA
msims@cs.umass.edu

Claudia V. Goldman
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003 USA
clag@cs.umass.edu

Victor Lesser
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003 USA
lesser@cs.umass.edu

ABSTRACT

We present a distributed approach to self-organization in a distributed sensor network. The agents in the system use a series of negotiations incrementally to form appropriate coalitions of sensor and processing resources.

Since the system is cooperative, we have developed a range of protocols that allow the agents to share meta-level information before they allocate resources. On one extreme the protocols are based on local utility computations, where each agent negotiates based on its local perspective. From there, a continuum of additional protocols exists in which agents base decisions on marginal social utility, the combination of an agent's marginal utility and that of others. We present a formal framework that allows us to quantify how social an agent can be in terms of the set of agents that are considered and how the choice of a certain level affects the decisions made by the agents and the global utility of the organization.

Our results show that by implementing social agents, we obtain an organization with a high global utility both when agents negotiate over complex contracts and when they negotiate over simple ones. The main difference between the two cases is mainly the rate of convergence. Our algorithm is incremental, and therefore the organization that evolves can adapt and stabilize as agents enter and leave the system.

*Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory Air Force Materiel Command, USAF, under agreement number #F30602-99-2-0525, by the National Science Foundation under Grant number DMI0122173, and in part by the Air Force Office of Scientific Research under Grant No. F49620-03-1-0090. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), Air Force Research Laboratory, NSF, Air Force Office of Scientific Research, or the U.S. Government.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AAMAS'03, July 14–18, 2003, Melbourne, Australia.

Copyright 2003 ACM 1-58113-683-8/03/0007 ...\$5.00.

Categories and Subject Descriptors

I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—*Multiagent Systems*

General Terms

Design, Experimentation, Performance, Reliability

Keywords

self-organizing systems, emergent organization, multi-agent systems, negotiation, coalition formation

1. INTRODUCTION

The process of self-organization in a large-scale, open system is of key importance to the performance of the system as a whole. An appropriate organization can limit control and communication costs, significantly improving performance. We have observed useful system performance with an organization of as few as sixteen agents [3]. A static organization, however, may not be able to handle a dynamic environment. Re-organization, therefore, is necessary during operation as agents and resources are removed or added, or when their characteristics change. In this paper we present a distributed, incremental approach to self-organization through bottom-up coalition formation that we have applied to the distributed sensor network (DSN) of the EW Challenge Problem [3]. The process uses negotiation iteratively to enable managers of coalitions to refine the set of coalitions in the system to achieve efficient allocations of sensors and adapt dynamically to environmental changes.

Horling et al. [3] describe the EW Challenge Problem domain in detail. It consists of homogeneous sensor agents distributed throughout a region. The agents are fixed and communicate using an eight-channel RF system in which each can use only one channel at a time. An organization in such a domain helps facilitate the efficient assignment of tracking tasks to particular agents and limit contention on communication channels. We employ a one-level hierarchy in which agents are distributively divided into sectors, each of which has a manager. The manager monitors what is currently being tracked by its sector and, as new data arises, determines whether it needs to assign a new tracking task to an agent in its sector. To do this the manager must model what is currently being tracked and the internal states of the agents in its sector. Furthermore, when it assigns tracking tasks, the manager attempts to minimize contention on any one channel. Therefore, the division of the agents into sectors helps to minimize not only the computational load on

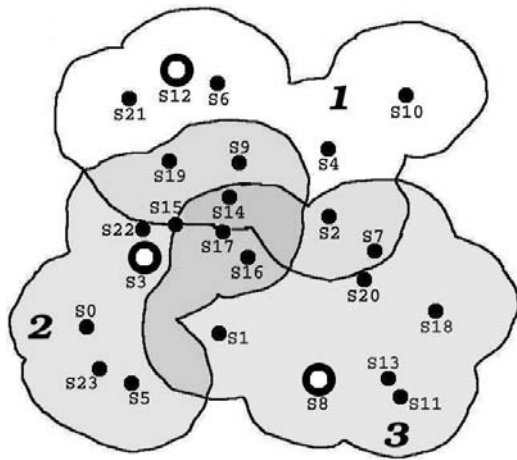


Figure 1: EWChallenge Domain

the managers but also the number of messages sent on any one channel. For our self-organization techniques, we assume all sector managers communicate with each other over channel zero and that the sector managers assign channels other than zero to agents as they enter the sector.

Figure 1 illustrates the domain. The empty circles are sector managers; the filled circles are sensor agents that are not managers. Although in actuality each agent has three separate sensor heads, for the purposes of this paper we assume agents have viewable areas of 360° . The sector areas represented by the “clouds” in the picture are defined by the intersection of the viewable areas of the sensors in the sectors. The areas of overlap show where the region covered by one sector intersects the region covered by another. Although sector boundaries overlap, each agent belongs to a single sector. To track a vehicle best, at least three sensor agents are required to triangulate the position of a vehicle moving through the region. If there are not three such sensors within the sector responsible for tracking, agents within that sector may request sensor data from other sectors.

We also assume that there is an overhead associated with passing a tracking task from one sector to another and that accessing sensor data of agents in different sectors may incur communication delays due to multiple hops or channel contention. It is desirable, therefore, for a sector to track well for as long as possible to minimize how often a tracking task is passed off to another sector, how often a tracking agent must access sensor data from different sectors, and how often tracking agents must negotiate over sensor allocation.

Given the need for an organization such as that described above, the motivation for applying self-organization techniques is the need to move from predefined, hand-generated configurations of sensors and organizational relationships as in [3] to arbitrary configurations and dynamic construction of organizational structure. To achieve this, we use a bottom-up coalition formation technique to enable the agents in the system to construct the organization dynamically in a decentralized manner.

Through coalition formation, agents in a large system faced with a set of tasks partition themselves to maximize system performance. By this process, the system moves from being a set of single agents to a set of either disjoint

or overlapping coalitions of agents. Our algorithm enables self-organization through coalition formation by having the agents discover their organizational relationships while partitioning themselves around the subtasks of a high-level task.

Our approach is similar to that of Shehory and Kraus [9, 10] in that it applies to a cooperative system of agents in a non-super-additive environment. We assume an overhead associated with each new member of a coalition and that a coalition reaches a point beyond which adding a member is no longer beneficial. Beyond these similarities our approach varies considerably from that of [9, 10]. In their work agents have a more global view of others in the system. The agents may not be aware of every other agent, but the assumption is that they know of a large number. Each agent then calculates a subset of the coalitions it may belong to, and the system engages in a greedy process of choosing coalitions based on their computed coalitional values. If the population changes, the coalition formation process must restart. In contrast, our approach is an incremental, local one in which agents need not know of that many other agents around them and the process of coalition formation can continue and adapt if the population changes. Another difference is that Shehory and Kraus [10] allow for overlapping coalitions. In our current work, we restrict our attention to disjoint coalitions although in future work, we plan to extend our techniques to overlapping coalitions where sensor agents may have membership in more than one sector.

Two other sets of related work are [2] and [4]. While the work of Horling et al. [2] does involve a local adaptation process, it uses evaluations of system performance to adapt an organization. Organizational adaptation in the work of Ishida et al. [4] is based on the tasks that enter the system and the system’s current load. It is not an iterative search process designed to converge on a good organization. In both sets of work the adaptations may be revised as the situation changes, but the process of adapting is a single shot. The Contract Net Protocol [11] is also related to our work; we discuss it in Sections 3 and 4.

Finally, a common coalition formation problem related to the distribution of tasks asks: Given a fixed set of tasks and a set of cooperative agents, how can we pick groups of agents best suited to those tasks? Our problem is slightly different and resembles the work in Goldman and Rosenschein [1] aimed at partitioning information domains to facilitate future information retrieval requests. In our DSN, the system is given the high-level task of providing coverage for a region. This task encompasses the future tracking tasks that the system will perform but does not know *a priori*. The goal is to subdivide the region and assign portions of it to sectors (coalitions) of agents so that each sector is best able to perform the tracking tasks that it encounters.

Sections 2 and 3 present our model. Section 4 describes empirical results from testing different negotiation protocols that lead to different stable organizations. Section 4, also analyzes the performance of the organizations evolved in terms of their rate of convergence, fault tolerance, and message traffic while tracking. We conclude in Section 6 after presenting a formal framework in Section 5 that allows us to analyze the decisions that the agents make as a function of the value of the information they hold. We distinguish between local information agents and k -social agents who may be able to obtain information about k other transactions happening at the same time.

2. PROBLEM DESCRIPTION

In order to formalize our problem, we present the following assumptions and definitions.

2.1 Assumptions

In addition to the assumptions stated in the Introduction, we make the following assumptions:

- Although agents are arbitrarily distributed, there are a sufficient number of them and they are arranged such that every point in the region assigned to the system has at least one sensor that can see it.
- Although agents may enter or leave the system at any time, the agent population does not vary dramatically from one instance to the next. If the population were to fluctuate wildly, attempting to build organizational structure would be futile.
- An ideal sector has between eight and ten agents in it and at least three agents can see every point in the region it is responsible for. As the first assumption suggests, this is not always possible.

2.2 Definitions

At any time t we have a set of agents in the system, $A = \{A_1, A_2, \dots, A_n\}$, and each agent A_i has a vector of capabilities $B = \{b_1^i, b_2^i, \dots, b_n^i\}$. For example, in the sensor domain presented, each agent controls the sensor associated with it and the capabilities are the regions each is able to cover.

Different organizations can result from a given set A of agents with their corresponding capabilities. These organizations are instantiations of different organizational templates. An organizational template is given by the organization's high-level goals, the roles that exist within the organization, the organization's control and communication hierarchies, and its evaluation function. Our implementation instantiates a simple template in which agent organizations are built using a single level hierarchy.

A sector S_i is a coalition of agents drawn from A that work together to accomplish a task. A sector manager SM is a representative of its sector that is responsible for handling negotiations with other sectors (as well as task and channel allocation within the sector). The manager may not remain constant throughout the life of the sector. Since the agents in A are homogeneous, any agent can serve as a sector manager.

Each sector S_i has an area defined by the viewable areas of the sensor agents that it is responsible for as shown by the "clouds" in Figure 1. We denote this as $Area_{S_i}$. Each sector has a utility value U_{S_i} that is a function of the number of agents in the sector and how well the sector can provide coverage of the sub-region it is responsible for. More specifically, let AN_i be the average number of sensors in S_i that can see each point in the region covered by S_i . Let NU_i be a function ranging between 0 and 1 dependent on the number of agents in S_i . Space limitations preclude a complete description of NU_i , but it is an empirically defined factor that equals 1 if S_i has eight sensors, falls off slowly at first so that between 6 and 10 agents still gets a fairly high rating, and then falls off quickly for sectors of other sizes. A sector with only one agent, for instance, has $NU_i = 0.001$. Finally, $U_{S_i} = AN_i \times NU_i$.

The coalition formation process results in a set of sectors called a coalition structure [6] $CS = \{S_1, S_2, \dots, S_m\}$ where S_i is the i^{th} sector in CS . A coalition structure's global utility is the sum of the utilities of the individual sectors in it:

$$U_{CS} = \sum_{i \in CS} U_{S_i}$$

The coalition formation process decomposes a high-level task T assigned to the system into subtasks $\{t_1, t_2, \dots, t_m\}$ which may overlap and are assigned to the different sectors. In the sensor network, two coverage subtasks overlap if $Area_{S_i} \cap Area_{S_j} \neq \emptyset$. Figure 1 shows that all three sectors illustrated have areas of overlap.

Each agent is able to perform a portion of the subtask assigned to its sector based on its capabilities. In the sensor network example, an agent is able to provide partial coverage of the region its sector is responsible for.

With the assumptions and definitions above, we can formulate the self-organization problem as follows: Given a high-level task T^1 and a set of agents A , subdivide T into m subtasks $\{t_1, t_2, \dots, t_m\}$ and A into a coalition structure $CS = \{S_1, S_2, \dots, S_m\}$ of m sectors such that each of the subtasks is assigned to one sector, where $\bigcup S_i = A$, $\forall i \neq j$ $S_i \cap S_j = \emptyset$, and U_{CS} is maximal.

2.3 Market Analogy Definitions

Because our approach involves an iterative negotiation process, comparing the system to a marketplace is useful. A **buyer** is a sector manager whose sector does not have the necessary sensors to perform its subtask adequately. In other words, the sensor agents that comprise the sector do not provide sufficient coverage of the area for which the sector is responsible. A **seller** is a sector manager whose sector has sensor agents able to provide coverage of a region the buyer would like to cover. Sector managers can be buyers and sellers simultaneously. The only agents involved in the negotiations of this marketplace are sector managers.

The **product** in the sensor network is the ability to provide coverage for a certain region and is transferred from one sector manager to another through the exchange of sensor agents between sectors. The product is the resource the buyer needs to improve its performance of its subtask. Finally, the **value** of a product to a buyer or seller is a function of the buyer's and the seller's marginal utility gains from the transaction and depends on the negotiation strategy they are using. When determining with whom to transact, buyers and sellers may consider either their own local marginal utility gains or the social marginal utility. The **local marginal utility** is the difference between a sector's utility before a transaction and the utility after the transaction. The **social marginal utility** is the sum of the local marginal utilities of both the buyer and the seller. In the local case, the buyer and seller value products differently. In the social case, they value products the same.²

With this analogy, the problem of bottom-up coalition formation translates into deciding which sellers the buyers

¹In our case the high-level task is to provide coverage for the entire region.

²Although the analogy to a marketplace is useful, it is worth noting that our system is indeed cooperative and, therefore, agents may be willing to negotiate at the social level. This is not reasonable in a competitive market.

should attempt to buy from and which buyers the sellers should sell their products to such that U_{CS} is maximized.

3. NEGOTIATION STRATEGIES

A well-known strategy for assigning resources that has also been used to organize a DSN is the Contract Net Protocol (CNET) [11]. CNET provides a general framework to describe negotiation processes between agents. In its original version it involved agents' making decisions based on each agent's own perspective. For the DSN domain, an example of how CNET enables agents to build an organization is as follows: A task manager with a task to be fulfilled (such as finding a sensor agent to provide signal data) broadcasts a task announcement with a deadline for receiving bids. Just before the deadline, agents capable of performing the task send their bids to the manager who then evaluates the bids and awards contracts appropriately. Once an agent receives a contract, that agent is committed to it.

In our example, many agents may be able to provide coverage for the same area, but assigning the task to different agents may lead to different global utilities. In CNET each task that is assigned by a task manager (at its highest abstraction level)³ was assumed to be independent of other tasks, so that the order of processing tasks by different task managers did not affect the global utility of the system.

We are interested in evaluating the performance of the whole organization in terms of the agents' decisions and the structure that results from these decisions. We assume that all agents are interested in maximizing the global utility of the system and, therefore, require a negotiation protocol to enable this. CNET in its original formulation is not sufficient for this purpose. For example, assume agents A_1 and A_2 are both able to cover a region that sector manager SM_1 needs covered, but only A_2 is able to cover a region SM_2 needs covered. If SM_1 awards a contract to A_2 , A_2 may no longer be available to SM_2 .

In order to correct for the above problem, we have developed two general classes of negotiation protocol for self-organizing through coalition formation in the marketplace of Section 2.3: local marginal utility based and social marginal utility based.⁴ In our case, because agents negotiate, even if A_2 initially joins SM_1 , SM_1 and SM_2 may be able to adjust the allocation of sensors to coalitions such that A_2 moves to SM_2 and A_1 joins SM_1 .

For an illustration of the dynamics of the protocols we have developed, refer to Figure 2. In the local marginal utility based protocols, a round of negotiation proceeds as follows: A buyer broadcasts a message (1) requesting coverage of a region. Each manager within range, who has an agent that can cover that region and whose local marginal utility of giving up the agent is positive, responds with a message (2) stating that it could provide coverage to the buyer. Unlike CNET, the seller is not bound to honor this offer. The seller is free to make offers to as many buyers that send requests as it likes.

The buyer waits for a period of time, collecting responses from sellers. When the period is over, the buyer selects

³This task may be sub-contracted and its sub-parts are indeed dependent.

⁴The idea of negotiating over marginal utility is similar to the TRACONET [8] extension of CNET in which bidding and awarding decisions are based on marginal cost calculations.

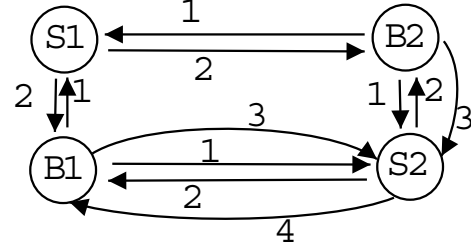


Figure 2: Negotiation Message Types

1. Buyers request. 2. Seller's potential utility change. 3. Buyer chooses seller. 4. Seller chooses buyer.

the seller whose product would provide the buyer with the greatest local marginal utility gain and sends a message (3) to that seller requesting the coverage offered.

Finally, given the multiple responses from buyers that the seller receives, the seller chooses to give its product to the buyer that maximizes the *seller's* local utility (4).

Negotiation in the social case is slightly different. As in the local case, the buyer sends its product request (1). This time, however, the seller responds with an offer even if its change in local utility would be negative and reports to the buyer what its local utility change would be (2). The buyer collects responses from sellers and chooses to request the product it needs from the seller that maximizes the sum of the buyer's local marginal utility and the seller's local marginal utility assuming the sum is positive. The buyer reports the sum (3) to the seller (i.e., the buyer requests the coverage offered).

Given the product requests the seller receives, it chooses to give the product to the buyer that reported the highest social marginal utility to it (4). Although the social marginal utility gain will be positive, the seller's or even the buyer's (but not both) local change in utility may be negative. In Figure 2 both Buyer 1 and Buyer 2 accept the offers Seller 2 made to them. Seller 2 then chooses (as seen by message (4) in the Figure) to give the product to Buyer 1 because the social marginal utility reported by Buyer 1 was higher than that reported by Buyer 2.

In addition to negotiation, we assume that a discovery process occurs when an agent enters the system; it must learn of the other agents near it and they must learn of it. To make this happen quickly, an entering agent joins the nearest sector to it, by listening for beacons on channel zero. If there is no sector within range, the agent elects itself manager of a new sector and begins attracting entering agents to it by broadcasting a periodic beacon on channel zero. It also starts negotiating with other managers over the resources it needs to perform its subtask.

We also assume that a maintenance process takes place throughout the life of the system. Sector managers must make sure that the members of their sectors still exist, and members must make sure that their managers still exist. In our approach each member of a sector periodically sends a brief message to its manager on the manager's channel. If the manager does not receive a message from a member, the manager assumes the agent is no longer a member of the

coalition and adjusts its evaluation of the coalition accordingly. Likewise, the manager periodically sends a message to each of its members on the channel the member uses. If the member does not receive a message from its manager, that member assumes the manager is no longer active as a manager and joins the nearest coalition to it (as if it were entering the system for the first time).

4. RESULTS

To examine the above classes of protocols on large numbers of agents, we built an asynchronous simulation testbed for the EW Challenge [3] DSN. One limitation of the simulation is that it does not model delays due to computation time. In order to deal with this, it has the ability to add random delays to messages that are sent. In future work we plan to explore how increased delays affect overall system performance.

Note that while we would like to compare our results to optimal, the sizes of the configurations tested in this work are too large to generate optimal values.

4.1 Organization Results

To compare the performance of local and social utility based negotiation mechanisms, we varied factors such as when agents can initiate and respond to requests, whether sellers can initiate negotiations by advertising coverage, and how many agents a seller can transfer to a buyer during a negotiation. For each variation, we compared our results to those generated by CNET. In the CNET adapted for our domain, a buyer broadcasts a request. The seller collects requests and responds with an offer that the seller is obliged to fulfill if the buyer accepts it. This differs from our protocols. Since we are dealing with cooperative agents whose priority is the welfare of the system, a seller need not honor an offer. In other words, a CNET agent will respond only to a single request, while an agent that uses our protocols may respond to several requests simultaneously.

In total we tested fourteen protocols. We ran 100 experiments each on 40, 70, and 90 node configurations in 100 x 100 foot regions populated by agents with viewable sensor regions with 20 foot radii. For a given number of nodes, we generated an arbitrary configuration and then ran each of the 14 protocols on that configuration. By far the best performing protocols were those that were socially based.

Because of space limitations, we include results from six protocols with the following characteristics:

- **Single-Node Social Protocol (SNSoc):** Only single nodes are transferred per negotiation cycle. Sector managers are simultaneously buyers and sellers. Sellers advertise regions of coverage they are willing to give up. Value is based on social marginal utility.
- **Multiple-Node Social Protocol (MNSoc):** Same as above, but either one or two nodes may be transferred per negotiation cycles.
- **CNET Single Social Protocol (CNETSoc):** Socially based CNET with single node transfer.
- **Single-Node Local Protocol (SNLoc):** Same as SNSoc except that value is based on local marginal utility.

	Single		Multiple		CNet	
	local	social	local	social	local	social
40 nodes						
% ΔU_{CS}	6.5	61.2	6.3	60.6	2.1	15.6
Cycles	3	18	2	13	1	3
70 nodes						
% ΔU_{CS}	24.6	50.0	23.6	47.8	14.7	14.7
Cycles	7	22	5	18	2	2
90 nodes						
% ΔU_{CS}	44.2	70.9	42.7	67.6	36.7	39.7
Cycles	7	24	5	15	3	4.3

Table 1: 40, 70 and 90 Node Configurations

- **Multiple-Node Local Protocol (MNLoc):** Same as MNSoc except that value is based on local marginal utility.
- **CNET Single-Node Local Protocol (CNETLoc):** Same as CNETSoc, but locally based.

Table 1 summarizes the results for the protocols above for 40, 70, and 90 node configurations. They show the average percent change in global utility % ΔU_{CS} from the initial state to a stable state and the approximate number of negotiation cycles required to reach the stable state. In this context the initial state is the set of rough sectors immediately after the discovery phase. The stable state occurs when agents are no longer able to engage in successful negotiations.

Of all the protocols SNSoc and MNSoc performed best. It makes sense that these would perform better than the locally based protocols because of their increased social context. We were surprised to find, however, that MNSoc achieved slightly lower global utility than SNSoc did since Sandholm [5] suggests that a contract over multiple objects can help avoid local maxima that result from single object contracts. One possible explanation is that transferring more than one agent in a single transfer causes the system to become stable more quickly. As a result, the system falls into local maxima more often than it does when only transfers of single agents are allowed. For example, if a sector manager gives up two nodes to another at time t , then the set of possible actions that the same sector manager can take at time $t + 1$ is reduced, and it may not be able to make a socially beneficial transfer that was unknown at time t . This conclusion is supported by the fact that the average number of negotiation cycles required to reach a stable state when MNSoc is used is less than the number when SNSoc is used. The conclusions in [5] consider a non-cooperative multi-agent system. In our cooperative organizations, the need for larger contracts is lessened by a more informative, social utility function.

While SNSoc does ultimately achieve higher global utility, it is interesting to note that early on in the self-organization process, MNSoc actually achieves higher utility. Figure 3 compares the average utility profiles of MNSoc and SNSoc for 90 node configurations. It gives the percentage of the final maximum utility achieved by SNSoc versus the average number of negotiation cycles. It shows that before the first 14 negotiation cycles, MNSoc does better. Only after this point does SNSoc pull ahead. The profile suggests a way of choosing a negotiation protocol and limiting how long the negotiation phase lasts. For instance, if the organization must form quickly due to high communication costs or other

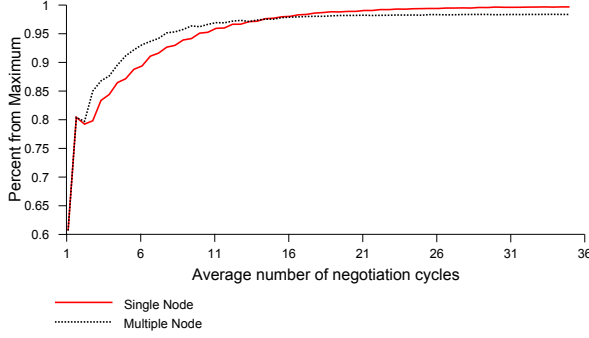


Figure 3: Utility profiles for SNSoc and MNSoc 90 node configurations.

constraints and achieving only 95% of the maximum utility is acceptable, then it is better to use MNSoc since it will reach this level of utility more quickly.

Also, evident in the graph is that in both cases, most of the utility increase occurs early on in the self-organization phase. This is corroborated by other results not shown here that show that the number of transfers of agents from one sector to another is high early on and then falls off rapidly.

Also of interest is that the fewer nodes in a configuration there are, the greater the difference is between the socially based protocols and the locally based ones (see Table 1). One explanation is that when there are many nodes in a fixed space, it is easier for these nodes to partition themselves to cover a given region. Thus, individual negotiation decisions in a dense region do not have as great of an effect on the ultimate social utility of the configuration as they do in less dense regions. While the more informed decisions possible through the socially-based utility functions certainly produce large improvements in utility in dense regions, their greatest impact is seen in less dense regions.

4.2 Fault Tolerance and Message Traffic

A DSN must be able to reorganize itself after several of its nodes go down. An additional concern of ours was that the maintenance process as described in Section 3 would hinder reorganization since if a sector manager fails, the nodes in its sector simply join other sectors near them rather than try to maintain the degraded sector. Therefore, we implemented a second maintenance scheme whereby if a manager fails, another node in its sector takes over its role.

To examine how well both mechanisms respond to node-failure, we performed the following experiment for SNSoc and MNSoc. For 100 different configurations we let 90 nodes organize until they reached a stable organization. Then starting from that stable state, we removed 10 nodes at random and let the system reorganize once using the original maintenance mechanism, once using the new one. Finally, using the remaining 80 nodes as a starting configuration, we let those nodes organize from scratch. We repeated this process three more times, each time starting from the stable 90-node organization, removing the same set of nodes as in the previous experiment plus an additional 10.

In all cases MNSoc organized more quickly than SNSoc with only minor differences in final utility. Figure 4 shows

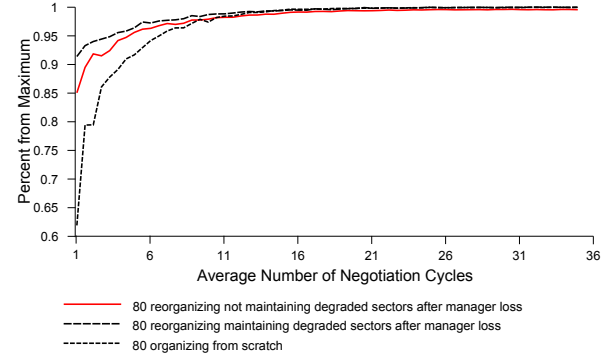


Figure 4: MNSoc utility profiles for 80 nodes reorganizing two ways plus 80 nodes organizing from scratch

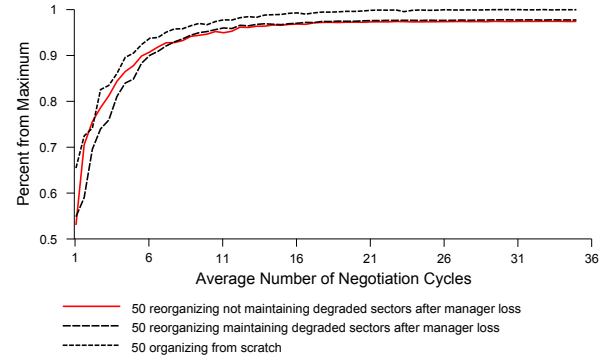


Figure 5: MNSoc utility profiles for 50 nodes reorganizing two ways plus 50 nodes organizing from scratch

the utility profiles for MNSoc when 10 nodes fail. The graph for SNSoc is similar. When only 10 nodes go down, maintaining degraded sectors enables the system to reorganize most efficiently. In other words, since losing only a small number of nodes does not perturb the structure of the stable organization much, it is best to work within that structure and make only minor adjustments after node failure.

The above does not hold as more nodes go down. When 20 nodes fail, the two mechanisms have almost identical profiles. When 30 and 40 nodes go down, SNSoc reorganizes most quickly with the original maintenance mechanism, followed closely by organizing from scratch. With MNSoc organizing from scratch actually does better than reorganizing with either maintenance mechanism; in fact it achieves better utility than the other two options (see Figure 5). When large numbers fail, trying to maintain a previous structure hinders reorganizing and prevents the system from finding a globally beneficial solution.

The final check of our self-organization procedure was to verify that it keeps inter-sector communication low. We built a simple message traffic model of the domain and tested it on the stable 70-node organizations. The model showed that on average approximately 85% of messages occur within

sectors, indicating that our algorithm successfully divides the agents such that most messages indeed occur within sectors.

5. THEORETICAL MODEL

In addition to experimentation, we developed a theoretical model of the negotiation process that builds on the idea that increased social context can improve system performance.

Because our system is cooperative, we assume that agents can share information, although the process of obtaining this information may be costly. In a DSN, decisions about which agent covers which area are affected by the interdependencies that exist between the agents.

DEFINITION 1 (INTERDEPENDENCY). *Given that sector managers SM_i and SM_j are responsible for tracking tasks in sectors S_i and S_j , we define an interdependency between these two sector managers if $Area_{S_i} \cap Area_{S_j} \neq \emptyset$*

A chain of interdependencies is given by an ordered list of sector managers $SM_{i_1}, SM_{i_2}, \dots, SM_{i_{n-1}}, SM_{i_n}$ such that $Area_{S_{i_1}} \cap Area_{S_{i_2}} \neq \emptyset \dots Area_{S_{i_{n-1}}} \cap Area_{S_{i_n}} \neq \emptyset$. We denote by n the maximal length of a possible chain of interdependencies in a given system.⁵

We distinguish agents based on their information horizon, the amount of information they can gather given by the length of an interdependencies' chain. We define an agent that knows the information in a chain of k interdependencies as follows:

DEFINITION 2 (K-SOCIAL, $k \leq n$). *Agent A is **k-social** if its decision about the action it will perform is based on information known by each agent in a chain of interdependencies of length k whose first element is A .⁶*

To explain the process in which an agent (a buyer or a seller) must decide which offer to accept or to whom it should sell a resource, we refer the reader to Figure 6.

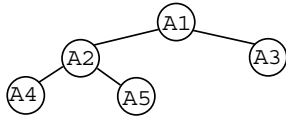


Figure 6: A Decision Tree example for $n = 3$.

We first build a tree describing the chain of interdependencies⁷. Each node represents a sector manager, and each edge represents an interdependency. In the figure, the numbers of each node represent the sector managers' names. The depth of the tree is the length of the interdependencies chain. Assume that $n = 3$. The root is at level $k = 0$, and the agent at the root is the one making a decision.

In Figure 6, agent $A1$ needs either to decide to whom to sell a resource that $A1$ currently owns, or it must decide

⁵In general, an interdependency exists if the set of resources needed by two agents making a decision intersect.

⁶In the setting analyzed so far, the agent referred to in this definition corresponds to the sector manager.

⁷The interdependencies may actually be represented by a graph, because there may be cycles of interdependencies among the resources. So this tree is a mapping from this graph to a tree, i.e., whenever a node that has already appeared in the tree needs to appear again it is set to a leaf.

from whom to buy a resource it needs. We denote by Δ_{ij} the change in i 's utility caused by agent j 's selling to or buying from agent i . If $A1$ is local ($k = 0$) it does not consider the other agents, and by comparing Δ_{12} to Δ_{13} chooses to interact with either $A2$ or $A3$ based which would produce the greater change in its own utility. This is analogous to the locally based protocols described in Sections 3 and 4.

If $A1$ is $k = 1$ social, it compares $\Delta_{12} + \Delta_{21}$ to $\Delta_{13} + \Delta_{31}$. In this case, $A2$ and $A3$ may be interacting simultaneously with other agents, but since $A1$ sees only up to horizon 1, it is unaware of any pending decisions further down the interdependency chain. So $A1$'s decision may be wrong. For example, if $A1$ accepts the transaction with $A2$ based on the above comparison, it may lose, if $A2$ chooses $A4$ instead of $A1$. We have not implemented a protocol in which the agents are $k = 1$ social. If we had, it would proceed as follows: 1) A buyer broadcasts a request for coverage. 2) A seller responds with an offer. 3) The buyer does not wait to collect responses from sellers; it simply responds to the seller, telling it what the buyer's local change in utility would be if it were to receive the seller's offer. 4) The seller ranks all responses it receives based on the sum of its local marginal utility and the reported buyers' utilities. The seller chooses the buyer that gives the highest sum.

If $A1$ is $k = 2$ social, it knows whether $A2$ is negotiating with $A4$ and $A5$ while $A2$ is negotiating with $A1$. Here $A1$ decides by computing $\max\{\Delta_{21} + \Delta_{12}, \max\{\Delta_{24} + \Delta_{42}, \Delta_{25} + \Delta_{52}\}\}$.

If the "winner" of this maximum is $\Delta_{21} + \Delta_{12}$, sector manager $A2$ will **not** transact with $A4$ or $A5$. Therefore, sector manager $A1$ to make his decision compares $\Delta_{21} + \Delta_{12}$ to $\Delta_{13} + \Delta_{31}$ and chooses $A2$ or $A3$ accordingly. If the "winner" of the above maximum is $\Delta_{24} + \Delta_{42}$ or $\Delta_{25} + \Delta_{52}$, then the value for $\max\{\Delta_{21} + \Delta_{12}, \max\{\Delta_{24} + \Delta_{42}, \Delta_{25} + \Delta_{52}\}\}$ is set to $-\infty$, so that $A1$ does not take $A2$ as an option in its decision because from its perspective $A2$ will accept the transaction with either $A4$ or $A5$ and not with $A1$; in this case $A1$ chooses $A3$.

The $k = 2$ social case is analogous to the social marginal utility protocols we have developed. In those protocols, the seller at the root does not actually do all of the calculations described above. Rather, parts of the calculation are done further down the tree and propagated up. For example, if $A1$ in Figure 6 is a seller, $A2$ is a buyer and calculates $\max\{\Delta_{21} + \Delta_{12}, \max\{\Delta_{24} + \Delta_{42}, \Delta_{25} + \Delta_{52}\}\}$. If the "winner" is $\Delta_{21} + \Delta_{12}$, $A2$ propagates this value up to $A1$. Otherwise, it does not, and $A1$ knows only to consider negotiating with $A3$.

Notice that Sandholm and Lesser [7] assume self-interested agents, which are necessarily $k = 0$ social⁸. In their case, agents must transact on complex deals in order to approximate the maximal utility. We take advantage of the cooperativeness of the system by allowing the agents to be social and, thus, obtain better deals in terms of the complete system without the need to transact over more complex deals.

Results obtained from our simulations show that social agents attain higher utilities than local agents and that in configurations with few agents, the difference between the organizational utility obtained by social agents and that ob-

⁸In this paper, we are assuming that communication between the cooperative agents is free. It is not reasonable to assume that self-interested agents will exchange this information for free, although they may benefit from it.

tained by local agents is greater than it is in configurations with many agents. We can understand these results from the theoretical model. We denote by $P(s_i)$ the probability of sector i 's manager's making the "correct" decision. If sector i 's manager chooses to transact with agent j and j also chooses to transact with agent i , then we say that agent i has made the right choice. Therefore, $P(s_i)$ is a conditional probability that the root of the interdependency tree has made the right decision. This probability is conditioned on the decisions of the other nodes in the tree. The more social the sector manager is, the more accurate $P(s_i)$ is. If the sector manager knew all the information in the complete tree then $P(s_i)$ would be 1 (in the decentralized version of the problem⁹). Hence, the result we obtained that being social is better than being local is supported by the theoretical model. The utility of the organization is higher as long as the value of $P(s_i)$ increases, and this value increases as long as the sector manager considers larger values of k (i.e., the agents are more social).

The result that being social in sparsely populated configurations has a greater effect than being social in a dense configuration is also supported by the model. In such a setting, because there are fewer interdependencies, $P(s_i)$ will be greater than in a dense configuration with many interdependencies. Therefore, the difference in utility will be greater as well.

6. CONCLUSIONS AND FUTURE WORK

This paper presents an incremental approach to self-organization based on bottom-up coalition formation. Agents negotiate to maximize the system's global utility by using a variety of protocols based on local or social marginal utility.

Our approach is novel in the sense that it allows for different levels of social agents to be tested. Our protocols can represent a continuum of agents from locally-oriented to fully-informed. Empirical results show that social agents do attain higher utilities than locally-based or CNET-based agents do. In other words, although the system achieves a stable organization in all the cases tested, negotiating with social awareness in an incremental fashion avoids many of the local maxima of non-social utility based negotiations. We also show that the organizations obtained are robust to agent failure; the agents do indeed reorganize after some number of them are deactivated, and as long as the number of nodes that fail is not so great as to obliterate the structure already in place, reorganizing to a stable state happens more quickly than simply organizing from scratch and achieves similar utility values.

Future work will look at other types of organization templates. We will study more complex topologies, such as organizations based on hierarchies with multiple levels which may require various communication models for the exchange of information at the different levels. Another cost model worth studying involves the computation of the global utility resulting from agents' negotiating based on combinations of local and social marginal utilities. Adding explicitly the cost of sending a message (e.g., given by delays) and analyzing the trade-off faced by agents between obtaining more accurate information and the time it may take to gather it deserves more research as well. In this work, we have as-

sumed that all the agents are homogeneous. Further work will look at systems where the sector managers require certain computational capabilities that only some of the agents have.

7. REFERENCES

- [1] C. V. Goldman and J. S. Rosenschein. Partitioned multiagent systems in information oriented domains. In O. Etzioni, J. P. Müller, and J. M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 32–39, Seattle, WA, USA, 1999. ACM Press.
- [2] B. Horling, B. Benyo, and V. Lesser. Using self-diagnosis to adapt organizational structures. In J. P. Müller, E. Andre, S. Sen, and C. Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 529–536, Montreal, Canada, 2001. ACM Press.
- [3] B. Horling, R. Vincent, R. Mailler, J. Shen, R. Becker, K. Rawlins, and V. Lesser. Distributed sensor network for real time tracking. In *Proceedings of the 5th International Conference on Autonomous Agents*, pages 417–424, Montreal, June 2001. ACM Press.
- [4] T. Ishida, M. Yokoo, and L. Gasser. An organizational approach to adaptive production systems. In *National Conference on Artificial Intelligence*, pages 52–58, 1990.
- [5] T. Sandholm. Contract types for satisficing task allocation: I theoretical results. In *AAAI 1998 Spring Symposium: Satisficing Models, 1998.*, pages 68–75, Stanford University, CA, March 1998.
- [6] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohme. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1-2):209–238, 1999.
- [7] T. Sandholm and V. Lesser. Issues in automated negotiation and electronic commerce: Extending the contract net framework. In V. Lesser, editor, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, pages 328–335, San Francisco, CA, USA, 1995. The MIT Press: Cambridge, MA, USA.
- [8] T. W. Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, pages 295–308, Hidden Valley, Pennsylvania, 1993.
- [9] O. Shehory and S. Kraus. Task allocation via coalition formation among autonomous agents. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 655–661, Montréal, Québec, Canada, 1995.
- [10] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200, 1998.
- [11] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transaction on Computers*, C-29(12):1104–1113, 1980.

⁹The optimal centralized organization may still be different from the optimal decentralized version.

FINAL TECHNICAL REPORT
Agreement No. F30602-99-2-0525
“Scalable Real-Time Negotiation Toolkit”

APPENDIX F

Separating Domain and Coordination Knowledge in Multi-Agent Organizational Design and Instantiation

Mark Sims,* Daniel Corkill, and Victor Lesser
University of Massachusetts
Multi-Agent Systems Laboratory
{msims,corkill,lesser}@cs.umass.edu

Abstract

Organizational design and instantiation is the process that accepts a set of organizational goals, performance requirements, agents, and resources and assigns responsibilities and roles to each agent. We present a prescriptive, knowledge-based organizational design and instantiation process for multi-agent systems. An important aspect of our approach is the separation of application-specific organizational knowledge from more generic organizational coordination mechanisms. We describe our model of organizational design and our search process. We also present example organizations generated by our automated system for the distributed sensor network domain for different environmental characteristics and performance requirements.

1. Introduction

The ability to create and maintain effective multi-agent organizations is key to the development of larger and more diverse multi-agent systems. Organizational control is a multilevel control approach in which long-term organizational goals, roles, and responsibilities are developed and maintained to serve as guidelines for making detailed operational control decisions by the individual agents. These organizational guidelines reduce the complexity of each agent's operational decision making, lower the cost of distributed resource allocation and agent coordination, help limit inappropriate agent behavior, and reduce communication requirements [2]. Designed organizations are created by applying organization-design knowledge, organizational goals and performance requirements, and task-environment information to generate explicit organizational responsibilities that are then elaborated by the individual agents into appropriate operational behaviors.

To date, multi-agent organizational structures used for control have been manually hand-crafted, sometimes assisted by automated template expansion [12] or computed adjustments made to a pre-determined structure [11]. In this paper, we describe recent work on developing an automated organizational design and instantiation system that is able to create appropriate, yet substantially different, organizational forms based on different requirements and task-environment expectations. One important aspect of our approach is **the separation of application-specific organizational knowledge from more generic organizational coordination mechanisms**. This separation will allow the reuse of organizational coordination mechanisms across a wide range of problem domains and environmental situations.

The multi-agent organizational design and instantiation problem can be summarized as follows. Given a problem-domain description of the organizational goals, environmental conditions, performance requirements, possible roles, agents, and resources, assign both problem-domain and coordination roles and responsibilities to each agent such that the organizational performance requirements are satisfied and the organization operates effectively over anticipated environmental conditions. This assignment constitutes the organizational structure. To solve this problem in an automated fashion, we have developed the prescriptive, knowledge-based design process illustrated in Figure 1, which we describe in detail in Section 2.

Before continuing, it is important to clarify the distinction between organizational and operational control and where our work fits into it. Organizational roles and responsibilities represent general long-term guidelines. Operational control, on the other hand, involves specific short-term agreements among agents to perform specific activities for specific periods of time. Our process does not pertain to operational activities. Rather than describe how particular control decisions are made, it ensures that sufficient resources and coordination mechanisms are available so that

* The first author is a student

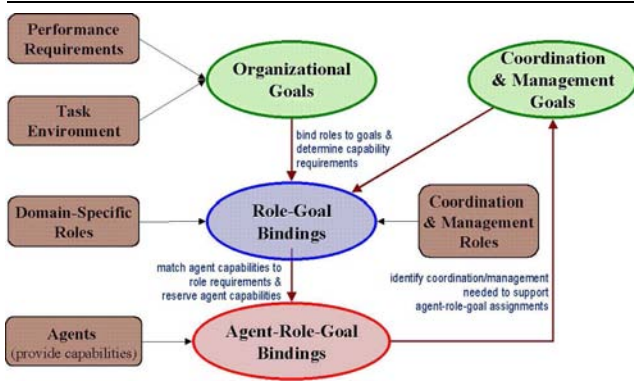


Figure 1. Organizational Design Process

agents can make efficient operational decisions throughout the life of the organization.

As mentioned above, our approach makes use of a separation we have observed between the problem-domain and organizational coordination. The former, shown on the left side of Figure 1, involves decomposing high-level organizational goals and matching them to problem-domain roles. The latter, shown on the right, pertains to the coordination mechanisms used by the agents performing those roles. The result is a set of bindings for each agent to both problem-domain specific and coordination-specific roles as illustrated in Figure 4(c).

As an example, consider a simple distributed sensor network (DSN) application. A problem-domain organizational goal might be to track all vehicles moving within a monitored area with a positional accuracy within 3 meters and a detection delay of at most 3 seconds. The environmental model indicates the expected traffic volume, spatial density, arrival rate, and movement characteristics. The available roles might be radar-based scanning and data processing which are clearly appropriate for a variety of scenarios. The best way to coordinate the agents playing the roles, however, is dependent on a number of factors. For example, if the area to be scanned is small enough that only a few agents are necessary, a peer-to-peer mechanism may be the right choice. If many agents are required, new vehicles arrive frequently, and the scanning resources are scarce, a multi-level hierarchical structure may be more appropriate.

It is our intuition that organizational coordination knowledge transcends the problem domain. Therefore, an automated system can include generic coordination knowledge, requiring the developer to supply information about the problem domain only. The system itself can then use both sets of knowledge in determining an appropriate organizational structure for the agents. Separating problem domain knowledge from coordination knowledge contributes to the field of organizational design in that it allows us to take a prescriptive, knowledge based approach to organizational

design and instantiation that does not pre-specify coordination mechanisms.

Past work in multi-agent organizational design has been purely descriptive, such as the organizational ontology of Fox, et. al. [4], or has used predetermined organizational forms as in Pattison, et. al. [12]. In our work only the problem domain features need to be specified; organizational structures are found based on domain-independent coordination knowledge. So and Durfee's work [11, 10] is the closest to ours in that they have a model based on the task environment, organizational structure, and performance metrics and explore the question of how to choose the best organizational structure for a given problem. However, they assume a hierarchical structure and are primarily concerned with making span of control decisions within it.

Still other multi-agent work deals with coordinating agent activity but emphasizes operational issues rather than organizational ones. STEAM [15], for instance, provides a hierarchical role-based framework for the quick formation of agent teams and coordination between them. Within our context, STEAM is an example of the type of coordination mechanism that could exist within the automated system's store of knowledge. Similarly, GPGP [9, 3] provides a family of coordination mechanisms each of which fits within the scope of the automated designer's knowledge.

The remainder of the paper is organized as follows. Section 2 formally defines our model and describes the design and search processes. Section 3 provides examples of organizational designs generated by a prototype designer we have built for a DSN under various environmental conditions and performance requirements. We conclude and describe future work in Section 4.

2. Model and Design Process

2.1. Problem-Domain Inputs

Refer once more to the left side of Figure 1. The environmental model M gives the general expectations of the environment over a period of time and is represented as a set of attribute-values pairs:

$$M = \{\langle f_i, v_{f_i} \rangle\} \quad (1)$$

where f_i is a user specified, domain specific environmental feature and $v_{f_i} \in \mathbf{R}$.

The set of performance requirements Q specify the requirements that must be met by the organization in order for it to satisfy the organizational goals. We represent Q as a set of attribute-value pairs similar to the environmental model:

$$Q = \{\langle q_i, v_{q_i} \rangle\} \quad (2)$$

where q_i is a feature and $v_{q_i} \in \mathbf{R}$ is its value.

Figure 2 shows an example environmental model and set of performance requirements for the DSN example that we will refer to throughout the paper. The example is a simplified version of the EW Challenge Problem domain [7] in which agents that control radar-based scanners must cooperate to track vehicles moving through a rectangular region. The environmental model indicates the expected traffic volume, spatial density, arrival rate, etc. The performance requirements are to track all vehicles with three meters of accuracy and a detection delay of at most 3 seconds.

Environmental Model			
maxNewArrivals	10	Performance Requirements	
maxTracks	10		
maxVelocity	$10 \frac{m}{sec}$		
vehicleWidth	$2m$		
(x,y)	(0,0)		
length	90		
width	90		
		Detect Delay	3sec
		Track Resolution	3m

Figure 2. Example environmental model

Returning to Figure 1, an organizational goal g is a high-level, long-term objective of an organization. We represent the decomposition of organizational goals in a tree T with root r . The nodes of T are goals and the edges represent subgoal relations such that there exists an edge from i to j if and only if $p(i, j)$ is true where $p(i, j)$ is a predicate indicating that i is the parent of j . Figure 3 illustrates a goal tree for our example DSN. It shows that the high-level root goal MONITOR can be decomposed into a subgoal for detecting new vehicles and one for tracking detected vehicles. Similarly, DETECT and TRACK can be further decomposed.

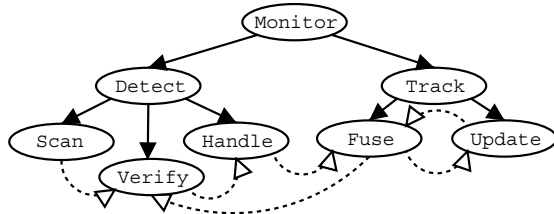


Figure 3. Example DSN goal tree and associated communication graph (dotted edges)

Each goal g is parameterized by the environmental model and the performance requirements. We represent the parameters of a goal as a set $P \subseteq M \cup Q$. For the

root r , $P = M \cup Q$. Every other goal g inherits the parameters of its parent unless otherwise specified by the developer. We also note that goals can be qualitatively different along a spectrum between continuous and triggered which can effect the most appropriate bindings for it. The DETECT goal in Figure 3 is a good example of a continuous goal. The organization must always look for new vehicles regardless of environmental or other features. DETECT can, therefore, be achieved with relatively static role assignments. This does not mean that the operational activities associated with performing these roles are static, but that a set of role-goal-agent bindings remains effective over time. TRACK on the other hand is a triggered goal. Each newly detected vehicle generates a new, more transient, tracking goal parameterized by the location of the detected vehicle. Therefore, it requires agents and roles to be assigned and modified in response to environmental dynamics. In addition both DETECT and TRACK are spatial goals in that their parameterizations include particular areas to cover.

In addition to its parameters and features, each goal g has a to-be-assigned list TAL of responsibilities that need to be assigned to an agent or agents in order for g to be satisfied. We define a goal $g \in L$ where L is the set of leaf goals to be *satisfied* if agents bound to it perform each of the responsibilities in its to-be-assigned list within the performance requirements on it. For all $g \notin L$, g is satisfied if all of its children are satisfied.

Figure 4(a) shows an example of the parameters and to-be-assigned list of the goal SCAN from Figure 3. It inherits all parameters except for maxTracks and Track Resolution. Although SCAN and the other goals in our example have single responsibilities in their to-be-assigned lists, in general a goal will have multiple responsibilities that must be fulfilled.

As in a traditional planning system, where goal decomposition continues until the subgoals can be achieved by primitive actions, organizational goal decomposition continues until the to-be-assigned lists of subgoals can be fulfilled by the assignment of roles. Roles are “atomic” job descriptions used to satisfy organizational goals. Each role r_i has an assignable-list AL_i of responsibilities that it can perform, a quality function f_i indicating how well it achieves a goal, a set of requirement functions, F_i , dependent on the parameters of the goals the role may be bound to, and a function D_i specifying how the role when bound to a goal can be distributed among a group of agents. Thus, we define the set of available problem-domain roles R as

$$R = \{r_i\} = \{\langle AL_i, f_i, F_i, D_i \rangle\}. \quad (3)$$

Figure 4(b) shows the roles and their assignable lists available in the DSN example. As with the goals, although each has only a single responsibility in its AL , in general roles

can have multiple responsibilities. For each role, the functions f_i and F_i are dependent on goal parameters (represented by P_X where X indicates one of the goals in Figure 3) and D_i is a function of the parameters of the goal the role is bound to and the set of available agents A (discussed below). F_i for RADARSCANNER, for instance, given the parameters of SCAN determines how often the region must be scanned in order to guarantee that vehicles are detected within the acceptable track delay requirement.

Certain goals require information from other goals. We represent such communication relationships as a directed communication graph $G = (L, E)$ where L is the set of leaf goals in T and E is a set of edges between the leaves such that there exists an edge (u, v) if information must flow from goal u to goal v . The dotted edges between goals in Figure 3 represent the communication graph for our DSN example. Suppose there is an edge in the communication graph from g_1 to g_2 and that agent sets A_1 and A_2 are bound to each respectively. If the goals are spatial in character, it is not necessarily the case that every agent in A_2 needs all of the information from every agent in A_1 . We represent this notion within the parameterization of each goal by specifying for each spatially defined goal the area the goal is responsible for. Thus, a goal requires information from another only if the information pertains to the goal's area. As we will see below, after the responsibility of handling a goal is distributed among a set of agents, each agent becomes responsible for a subregion of the whole and sends information to the relevant agents participating in connected goals. For example, suppose g_1 is responsible for area X and sends information to g_2 responsible for Y with the constraint that $X \subseteq Y$. When g_1 and g_2 are distributed among a set of agents, they are broken into $g_1^1 \dots g_1^n$ with areas $X_1 \dots X_n$ and $g_2^1 \dots g_2^m$ with areas $Y_1 \dots Y_m$ respectively. g_1^i sends information to g_2^j with the constraint that $X_i \subseteq Y_j$.

To complete the problem-domain input, let $A = \{a_i\}$ be the set of agents available to the organization. For a_i we specify a set ϕ_i of features such as its location, plus a set $\rho_i = \langle r_k, d_k, m_k \rangle$ of each role r_k that the agent is able to play, the percent drain d_k on the agent's resources caused by r_k , and the number of messages per time m_k the agent sends during its operational performance of r_k (m_k may be a function). In addition we specify a set C_i of capabilities:

$$C_i = \{\langle c_j, v_{c_j} \rangle\} \quad (4)$$

where c_j is a capability and $v_{c_j} \in \mathbf{R}$ is its value. Thus,

$$a_i = \langle \phi_i, \rho_i, C_i \rangle \quad (5)$$

2.2. Problem-Domain Matching

With the above input, the design process first attempts to match problem-domain roles to organizational goals to form

role-goal bindings, assignments of specific roles to each leaf goal in T . Any role whose assignable-list contains a goal's to-be-assigned list may be bound to that goal. In the DSN example the RADARSCANNER role can be bound to the SCAN goal forming the RADARSCANNER→SCAN role-goal binding. When a role is bound to a goal, it produces requirements as specified by the role's requirement functions. We define the set of role-goal bindings within an organization as a set of triples:

$$RGB = \{\langle r_i, g_j, \mu_k \rangle\} \quad (6)$$

where $r_i \in R$ and g_j is a leaf goal of T such that $TAL_j \subseteq AL_{r_i}$, and $\mu_k = \{\langle \mu_h, v_{\mu_h} \rangle\}$ is a set of requirement attribute-value pairs determined by r_i 's requirement function parameterized by g_j and its parameters. For RADARSCANNER→SCAN, μ_h and μ_v specify the scan frequency that must be maintained.

The next step in the process is to bind agents to each role-goal binding. Continuing with the RADARSCANNER→SCAN example, the design process identifies agents that can meet the requirements of the role-goal binding according to the agents' capabilities and RADARSCANNER's distribution function to form a set of role-goal-agent bindings. The particular binding specifies the role the agent is bound to, the decomposed sub-goal it is responsible for, and the sets of agents it receives information from and sends information to. Thus, we define the set of role-goal-agent bindings of agent $a_i \in A$ as

$$RGAB_{a_i} = \{\langle r_k, g_j, g'_j, f_{g'_j}, t_{g'_j}, team \rangle\} \quad (7)$$

where $r_k \in R$, g_j is a leaf goal of T , g'_j is a subgoal of g_j as determined by r_k 's decomposition method, $f_{g'_j}$ is the set of agents a_i receives information from pertaining to this binding, $t_{g'_j}$ is the set of agents a_i sends information to, and $team$ is a boolean flag indicating that this binding is a teaming role assignment (described below).

2.3. Coordination-Domain Matching

Up to this point, the design process involves problem-domain specific knowledge of goals, roles, performance requirements, agent capabilities, etc., and is shown in the left half of Figure 1. What is advantageous to our approach is that the remainder of the organizational-design process can be addressed using more domain-independent organizational coordination knowledge. In general, a role will require multiple agents to fulfill the performance requirements of an organizational subgoal. In our example, sensor agents have limited range and synchronized scanning by at least three agents is required throughout the coverage area. Not only must role-goal-agent bindings be

found as above, but those agents must also be coordinated in performing their roles. The agents bound to RADARSCANNER→SCAN have the necessary capabilities to satisfy the requirements, but unless their scanning is synchronized correctly, holes may exist in the coverage.

The need to coordinate these agents causes the system to generate a new *coordination goal* that was not part of the original goal decomposition. This organizational-coordination goal must be fulfilled by more problem-domain-independent coordination roles, as shown on the right side of Figure 2. Possible roles for coordinating our set of sensing agents include: peer-to-peer negotiation of scan schedules and a simple, one-level hierarchy where a manager agent (potentially, but not necessarily, one of the sensing agents) develops the scan schedule for the group. A coordination role-goal-binding can, itself, require a set of agents to satisfy it, causing the creation of another higher-level coordination goal. For example, if the span of control of potential manager agents requires the use of multiple managers, the activities of these managers would also need to be coordinated again, potentially using a peer-to-peer or hierarchical approach. If the latter is chosen, management of our sensing agents would involve a multi-level hierarchy of sensing, middle-manager, and overall manager roles.

The sets of role-goal-agent bindings and their parameters specify the long-term structure, role assignments, authority relationships, and communication paths of the designed organization. These are particularly applicable to static goals. However, these long-term bindings alone can be insufficient to satisfy the organizational goals when we consider dynamic goals which may be better satisfied by establishing teams [4, 15, 13, 14, 8, 1]. A team, coalition, or congregation is a temporary structure that is formed as needed to satisfy a particular task when it enters the environment and is disbanded when the task leaves the environment or is completed. In our simple example, tracking a newly detected vehicle might be done by creating a team whose membership changes as the vehicle moves through the monitored area. Teams are not strictly part of the organizational structure since the assignment of agents to roles associated with the team will not be as long lived as the assignment of agents to roles to satisfy organizational goals. However, a team is not a purely operational construct either, since sufficient resources must be set aside organizationally to allow for generating and participating in teams. Furthermore, when an agent within an organization is participating in a team, its team activities will have an effect on how it satisfies its other roles. Therefore, the organizational structure must account for and be prepared for team activity by its members.

We do not generate transient teams in our organization-

design process, but we must ensure that the organizational structures and resources exist to generate effective teams as needed. Thus, in our design process we reserve resources within agents capable of participating in teams. For the DSN example, this means finding role-goal-agent bindings for the leaf goals of TRACK, but setting the *team* flag to true to indicate that the agent participates in the team only as needed. A team role is similar to an organizational role in that the agent with a team-role responsibility will have an expected number and frequency of messages to send and amount of work to do. The difference is that the agents bound to these roles will only be expected to perform those activities if and when they are called upon to join a team. Furthermore, we must also specify appropriate coordination roles in order to enable teams to form. In this work, we define a TEAMINITIATOR role that is responsible for generating teams operationally.

Figure 4(c) shows an example set of bindings for a single agent participating in the DSN. For each binding, it specifies the which organizational subgoal it is bound to, and the role and agents in that role to which it sends information. If the role is a teaming assignment such as FUSER→FUSE, it is signified with a superscript *T*.

2.4. Search and Suitability

In general, multiple roles can satisfy the same organizational subgoal, many agents can be bound to the same role-goal binding, and a single agent can play multiple roles simultaneously, making it computationally infeasible to generate all possible bindings. Therefore, we have developed a prototype system that uses organization-design knowledge and heuristics to generate a reasonable set of bindings. For the domain-specific portions of the design process, the heuristics rely on information provided by the developer in the quality, requirements, and decomposition specifications of the roles plus the capabilities of the agents. In general the heuristics consider which roles should be bound to the organizational goals, which agents can be bound to particular role-goal bindings and the computational and communication loading on agents that would result under different assignments.

For coordination goals, the design system goes through a similar process of finding role-goal-agent bindings for the coordination goals. The main difference is that the roles available for satisfying the coordination goals and the search heuristics exist within the organizational-coordination library as domain-independent knowledge. In the current prototype the parameters on coordination roles and goals are not fully generalized; some parameterization values still refer to problem-domain parameters. In future research, we plan to develop generic abstractions of problem-domain parameters (that would

SCAN((x,y), length, width, maxNewArrivals,
maxVelocity, vehicleWidth, detectDelay)
TAL: Scanning

(a) Parameters and to-be-assigned list of SCAN goal

Role	AL	f_i	F_i	D_i
RADARSCANNER	Scanning	P_S	P_S	P_S, A
VERIFIER	Verifying	P_V	P_V	P_V, A
HANDLER	Handling	P_H	P_H	P_H, A
FOCUSSEDRADAR	Updating	P_U	P_U	P_U, A
FUSE	Fusing	P_F	P_F	P_F, A

(b) Problem-Domain Roles for the DSN example showing each role's assignable list and the parameters to each of the functions in Equation 3. P_X represents the parameters of a goal where X represents one of the goals in Figure 3. A is the set of agents.

Agent S24 (82.5, 52.5)

RADARSCANNER → SCAN((62.5,32.5),40,40)
TO: VERIFIER S22
FUSER → FUSE((45,60),45, 30)^T
TO: FOCUSSEDRADAR S22 S24 S23 S18 ...
TO: VERIFIER S22
FROM: FOCUSSEDRADAR S22 S24 S23 ...
FROM: HANDLER S22
SUBORDINATE → COORDGOAL(SCAN)
TO: MANAGER S22
FROM: MANAGER S22
...

(c) Set of problem-domain and coordination role-goal-agent bindings for a single agent

Figure 4. Representation of goals, roles, and role-goal-agent bindings

be included as part of the problem-specific knowledge) that would provide a completely clean separation of problem-domain and coordination parameters.

Although the heuristics above should lead to an organization that meets the performance requirements, they do not give enough information to rank a set of feasible candidate organizations all of which satisfy the requirements. We must consider other factors in how we evaluate them. For that it is important to have an organizational evaluation function that is based on user specified criteria to determine the utility of a particular candidate. In future work, we plan to develop a detailed evaluation capability both to eval-

uate fully specified organizations and to prune the search through partially complete bindings. For now, we rely on simple utility criteria stemming from the relative costs of agent load and communication.

3. Example Organization Designs

We present below four example organizational designs generated by our automated system on the goal tree and communication graph in Figure 3, the parameters in Figure 2, and the roles in Figure 4(b). We varied the input along several dimensions: size of the area to be scanned and number of agents available, the value of the acceptable track delay performance requirement, and the relative costs of communication and agent load. In all cases the agents we used were evenly spaced throughout the region, each with identical features, roles they can be bound to, and capabilities. Figure 5 summarizes the results.

Agents	Area	Delay	Com. Cost	Load Cost
36	90' × 90'	3s	0.6	0.4
Single-level hierarchy: 6 Managers. Verifier and Handler roles multiplexed within same agent as Manager. Managers coordinate peer-to-peer.				
Agents	Area	Delay	Com. Cost	Load Cost
36	90' × 90'	3s	0.4	0.6
Single-level hierarchy: 6 Managers. Verifier and Handler roles not multiplexed with Manager. Managers coordinate peer-to-peer.				
Agents	Area	Delay	Com. Cost	Load Cost
36	90' × 90'	2s	0.6	0.4
Two-level hierarchy: 6 mid-level Managers. Verifier and handler roles multiplexed within mid-level Managers. One upper-level Manager to coordinate mid-level Managers.				
Agents	Area	Delay	Com. Cost	Load Cost
100	150' × 150'	3s	0.6	0.4
Two-level hierarchy: 9 mid-level Managers. Verifier and handler roles multiplexed within mid-level Managers. Two upper-level Managers to coordinate mid-level Managers. Upper-level Managers coordinate peer-to-peer.				

Figure 5. Example Organizational Designs

In the first design scenario, we used 36 agents in a 90' × 90' rectangular area with an acceptable track delay performance requirement of 3 seconds, and the cost of communication greater than that of agent loading. The resulting organization was a single-level hierarchy with 6 managers each managing 6 agents. The managers coordinated among themselves using a peer-to-peer mechanism. Furthermore, in order to minimize communication, there were 6 verifying and handling roles each multiplexed within the

same agents as the managing roles. This organization corresponds closely to the hand-crafted organizational structure used for the EW Challenge Problem [7] where communication cost was a major concern. The performance of this organizational form relative to others was recently tested experimentally [5, 6]. Also, in this scenario and the others, the FUSER and FOCUSSEDRADAR roles were set as team roles with the TEAMINITIATOR role distributed among the HANDLER agents.

When we switched the relative costs of communication and load, the resulting organizational design was still a single-level hierarchy, but the verifying and handling roles were no longer multiplexed within the same agents as the manager roles. Instead they were distributed to separate agents in order to minimize load. In effect because communication was inexpensive, the organization could afford to use more communication in order to balance the computational load among the agents.

For the third scenario, we used the same costs as in the first, but reduced the acceptable track delay to 2 seconds. This time the generated organization was a two-level hierarchy with 6 mid-level managers and 1 upper-level manager to coordinate them. At first this may seem counter-intuitive since increasing the level of hierarchy can often introduce delays. However, in this problem with a small acceptable delay on new detections, it is critical that the scanning agents have tightly synchronized scan-schedules. Because producing a shared scan-schedule can be done in advance of detection activities, the design system added a second level of hierarchy in order to resolve scan-schedule conflicts among the managers in a centralized fashion.

In the last scenario, the parameters were also the same as in the first run except that we increased the number of agents to 100 and the size of the region to $150' \times 150'$. In this case the system generated another two-level hierarchy this time with 9 managers and 2 upper-level managers which coordinate using a peer-to-peer mechanism.

Overall, we were quite pleased that our design system produced such different organizational forms given the changes to the environmental characteristics and performance requirements we presented it with. These results confirm for us the usefulness of our approach in generating organizational forms without pre-specified organizational information.

4. Conclusions and Future Work

We believe that the prescriptive, knowledge-based organizational design process we have presented has great promise for the field of multi-agent organizational design. It relies on a separation between problem-domain and organizational coordination-domain knowledge to generalize coordination mechanisms across domains, requiring a devel-

oper only to supply problem-specific information. The results from our prototype system show that through this process we are able to design organizations of different forms by varying performance requirements and environmental characteristics. We believe, this is the first work to do so.

We have identified several areas of future work stemming from the initial research presented here. First, we will add an evaluation capability to our system that, given the bindings of a set of candidate organizations, performance requirements, and more detailed evaluation criteria specified by the developer, will rank the candidate organizations. We also hope to apply the evaluation capability to partial bindings in order to prune the search for a suitable organization. Another long-term goal is that in addition to evaluating generated organizations, we would like the system to suggest what additional resources and capabilities, if they were provided, would have supported a better organization.

In addition, we must continue to refine our understanding of coordination-domain knowledge so as to parameterize the coordination roles more appropriately. Part of this will involve understanding the distinguishing features of goals and how those features relate to the mechanisms available to coordinate the agents bound to those goals. In part this will involve a greater understanding of aspects such as how resource contention, the number of agents bound to a goal, and the interdependency among agents and goals interrelate.

References

- [1] C. H. Brooks and E. H. Durfee. Congregation formation in multiagent systems. *Journal of Autonomous Agents and Multiagent Systems*, 7:145–170, 2003.
- [2] D. D. Corkill. *A Framework for Organizational Self-Design in Distributed Problem-Solving Networks*. PhD thesis, University of Massachusetts, Amherst, Massachusetts 01003, Feb. 1983. (Also published as Technical Report 82-33, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, December 1982.).
- [3] K. Decker and V. Lesser. Generalizing the partial global planning algorithm. *International Journal on Intelligent Cooperative Information Systems*, 1(2):319–346, June 1992.
- [4] M. S. Fox, M. Barbuceanu, M. Gruninger, and J. Lin. An organization ontology for enterprise modelling. In M. Prietula, K. Carley, and L. Gasser, editors, *Simulating Organizations: Computational Models of Institutions and Groups*, pages 131–152. AAAI/MIT Press, 1998.
- [5] B. Horling, R. Mailler, and V. Lesser. A Case Study of Organizational Effects in a Distributed Sensor Network. 2004.
- [6] B. Horling, R. Mailler, M. Sims, and V. Lesser. Using and Maintaining Organization in a Large-Scale Distributed Sensor Network. *Proceedings of the Workshop on Autonomy, Delegation, and Control (AAMAS03)*, July 2003.

- [7] B. Horling, R. Vincent, R. Mailler, J. Shen, R. Becker, K. Rawlins, and V. Lesser. Distributed sensor network for real time tracking. In *Proceedings of the 5th International Conference on Autonomous Agents*, pages 417–424, Montreal, June 2001. ACM Press.
- [8] M. Klusch and A. Gerber. Dynamic coalition formation among rational agents. *IEEE Intelligent Systems*, 17(3):42–47, May/June 2002.
- [9] V. R. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. N. Prasad, A. Raja, R. Vincent, P. Xuan, and X. Q. Zhang. Evolution of the GPGP/TÆMS domain-independent coordination framework. *Autonomous Agents and Multi-Agent Systems*, 2003. (Accepted, conditionally, 8/02).
- [10] Y. pa So and E. H. Durfee. Designing tree-structured organizations for computational agents. *Computational and Mathematical Organization Theory*, 2(3):219–246, 1996.
- [11] Y. pa So and E. H. Durfee. Designing organizations for computational agents. In *Simulating Organizations: Computational Models of Institutions and Groups*, pages 47–64. AAAI Press/MIT Press, 1998.
- [12] H. E. Pattison, D. D. Corkill, and V. R. Lesser. Instantiating descriptions of organizational structures. In M. N. Huhns, editor, *Distributed Artificial Intelligence*, Research Notes in Artificial Intelligence, chapter 3, pages 59–96. Pitman, 1987.
- [13] T. Sandholm and V. Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence, Special Issue on Economic Principles of Multi-Agent Systems*, 94(1):99–137, Jan. 1997.
- [14] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1–2):165–200, 1998.
- [15] M. Tambe, J. Adibi, Y. Alonaizon, A. Erdem, G. Kaminka, S. Marsella, and I. Muslea. Building agent teams using an explicit teamwork model and learning. *Artificial Intelligence*, 110:215–240, 1999.