Chapter 1

USING AUTONOMY, ORGANIZATIONAL DESIGN AND NEGOTIATION IN A DISTRIBUTED SENSOR NETWORK

Bryan Horling[1], Roger Mailler[1], Jiaying Shen[1], Régis Vincent[2], and Victor Lesser[1]

[1]*Department of Computer Science, University of Massachusetts*
Amherst, MA 01060
{bhorling,mailler,jyshen,lesser}@cs.umass.edu

[2]*SRI International, 333 Ravenswood Ave.*
Menlo Park, CA 94025
vincent@ai.sri.com

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 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 the hardware-based system.

1. Overview

The UMass approach to the distributed sensor network challenge problem consists of three major architectural and behavioral contributions. First, each of the sensors is controlled by a single agent which exists as part of a larger, heterogeneous organizational structure. This structure helps bound and target the computation necessary to solve the distributed tracking problem by associating individual agents with one or more roles within that organization, each 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 quantities and qualities of data are achieved for all targets to be tracked, if at all possible.

Each of these technologies plays an important role solving the sensor allocation allocation problem. The SPAM protocol [Mailler et al., 2001] 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 [Horling et al., 2002] resolves conflicts that exist within an agent, by modeling tasks and commitments and using several scheduling techniques to manage local activities as best possible. This allows arbitrary allocations to accrue quality, even if some level of unresolved conflict exists. The organizational design acts to limit the distance over which information must be propagated, which both reduces communication effort and facilitates the allocation and negotiation process. All of these technologies must operate in real-time to be effective in this distributed sensor network environment.

A high-level view of the solution described in this chapter can be seen in Figure 1.1. Each sensor is controlled by a single agent, and the organizational design divides these sensor agents into location-based sectors. Each of these sectors 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. This information 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 negotiating over the sensors, and fusing the data they produce. SPAM and the problem solver identify desired sensors, and request commitments from them. Upon receipt of such a commitment, a sensor takes on a *data collection* role. Like the scan schedule, these commitments are used to generate 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 data are gathered, they are fused and interpreted to estimate the target's location, which allows the process to continue.

The connection between resource allocation and tracking is straightforward - in order to track a particular target, sensor resources must be allocated in such a way that the data they produce can be used to determine a target's position. Furthermore, this needs to be done in such a way that tasks competing for a sensor's attention are not starved. The need to triangulate a target's position requires frequent, coordinated actions among the agents - ideally three or more sensors taking measurements of a target at approximately the same time. Early versions of the ANTs environment required at least three coordinated measurements per track data point. While the current implementation can interpret based on a single measurement, multiple coordinated measurements will better reduce uncertainty, so coordinating the activities of the agents is still a beneficial strategy. 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 as frequently as possible. Limited communication and competition from other tracking and scanning tasks, however, restrict 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 tradeoff between discovering new targets and tracking existing ones.

The real-time nature of this problem presents another set of complications. While schedules and commitments may specify precise instants in time, it can be difficult to meet these deadlines in the face of uncertain action durations and competing meta-level activities or local processes. For example, the scheduling and reasoning mechanisms of the agent clearly require some amount of time and computational resources, which will then compete with the agent's scheduled activities on a single processor system. A viable solution must be able to work effectively in an environment where a certain amount of temporal imprecision may exist, and where commitments or actions may fail because they do not meet timing constraints. This architecture does not directly reason about or estimate the effects of meta-level activities, but instead copes with such uncertainty by generating schedules which generally contain a sufficient amount of slack time for all activities to coexist. A more explicit, and likely more reliable solution would reason about the effects of meta-level activities directly, as in [Raja and Lesser, 2002].

These and other characteristics of the environment contribute to a large degree of uncertainty the solution must handle. Noisy measurements, unreliable communications, varying hardware speeds, and sensor availability also make knowing a target's precise location and velocity very difficult. This in turn makes predicting and planning for future events more difficult, which subsequently increases usage of resources when unreliable data directs high level reasoning to incorrect conclusions and actions.

In the remainder of this chapter, we will describe our solution to these complicated problems. The three main components of this solution, it's organization, agent control mechanisms and resource allocation protocol, are described in more detail in the following sections. Section 1.5 is a discussion of our results and experiences, and the chapter will conclude with some final thoughts and discussion of future work.

2. 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 towards scalability, reliability, speed, or efficiency, among other things.

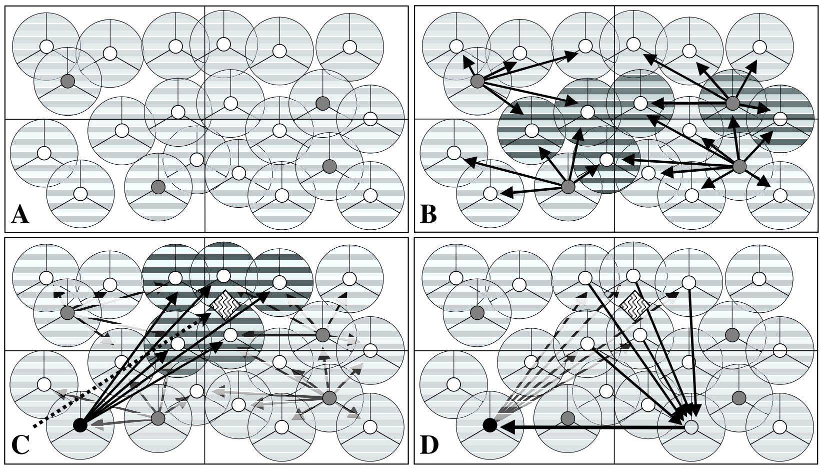

Figure 1.1. High-level architecture. A: sectorization of the environment, B: distribution of the scan schedule, C: negotiation over tracking measurements, and D: fusion of tracking data.

The organizational design used in this solution primarily attempts to address the scalability problem, by imposing limits on how far certain classes of information must 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.

The environment itself is divided into a series of sectors, each a non-overlapping, identically sized, rectangular portion of the available area, shown in figure 1.1A. The purpose of this division, as will be shown below, is to limit the interactions needed between sensors, an important element of our attempt to make the solution scalable. In this figure, 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 positions and sensor radii, 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 1.2. The organizational leader of each

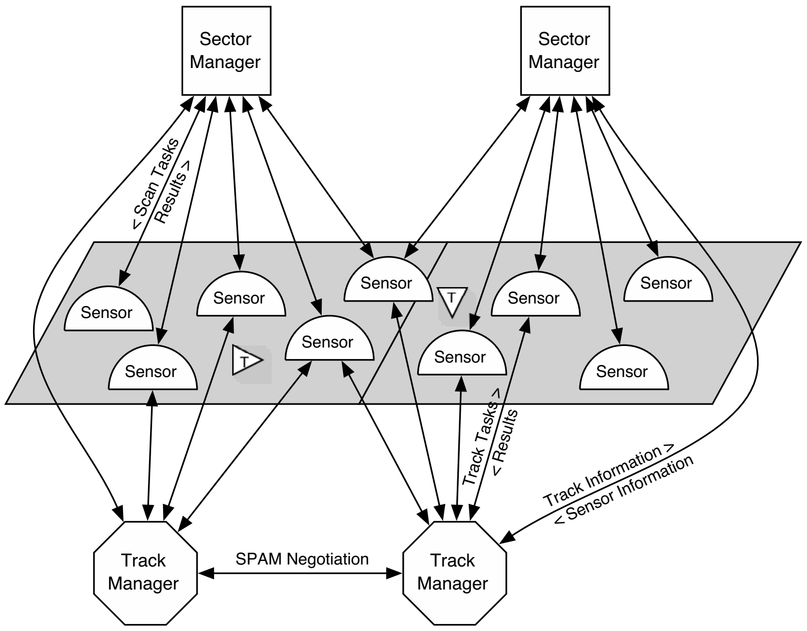

Figure 1.2. Overview of the agent's organizational hierarchy, with some information flows represented.

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 target 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. We found empirically that a reasonable sector would contain 8 sensors, but would still function 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 [Sims et al., 2003].

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 priori[1]. In figure 1.1, 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 negotiates with each agent over their respective responsibilities in that schedule (see figure 1.1B). 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.

[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.

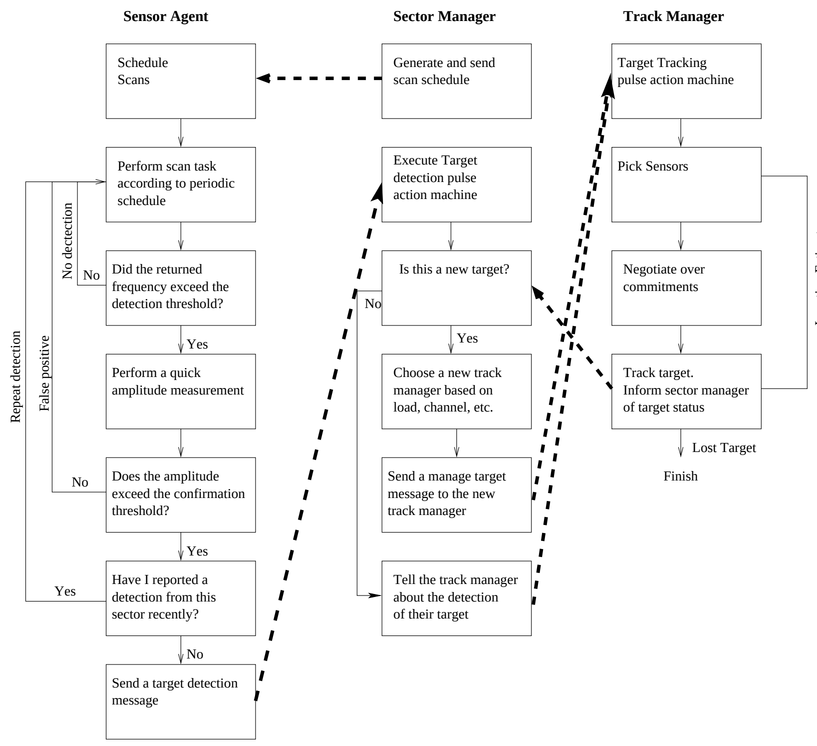
Sensor Agent
Sector Manager
Track Manager
Schedule Scans
Generate and send scan schedule
Target Tracking pulse action machine
Perform scan task according to periodic schedule
Execute Target detection pulse action machine
Pick Sensors
No dectection
No
Did the returned frequency exceed the detection threshold?
Is this a new target?
No
Negotiate over commitments
Location Estimate
Yes
Yes
Repeat detection
False positive
Perform a quick amplitude measurement
Choose a new track manager based on load, channel, etc.
Track target. Inform sector manager of target status
Lost Target
Finish
No
Does the amplitude exceed the confirmation threshold?
Send a manage target message to the new track manager
Yes
Yes
Have I reported a detection from this sector recently?
Tell the track manager about the detection of their target
No
Send a target detection message

Figure 1.3. An abstraction of the messages and reasoning used for target detection by sensor agents, sector and track managers.

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. These detections then be used to spawn a new track manager as shown in Figure 1.3. 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. This location estimate is 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 uniformed 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 loads, communication channel assignments, geographic locations and activity histories. Ideally, it will select an agent which has minimal channel overlap and is not currently tracking a target, but which 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 1.1C with a blackened inner circle) is responsible for organizing the tracking of the given target. To do this, it first discovers sensors capable of detecting the target, and then negotiates 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 negotiate with the discovered agents to disseminate this goal (see figure 1.1C). As with scanning, conflicts can arise between the new task and existing commitments at the sensor, which the agent must resolve locally. The source of a given commitment can identify how important its task is to it, which is normalized in such a way that it has the correct importance relative to others in a more global sense. For instance, if a track manager determines that a sensor is particularly useful, based either on its location relative to the estimated position of the target or the scarcity of viable alternative sensors, this can be reflected in the importance value of the commitment. These importance values then allow the local agent to effectively discriminate among conflicted tasks with an eye towards global social welfare.

The data gathered from individual sensors is collected by an agent responsible for fusing the data and extending the computed track (see figure 1.1D). 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 itself is fairly lightweight, and thus does not benefit from distribution for load balancing purposes, and transferring the fusion data results in an additional delay while it is being communicated to the track manager. Therefore, our organization assigns this fusion task to the track manager itself, which avoids this delay with relatively little overhead. 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 the position of the target at that time. This data point is then added to the track, which itself is distributed back to the track manager to be 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. Further details describing the track manager, including the tracking process in general and the tradeoffs associated with communication decisions and predictive capacity will be covered in section 1.4.

Partitioning the environment reduces the amount of information and processing 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. In fact, the partitioning makes nearly every aspect of this solution scalable to arbitrary numbers, with the exception of negotiation, 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 book, 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. The technique of migrating the tracking responsibility through the agent population as the target moves, which is also covered in section 1.4, is a direct result of this.

3. Agent Architecture

The structure and function of agents used in this solution can be roughly divided into two parts: elements that are roughly problem-independent, and those that are relatively problem-dependent. In this section we describe the former part, consisting of the basic framework used to build the agents, and the control engine which drives much of its behavior. The latter part, consisting of the domain problem solving expert and the resource allocation protocol, are covered in the subsequent section.

3.1 Java Agent Framework

We use the Java Agent Framework (JAF) [Horling and Lesser, 1998] as the foundation to our implemented solution. JAF is a component-oriented framework, similar to Sun's JavaBeans technology. The JAF framework consists of a number of generic components that can be used directly or by subclassing them, along with a set of guidelines specifying how to implement, integrate, and use new components. Components can interact in three different ways, each having different flexibility and efficiency characteristics: direct method invocation, through event (message) passing among components, or indirectly through shared data.

JAF was designed with extensibility and reusability in mind. The use of generic components, or derived components with similar APIs, allows for a plug-and-play type architecture where the designer can select those components they need without sacrificing compatibility with the remainder of the system. The designer can therefore pick and choose from the pre-written components, derive those that aren't quite what he or she needs, and add new components for new technologies. For example, generic components exist to provide services for such things as communication, execution and directory services. In the environment presented in this paper, specialized facilities are needed for communication and execution. Derived versions of these two components were written, overriding such things as how messages are sent or how certain actions are performed. The communication component was also extended to provide a reliable messaging service, using sequence numbers, acknowledgments and retransmits to cope with the unreliable RF medium. These derived components were then inserted in place of their generic counterparts within the agent. The unmodified directory service component can still make use of the communication component, and if needed, communication can also use the directory services. In all, 17 components were used in the agents described in this paper: 10 were generic, 3 were derived, and 4 were new. This translates to roughly 20,000 lines

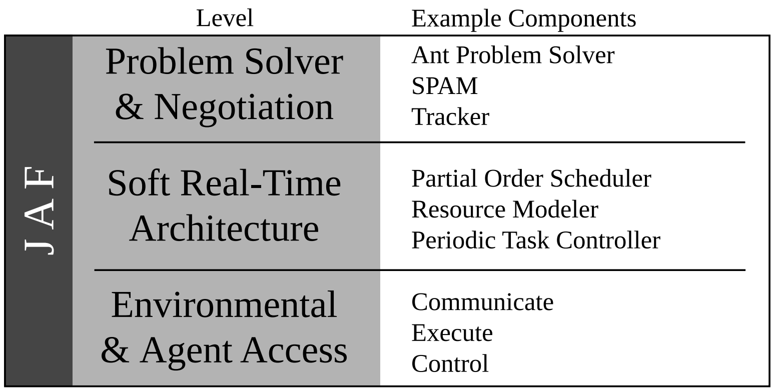

Figure 1.4. A JAF agent architecture, modeled after the one used by the agents in the DSN environment. Broad levels of abstraction are shown, along with typical components which reside at those levels.

of reused, domain independent code, and 8,000 lines of domain dependent code. The specific components which were used to create the agents are: Control, Log, State, Execute, Communicate, WindowManager, Observe, Sensor, ActionMonitor, PreprocessTaemsReader, DirectoryService, ResourceModeler, PartialOrderScheduler, PeriodicTaskController, ScanScheduler, Coordinate, and AntProblemSolver.

JAF was used in this architecture to create a layered agent, as shown in figure 1.4. Here, the problem solver and negotiation components reside at the highest level, where they deal with problems at a coarse, conceptual level of abstraction. These components will typically interact with those below it, in this case by passing goals and tasks to the soft-real time architecture, which reasons about actions and schedules. This in turn will make use of the lowest agent level to gather data and actually perform those actions.

While layers of abstraction and encapsulation certainly are not new ideas, their incorporation into this architecture is important because they both facilitate construction and motivate reusability and clean software design. A variety of components currently exist in JAF, providing services from logging and state maintenance to scheduling and problem solving.

3.1.1 Communication.

The communication component used by the agent is worth examining because of the adaptations that were made to work in the ANTs environment. The designed communication medium used between sensors is a radio-frequency, wireless broadcast technology. It consists of eight independent channels, each of which offers a theoretical data rate of approximately 14.4 Kbp. Data transfer is unreliable, so collisions may result in data loss, although a checksum prevents garbled messages from being given to the agent. In our architecture, agents were uniformly assigned channels to receive messages on, and were required to change their transmission channel to match the receive channel of the destination.

The central issue when using the RF communication medium (or operating in simulations of it) is the relatively low bandwidth. The theoretical data rate, combined with the potential for message loss caused by collisions, permitted only a handful of messages to be successfully transferred per second on a given channel. When one considers that a track manager might need to receive messages from three or four data sources, negotiate with one or more track managers, and get state information from the sector manager - all in the span of a second or two - this limitation can greatly restrict communication, and we have implemented strategies at several levels within our agents to cope with it.

Several mechanisms were built into the communication component used by the agent to work under these conditions. The first was a reliable messaging system, using an ALOHA-like retransmission scheme. When a message is being composed, the source can mark it with a flag indicating it should be sent reliably, while other messages are assumed to be unreliable. Before transfer, the communication component adds a sequence number to such messages. Upon receipt of a sequenced message, the destination will reply with an ack, so that the source can track which messages have been successfully delivered. If no acknowledgment is received, the component waits a random amount of time, typically around 1500 ms, before resending the message.

While this solved the reliable messaging problem, the potential for retransmissions exacerbated the aggregate bandwidth limitation. In particular, spikes in an agent's communication level could result in a cascade of failures as retransmissions collide with new data. To help with both this and the bandwidth restrictions in a more general sense, a byte-level rate limiter was added, which enforces a limit on the maximum number of bytes an agent could transmit per interval. Any attempted transmissions past this cap are postponed until the next interval. The level used in practice was empirically selected, typically around 1000 bytes/second, but could also be learned.

With the addition of the rate limiter, it then became an issue of outgoing message queues potentially growing without limit. To overcome this, the communication component provides a communication load estimate, which can be used by other components to restrict unnecessary communication. The SPAM protocol, for instance, uses this to decide how aggressively it is willing to search for a solution. In addition, a lightweight prioritization scheme was added which affected which messages from the queue would be selected for delivery. It would also proactively drop unreliable messages if they were too old. Although some are relatively domain-specific, the mechanisms used work fairly well, and we feel these sorts of techniques could be generalized for use in other bandwidth-challenged environments.

3.1.2 Directory Services. Another component worth covering in more detail is the directory service component, which provides an agent with the ability to store arbitrary textual information from many sources and process queries over that data. Individual *entries* consist of one or more named fields, each of which will contain data. The directory also possesses a set of one or more *descriptions*, which specify the type of entries they are willing to accept. As a directory receives an entry to be added, it checks it against each of its descriptions, and if any match, the entry is added. Queries may be made to local or remote directories. The syntax for both entry descriptions and queries is the same, consisting of a series of boolean, arithmetic or string expressions. The functionality of the directory itself is generic, and thus can serve as the supporting structure for a number of different directory paradigms, such as yellow pages, blackboards or brokers [Sycara et al., 1997].

In our system, directory services are used primarily in a yellow pages capacity, to centralize and disseminate information, thereby limiting the amount of communication needed to gather information. Individual agents post their capabilities to their sector manager's directory, which allows other agents (particularly track managers) to easily search for agents that can sense within that sector. This type of query is used to both construct the scanning schedule, and determine which agents are capable of sensing a target at a particular location. Each agent in the environment also has its own, smaller directory service, where it locally stores descriptions of sector managers, providing an easy mechanism to find the managers of their own and neighboring sectors.

As an example, a typical sector manager might have several directory entries for sensors capable of scanning in its sector. One such entry would take the following form:

```
[E SA1 [Name->SA1] [Task->Scan] [R->20]
    [X->10] [Y->10] [O->60] [C->1]]
```

This contains information such as the sensor's name, task, sensing radius, x and y position, orientation and communication channel. Later, when, for instance, a track manager needs to determine which nodes can be used to track a target in a given region, it might generate the following query:

```
                 ((((((20 + R) >= X) & ((10 - R) <= X)) &
           ((10 + R) >= Y)) & ((0 - R) <= Y)) & (Task == "Scan"))
```

This query matches entries whose x,y locations fall within a given area, offset by the sensor's radius. In this case, it will return all sensors which are capable of scanning within the area (10,0),(20,10). If the region in question spanned multiple sectors, the track manager would assimilate the results from several queries to different sector managers.

The directory service provided by the sector manager is also used to create some measure of fault tolerance in the system. When a track manager submits a query like the one described above, the directory service remembers this event, and stores the query. If the contents of the directory are updated in such a way that the response to that query changes, it automatically forwards the new information to the agent which originally produced the query. Therefore, as the sector manager recognizes changes in the agent population, that information is automatically propagated to the affected agents. For example, if a sensor goes offline for some reason (diagnosing such a fault is a challenge we do not address here), that information will be reflected in the directory, which will then be sent to all relevant track managers. Once it has that information, the track manager can adapt to or compensate for that change as necessary.

3.2 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 ANTs solution:

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 efficiently handle deviations in expected plan behavior that arise out of variations in resource usage patterns and unexpected action characteristics.

The system is implemented as a set of interacting components and representations, as shown in Figure 1.5. A domain independent task description language is used to describe goals and their potential means

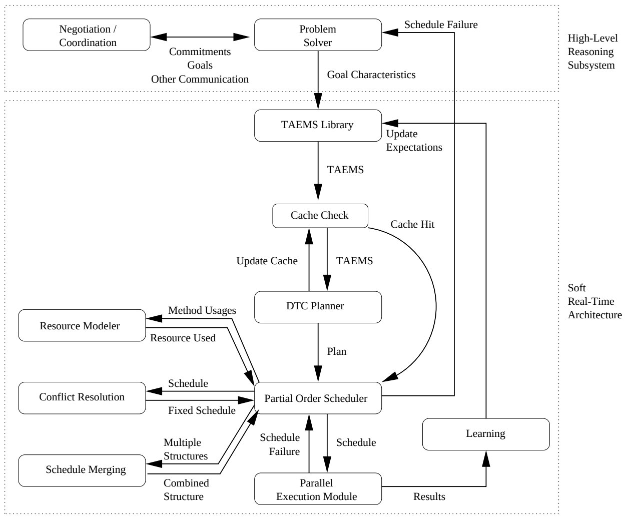

Figure 1.5. The soft real-time control architecture.

of completion, which includes a quantitative characterization of the behavior of alternatives. A planning engine can determine the most appropriate means of satisfying such a goal within the set of known constraints and commitments. This permits the system to be able to adjust which goals it will achieve, and how well it will achieve these chosen goals based on the dynamics of the current situation. Scheduling services integrate these actions and their resource requirements with those of other goals being concurrently pursued, while a concurrent execution module performs the actions as needed. Exception handling and conflict resolution services help repair and route information when unexpected events take place. Together, this system can assume responsibility for the majority of the goal-satisfaction process, 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 negotiate over an abstraction of their activities or resource allocations, and only locally translate those activities into a more precise form [Mailler et al., 2001]. SRTA can then use this description to both enforce the semantics of the commitments which were generated, and automatically attempt to resolve conflicts that were not addressed through negotiation.

SRTA is exploited in the agent to create a virtual agent organization of simple, single-purpose agents which are mapped onto the real agent population. For example, recall the track manager described earlier and shown in figure 1.2. While this role could be assigned to an agent who had no other responsibilities, we instead allow these roles to coexist with others, which are addressed in parallel by a single, sophisticated agent which uses SRTA to work on multiple goals simultaneously. The manager and sensor control roles described previously are in fact "virtual" agents, which are implemented as goals that are created as needed and dynamically assigned to a specific sophisticated "real" agent. The "real" agents in this case are the processes residing at and controlling the sensors in figure 1.2. The scanning activities described in the previous section are also created this way, when the sector manager provides an agent with details from the scan schedule, as are the tracking tasks described in the next section. With this information, the "real" agent then performs detailed planning/scheduling for all of its goals based on local resource availability and goal priority, and multiplexes among the different concurrently executing tasks in order to meet soft real-time requirements.

SRTA operates as a functional unit within a JAF-based agent, which itself runs on a conventional (i.e. not real-time) operating system. More generally, the SRTA controller is designed to be used as part of a layered architecture, occupying a position below the high-level reasoning component in an agent [Zhang et al., 2000, Bordini et al., 2002]. In this role, it will accept 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. Within this context, SRTA offers a range of features designed to provide support for operating in a distributed, intelligent environment. The goal description language supports quantitative, probabilistic models of activities, including non-local effects of actions and resources and a variety of ways to define how tasks decompose into subtasks. In particular, the uncertainty associated with activities can be directly modeled through the use of quantitative distributions describing the different outcomes a given action may produce. Commitments and constraints can be used to define relationships and interactions formed with other agents, as well as internally generated limits and deadlines. The planning process uses this information to generate a number of different plans, each with different performance characteristics, and ranked by their predicted utility. A plan is then used to produce a schedule of activities, which is combined with existing schedules to form a potentially parallel sequence of activities, which are partially ordered based on their interactions with both resources and one another. This sequence is used to perform the actions in time, using the identified preconditions to verify if actions can be performed, and invoking light-weight rescheduling if necessary. Finally, if conflicts arise, SRTA can use an extendable series of resolution techniques to correct the situation, in addition to passing the problem to higher level components which may be able to make a more informed decision.

3.3 TÆMS

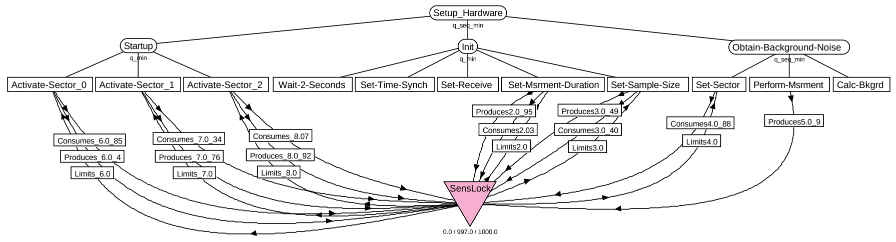

Figure 1.6. An abbreviated view of the sensor initialization TÆMS task structure.

TÆMS , the Task Analysis, Environmental Modeling and Simulation language, is used to quantitatively describe the alternative ways a goal can be achieved [Decker and Lesser, 1993, Horling et al., 1999]. 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 1.6 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 functions (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, and affected resources are also quantitatively described. The curved lines in figure 1.6 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 1.6. 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, 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. 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. 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. An important class of actions, so called meta-level activity, is missing from this list. Meta-level activities are the high-level functions which enable the lower-level activities. These include such things as scheduling, negotiation, communication, problem solving and planning. Without accounting for the time and computational resources these actions take, the schedule will be incomplete and susceptible to failure. In this study, we have begun accounting for these activities by including negotiation and coordination activities in our TÆMS task structures. From a scheduling and execution perspective, a negotiation sequence is just like any other action - it will have some expected duration and cost, a probability of success, and some level of required computational resources. By modeling negotiation sessions as a task structure, we are able to cleanly account for and schedule the time required to perform them, thus improving the accuracy of our schedules. In the future we will explore additional modeling of other meta-level activities, including planning and scheduling. We currently handle the time for these activities implicitly by adding slack time to each schedule. This is accomplished by reasoning with the maximum expected duration time for a given schedule, rather than the average time [Raja and Lesser, 2002].

3.3.1 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 be handled through changes to only a small subset of the schedule. Figure 1.5 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 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 [Lesser et al., 2000], or making use of an approximate base structure and then employing learning techniques to refine it over time [Jensen et al., 1999].

SRTA uses the Design-To-Criteria component [Wagner et al., 1998] to generate linear plans solving the goal described in the TÆMS structure. 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. Several components are used during this final scheduling phase. A resource modeling component is used to ensure that resource constraints are respected. A conflict resolution module reasons about mutually-exclusive tasks and commitments, determining the best way to handle conflicts. Finally, a schedule merging module allows the partial order scheduler to incorporate the actions derived from the new goal with existing schedules. Failures in this process are reported to the problem solver, which is expected to handle them. 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 decommiting from a lower priority goal or the goal causing the failure.

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. 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. The primary advantage of the partial order scheduler is its ability to quickly shift methods' execution order at any point in time instead of performing costly re-planning [Vincent et al., 2001]. 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. We also attempt to reduce scheduling overhead by caching and reusing plans from similar task structures.

3.3.2 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.

The slot-based notion of time is tightly connected with the resource allocation protocol, and will be covered in more detail in section 1.4.2.

4. Resource Allocation

4.1 Problem Solver

The problem solver is one of only a few of the components in the agent that is strictly domain specific. Its principle purpose is to coordinate and direct the activities of the agent as a whole, by initiating and providing input to other agent components. More specifically, the problem solver is responsible for ensuring the agent performs its duties within the organization, by reasoning about the environment and goals at hand. This includes managing the local sensor resource, managing the tracking of one or more targets (if assigned as a track manager), and handling the details of managing the sector (if acting as a sector manager). To accomplish each of these roles, the problem solver is composed of a Finite State Machine (FSM) controller, called the Pulse Action Controller (PAC), a set of FSMs, called pulse action machines (PAM), and a set of simple message handlers.

The Pulse Action Controller (PAC) is the heart of the problem solver. It is a pseudo-generic control component that manages the division of execution time to each of the concurrently executing roles of the agent. The name Pulse Action is derived from the execution cycle in JAF which "pulses" the problem solver at a somewhat fixed interval by passing control to it. When this happens, the PAC passes execution control to the individual PAMs that are currently registered within the agent. This not only allows for the division of execution time, but provides the basis for how our agents dynamically add and remove organizational roles. In addition to dividing execution time, the PAC provides PAM specific message routing. This feature allow messages to be specifically addressed to a role that the agent is handling. For example, when an agent is managing two tracks concurrently, the sensors can address the results of measurements they are taking to one of the two roles, avoiding the need to disambiguate the information. Lastly, the PAC provides a priority based shared locking mechanism. This feature allows a PAM to "lock" on specific keywords, which then cannot be locked on by a lower priority lock requests. This mechanism is most frequently used to signal that a critical section of processing on shared state has been entered and changes to that state should be avoided unless absolutely necessary.

A PAM is defined as $PAM = (A, C, \delta, a_0, F)$ where A is a set of actions, C is the set of input conditions, $\delta : A \times C \rightarrow A$ is the transition function, $a_0 \in A$ is an initial action, and $F \subseteq A$ is the set of final actions. Actions are not just a representation of state, but actual executing code that performs an action or set of actions before completing. Each of the PAMs in our system has been constructed by extending a basic or abstract PAM. The abstract PAM provides message and lock queue maintenance, action and transition monitoring, as well as PAM specific attribute/state storage. There are currently five possible transition conditions specified in the abstract PAM, which may be combined with each other using an implicit "or". They are time (transition to a new state once a specific time has been reached), messages (transition once the message queue has a particular number of messages), directory search (transition once the directory search has been completed), negotiation (wait until the negotiation has completed), and locks (wait until all of the locks have been obtained from the PAC). Negotiation is actually just a special form of lock, which is distinguished for better clarity. Figure 1.7 is an example of a PAM within our system. Here, A is represented by the two blocks, C is a target detect message, δ is represented by the arrows, etc.

The problem solver is also composed of a separate set of message handlers, which deal with most of the communication not addressed by a PAM. Based on the incoming message type, these message handlers perform a number of actions, which range from simply updating the agent's internal state, to spawning a new role for the agent.

4.1.1 Sensor Agent. The simplest role for an agent to handle, from the problem solver's perspective, is that of sensor agent. As a sensor, the agent is responsible for performing scan and tracking tasks that are assigned to it by the sector manager or a track manager, and for returning the results of those measurements to the agent that assigned the task. For the most part, these tasks are all performed through the use of the SRTA module described in section 1.3.2. The problem solver is only required to provide two functions as a sensor agent. The first is that of pre-filtering results from measurements being taken by the sensor heads, removing measurements which indicate the absence of a moving target. The second is to provide resource usage information to track managers that are currently in its schedule. This is accomplished by piggy-backing current schedule information onto results messages, which can then be used by the track managers to detect resource conflicts. Piggy-backing is a technique which is used to reduce the total number of messages being transmitted in the system by adding information onto routine, regular messages, at the expense of making those messages longer. For example, sensors piggy-back their current schedules onto results messages occasionally, thereby avoiding the need for a separate message.

4.1.2 Sector Manager. The sector manager plays a pivotal role in the organization of the agents in the system. As mentioned earlier, each sector manager is assigned a geographic area of the environment for which it provides directory services, scan scheduling, target resolution and role assignment. Directory services was covered in section 1.3.1.2, and scan scheduling mentioned in section 1.2. This section covers the last of these roles: target resolution and role assignment (see figure 1.7).

Target resolution is the process of determining if a reported positive detection discovered during scanning is actually a new target or one that is currently being tracked. When a sensor agent detects a target, it sends a target detection message to the sector manager, with the observed amplitude and frequency measurements, as well as the sector and time the detection occurred. The information contained in this message is sufficient to give the sector manager an estimated position of the potentially new target. It is the job of the sector manager to determine whether, given this vague information, the detection represents a new target in the environment or if the target is currently assigned to a track manager. To do this, the sector manager has to have an estimated position of each of the targets within its sector. This is accomplished by having the track managers occasionally report the location of their targets to their sector manager. Location reports come with a great deal of uncertainty; track managers can estimate the position of their targets incorrectly, reports become outdated, etc. We categorize this uncertainty by having a bounding circle that encompasses a region that the target is most likely to be in. As time goes on, with no further information updates, the bounding circle grows, representing increased uncertainty. The rate of the circle growth is in part associated with the last speed estimate of the target, and the center of the circle moves as a function of previous estimated target locations and headings. What should be clear from this is that target detection is an exercise in uncertainty. Setting the parameters for bound growth, initial detection bounds, reliance on estimated target velocity, and the frequency of positional updates were all done through extensive empirical testing.

Role assignment occurs whenever a detection is determined to actually be a new target. When this happens, the sector manager must choose an agent to take the role of the track manager. Choosing a track manager is done using several criteria. First, the sector manager uses estimates of the work that is being done by each of the agents. For example, the sector manager is less likely to choose itself as a track manager because it is already playing an important, time-consuming role in the environment. Next, agents which have previously been selected as track managers, but are no longer tracking a target become more likely to be assigned the new role. The reason for this is that agents that have previously tracked have more cached local knowledge of the agents within their sector than agents that have never tracked. Choosing an agent with this prior knowledge saves the sector manager the associated message traffic to update the new track manager's internal knowledge. Lastly, the sector manager chooses based on the new track manager's communication channel. Because the role of track manager is communication intensive, this acts to spread the messaging load of tracking multiple targets as evenly as possible across the available channels.

4.1.3 Track Manager. Because the system's resource allocation task depends on having good predictive data the role of track manager is a critical part of the architecture, and like other problem solver activities is performed using a PAM (see figure 1.8). The tracking behavior implemented in this system is a multi-part closed loop process. Positional estimates are created by fusing measurements from three or more sensors. These estimates are then used to model the target's behavior, which in turn is used to select the most appropriate sensors to

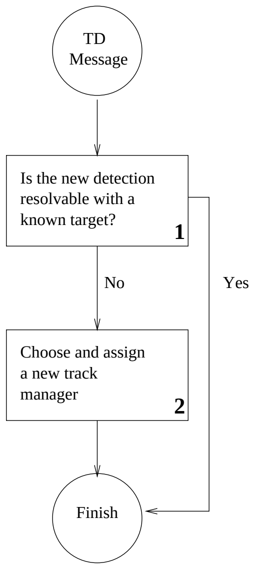
TD Message
Is the new detection resolvable with a known target?
1
No
Yes
Choose and assign a new track manager
2
Finish

Figure 1.7. The target detection pulse action is used to determine if a new track manager should be assigned and to whom the new role belongs.

take measurements. Ideally, if the positional estimates are accurate and the target's motion model is correctly created and applied, the track manager should be able to choose the best sensors to track the target in the future. If, on the other hand, the sensor selection is made poorly, such as one that cannot see the target, an inaccurate position estimate may be generated. This flawed position will likely corrupt the motion model and may, for instance, cause the track manager to choose two wrong sensors at some time in the future or to choose a sensor that can accidentally see a different target. Through such a cascade of small failures, the uncertainty in any part of the process can quickly degrade the tracking performance to a point where the target is no longer being tracked. A great deal of effort was spent in building, testing and modifying each of the components involved in the tracking process. This section discusses three of these components, namely the selection of resources (sensors), the fusion of data these sensors produce, and the construction of the motion model from that fused data.

Choosing the sensors needed to track a target is seemingly a straight forward process: pick the best sensors given the location of the target. Often, it is the case that actions that are simply stated and understood by humans are the hardest to accomplish in an intelligent software system. It turns out that this is the case for this problem as well. There are actually a number of steps and information that need to be obtained

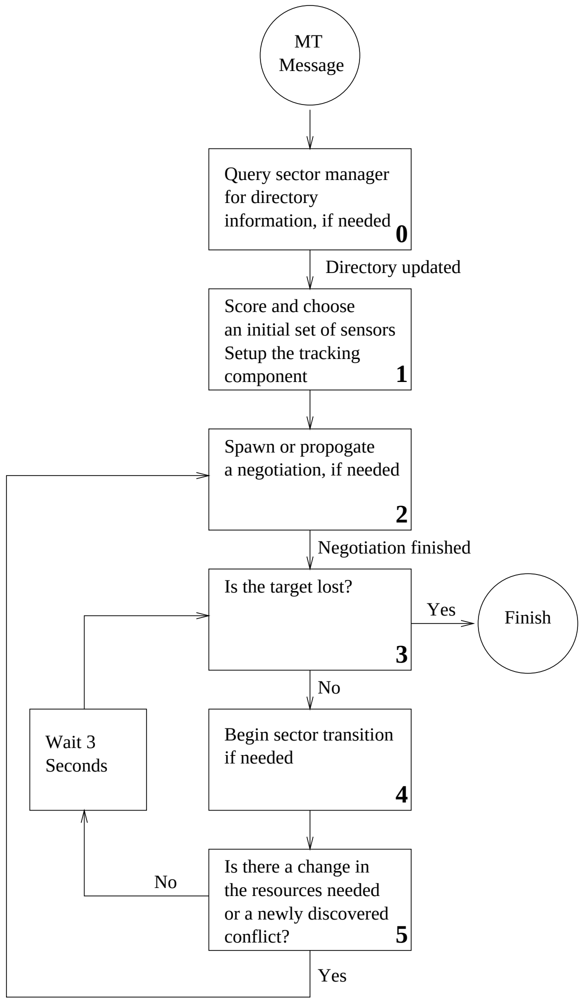
MT Message
Query sector manager for directory information, if needed
0
Directory updated
Score and choose an initial set of sensors Setup the tracking component
1
Spawn or propogate a negotiation, if needed
2
Negotiation finished
Is the target lost?
3
Yes
Finish
No
Wait 3 Seconds
Begin sector transition if needed
4
No
Is there a change in the resources needed or a newly discovered conflict?
5
Yes

Figure 1.8. The track pulse action starts and monitors the track until the target is determined to be lost. It also spawns negotiations and handles sector transitions when needed.

before an informed decision about which sensors are most appropriate can be made. These include knowing what and where the sensors are in the environment, knowing where the target is and will be, scoring the sensors to find the best among the available, and ensuring that the sensors are actually available to be used. The last of these will be covered in the next section.

As can be seen from figure 1.8, the first time a target is assigned to an agent, the track manager requests information from the sector

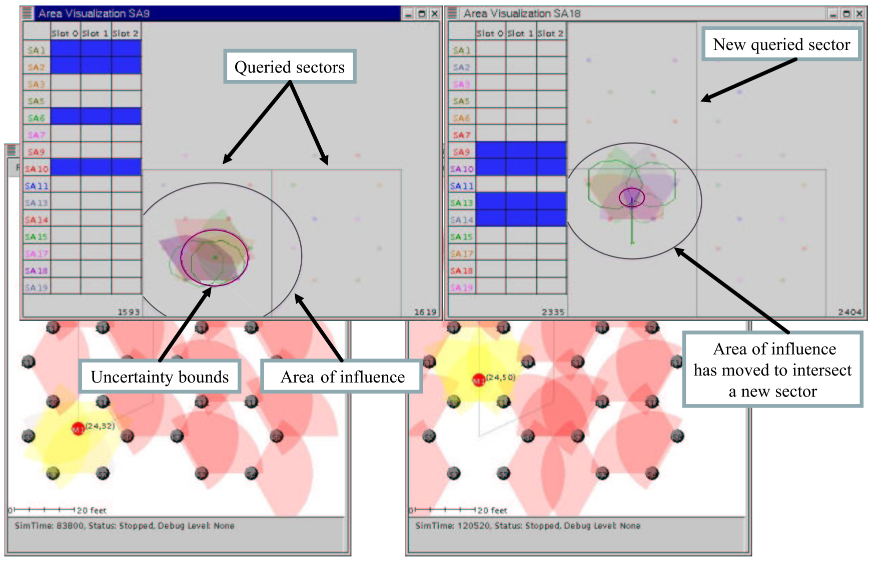
Area Visualization SA9
Area Visualization SA18
Queried sectors
New queried sector
Uncertainty bounds
Area of influence
Area of influence has moved to intersect a new sector
20 feet
SimTime: 83800, Status: Stopped, Debug Level: None
SimTime: 120520, Status: Stopped, Debug Level: None

Figure 1.9. A sector transition in progress. The figure on the left shows the area of influence before it intersects the top-left sector. After this happens, the right figure shows that sector and its sensors based on newly acquired directory information.

manager in the form of a directory service query. The query, similar to the one described in section 1.3.1.2, returns all the sensors available in that sector. This query serves two purposes. First, it provides the information needed to do sensor selection for the current target. Second, it updates the local directory services cache, which both makes that data available indefinitely and allows the agent to avoid future queries to the sector manager. As was mentioned earlier, once an agent obtains this sensor information locally, it becomes a more likely candidate for tracking future targets within the sector.

Obtaining information about the originating sector of a target is only part of knowing what sensors are available. As the target moves through the environment, it enters and exits other sectors. Two mechanisms exist to deal with this. The first is simply an area of influence (AOI) circle around the target, as shown in figure 1.9. This bounding circle, which is similar to, but much larger than the uncertainty bounding circle, is used by the track manager to signal that the target may be moving into a new sector or has moved far enough outside of an old sector to stop considering the target as being part of that sector. When the AOI of a target enters a new sector, the track manager registers its target with the sector manager of that sector. If the track manager does not already have the information, it may also query the sector manager to update its local listing of sensors available in that area. The second method involves actually moving the responsibility for the track to a new track manager closer to the target. This technique, which we call migrating, was incorporated because the communications range is both limited and theoretically degrades over distance. So, when a target and the sensors needed to track it are far enough away from the track manager, the track manager contacts the remote sector manager and hands off the tracking responsibility.

Once the track manager knows of the available sensors, it will select a subset with which it will coordinate to take measurements (this process is covered in detail in the following section). Each of these sensors will then take measurements and return them to the track manager, which must fuse the data in order to produce a track. Much of the complexity involved in this process is handled by a dedicated tracking component, which is covered elsewhere in this book. However, a few details are pertinent at the agent level. For instance, particular measurements may be excessively noisy - a characteristic which can be determined by the originating agent. A simple heuristic is to not send such measurements, which reduces load on both the communication medium and data fusion agent. Another particularly difficult problem faced by the sensor network is associating measurements with their correct targets. In a multi-target environment, the potential exists for an agent to unknowingly fuse measurements from one target into the track of another, which will adversely affect that track's quality. Our solution attempts to detect this situation by using the sector managers to distribute known targets' locations to track managers if their respective targets are close to one another. When a track manager receives a measurement, it can use that information to determine the probability that it is ambiguous, and may not correspond to the target it is responsible for. That information can be used to determine if the measurement should be discarded.

The next component of the sensor selection process is knowing where the target is or will be. While tracking the target's current location is important, actually obtaining that information relies upon measurements from the appropriate sensors, which requires an accurate prediction of the future location of the target so those sensors may be allocated. This prediction is necessary because of the latency introduced by communications. From the time a measurement is taken to the time it is used in determining a target's position, some delay occurs. So the "current location" of a target known by the tracking agent is actually its location some moments beforehand, and the delay imposed by selecting and committing with sensors means the tracking agent actually needs the location of the target at some point in the future.

Motion modeling is accomplished by obtaining a small sampling of recent location estimates from the track and performing a linear regression on them. There are actually two regressions happening: change in X position over time and change in Y position over time. This simple regression assumes that the X and Y relative velocities are independent of one another. Small sample sizes are used to allow the system to be more reactive to changes in both the speed and direction of the target. This, of course, causes the predictive model to be subject to large small-sample statistical errors. Some of these affects are compensated for by a second model of target speed maintained by the agent, which is used to appropriately scale the regressions in cases where the independence assumption and sampling error causes the speed of the target to be excessive. This has the overall effect of smoothing the speed while not effecting the rate of direction changes.

Even with these and other techniques, noisy data, lost communication, or incorrect interpretation can still cause the target to be lost. We have incorporated two ways of dealing with targets that are becoming untrackable. First, we use a fail-safe area intersection technique. The area intersection model takes measurements which are above the noise threshold, interprets them as graphical regions, and then computes the intersection of those regions. This provides a rough estimate of where the target is at any given moment, and can allow the track manager to refocus its attention in the right direction. This area intersection model is used as an override technique to the linear regressions. The other technique is simply to report that the target is lost. Although not an optimal technique, this method frees up potentially wasted resources and allows the sensors to scan their areas to potentially rediscover the missing target.

Ultimately, the target's location estimate is simply that - an estimate. To represent this fact, target locations are always reasoned about as a bounding circle, centered around the maximum likelihood predictor of the linear regression motion model. The radius of the circle is associated with the relative velocity of the target. The degree of association or, put another way, our belief in the predictive capabilities of the motion model is a tunable parameter. In addition, the size of the circle has a lower bound to model the inherent uncertainty introduced by the noise of the sensors and the triangulation process, even if the supporting data are seemingly accurate. This parameter is also tunable.

Finally, the last part of selecting the sensors is scoring or ranking their appropriateness for tracking the target. Given the target bound, the sensor locations and orientation, the scoring function calculates the average expected amplitude of taking a measurement from a particu-

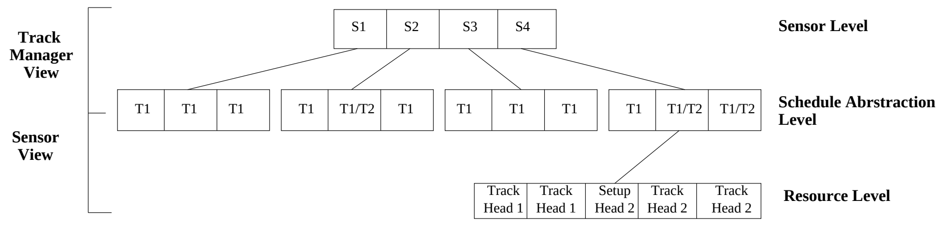

Figure 1.10. The three levels of abstraction used to negotiate when tracking a target. The top level shows track manager T1's sensor level abstraction wanting to use sensors S1, S2, S3, and S4. All conflicts could not be resolved at a higher level, so sensors S2 and S4 are left to resolve the conflict between T1 and T2 at the next lower level. Sensor S4 chooses to resolve its conflict at the resource level by squeezing in two track tasks from T1 followed by two from T2.

lar sensor/platform combination. The expected amplitude is calculated using a model of the sensor's performance that takes into account the distance and relative angle of the target from the sensor head. Expected amplitude is appropriate because higher signal to noise ratios provide greater ability to distinguish between small variations in distance or angle which means less uncertainty in the the final location estimate. In the end, the score results in a ranking of the sensors relative to one another[2]. The next section discusses the final part of the tracking problem, namely, how to ensure enough measurements are taken for each target when conflicts for the same sensors exists.

4.2 SPAM

4.2.1 Abstraction. Our approach to solving the allocation problem created by this environment involves viewing the problem at different levels of abstraction (see figure 1.10) depending on the amount of available time. The highest level, the sensor level, is maintained by the individual track managers and strictly focuses on which sensors are needed and desired to track the target. Solutions created at this level ignore the details of the individual sensors' schedules in making choices of how to allocate resources and simply choose based on the track managers internal requirements.

Below the sensor level is the schedule abstraction level. Here, tasks are viewed as periodic and resource scheduling is done at a coarse, slot-

[2]Certainly, more advanced methods of describing the uncertainty of a target's location and the subsequent selection of sensors could be used. A change of this nature does not affect the overall architecture in a significant way.

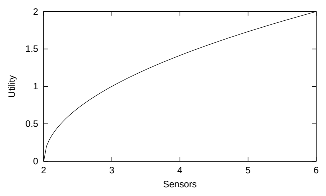

Figure 1.11. Utility of taking a single measurement from T_a sensors.

based granularity. Within the sensor platforms, agents maintain this abstraction level by using the Periodic Task Controller (PTC), which provides SRTA with tasks at times that are appropriate to the schedule. The PTC is capable of autonomously resolving conflict by using one of several techniques, including shifting slot boundaries, selecting tasks to execute based on importance level, or temporarily shifting a task to empty slots in its schedule. It also implicitly uses the alternative solution generation capabilities provided by the Design-to-criteria planner used by SRTA. This allows the PTC to handle some level of resource conflict, which we call slot co-binding, that may be left unresolved by high level negotiation.

At the lowest level, the resource level, all of the minute details of task execution and resource usage within the sensors are scheduled using SRTA. If scheduling conflicts reach this level of abstraction, the partial order scheduler (POS) can shift the task execution to try to eliminate any remaining conflict. Conflicts at this level can be created when the sensor is working on meta-level tasks that are not explicitly reasoned about at the schedule abstraction level.

4.2.2 Utility. To clarify what our protocol is attempting to achieve it helps to see how utility is measured in the tracking domain. As mentioned previously, 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 the estimate by the function given in figure 1.11. Increasing the measurements taken in a given period of time yields a linear increase in the overall quality of the track.

If we say that T_a is the number of sensors that took measurements leading to the positional estimate and T_s is the number of times they are taken in a given period of the abstract periodic schedule, then we can quantify this relationship by the following formula:

$$Util(Track) = Util(T_a) \times T_s$$

In fact, track managers within the system use this measurement as the basis for deciding what objective utility level to try to achieve for tracking a specific target. We will often denote the objective level as $D_a \times D_s$ denoting the number of agents desired for the number of slots in the schedule abstraction level. For example, a track manager may wish to have three agents for two slots of the schedule abstraction level denoted 3×2. For this domain, we set the number of slots in the schedule abstraction level to match the number of sensor heads on each platform, which is three.

Looking at this utility function it should be noted that co-binding can have a profound effect on the quality of a track. In fact, because the sensors make the decision about which track to satisfy on each period of their periodic schedule, having more than one sensor co-bound for a particular slot causes a near random occurrence of synchronization. This relationship can be seen in the following formula. Here S is the set of slots in the abstract schedule level and T_i^s is the number of actual measurements that are taken during a given slot s .

$$Util(Track) = \sum_{s \in S} \sum_{a=3}^{\infty} Prob(T_i^s = a) Util(T_i^s)$$

Note that if the utility of a particular track is 0 by the above formula, we actually penalize ourselves for not tracking the target by returning a value of -1 instead. In addition, the lower bound on the number of sensors needed to track is three. As the formula specifies, tracking with 0, 1 or 2 sensors does not add to the utility of the track[3].

Finally, the global utility can be calculated by summing the utilities for the individual tracks (one track per target).

4.2.3 Protocol. To meet the objectives of the environment and to incorporate the techniques that were discussed in the previous sections, the Scalable Protocol for Anytime Multi-level (SPAM) negotiation is divided into three stages. As the protocol transitions from stage to stage, the agent acting as the track manager gains more context information and therefore is able to improve the quality of its overall decision. After each stage or at any time during stage 2, the track manager can choose to stop the protocol and is ensured to have a solution - albeit not necessarily a good one (not necessarily conflict free). Figure 1.12 shows the amount of information that the track manager has at each stage of the protocol. The figure shows that as the amount of information obtained increases, the track manager is able to shift its negotiation

[3]This isn't strictly the case using the Bayesian tracking module provided by USC.

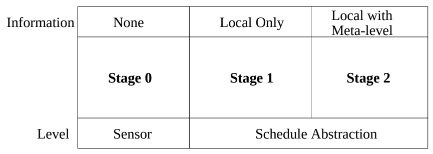

Figure 1.12. The three stages of SPAM showing the information that is available and the level of abstraction the track manager uses in generating possible solutions.

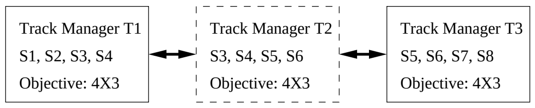

Figure 1.13. Example of a common contention for resources. Track manager T2 has just been assigned a target and contention is created for sensors S3, S4, S5 and S6.

abstraction level. This means that if the track manager chooses to terminate the protocol before stage 1, it acts at the sensor level of abstraction (deciding on only which sensors it desires) and leaves the decision of how to handle the actual scheduling to the sensor agents themselves as was discussed in the previous sections.

4.2.4 Stage 0 & 1. The best way to explain the operation of the protocol is through an example. Consider figure 1.13, 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 and there are no other targets in the environment. T1 and T3 have already bound every one of the slots in each of their desired sensors. This creates contention for sensors S3, S4, S5, and S6.

The protocol is started whenever the track manager (T2) is assigned the new target and begins stage 0. Stage 0 is primarily responsible for viewing the problem at the sensor level. Because of this, each of the sensors that have the potential to track the target are evaluated and ordered using domain specific measures. In this stage, the track manager also assigns an initial objective level to the track. Objective levels in general are derived from the track manager's objective function. This function, which may be different for every track manager, defines the order of the objective levels, the initial objective level for a track, and a lower bound of the objective level before giving up on an unconflicted solution. In our example, T2 prunes the list of available sensors down to just 4 that are able see the target, namely S3, S4, S5, S6, and sets its initial objective level to the highest possible level, desiring all 4 sensors

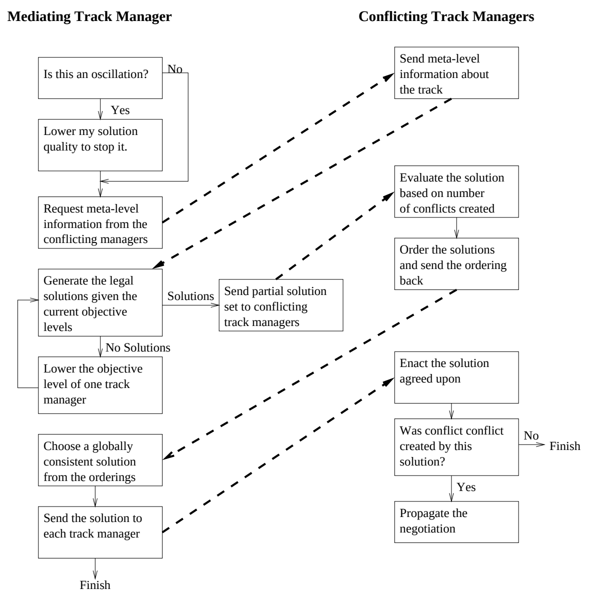
Mediating Track Manager
Conflicting Track Managers
Is this an oscillation?
No
Yes
Lower my solution quality to stop it.
Request meta-level information from the conflicting managers
Generate the legal solutions given the current objective levels
Solutions
No Solutions
Lower the objective level of one track manager
Send partial solution set to conflicting track managers
Choose a globally consistent solution from the orderings
Send the solution to each track manager
Finish
Send meta-level information about the track
Evaluate the solution based on number of conflicts created
Order the solutions and send the ordering back
Enact the solution agreed upon
Was conflict conflict created by this solution?
No
Finish
Yes
Propagate the negotiation

Figure 1.14. Stage 2 of the SPAM negotiation protocol resolves all local conflict at the schedule abstraction level through negotiation with conflicting track managers

for each of the 3 potentially available slots. The manager perceives itself to have more time so it goes onto stage 1.

Stage 1 of the SPAM protocol begins by obtaining abstract schedule information from the PTC in each of the sensor agents. This information is used in two ways. First, if a solution at the current objective level can be obtained, the track manager can bind the solution and avoid a more costly track manager-to-track manager negotiation. Second, if a solution cannot be found at the current objective level, the track manager has enough information to bind a solution that minimizes the unresolved conflict given the current objective level. Like stage 0, the negotiation session can be terminated at the end of stage 1 if insufficient time is available to continue. Track manager T2, following the protocol, obtains the current schedule information for each of the sensors it desires. In this example, it is unable to find a conflict free solution, binds at its current objective level, causing a great deal of conflict in the sensors and therefore it proceeds to stage 2. This binding, though not conflict free, will allow tracking to begin for T2 at some degraded level, depending on the local sensor agent's ability to resolve the conflict.

4.2.5 Stage 2.

Stage 2 is the heart of the protocol (see figure 1.14). This stage attempts to resolve all local conflict that a track manager has by elevating the negotiation to the track managers that are in direct conflict over the desired resources. To do this, the originating track manager takes the role of the negotiation mediator for the local conflict (multiple negotiations 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 possible solutions which may involve changes to the objective levels of the managers involved, and finally choosing a solution to apply to the problem. Because the solutions are generated without full global information, however, the final solution may lead to newly introduced non-local conflict. If this occurs, each of the track managers can choose to propagate the negotiation by becoming the mediator of a new negotiation to resolve the newly introduced conflict if they have the time. So, what started out as a new target or resource requirement, may lead to the negotiation propagating across the problem landscape.

Viewing the behavior of SPAM stage 2 from the global perspective, where there may be long chains of interacting subproblems, SPAM works by solving local resource conflicts while attempting to prevent the conflicts from spreading through the chain. This behavior is similar to the work of [Minton et al., 1992], who randomly chose a queen and used a min-conflict heuristic to solve the n-queen problem. The chief difference between that work and this is that the mediator, with only a limited view, has no way to measure the overall effect of attempting to minimize its own conflict.

Continuing the example in figure 1.13 and following the protocol, track manager T2, as the originator of the conflict, takes on the role of negotiation mediator. After the mediator concludes the oscillation detection phase (explained later in this section), it begins the solution generation phase by requesting abstract information from all of the track managers that are involved in the resource conflict. This information includes the manager's current objective level, the number of sensors that can see the target, the names of the sensors that are in direct conflict with the mediator, and the number of external conflicts (i.e. conflicts on the sensors not being mediated over) 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.

Using this information, T2 begins to generate solutions to the resource problem. Here, a solution refers to one that includes all of the track managers (T1, T2, T3) for all of the sensors (S3, S4, S5, S6) that the mediator is able to directly interact with (an example can be seen in figure 1.16). In addition, when a solution is created, it is unconflicted over those sensors. These solutions represent complete solutions for the mediator (T2) and potential partial solutions for the other track managers (T1 and T3) since the solution does not involve sensors that are not in conflict with T2's desired allocation (i.e. S1 and S2 for T1, and S7 and S8 for T3). In fact, the mediator assumes that sensors not being directly considered as part of the mediation are freely available to be used. In other words, T2 assumes that sensors S1, S2, S7, and S8 currently only have commitments for tracks T1 and T3.

Track manager T2 enters a loop that involves attempting to generate solutions followed by lowering one of the track manager's objective level, if no full solutions are possible 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 non-conflicting solutions are unavailable. Currently this is done by first choosing the track manager with the highest objective level and lowering their level. This has the overall effect of balancing the objective levels of the track managers involved in the negotiation. 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. External conflict is measured by information given to the mediator by track managers about current conflicts over sensors not currently being considered in the mediation. By doing this, it is our belief, that track managers with more external conflict will maintain higher objective levels, which leaves them more room to compromise in subsequent negotiations as a result of propagation of these conflicts.

The solution generation loop is terminated under one of two conditions. First, if given the current objective levels for each of the managers, a set of full solutions is available, the negotiation enters the next phase. Second, the objective levels of the track managers cannot be lowered any further and no full solutions are available. Under this condition, the negotiation session is terminated and the mediator takes a solution at its lowest objective level, 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 of set 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 full solutions to the problem (an example of which can be seen in figure 1.15).

	Slot 1	Slot 2	Slot 3
S3	T2		T1
S4	T2	T1	T2
S5	T2		T2
S6	T3	T3	T2

Figure 1.15. One of the proposed solutions derived by mediating manager T2 to the problem in figure 1.13.

During the solution evaluation phase, the mediator sends each of the track managers its set of partial solutions that are part of the full solutions generated in the previous phase. Each track manager, using this information and the proposed objective level, can then determine what partial solutions, if any, are acceptable. In our example, T2 sends 24 partial solutions to T1 for sensors S3 and S4, 24 partial solutions to itself for sensors S3, S4, S5, and S6, and 24 partial solutions to T3 for sensors S5 and S6. In our current implementation, each of the track managers orders its partial solutions from best to worst based on the number of new conflicts that will be created and the number of changes that will have to be made in order to implement the new allocation. The ordering is then returned to the mediator. Currently, we are 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 partial solution orderings from the track managers, it is able choose the final solution to apply to the problem. Using the orderings, the mediator prunes the solution set generated in the solution generation phase by only keeping solutions that contain the highest ranked partial solution for the track manager with the most external conflicts. This new reduced set of solutions is then pruned by the mediator to contain only solutions that have the highest ranked partial solution from the second most externally conflicted track manager. This process continues until only one solution remains in the solution set.

In our example, T2 collects the ordering from T1, T2, and T3. Choosing based on the same ordering that was used to reduce the objective levels, T3 is given first choice. By its ordering it ranked its partial solution 0 the highest. This restricts the choice for T2 to its partial solutions

	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

$\longrightarrow$

	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 1.16. A solution derived by SPAM to the problem in figure 1.13. The table on the left is before track manager T2 negotiates with T1 and T3. The table on the left is the result of stage 2 negotiation.

0, 1, 2, and 3 because only these partial solutions continue to provide a solution. T2 ranked 0 ranked from this set, leaving T1 to choose between its 0th, 1st, and 2nd partial solutions. It turns out that T1 likes its 0th solution the best; so the final solution that is applied is composed of T3's partial solution 0, T2's partial solution 0, and T1's partial solution 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 a negotiation if it chooses to. Currently, track managers will propagate if a new conflict has been created as a result of the final solution choice. In future versions, it is our hope that utility, and not conflicts, will form the basis for determining when to propagate. Figure 1.16 shows the original configuration of the sensors before T2 was introduced and after stage two completes.

As mentioned earlier, stage 2 starts in the oscillation detection phase. Oscillation can occur because conflicts are resolved locally without regard to the global context. Consider if in our previous example, track manager T1 originated a negotiation 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 negotiates with T2, T1 in the end gets a locally unconflicted solution, but in order for that to occur, T2 ended up in conflict with T3. It is possible that when T2 propagates the negotiation, that the original conflict between T1 and T2 is reintroduced, leading to an oscillation.

To prevent this from happening, each track manager maintains a history of the sensor schedules that are being negotiated over whenever a negotiation terminates. By doing this, managers are able determine if they have previously been in a state which caused them to propagate a negotiation in the past. To stop the oscillation, the propagating manager lowers its objective level to force itself to explore different areas of the solution space. It should be noted that in certain cases oscillation may be incorrectly detected using this technique which can result in having the track manager unnecessarily lower its objective level.

4.2.6 Generating Solutions. Generating full solutions for the domain described earlier involves taking the limited information that was provided through communications with the conflicting track managers and assuming that the sensors which are not in direct conflict, 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. The formula below gives the basic form for how partial solutions are generated for each track manager. Here, A_s is the number of slots that is available in the schedule abstraction layer, D_s is the number of slots that are desired based on the objective level for the track manager, A_a is the number of sensors available to track the target (those that can see it), D_a is the number of sensors desired in the objective function, and finally C_a is the number of sensors under direct consideration because they are conflicting.

$$Solutions = \binom{A_s}{D_s}\left(\sum_{i=max(0,D_a-A_a+C_a)}^{min(C_a,D_a)} \binom{C_a}{i}\right)^{D_s}$$

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. 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 generating partial solutions in stage 0 and 1 of the protocol. This special case generation is actually done by simple setting $C_a = A_a$. The formula above, in this case reduces to:

$$Solutions = \begin{pmatrix} A_s \\ D_s \end{pmatrix} \begin{pmatrix} A_a \\ D_a \end{pmatrix}^{D_s}$$

Partial solutions can also be generated when there are a number of pre-existing constraints on the usage 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 partial solutions generated for a given track manager allows us to generate full solutions for the track managers in stage 2. By ordering the track managers, we can generate partial solutions for them one at a time using the results from higher precedence track managers as constraints for lower precedence ones. Continuing our example from figure 1.13, say that T1 had one external conflict and T3 had two. When the full solution set is generated, T2 generates partial solutions for manager T3 first. T2 then uses the results from this as constraints on the creation of partial solutions for T1. The resulting full solutions (now with solutions for T1 and T3) are used as constraints for generating the partial solutions for T2 (which only has local conflict because it is the mediator).

This process forms the basis of a search for full solutions to the local conflict. You can view this as a tree based search where the top level of the tree is the set of partial solutions for the most constrained track manager. Each of the nodes at this level may or may not have a number of children which are the partial solutions available to the second most constrained track manager and so on. Only branches of the tree that have a depth equal to the number of track managers - 1 are considered full. If there are no branches that meet this criteria, then the problem is considered over constrained.

In the end, we are left with a directed acyclic graph where every path from the root nodes to the leaves has equal length (number of track managers - 1) and represents one unique full solution. The nodes at a particular path length represent the set of partial solutions that a track manager has to choose from during the solution evaluation phase of stage 2.

5. Results

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 look at the results produced just before the tracking results are produced. Specifically, we look at the number and quality of measurements that were taken, which reflects how well both the allocation process (SPAM) and the control architecture (SRTA) performed in practice independent of the actual tracking output.

To perform the evaluation, we quantitatively determine the quality of a scenario by traversing each track's timeline and calculating a "goodness" metric for it. This goodness metric attempts to capture how well that target was tracked by first partitioning the measurement data into a number of one second slots based on the time each measurement was taken. The analysis iterates through these slots, determines how many measurements were taken for that track during the slot's window of time, and computes the slot's score with the following function

$$slot_i = \sqrt{|slot_i| - 2}$$

where $|slot|$ is the number of measurements in the slot. To compute the track's score, the function groups the sequence of slots into sets of three, and determines the average score for that period.

$$period_j = (\sum_{i=0}^{2} slot_{3j+i})/3$$

If the average score for a period is 0, which indicates that no measurements were taken for that target during those three slots, the period score is set to -1, which is effectively a penalty for failing to track the target. The score for the track as a whole is computed as the average scores of that track's periods.

$$track_k = (\sum_j period_j)/|track_k|$$

where $|track|$ is the number of periods in that track. Finally, the aggregate score for the scenario as a whole is the average goodness of the tracks in that scenario, each weighted by the length of the track:

$$scenario_l = 1 + \sum_k (track_k \times |track_k|)/\sum_k |track_k|$$

Figure 1.17 shows results using this goodness metric. This graph represents data gathered from 400 experimental runs, which compare three

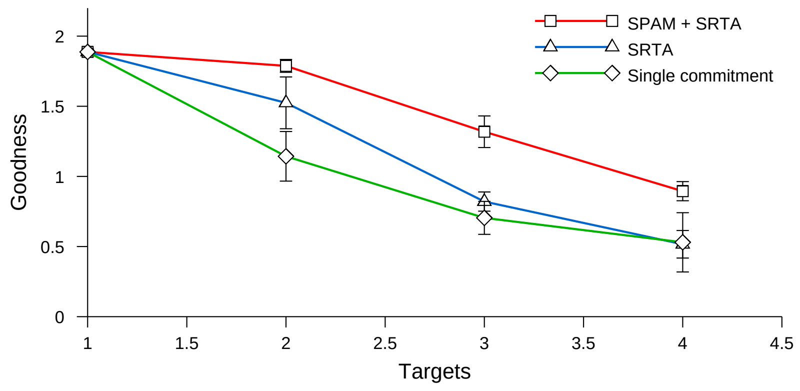

Figure 1.17. Goodness metrics for three allocation styles under various conditions.

different allocation techniques under different target densities in order to highlight the benefits of local and negotiation-based resource allocation. The environment, shown in figure 1.18, consists of eight sensors and from one to four targets, and each run was three minutes in duration. This particular setup was selected to maximize contention with minimal possibility of target ambiguity, creating a need for proper sensor allocation to effectively track. In the simplest allocation process, single commitment, each sensor can work on only one commitment at a time. New commitments accepted by a sensor override any existing ones. In the SRTA experiments, agents 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 SPAM + SRTA allocation style represents the full solution as presented in this chapter. 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.

As seen in the graph, when there is only one target, all three of the methods do equally well. Of course, this is because there is no contention for the sensors. The score of about 1.8 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 period score becomes -1, lowering the average for the track.

As the number of targets increases, the effects of contention for resources becomes clear and the difference in the methods also becomes clear. For two targets, the single commitment strategy, essentially ig-

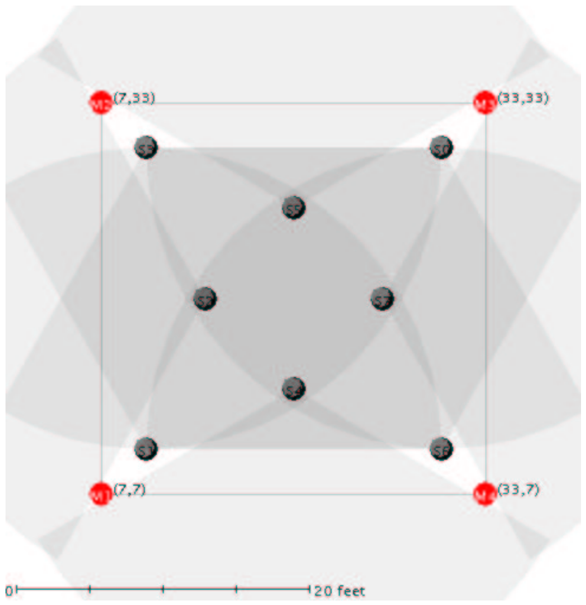

Figure 1.18. Radsim environment with 8 sensor nodes and 4 targets.

nores one of the targets whenever contention for the sensors occurs. The SRTA only strategy has better performance than the single commitment strategy because although not a conflict free solution, the local agents are able to get coordinated measurements occasionally. 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 performance characteristics of the strategies become more distinguished. 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 to a level where at least one of the targets is essentially ignored. However, the degradation is not as serious as for the other two techniques.

6. Conclusions

In this chapter we have described 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 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 the 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.

Although not mentioned here, the use of both general and DSN-specific debugging tools was of critical importance. The complexity of individual agents in addition to their distributed interactions made it very difficult to identify symptoms and diagnose problems. A number of visualization and debugging tools were therefore used to facilitate this process. Our experiences using these tools are briefly covered in the Visualization chapter earlier in this book.

Our solution as described covers many different aspects of the distributed sensor interpretation problem, including several which we did not anticipate when starting this project. Despite this, there remain parts of the domain which we have not yet explored that seem to offer the possibility of additional 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.

Acknowledgments

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.

We would like to acknowledge the following people for their important contributions to this research. 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.

References

[Bordini et al., 2002] Bordini, R., Bazzan, A., Jannone, R., Basso, D., Vicari, R., and Lesser, V. (2002). Agentspeak(xl): Efficient intention selection in bdi agents via decision-theoretic task scheduling. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2002)*, Bologna, Italy.

[Decker and Lesser, 1993] Decker, K. S. and Lesser, V. R. (1993). Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance, and Management*, 2(4):215–234. Special issue on "Mathematical and Computational Models of Organizations: Models and Characteristics of Agent Behavior".

[Horling and Lesser, 1998] Horling, B. and Lesser, V. (1998). A reusable component architecture for agent construction. Master's thesis, Department of Computer Science, University of Massachusetts, Amherst. Available as UMASS CS TR-98-30.

[Horling et al., 1999] Horling, B., Lesser, V., Vincent, R., Raja, A., and Zhang, S. (1999). The tæms white paper. http://mas.cs.umass.edu/res-earch/taems/white/.

[Horling et al., 2002] Horling, B., Lesser, V., Vincent, R., and Wagner, T. (2002). The soft real-time agent control architecture. Computer Science Technical Report TR-02-14, University of Massachusetts at Amherst.

[Jensen et al., 1999] Jensen, D., Atighetchi, M., Vincent, R., and Lesser, V. (1999). Learning quantitative knowledge for multiagent coordination. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, Orlando, FL. AAAI.

[Lesser et al., 2000] Lesser, V., Horling, B., Klassner, F., Raja, A., Wagner, T., and Zhang, S. (2000). Big: An agent for resource-bounded information gathering and decision making. In *Artificial Intelligence Journal, Special Issue on Internet Information Agents (also available as UMass Computer Science Technical Report 1998-52)*, volume 118, pages 197–244. Elsevier Science.

[Mailler et al., 2001] Mailler, R., Vincent, R., Lesser, V., Shen, J., and Middlekoop, T. (2001). Soft real-time, cooperative negotiation for distributed resource allocation. In *Procceedings of the 2001 AAAI Fall Symposium on Negotiation.*

[Minton et al., 1992] Minton, S., Johnston, M. D., Philips, A. B., and Laird, P. (1992). Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3):161–205.

[Raja and Lesser, 2002] Raja, A. and Lesser, V. (2002). Meta-level control in multi-agent systems. *Proceedings of AAAI/KDD/UAI-2002 Joint Workshop on Real-Time Decision Support and Diagnosis Systems. (Also UMass Computer Science Technical Report 01-49, Nov. 2001)*, WS-02-15:47–53.

[Sims et al., 2003] Sims, M., Goldman, C., and Lesser, V. (2003). Self-organization through bottom-up coalition formation. In *Proceedings of Second International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2003).* To appear.

[Sycara et al., 1997] Sycara, K., Decker, K., and Williamson, M. (1997). Middle-agents for the internet. In *Proceedings of IJCAI-97.*

[Vincent et al., 2001] Vincent, R., Horling, B., Lesser, V., and Wagner, T. (2001). Implementing soft real-time agent control. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 355–362.

[Wagner et al., 1998] Wagner, T., Garvey, A., and Lesser, V. (1998). Criteria-Directed Heuristic Task Scheduling. *International Journal of Approximate Reasoning, Special Issue on Scheduling*, 19(1-2):91–118. A version also available as UMASS CS TR-97-59.

[Zhang et al., 2000] Zhang, X., Raja, A., Lerner, B., Lesser, V., Osterweil, L., and Wagner, T. (2000). Integrating high-level and detailed agent coordination into a layered architecture. Lecture Notes in Computer Science: Infrastructure for Scalable Multi-Agent Systems, pages 72–79. Springer-Verlag, Berlin. Also available as UMass Computer Science Technical Report 1999-029.