Cooperative Negotiation for Optimized Distributed Resource Allocation in Soft Real-Time [*]

Roger Mailler
University of Massachusetts
Department of Computer Science
Amherst, MA 01003
mailler@cs.umass.edu

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

ABSTRACT

In this paper we describe a cooperative negotiation protocol that solves a distributed resource allocation problem while conforming to soft-real time constraints in a dynamic environment. By framing the allocation problem in terms of optimization, we have been able to develop a number of techniques to address an ever changing problem landscape. Amongst these techniques are the ability to resolve conflict in the allocation of resources on multiple levels, temporarily binding and, given time constraints, incrementally improving the quality of the solution (a form of distributed hill climbing), and restricting the context of negotiations to only use local information with extended meta-level data to generate and propose possible solutions to the problem. We describe the implementation of a simulator for the protocol, the more pragmatic experiences of implementing it in a real system, and present experimental results.

1. INTRODUCTION

Resource allocation is a classical problem that has been studied for years by Multi-agent Systems researchers [4]. The reasons for this is that resource allocation is difficult and time consuming to do in a centralized manner when the environment is dynamic and the time or cost of centralizing the information needed to generate a solution is considerable. Negotiation, a form of distributed search [3], has been viewed as a viable alternative to handling this complex search through a solution space that includes multi-linked interacting subproblems.[1] Researchers in this domain have focused primarily on resource allocation problems formulated as distributed constraint satisfaction problems[6]. In this work, we extend this classic formulation in two ways. First, we introduce dynamic soft-real time constraints which requires the negotiation protocol to adapt to the available time left. This estimation is determined dynamically as a result of emerging environmental conditions. Second, we reformulate the resource allocation problem as an optimization problem in which there are a range of acceptable solutions with varying preferences.

In this paper, we present a negotiation protocol that exploits the fact that the agents within the system are cooperative and have the ability to resolve conflicts internally. This means that some level of conflict can be left unresolved as a result of negotiation, but the agent faced with the conflict must resolve it based on its local perspective. We are not making the assumption that these internally based solutions obtain the best possible results, but that they are capable of providing some measure of utility while a better solution is obtained. The ability to create temporary solutions and incrementally improve them both locally and globally forms a distributed hill climbing search through the solution space that optimizes based on the demands of the soft real-time environment. In this context, soft real time should be interpreted as soft deadline, which means that finishing a task a bit early or late does not result in detrimental effects.

Our negotiation protocol is based on three major principles which allow it to operate under soft-real time constraints in a dynamic environment. First, we limit the context of the negotiation such that allocation problems are always resolved locally with only limited information about interacting subproblems being considered. After local negotiation is finished, each of the agents can choose to propagate the negotiation in an attempt to resolve conflict that may have been created as a result of the originating negotiation. Viewing this activity from the perspective of the global problem, each of the agents that propagates the negotiation is in essence locally optimizing in an attempt to reach a global optimum, which is form of distributed hill climbing. Second, local negotiations are conducted at multiple levels of abstraction. Agents can choose to resolve the conflict at different granularities and if they are unable to resolve it at one level, because of limited time, can leave

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

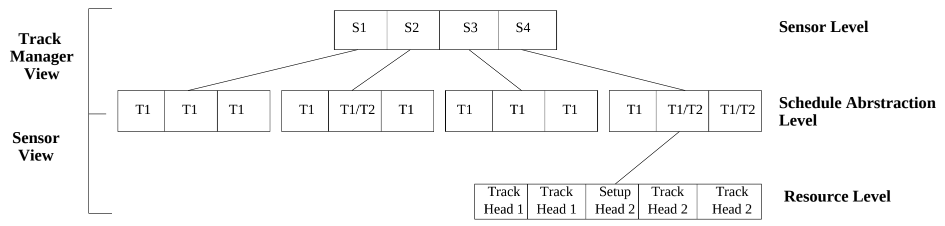


Figure 1: *The three level of abstraction used to negotiate when tracking a target. The top level shows track manager T1's sensor level abstraction wanting to use sensors S1, S2, S3, and S4. All conflict could not be resolved at a higher level so sensors S2 and S4 are left to resolve the conflict between 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.*

it to be handled at lower levels. Lastly, we have designed the negotiation protocol to exploit the previously mentioned techniques to have an anytime flavor. By having the ability to take solutions that may have unresolved conflict and still obtain some quality, we can actually bind temporary solutions while attempting to resolve the conflicts at a lower level. Clearly in a dynamic domain this has the ability to "buy some time" so that a good solution can be obtained while not completely abandoning processing that has some value in the mean time.

In the remaining sections of this paper, we introduce a distributed monitoring and tracking application which motivated the development of our protocol. Next, we describe the Scalable Protocol for Anytime Multi-level negotiation (SPAM). In section 4, we will describe an abstract model of the task environment that was used to develop and test SPAM in addition to some of the early results we have obtained. In section 5, we will discuss some of the details and difficulties of implementing a protocol of this type in a real system. The last section of the paper will present conclusions and future directions for this work.

2. DOMAIN

The problem that we are exploring is that of allocating sensor time to the task of tracking targets. In this problem, multiple sensors platforms are distributed with varying orientations throughout a real time environment [2]. These platforms have three radar based sensors each with a viewable 120 degree arc, which are capable of taking amplitude (measuring distance from the platform) and/or frequency (measuring the relative velocity of the target) measurements. In order to track a target, and therefore obtain utility, at least three of the sensor platforms must take a coordinated measurement of the target which is then fused to triangulate the target's position. Having more sensor heads, taking measurements more often, or having tighter relative synchronization of the measurements yields better overall quality in estimating the targets location and a more optimal result. The sensor platforms are restricted to only taking measurements from one sensor head at a time with each measurement taking about 500 millisecond. These key restrictions form the basis of the resource allocation problem.

Each of the sensor platforms is controlled by a single agent which may take on multiple organizational roles in addition to managing its local sensor resources. Each of the agents in the system maintain a high degree of local autonomy, being able to make trade-off decisions about competing tasks using our Soft Real Time Architecture (SRTA pronounced Serta)[5].

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

To lend to the dynamic characteristics of this problem, targets continuously moves through the environment as a scenario unfolds. This means that during the course of a run, targets move from the viewable range of some sensors to others. This, of course, means the actual allocation problem changes in structure during the course of a run as the track managers alter their resource requirements due to the discovery of new targets and the movement of existing ones. In addition, the dynamics drive the need for real-time negotiation because a particular problem structure is valid for only a limited amount of time.

Contention is introduced when more than one target enters the viewable range of a single sensor platform. Because of the time it takes to perform a measurement and the one measurement at a time restriction, track managers have to come to some sort of agreement about how to split the resource while still being able to track their target. This local agreement can have profound global implications. For example, what if as part of the local agreement one track manager completely relinquished control of a sensor platform and takes another instead? This may introduce contention with another track manager which could propagate through the entire environment.

2.1 Abstraction

The actual resource allocation problem that is created by this environment can be view at different levels of abstraction (see figure 1). At the highest level is the sensor level. This 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. Of course, since these solutions are created without information about what the sensors are actually doing, they are almost never free of conflict.

The next level of abstraction is the schedule abstract level. At this level, tasks can be viewed as periodic (which track-

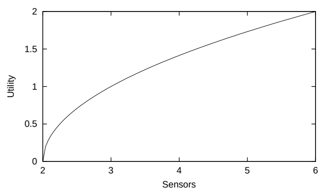


Figure 2: *Utility of taking a single measurement from T_a sensors.*

ing is) and resource schedules can be viewed at a coarse slot based granularity (all measurements take approximately the same amount of time). Within the sensors platforms, our agents maintain the schedule abstraction level by using the Periodic Task Controller(PTC). The PTC is a slot based, periodic scheduler that feeds SRTA with tasks at times that are appropriate to its schedule. The PTC is capable of 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 is easy to see that if a negotiation ends in unresolved conflict, which we call a co-binding, that the PTC has some capability to resolve the conflict. For instance, if two track managers T1 and T2 are in conflict over sensors S4, they may, due to time constraints, be unable to resolve the issue and may leave one of the slots of S4 co-bound. This means that when the PTC in S4 attempts to schedule that slot, it is forced to make the local determination as to which of the track managers gets the slot for that period.

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), a component of SRTA, can shift the task execution to try to eliminate any remaining conflict. Conflicts at this level can be created because the sensor is working on a non-tracking task that is not explicitly reasoned about at the schedule abstraction level.

During the course of negotiation, due to time constraints, the track manager can choose to operate at either the sensor or schedule abstraction level of negotiation. Leaving unresolved conflict at these levels of abstraction, though, introduces a great deal of uncertainty about the exact nature of the final solution. The deeper the track manager is able to go and resolve conflict, the greater the guarantee about the solution quality obtained in the end.

2.2 Utility

To help 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 2. 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:

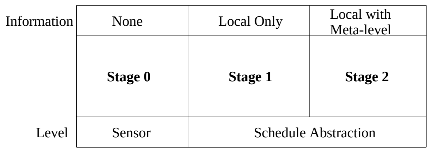


Figure 3: *The three stages of SPAM showing the information that is available and the level of abstraction the track manager uses in generating possible solutions.*

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

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 for 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 typically set the number of slots at 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 bound for a particular slot causes a near random occurrence of synchronization. For example, if a track manager T1 uses sensors S1, S2, S3, and S4 each for one slot of their schedule and sensors S2 and S4 are co-bound on that slot with one other track manager, it is easy to see that T1 has a 25 percent chance of getting four sensors for the slot and a 50 percent chance of getting three sensors for that slot during any given period. 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.

Finally, the global utility can be calculate from the following formula which just says that the overall utility is the sum of the utilities for the individual tracks (one track per target).

$$Utility = \sum_{Track \in Targets} Util(Track)$$

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 anytime 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 optimal and not necessarily conflict free). Figure 3 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 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 sensors themselves as was discussed in the previous sections.

3.1 Stage 0

On target detection, stage 0 of the negotiation protocol is activated. Stage 0 is primarily responsible for viewing the problem at the sensor level of abstraction. Because of this, each of the sensors that have the potential to track the target are evaluated and ordered. In this stage, the track manager also assigns an initial objective level to the track. Objective levels in general are derived from the track managers 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. Changing these parameters can alter the characteristics of the search process to make it faster (start at a lower objective level) or better (start at the best possible objective level).

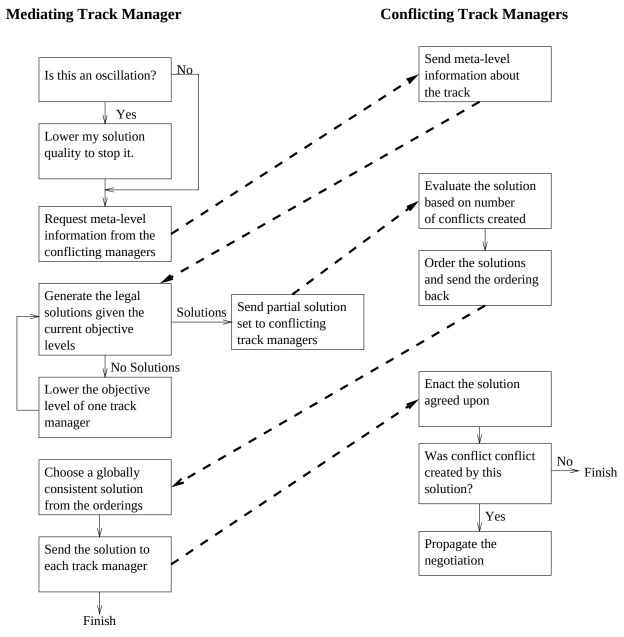
The actual activity of choosing a solution at this level of abstraction is primarily domain specific. For example in the tracking domain, criteria for solution choice might be the relative proximity of the target to the sensor, whether the target is moving toward or away from the sensor, etc. The solution choice, however, is based on internal information only.

Stage 0 ends, either by determining that enough time is available to go to stage 1 or by finishing the negotiation and binding a solution at the current level of abstraction. In the second case, the track manager leaves the conflict to be resolved by the agents residing in the sensors platforms.

3.2 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 process. We discuss how possible solutions are generated in a later section. Second, if a solution cannot be found at the current objective level, the track manager has enough information to bind a good solution which minimizes the amount of unresolved conflict and maximized the track manager's local objective level. Like stage 0, the negotiation session can be terminated at the end of stage 1 if enough time is not available to continue.

Solutions in stage 1 are only considered at the original objective level set forth in stage 0. The reason for this is that if the track manager were to lower its objective function without considering additional information then in all



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

Figure 4: *Stage 2 of the SPAM negotiation protocol resolves all local conflict at the schedule abstraction level through negotiation with conflicting track managers.*

likelihood it would end up with a utility that was lower than it should have been. For example, consider the conflict between the two tracking tasks, T1 and T2, in figure 5. Track manager T2 is assigned the role of tracking a new target and during stage 0 it determines that it wishes to have sensors S3, S4, S5, and S6 to track the target. In addition, it assigns an initial objective level of 4×3. After obtaining the abstract schedule of all four sensors the track manager finds that this solution is not possible because manager T1 has all three slots of sensor S3 assigned. As the protocol stands now, T2 binds a temporary solution and moves into stage 2 to begin negotiation with T1. Clearly, though, if T2 had lowered its objective function to 3×3 a solution (S4, S5, S6) with no conflict could have been obtained without expending time by going into stage 2. From a utility perspective, say that the other track manager, T1, was actively using a 5×3 objective level in tracking its target. If T2 accepts a 3×3 then the global utility would be around 8.2. If, however, T2 co-binds, while negotiating, then both managers obtain a 4×3 configuration with a global utility of about 8.5. Although the difference seems minimal, our belief is that in order to maintain the hill climbing nature of the search, agents must always try to locally maximize their utility until such a time where it is determined that to do so actually harms the global utility.

3.3 Stage 2

Stage 2, the final stage of SPAM, is the heart of the negotiation protocol (See figure 4). Stage 2 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 gener-

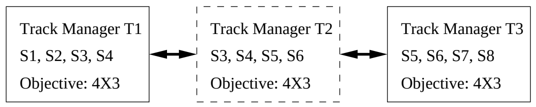


Figure 5: *Example of a common contention for resources. Track manager T2 has just been assigned a target and contention is created for sensors S3, S4, S5 and S6.*

ate 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 in order to resolve this 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.

The best way to explain how stage 2 operates is through an example. Again, consider figure 5. This figure depicts a commonly encountered form of contention. Here, track manager T2 has just been assigned a target. The target is located between two existing targets that are being tracked by track managers T1 and T3. This creates contention for sensors S3, S4, S5, and S6.

Following the protocol for the example in figure 5, 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 meta-level information from all of the track managers that are involved in the resource conflict. The information that is returned includes the current objective level that the track manager is using, the number of sensors which could possibly track the target, the names of the sensors that are in direct conflict with the mediator, and any additional conflicts that the manager has. To continue our example, T2 sends a request for information to T1 and T3. T1 and T3 both return that they have 4 sensors that can track their targets, the list of sensors that are in direct conflict (i.e $T1(S_3, S_4)$, $T3(S_5, S_6)$) their objective level (4×3 for both of them) and that they have no additional conflicts outside of the immediate one being considered.

Using this information, T2 begins to generate full solutions to resource problem (see section 3.4). Here, a full solution refers to a solution that includes all of the track managers for all of the sensors that the mediator is able to directly interact with (an example can be seen in figure 6). In addition, when a full solution is created, it is unconflicted over those sensors. This is in comparison to a partial solution which refers to a solution for a specific track manager over the proper subset of sensors that the mediator has direct interaction with. Note that each full solutions is composed of a unique set of partial solutions. The reciprocal is not necessarily true in that a partial solution can belong to a number of full solutions.

As you can see in figure 4, T2 enters a loop that involves attempting to generate full 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 full solutions are unavailable. Right now, this is done by first choosing the track manager with the highest current objective level and lowering them. 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. By doing this, it is our belief, that track managers with more external conflict will maintain higher objective levels, which leaves them more leverage to use in subsequent negotiations as a result of propagation.

The solution generation loop is terminated under one of two conditions. First, if given the current objective levels for each of the track mangers, a set of full solutions is available, the negotiation enters the solution evaluation 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 partial solution at the lowest objective level that minimizes the resulting conflict, conceding that it cannot find a full solution.

Continuing our example, T2 first lowers the objective level of T1 (choosing T1 at random because they all have equal external conflict). No full solutions are possible under the new 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.

During the solution evaluation phase, the mediator sends each of the track managers their set of partial solutions that are part of 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 their 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 full solution to apply to the problem. Using the orderings, the mediator prunes the full solution set generated in the solution generation phase by only keeping full solutions that contain the highest ranked partial solution for the track manager with the most external conflicts. This new reduced set of full solutions is then pruned by the mediator to contain only full 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 full 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.

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

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

Figure 6: *A solution derived by SPAM to the problem in figure 5. 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.*

By its ordering it ranked its partial solution 0 the highest. This restricts the choice for T2 to its partial solutions 0, 1, 2, and 3 because only these partial solutions continue to provide a full 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 full 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 it chooses to. Currently, track managers will propagate if 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 6 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 occurs because conflicts are resolved locally without regard to the global context. Say that from our previous example, track manager T1 originated a negotiation with track manager T2. In addition let's say 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.

3.4 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

$$Solutions = \binom{A_s}{D_s} \binom{A_a}{D_a}^{D_s}$$

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

Using the ability to impose constraints on the 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 5, 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)

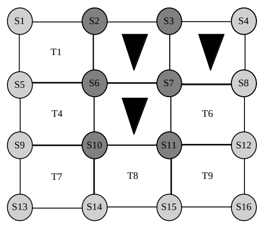


Figure 7: *The environmental layout used for testing the SPAM protocol. Track managers were defined by the sensors they needed to track targets in their region. In this figure, circles represent sensors, triangles represent targets and squares represent track manager regions.*

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 (DAG) where every path from the root nodes to the leafs has equals 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.

4. SIMULATION

To evaluate the SPAM protocol, we developed a simulator that uses a model of the tracking environment described earlier in the paper (see figure 7). In this simulator, we concentrated on evaluating primarily stage 1 and 2 of the protocol. To do this, the simulation was constructed using two major pieces, the environmental simulator and the track managers themselves. The environmental simulator manages the state of the sensors, spawns the track managers, introduces new targets, and manages propagation requests from the track managers. The track managers, which are defined by a set of sensors that they desire once they are given a target, handle any incoming negotiation requests from other track managers, spawn new negotiations when assigned a target, and request to be placed on the propagation list when they have unresolved conflict.

Using the environment in figure 7, we ran every possible configuration of targets and every possible order of target introduction in order to test the convergence, communication and, utility properties of the algorithm. In this environment, that equates to 9! = 362,880 tests.

From the simulation tests we determined that the SPAM protocol, when going from 0 to 9 targets, converges on a solution in an average of 18 discrete negotiation sessions which includes the 9 original local negotiations that take place due to target introduction. In addition, on average each track manager obtains a local objective level of better than 3×1 and receives a utility of approximately 1.51. On average, the overall solution has less than 1 unresolved conflict. Communication cost is dominated by track manager to sensor communications. On average to complete a 9 target problem, it takes 163 sensor schedule requests and 155 bind messages. These numbers may appear large, but considering that the activity is being done in parallel, the bind message counts include the temporary bind messages sent out at the end of stage 1, and that schedule requests occur in several place during the protocol, these numbers seem very reasonable. In fact, in the actual implementation of the protocol, sensor schedule requests are not actually made.

5. IMPLEMENTATION

Implementing and evaluating a protocol like SPAM in the domain described in section 2 with multiple agents running in parallel, an every changing environment with uncertainty about the exact resource needs, slow, lossy communications, and message length restrictions turned out to be quite a challenge. Principally, the amount of time needed to complete the entire protocol, including gathering the needed information to compute alternative solutions turned out to be far too large. With message transit times as long as 500 milliseconds, completing stage 1 of the protocol took almost a full second and completing stage 2 took over 3 seconds. In an environment where the resource requirements change almost every second (mostly due to uncertainty about the target location because of the underlying tracking components), a protocol that takes even a full second to complete is not appropriate.

To handle this, we changed many of the pull-based communications to push-based. For example, instead of a track manager asking for a schedule from a sensor agent, the sensor agent transmits changes to its schedule to all track managers that have slots bound by piggy-backing the information on top of measurement messages. This had three immediate implications for the protocol. First, stage 0 and stage 1 collapsed into a single stage. Since there is essentially, no cost for asking for sensor schedules, the track managers is always able to do stage 1 of the protocol. Second, the quality of binding a stage 1 solution was drastically decreased. Since track managers only receive schedule updates from sensors that they are currently using, whenever a new sensor is added to the allocation, the track manager assumes that the sensor is currently not in use by another track manager. This has the effect of causing more conflicts to occur. Third and most importantly, the protocol was able to bind a temporary solution immediately (or as long as it takes the bind messages to be sent). We also reduce the amount of time it took to complete stage 2 by employing a similar technique for gathering track manager meta-level information.

Even with this change, we found that often during the negotiation process the environment changed making the negotiated solution inconsistent with the current state. Initially, we considered a simple fix to the problem in which the target monitoring Finite State Machine(FSM) would be suspended until the negotiation session had terminated. When the target monitoring FSM was restarted, it should immediately detect a misallocation and re-start the negotiation process in order to rectify the situation. We determined after a very short period of time that this strategy, although simple, was not able to keep up with rapid changes in the environment. In fact, on many occasions, the target was lost because the track managers were constantly re-negotiating about a resource requirement that was no longer applicable.

SPAM, it turns out, was easy to adapted to handle this particular problem because the actual allocation decision is being done by each of the track managers based on their local view. If a shift in the resource requirement occurs during the negotiation, track managers can simple alter the allocation of the sensors not in direct conflict to fix their allocations. This allows the managers to adhere to their commitments while actually fixing context shifts that occur during the time it takes to negotiate quite gracefully.

The last problem we encountered was that of message length restrictions. During stage two, it was often the case that transmitting the potential solutions for a large number of conflicted sensors (even after compressing them), exceeded the 150 byte message length limit. One obvious, but bad, approach that we investigated for fixing this problem was to send two or more messages full of solutions. Without even implementing the solution, we determined that the amount of extra time needed to send additional message made the solution undesirable.

In the end, we decided that the only option was to reduce the number of alternatives that were being sent. Choosing which alternative to send and which not to send is actually an interesting problem in its own right. Recall from section 3.4 that the set of consistent solutions can be viewed as a DAG were each node is a particular partial solution for a particular track manager and paths through the DAG represent consistent unconflicted solutions. Now, imagine that the number of nodes (partial solutions), j, at path length i (for track manager i) is greater than the number, x, that can fit in a single message. The problem is to prune the "worst" $j - x$ nodes from the DAG at level i. Note that if a node is removed from the DAG, that a path (or consistent) solution is removed. Also, because the number of nodes on each path length must be equal to the number of track managers, nodes at lengths less than and greater than i may also be removed if they are not part of at least one complete path.

The key issue with this problem is to define what is meant by "worst". We are currently evaluating a number of different pruning methods, which include choosing to prune nodes that do not remove or minimize the number of additional nodes removed by them and choosing to remove nodes that have a high probability of not being part of the final solution. The problem with the first is that a really good solution might be removed, the problem with the second is that it is hard or impossible to calculate the probabilities.

As mentioned earlier, implementing is only half of the difficulty of migrating a protocol like SPAM to a real domain. The other half is in evaluating the quality of solutions that the protocol obtains. Particularly, we have not been able to evaluate the protocol as it relates to optimality. We have also not been able to correctly evaluate the effects of changes to the protocol on the relationship between optimality and our solution. The problem lies in choosing a metrics for evaluation because each of the metrics we have come up with is in part dependent on some other part of the system or has the potential to be influenced by components that compete with the protocol for processing or communications. Simply evaluating the protocol without considering the effects that various components have on the metric, in a system that is non-deterministic, makes determining how good the protocol is in practice very unreliable. For example, using the error of where the target was compared to where we thought is was (referred to as RMS tracking error), is influenced by the quality of the underlying tracker, the messages that were lost in the communications system, the processing load and the processor that the track manager were running on, etc. Most of the other metrics that could be used fall prey to similar difficulties. We continue to strive for fair measurement tool and in the mean time use the simulator we presented in the previous section to evaluate new techniques.

6. CONCLUSION

In this paper, we have described the SPAM protocol which was built to solve coordinated resource allocation problems in a soft-real time environment. The protocol exploits the fact that agents within the environment are both cooperative and autonomous and employs a number of techniques to operate in highly dynamic environments.

Much work remains to be done on this protocol. We are currently evaluating different methods for assigning precedence between track managers, pruning the consistent solution set, managing uncertainty and dynamics, and making the protocol more utility based. In addition, we are trying to find methods for evaluating the protocol in the real system where the state space is very large because of the interaction of different components and the protocol. Lastly, we are currently trying to create a more formal foundation from which to talk about the protocol and its suitability to a variety of domains.

7. ACKNOWLEDGMENTS

Thanks to Tim Middlekoop and Regis Vincent for helping during the initial phases of the protocol development, to Jiaying Shen for implementing the PTC that was used in the final implementation and to Bryan Horling for his efforts in adapting SPAM to work in the real system.

8. REFERENCES

[1] S. E. Conry, K. Kuwabara, V. R. Lesser, and R. A. Meyer. Multistage negotiation for distributed constraint satisfaction. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6), Nov. 1991.

[2] B. Horling, R. Vincent, R. Mailler, J. Shen, R. Becker, K. Rawlins, and V. Lesser. Distributed sensor network for real time tracking. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 417–424, 2001.

[3] T. Moehlman, V. Lesser, and B. Buteau. Decentralized negotiation: An approach to the distributed planning problem. *Group Decision and Negotiation*, 1(2):161–192, 1992.

[4] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transctions on Computers*, 29(12):1104–1113, 1980.

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

[6] M. Yokoo. *Distributed Constraint Satisfaction*. Springer Series on Agent Technology. Springer, 1998.