Hierarchical organizations for real-time large-scale task and team environments

Osher Yadgar
Dept. of Math and CS,
Bar Ilan University,
Ramat Gan, 52900 Israel
yadgar@macs.biu.ac.il

Sarit Kraus
Dept. of Math and CS,
Bar Ilan University,
Ramat Gan, 52900 Israel
sarit@macs.biu.ac.il

Charles L. Ortiz, Jr.
Artificial Intelligence Center,
SRI International,
Menlo Park, CA 84025 USA
ortiz@ai.sri.com

Abstract

In this paper, we describe the Distributed Dispatcher Manager (DDM), a system for monitoring large collections of dynamically changing tasks. We assume that tasks are distributed over a virtual space. Teams consist of very large groups of cooperative mobile agents. Each agent has direct access to only local and partial information about its immediate surroundings. DDM organizes teams hierarchically and addresses two important issues that are prerequisites for success in such domains: (i) how agents should process local information to provide a partial solution to nearby tasks, and (ii) how partial solutions should be integrated into a global solution. We conducted a large number of experiments in simulation and demonstrated the advantages of the DDM over other architectures in terms of accuracy and reduced inter-agent communication [*].

Introduction

This paper considers the problem of monitoring large collections of dynamically changing tasks. The tasks are distributed over a large (possibly, virtual) environment and are to be executed by large teams of mobile cooperative agents. These agents have direct access to only local and partial information about their immediate environment. There are several domains where such problems arise: satellites that are tasked to form a general picture of a large area; satellites that form weather maps; agents that control air pollution or ocean pollution; sensor webs that monitor geographic areas for passing aircrafts; and unmanned air and ground vehicles that must be jointly tasked for surveillance missions. In such domains, there are two central issues that represent prerequisites for success: (i) how agents should process local information to provide a partial solution to nearby tasks, and (ii) how partial solutions should be integrated into a global solution.

We describe the Distributed Dispatcher Model (DDM), an agent based computational model. DDM is designed for efficient coordinated task allocation in systems consisting of hundreds of agents (resources); the model makes use of hierarchical group formation to restrict the degree of communication between agents. Our main contribution is in use of a hierarchical organization of agents to combine partial information. The hierarchical team organization supports processes for very quickly combining partial results to form an accurate global solution. Each level narrows the uncertainty about the solution based on the data obtained from lower levels. We proved that the hierarchical processing of information reduces the time needed to form the accurate global solution.

We tested the performance of the DDM through extensive experimentation in a simulated environment involving many sensors. The simulation models a suite of Doppler sensors used to form a global information map of targets moving in a steady velocity as a function of time. A Doppler sensor is a radar which is based on the Doppler effect. Due to its nature, a Doppler sensor may provide information only about an arc that a detected target may be located on as well as the velocity towards that sensor, that is, the *radial velocity* (Thomas 1965). Given a single Doppler measurement, one cannot establish the exact location of a target and its exact velocity; therefore, multiple measurements must be combined for each target. This problem was devised as a challenge problem by the DARPA Autonomous Negotiating Teams (ANTS) program to explore realtime distributed resource allocation algorithms.

We compared our hierarchical architecture to other architectures and showed that the monitoring task is faster and more accurate in DDM. We have also shown that DDM can achieve these results when using a low volume of noisy communication.

The DDM model

We consider environments with tasks and agents distributed over some area. We present a formal specification of the environment and demonstrate the formalism using the ANTS challenge problem. We assume that there is a set T of time points.

[*] This work was supported by DARPA contract F30602-99-C-0169 and NSF grant IIS-9907482. The second author is also affiliated with UMIACS.

Tasks: There are many tasks in the environment. At any time, each task has an associated state. The state of a task may change over time. We denote the set of all task-states by S. There is a boolean function *ResBy* that given two task-states s1 and s2 and two time points t1 and t2 where t2 ≥ t1, returns true if it is possible that if the state of a task was s1 at t1, then it could be s2 at t2. *ResBy* satisfies the constraints:

Let $t_1, t_2, t_3 \in T$ and $s_1, s_2, s_3 \in S$, $t_1 < t_2 < t_3$ such that

if *ResBy*($< t_1, s_1 >, < t_2, s_2 >$) and *ResBy*($< t_2, s_2 >$, $< t_3, s_3 >$) then *ResBy*($< t_1, s_1 >, < t_3, s_3 >$)

if *ResBy*($< t_1, s_1 >, < t_2, s_2 >$) and *ResBy*($< t_1, s_1 >$, $< t_3, s_3 >$) then *ResBy*($< t_2, s_2 >, < t_3, s_3 >$)

if *ResBy*($< t_1, s_1 >, < t_3, s_3 >$) and the *ResBy*($< t_2, s_2 >, < t_3, s_3 >$) then *ResBy*($< t_1, s_1 >$, $< t_2, s_2 >$)

if *ResBy*($< t, s_1 >, < t, s_2 >$) then $s_1 = s_2$.

The constraints (i)-(iii) of ResBy consider the way the state of a target may change over time. They refer to three points of time t_1, t_2, t_3 in an increasing order and to the possibly that a task was at state s_1, s_2 and s_3 at these time points, respectively. If the task has been in these states at the corresponding time then s_2 at t_2 should be a result of s_1 at t_1, i.e. *ResBy*($< t_1, s_1 >, < t_2, s_2 >$). Similarly *ResBy*($< t_2, s_2 >, < t_3, s_3 >$) and *ResBy*($< t_1, s_1 >$, $< t_3, s_3 >$). The constraints indicate that it is enough to check that two out of the three relations hold, to verify that the task was really at s_1 at t_1, s_2 at t_2 and s_3 at t_3. That is if two of the three relations hold, also the third one. The last constraint (iv) is based, intuitively, on that a task cannot be in two different states at the same time.

Tasks and *ResBy* relation in the ANTS domain: In the ANTS domain we map each target to a task. The target state structure is $s =< \bar{r}, \bar{v} >$. $\bar{r}$ is the location vector of the target and $\bar{v}$ is the velocity vector. If a target state s_2 at t_2 resulted from target state s_1 at t_1 and the velocity of the target remains constant during the period $t_1 . t_2$, then $\bar{r}_2 = \bar{r}_1 + \bar{v}_1 \cdot (t_2 - t_1)$. We assume that no target is likely to appear with the same exact properties as another target. That is, there cannot be two targets at the exact same location moving in the same velocity and direction. Thus where $s_i =< \bar{r}_i, \bar{v}_i >$ *ResBy*($< t_1, s_1 >, < t_2, s_2 >$) in ANTS is true iff: (i) r_2 may be derived from r_1 using the motion equation of a target and given $\bar{v}_1$ during the period $t_2 - t_1$ and (ii) $\bar{v}_1 = \bar{v}_2$.

The physical motion of a moving body in a steady velocity follows the four constraints of the ResBy relation. In general in any domain every task state that combines out of a singular state along with the first derivative of this state by time where this derivative is not depended on time satisfies the four constraints. For instance, in the monitoring satellite domain the task state may be a combination of an image and a derivation of the image by time.

Agents: There is a set of sampling agents, A. Each agent is capable of taking measurements. Each $a \in A$ is associated with an agent state that may change over time. The set of all possible sampling-agent states is denoted Sa.

In the ANTS domain we map each Doppler to a sampling agent. The sampling agent state is the location of the sensor and its orientation.

Agents observing tasks: Sampling agents are able to obtain measurements on tasks if they are located near the tasks. The measurements provide only partial information on the task-states and may be incorrect. When an agent takes measurements we refer to its agent state as a viewpoint in which a task state was measured. We assume that there is a function *PosS* that given k consecutive measurements taken by the same agent, up to time t returns a set of possible states, $S' \subseteq S$, for a task at time t where exactly one $s \in S'$ is the right task state and there is an $m \geq 1$ such that $| S' | \leq m$.

A path, p, is a sequence of triplets $<< t_1, sa_1, s_1 > ... < t_n, sa_n, s_n >>$ whereas $t_i \in T, s_i \in S, sa_i \in Sa$ and for all $1 \leq i < n$, either $t_i < t_{i+1}$ and *ResBy*($< t_i, s_i >, < t_{i+1}, s_{i+1} >$) is true or $t_i = t_{i+1}$, $sa_i \neq sa_{i+1}$ and $s_i = s_{i+1}$.

We refer to points of a path as path points. Each path represents task state discrete change over time as measured by sampling-agents in states, $sa_1 ... sa_n$. Constraint (i) considers the case where the two points in the path captures the change of the state of the task from s_i at time t_i to s_{i+1} at time t_{i+1}. In this case, where the path specifies the way the state was changed, *ResBy*($< t_i, s_i >, < t_{i+1}, s_{i+1} >$) must hold, i.e. the task could be at s_i at t_i and then at s_{i+1} at t_{i+1}. On the other hand, constraint (ii) considers the case of two points $< t_i, sa_i, s_i >$, $< t_{i+1}, sa_{i+1}, s_{i+1} >$ of the path that does not capture a change of the task's state. Rather, it captures two different observations of the task. That is, that task was at a given state s_i at time t_i, but was observed by two agents. The two agents were of course in different states, and this is indicated by $sa_i \neq sa_{i+1}$.

A path consists of only very few states of the task that were observed by agents. However, from a path the agent would like to induce the state of the task at any given time. This is formalized as follows.

A task state function f_{π_s, π_e}, with respect to two path points $\pi_s =< t_s, sa_s, s_s >$, $\pi_e =< t_e, sa_e, s_e >$ where $t_s \leq t_e$, associates with each time point a task state (i.e. $f_{\pi_s, \pi_e} : T \rightarrow S$) such that

(i) $f_{\pi_s, \pi_e}(t_s) = s_s$ and $f_{\pi_e, \pi_e}(t_e) = s_e$

(ii) $\forall t_1, t_2$, $t_1 < t_2$

ResBy($< t_1, f_{\pi_s, \pi_e}(t_1) >, < t_2, f_{\pi_s, \pi_e}(t_2) >$).

A task state function represents task state change over time points of T with respect to two path points.

Finally, to move from a path to an associated function, we assume that there is a function *pathToFunc:* $P \rightarrow F$ such that given a path $p \in P$, $p =<< t_1, sa_1, s_1 >, ..., < t_n, sa_n, s_n >>$, if f_{π_s, π_e} = *pathToFunc* (*p*) then, $\pi_s =< t_1, sa_1, s_1 >$, $\pi_e =< t_n, sa_n, s_n >$ $\forall t_i, f_{\pi_s, \pi_e}(t_i) = s_i$.

In ANTS we have developed a method involving k consecutive measurements taken by a single Doppler to compute possible locations and velocities of targets.

***PosS* implementation in ANTS.** A measurement in the ANTS domain is a pair of amplitude and radial velocity values for each sensed target. Given a measurement of a Doppler radar the target is located on the Doppler equation:

$$r_i^2 = \frac{k \cdot e^{\frac{-(\vartheta_i - \beta)^2}{\sigma}}}{\eta_i}$$

where, for each sensed target, *i,* r_i is the distance between the sensor and i; θ_i is the angle between the sensor and i; η_i is the measured amplitude of i; β is the sensor beam angle; and k and σ are characteristics of the sensors and influence the shape of the sensor detecting area (1). Given k consecutive measurements one can use the Doppler equation to find the distance r_i. However, there are two possible θ_i angles for each such distance. Therefore, for PosS function in ANTS domain returns two possible task states, i.e. m=2. For space reasons we do not present the proofs of the lemmas and theorems.

Theorem 1: *(PosS)* Assuming that the acceleration of a target in a short time period is zero. The next target location after a very short time is then given by

$$\theta_1 = \alpha_1 + \sqrt{-\sigma \cdot \ln\left(\frac{\eta_1}{k}\left(r_0 + v_{r0} \cdot t_{1,0}\right)^2\right)} \qquad \frac{\bar{r}_2(\theta_0) - \bar{r}_1(\theta_0)}{t_{2,1}} = \frac{\bar{r}_1(\theta_0) - \bar{r}_0(\theta_0)}{t_{1,0}}$$

where $r_0, \theta_0, \eta_0, v_{r0}$ and α_0 are values of the target at time $t = 0$ and θ_1, η_1 and α_1 represent values of the target at time $t = 1$. $t_{i,j}$ is the time between t=i and t=j.

Only certain angles will solve the equations. To be more accurate, the sampling agent uses one more sample and applies the same mechanism to θ_1, θ_2 and θ_3. The angles are used to form a set of possible pairs of location and velocity of a target (i.e., the PosS function values). Only one of these target states is the correct one.

DDM's goal

The aim of the DDM is to describe the way the states of the task in the area change over time. To monitor tasks we use many agents. Each of these agents can obtain partial information about tasks in its close environment. The DDM uses the partial and local information to form an accurate global description on how the tasks are changing over time. We can apply the DDM model to many problems by mapping the DDM entities to the domain entities. For example, in the case of satellites used to track forest preservation we can map each sampling agent to a satellite and each forest segment under surveillance to a task. In that case a ResBy relation can be a function that uses an image processing methods and a physical logic to deduce whether one state is resulted by another. A f_{π_s,π_e} function will be a function that describes the changes of the image by time using the same deductions. The function will describe the changes of the image between the two time points as collected by a path of states and a prediction about a further changes.

In the ANTS domain the agent's goal is to find the location of the targets at any time point. Here $f_{\pi_s,\pi_e}(t) = <\bar{r}_s - \bar{v}_s \cdot (t - t_s), \bar{v}_s> \equiv <\bar{r}_e - \bar{v}_e \cdot (t - t_e), \bar{v}_e>$.

Formally, the DDM aims at forming the information map structure.

Definition 1: An *information map, infoMap,* is a set of task state functions $< f^1_{\pi_s,\pi_e}, ..., f^h_{\pi_s,\pi_e} >$ such that for every $1 \le i, j \le h$ and $t \in T$ $f^i_{\pi_s,\pi_e}(t) \neq f^j_{\pi_s,\pi_e}(t)$

Intuitively, *infoMap* represents the way the states of a set of tasks change over time. The condition on the information map specifies the assumption that two tasks cannot be at the same state and time. Because each agent has only partial and uncertain information on its local surrounding an agent may need to construct the *infoMap* in stages. In some cases they may not be able to construct the entire *infoMap*. The process of constructing the *infoMap* will use various intermediate structures.

Intermediate structures: Due to uncertainty of the sensing information at a given time every sampled task may be associated with several possible *task states*. Each such task state is derived from the same raw sensed data, i.e. *PosS* returns S' such that $1 < |S'| \le m$. All the possible *task states* are combined with the sampling agent state to form a capsule.

Definition 2: A *capsule* is a triplet of a time point, a sampler agent state and a sequence of up to m task-states, i.e., $c = < t, sa, \{s_1, ..., s_l\} >$ where $t \in T, sa \in Sa, s_i \in S, l \le m$. We denote the set of all possible capsules by C.

A capsule represents a few possible states of a task at time t as derived from measurements taken by an agent in a given state. Capsules are generated by the sampling agents using the domain dependent function *PosS* and k consecutive samples.

The problem faced by DDM is how to choose the right state from every capsule. It is impossible to determine which state is the right one using one viewpoint since measurements from one viewpoint results up to m task states each could be the correct state. Therefore, capsules from different viewpoints are needed. A different viewpoint may correspond to a different state of the same sampling agent or of different sampling agents. To choose the right task state from each capsule we connect states from different capsules using the *ResBy* relation and form a path. We then try to evaluate each of these paths and use the ones with the best probability to represent changes of task states and form state functions. We will use the following.

Definition 3: A *localInfo* is a pair of *infoMap* and a set of capsules, <*infoMap*, *unusedCapsules*> where *unusedCapsules*= $< c_1, ..., c_m >$ s.t. for all $1 \le i \le m$ and for all $1 \le j \le l$ and $c_i = < t_i, sa_i, \{s_{i1}, ..., s_{i,l}\} >$ and for every $f_{\pi_s,\pi_e} \in$ infoMap $f_{\pi_s,\pi_e}(t_i) \neq s_{ij}$.

At any time, some of the capsules can be used to form task state functions that have a high probability of representing tasks. These functions are kept in the *infoMap; we* refer to

them as *accurate representatives*. The rest of the capsules will be kept in the *unusedCapsules* set and will be used for further attempts to identify state functions. That is, the condition of definition 3 intuitively says that a task associated with a function f_{π_s,π_e} was not constructed using one of the measurements that were used to form the capsules in the *unusedCapsules* set.

2.2 The DDM hierarchy structure

In a large-scale environment we will have to attempt to link many capsules from all the area. Using the relation *ResBy* many times is time consuming. However, there is low probability that capsules created based on measurements taken far away from one another will fit. Therefore, we will distribute the solution. The DDM uses hierarchical structures to construct a global *infoMap* distributively. The lower level of the hierarchy consists of the sampling agents. These agents are grouped according to their area. Each group has a leader. Thus, the second level of the hierarchy consists of the sampler group leaders. These sampler group leaders are also grouped according to their area. Each such group of sampler leaders is associated with a zone group leader. Thus, the third level of the hierarchy consists of these zone group leaders, which in turn, are also grouped according to their area associated with a zone group leader and so on and so forth. We refer to members of a group as group suborinates. Sampling agents are mobile; therefore, they may change their group when changing their area. The sampler leaders are responsible for the movements of sampling agents. For space reasons we do not discuss this process here, but rather focus on the global *infoMap* formation. We also do not discuss the methods we developed that are used to replace group leaders that stop functioning. Leader agents are responsible for retrieving and combining information from their group of agents. All communication is done only between a group member and its leader.

A sampler agent takes measurements occasionally and forms capsules. It sends its capsules to its sampler leader every specified time period. A sampler leader collects the capsules from all the sampler agents in its area and forms a *localInfo*. In this formation it uses its *localInfo* from previous round. It then sends its *localInfo* to its zone leader. A zone leader collects the *localInfo* of all the sub-leaders of its zone and forms a *localInfo* of its entire zone. In turn it sends it to its leader and so on. The top zone leader, whose zone consists of the entire area, forms a *localInfo* of all the tasks in the entire area. In the next section we present the agent algorithms.

Algorithm descriptions

The formation of a global information map integrates the following processes:

Each sampling agent gathers raw sensed data and generates capsules.

Every dT seconds each sampler group leader obtains from all its sampling agents for their capsules and integrates them into its *localInfo*.

Every dT seconds each zone group leader obtains from all its subjected group leaders their *localInfo* and integrates them into its own *localInfo*.

As a result, the top-level group leader *localInfo* contains a global information map.

We have developed several algorithms to implement each process. We will use a dot notation to describe a field in a structure, e.g., if $c =< t, sa, \{s_1,...,s_l\} >$ then *c.sa* is the sampling agent field of the capsule *c*.

Figure 1 : Step 1 - Obtaining new information algorithm

In: localInfo = <infoMap, unusedCapsules> **Out:** updated localInfo

```
if activated as Sampler group leader
    for each subjugated sampler, sampler
        additionalCapsules =obtain set of capsules from each sampler
    localInfo.unusedCapsules = localInfo.unusedCapsules ⋃ additionalCapsules
else // activated in Group leader
    for each subjugated leader, leader
        // in this part we identify identical functions and
        // leave only one of them
        additionaLocalInfo = ask each leader for its local info
        additionalCapsules =  additionaLocalInfo.unusedCapsules
        additionalInfoMap = additionaLocalInfo.infoMap
        localInfo.unusedCapsules = localInfo.unusedCapsules ⋃ additionalCapsules
        mergeFunctions (localInfo.infoMap , additionalInfoMap);
return infoMap, unusedCapsules
```

Figure 2 : mergeFunctions algorithm

In: F, F' **Out:** updated F

for each state function, $f^i_{\pi^i_{si},\pi^i_{ei}}$, in F'

let $\pi^i_s =< t^i, sa^i, s^i >$

merged = false

for each state function, $f^j_{\pi^j_{sj},\pi^j_{ej}}$, in F && not merged

let $\pi^j_s =< t^j, sa^j, s^j >$

if ($ResBy(< t^i_e, s^i_e >, < t^j_s, s^j_s >)$ or $ResBy(< t^j_s, s^j_s >, < t^i_e, s^i_e >)$)

change $\pi^j_{s_j}$ of $f^i_{\pi^i_s,\pi^j_e}$ to be the triplet of $\min(\pi^i_s, \pi^j_s)$ by time

change π^j_e of $f^j_{\pi^j_s,\pi^j_e}$ to be the triplet of $\max(\pi^i_e, \pi^j_e)$ by time

merged = true

if (not merged)

F = F ⋃ $\{f^i_{\pi^i_s,\pi^i_e}\}$

return F

Sampler capsule generation algorithm. We use one sampling agent to deduce a set of possible task states at a given time in the form of a capsule. A sampling agent takes k consecutive measurements. Then it creates a new capsule, c, such that the time of the capsule is the time of the last measurement. The state of the sampling agent while taking the measurements is assigned to *c.sa*. The task states resulting from the application of the domain function *PosS* to the k consecutive measurements is assigned to *c.states*. The agent stores the capsules until it is time to send them tp its sampler group leader asks for them. After delivering the capsules to the group leader the sampler agent deletes them.

Leader localInfo generation algorithm. Every dT seconds each group leader performs the *localInfo* generation algorithm. Each group leader holds its own *localInfo*. The leader starts by purging data older than τ

seconds before processing new data. Updating the *localInfo* involves three steps: (i) obtaining new information from the leader's subordinates; (ii) finding new paths; (iii) and merging the new paths into the *localInfo*.

Figure 1 presents the algorithm for (i) in which every leader obtains information from its subordinates. The sampler group leader obtains information from all of its sampling agents for their *unusedCapsules* and adds them to its *unusedCapsules* set. The zone group leader obtains from its subordinates their *localInfo*. It adds the *unusedCapsules* to its *unusedCapsules* and merges the *infoMap* of that *localInfo* to its own *localInfo*.

Merging of functions is performed both in steps (i) and (iii). Merging functions is needed since, as we noted earlier, task state functions that a leader has inserted into the information map are accepted by the system as correct and will not be removed. However, different agents may sense the same task and therefore it may be that different functions coming from different agents will refer to the same task. The agents should recognize such cases and keep only one of these functions in the *infoMap*. We use the next lemma to find and merge identical functions.

Lemma 1:

Let $p^1 =< \pi_s^1,...,\pi_e^1 >$, $p^2 =< \pi_s^2,...,\pi_e^2 >$ be two paths, where $\pi_j^i =< t_j^i, sa_j^i, s_j^i >$ and $f_{\pi_s^1,\pi_e^1}^1 = pathToFunc(p^1)$, $f_{\pi_s^2,\pi_e^2}^2 = pathToFunc(p^2)$.

If $ResBy(<t_s^1, s_s^1>, <t_s^2, s_s^2>)$ then for any $f_{\pi_s^1,\pi_e^1}^1(t) = f_{\pi_s^2,\pi_e^2}^2(t)$

Using lemma (1) we developed the *mergeFunctions* algorithm that is presented in Figure 2. In this function, the leader uses the *ResBy* relation to check whether the first state of the task state function resulting from the first state of a different task state function. If one of the states is resulted by the other the leader changes the minimum and the maximum triplets of the task state function. The minimum triplet is the starting triplet that has the lowest time. The maximum triplet is the ending triplet that has the higher time. Intuitively, the two state functions are merged and the new function that is with respect to the largest range given the found points. In case a leader cannot find any task state function to meet the subordinate's function, the leader will add it as a new function to its *infoMap*.

The second step, as presented in Figure 3, is conducted by every leader to find paths and extend current paths given a set of capsules. In order to form paths of capsules, the agent should choose only one task state out of each capsule. This constraint is based on the flowing lemma.

Lemma 2:

Let $C^1 =< t^1, sa^1, < s_1^1,...,s_{h^1}^1 >>$,
$C^2 =< t^2, sa^2, < s_1^2,...,s_{h^2}^2 >>$ and
$ResBy\ (<t^1, s_i^1>, <t^2, s_j^2>)$ and $ResBy\ (<t^1, s_{i'}^1>, <t^2, s_{j'}^2>)$ then

(i) if $s_i^1 \neq s_{i'}^1$ then $s_j^2 \neq s_{j'}^2$ (ii) if $s_j^2 \neq s_{j'}^2$ then $s_i^1 \neq s_{i'}^1$
(iii)if $s_i^1 = s_{i'}^1$ then $s_j^2 = s_{j'}^2$
(iv) if $s_j^2 = s_{j'}^2$ then $s_i^1 = s_{i'}^1$

According to this lemma one state of one capsule cannot have a relation of ResBy with two different states in another capsule with respect to the capsule's time. Having a case of such two different states violates the ResBy constraints.

Figure 3 : Step 2 - Finding new paths algorithm

In: unusedCapsules **Out:** updated unusedCapsules, paths, *mediocreFunctions*

// phase 1: make links
sort(unusedCapsules) // by time stamp
allPaths = {}
for each capsule, $c =< t, sa, \{s_1,...,s_l\} >$, in unusedCapsules
  cap.mark = false // marking for phase 2
  for each task state, si, in cap states
    linked = false
    // because of the above assumption and given that eh path
    // elements came from capsules there will be only one suitable
    // path. Therefore,we exit the loop after finding such path
    for every last triplet, $< t-last, sa-last, s-last >$, in each path, p, in *allPaths* && not linked
      if (*ResBy* ($< t-last, s-last >, < t, si >$) or (t-last=t && sa-last $\neq$ sa))
        $p = p* << t, sa, si >>$
        linked = true
    if (not linked)
      $p =<< t, sa, si >>$
      $allPaths = allPaths \cup \{p\}$

// phase 2: collect task representing paths that has no common capsules
// when giving a greater priority to paths with more viewpoints.
sort(*allPaths*) // by number of viewpoints
paths = {}
for each path, p, in *allPaths*
  if (not isAnyCapsuleMarked(p) && numberOfViewpoint(p) > 1)
    markAllCapsules(p)
    unusedCapsules.= unusedCapsules - allCapsules(p)
    $paths = paths \cup \{p\}$

if activated as top-level leader
  mediocreFunctions = collectMediocreFunctions(*allPaths*)
else
  mediocreFunctions = {}
return unusedCapsules, paths, *mediocreFunctions*

For the algorithm in Figure 3 that creates the new paths we add two temporary fields to two of the structures only or the purpose of the algorithm below. The first is a boolean flag named *mark* that will be added to the capsule structure. The second is a pointer to the originated capsule that will be added to every triplet stored in a path. Every leader keeps the correct paths as part of its *infoMap* structure. In the top-level leader we would also like to have the paths with a mediocre probability to represent tasks. The top leader knows that some of these paths are correct but it cannot decide which are correct. Paths with only one viewpoint are paths that may be correct. For instance, due to the characteristics of the sensors in the ANTS domain, paths with one viewpoint will have a 50% probability to be correct. In other domains the characteristics of the sensors

may lead to a different probability. The top-level leader will use these mediocre paths to form a set of functions that have a partial probability to be correct.

Simulation environment

We developed a simulation of the ANTS domain to test the model. The simulation consists of an area of a fixed size in which Dopplers attempt to identify the task state functions of moving targets. Each target had an initial random location and an initial random velocity up to 50 km. per hour. Targets leave the area when reaching the boundaries of the zone. Each target that leaves the area causes a new target to appear at the same location with the same velocity in a direction that leads it inwards. Therefore, each target may remain in the area for a random time period. Each Doppler has initial random location and a velocity that is less than 50 km. per hour. When a Doppler gets to the border of the controlled area it bounces back with the same velocity. This ensures an even distribution of Dopplers.

Evaluation Methods. We collected the state functions produced by the agents during a simulation. We used two evaluation criteria in our simulations: (1) target tracking percentage and (2) average tracking time. We counted a target as being tracked if the identified path by the agent satisfied the following: (a) the maximum distance between the calculated location and the real location of the target was less than 1 meter, and (b) the maximum difference between the calculated v(t) vector and the real v(t) vector was less than 0.1 meter per sec. and 0.1 radians in angle.

In addition, the identified task state functions could be divided into two categories: (1) Only a single function was associated with a particular target and was chosen to be part of *the infoMap*. Those functions were assigned a probability of 100% corresponding to the actual task state function. (2) Two possible task state functions based on one viewpoint were associated with a target. Each was assigned a 50% probability of corresponding to the actual function. We will say that one set of agents *did better* than another if they reach higher tracking percentage and lower tracking time with respect to the 100% functions and the total tracking percentage was at least the same.

The averages reported in the graphs below were computed for one hour of simulated time. The *target tracking percentage time* was calculated by dividing the number of targets that the agents succeeded to track, according to the above definitions, by the actual number of targets during the simulated hour. In total, 670 targets passed through the controlled area within an hour in the basic settings that described below. The *tracking time* was defined as the time that the agents needed to find the task state function of the target from the time the target entered the simulation. Tracking average time was calculated by dividing the sum of tracking time of the tracked targets by the number of tracked targets. Note that 29% of the targets in our experiments remained in the area less than 60 seconds in our basic settings.

Basic Settings. In the *basic setting* of the environment the area size was 1200 by 900 meters. In the experiments we varied one of the parameters of the environment, keeping the other values of the environment parameters as in the basic settings. The Dopplers were mobile and moved randomly as described above. Each Doppler stopped every 10 seconds, varied its active sensor randomly, and took 10 measurements. The maximum detection range of a Doppler in the basic setting was 200 meters; the number of Dopplers was 20 and the number of targets at a given time point was 30. The DDM hierarchy consisted of only one level. That is, there was one sampler-leader that was responsible for the entire area.

We first compared several settings to test the hierarchy model and the sampling agents characterizations. Each setting was characterized by (i) whether we used a hierarchy model (H) or a flat model (F); (ii) whether the sampler-agents were mobile (M) or static (S); and (iii) whether Dopplers varied their active sectors from time to time (V) or used a constant one all the time (C). In the flat model the sampler agents used their local capsules to produce task state functions locally.

Mobile and dynamic vs. static Dopplers. In preliminary simulations (not presented here for space reasons) we experimented with all the combinations of the parameters (i)-(iii) above. In each setting, keeping the other two variables fixed and varying only the mobility variable, the mobile agents did better than the static ones (with respect to the evaluation definition above).

Hierarchy vs. flat models. We examined the characteristics of 4 different settings: (A) FSC that involves static Dopplers with a constant active sector using a nonhierarchical model; (B) HSC as in (A) but using the hierarchical model; (C) FMV with mobile Dopplers that vary their active sectors from time to time, but with no hierarchy; (D) HMV as in (C) but using the hierarchical model. We tested FSC on two experimental arrangements: a random located Dopplers and Dopplers arranged in a grid formation to achieve a better coverage. There was no significant difference between these two FSC formations. Our hypothesis was that the agents in HMV would do better than in all the other settings.

The first finding is presented in the left part of Figure 4. This finding indicates that the setting does not affect the overall tracking percentage (i.e., the tracking percentage of the 50% and 100% functions). The difference between the settings is with respect to the division of the detected target between accurate tracking and mediocre tracking. HMV performed significantly better than the other settings. It found significantly more 100% functions and did it faster than the others. This supports the hypothesis that a hierarchical organization leads to better performance. Further support for a hierarchical organization comes from HSC being significantly better than FMV even though, according to our preliminary results, HSC uses Dopplers that are more primitive than the Dopplers FMV.

Another aspect of the performance of the models is the average tracking time as shown in the right part of Figure 4.

Once again, one can see that the hierarchically based settings lead to better results. We found that by considering only targets that stayed in the controlled zone at least 60 seconds, HMV reached 87% tracking percentage, 83% were accurately detected

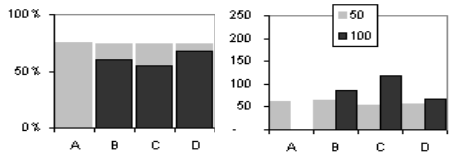

Figure 4: Target tracking percentage and average time by the settings.

We also considered a hierarchy with two levels: one zone leader leading four sampling leaders. The area was divided equally between the four sampling leaders, and each obtains information from the many mobile sampling agents located in its area. In that configuration Dopplers were able to move from one zone to another. In that case, Dopplers changed their sampling leader every time they moved from one zone to another. Comparing the results of the two level hierarchy simulations (not presented here because of space reasons), with the one level hierarchy simulations we found that there was no significant difference in the performance (with respect to the evaluation definition) of the system when there were two levels of the hierarchy of when there is only one level in the hierarchy. However, consistent with theorem 1, the computation time of the system was much lower.

Communication and noise. While the performance of the hierarchy-based models are significantly better than the non-hierarchy ones, the agents in the hierarchy model must communicate with one another, while no communication is needed for the flat models. Thus, if no communication is possible, then FMV should be used. When using communications messages may be lost or corrupted. The data structure exchanged in messages is the capsule. In our simulations using a hierarchy model, each sampling agent transmitted 168 bytes per minute. We checked the influence of randomly corrupted capsules on the HMV's behavior. Figure 5 shows that as the percentage of the lost capsules increases the number of tracked targets decreases; however, up to a level of 10% noise, the detection percentages decreased only from 74% to 65% and the accurate tracking time increased from 69 seconds to only 80 seconds. Noise of 5% results in a smaller decrease to a tracking accuracy of 70% while the tracking time is slightly increased to 71.DDM could even mange with noise of 30% and track 39% of targets with average tracking time of 115 seconds. In the rest of experiments we used the HMV settings without noise.

Varying the number of Dopplers and targets. We examined the effect of the number of Dopplers on the performance. We found that, when the number of targets is kept fixed, as the number of Dopplers increases the percentage of accurate tracking increases. The significant of the result is that it demonstrates that the system can make good use of additional resources that it might be given. We also found out that as the number of Doppler sensors increases the 50% probability paths decrease. That may be explained by the fact that 100% paths result from taking into consideration more than one point of view of samples. We also found that increasing the number of targets, while keeping the number of Dopplers fixed does not influence the system's performance. We speculate that this is because an active sector could distinguish more than one target in that sector.

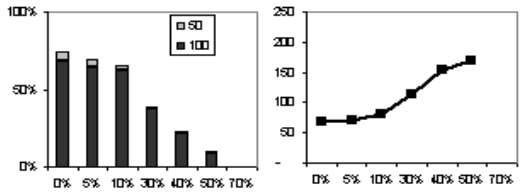

Figure 5: Target detection percentage and average time as function of the communication noise.

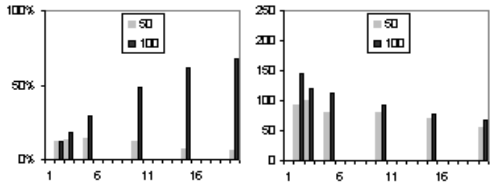

Figure 6: Tracking percentage and average time as a function of the number of Dopplers.

Maximum detection range comparison. We also tested the influence of the detecting sector area on performance. The basic setting uses Dopplers with detection range of 200 meters. We compared the basic setting to similar ones with detection ranges of 50,100 and 150 meters. We found that as the maximum range increases the tracking percentage increases up to the range of covering the entire global area. As the maximum radius of detection increased the tracking average time decreases. This is a beneficial property, since acquiring better equipment leads to better performance.

Conclusions and related work

We have introduced a hierarchical approach for combining local and partial information of large-scale task and team environments where agents must identify the changing states of tasks. To apply the DDM model in a different environment, it is only necessary to define three domain specific functions: *PosS* that maps measurements to possible states, *ResBy* that determines whether one given task state associated with a time point can be the consequence of another given task state associated with an earlier time point and *pathToFunc* that given a path returns a function to represent it. Given these functions, all the DDM algorithms implemented for the ANTS domain can be applied, as long as the complexity of these functions is

low. Thus, we believe that the results obtained for the ANTS simulations will carry over to any such domain. In particular, we showed that (i) the hierarchy model outperforms a flat one; (ii) the flat mobile dynamic sector setting can be used in situations where communications is not possible; (iii) increasing resources increases the performance; (iv) under the identified constraints, it is beneficial to add more levels to the hierarchy; and (v) the DDM can handle situations of noisy communications.

Agent and organizational designs typically vary along a number of dimensions. Agent architectures can vary from very simple designs, usually deployed in large numbers and crafted so that some set of desirable global properties might emerge (Shehory et al. 99), to complex designs usually based on some variation of the BDI agent model in which agents behaviors are linked to team roles and commitments.

On the organizational side, societal structures are usually either prescriptive in design - in the sense that a particular structure is assigned to a team - or dynamic, adapting to problem changes. In the latter case, organizations can be identified as consequences of role persistence in team interactions (.Durfee et al. 87). Prescriptive structures might derive from task decomposition structures or segmentations based on capabilities, commitments, spatio-temporal divisions, or *a priori* authority structures.

The benefits of hierarchical organizations have been argued by many. So and Durfee draw on contingency theory to examine a variety of hierarchical organizations benefits; they portray a hierarchically organized network monitoring system for task decomposition and also consider organizational self-design (So and Durfee 92, 96). DDM differs in its organization use to dynamically balance computational load and also in its algorithms for support of mobile agents.

The idea of combining partial local solutions into a more complete global solution goes back to early work on the distributed vehicle monitoring testbed (DVMT) (Lesser et al. 87). DVMT also operated in a domain of distributed sensors that tracked objects. However, the algorithms for support of mobile sensors and for the actual specifics of the Doppler sensors themselves is novel to the DDM system. Within the DVMT, Corkill and Lesser investigated various team organizations in terms of *interest areas* which partitioned problem solving nodes according to roles and communication, but were not initially hierarchically organized (Scott 92). Wagner and Lesser examined the role that knowledge of organizational structure can play in control decisions (Wagner and Lesser 00).

Alternative approaches to realtime distributed resource allocation are being explored within the ANTS program (Soh and Tsatsoulis 01). All of those approaches assume that agents are stationary. Vincent *et al* use a limited team organization, assigning agents to sector managers (Vincent et al. 01). Each sector manager is responsible for fusing information for tracking; a hierarchical organization serves to limit communication among agents and to provide a measure of fault tolerance: if a sector manager is disabled, another can fill in.

References

ANTS Program Design Document, unpublished.

D. Corkill and V. Lesser, "The use of meta-level control for coordination in a distributed problem solving network," IJCAI 1983.

Richard J. Doviak and Dusan S. Zrnic. Doppler radar and weather observations. Orlando, Academic Press, 1984.

E. Durfee, V. Lesser, and D. Corkill, "Coherent cooperation among communicating problem solvers,"*Readings in DAI,262-284,* 1987.

R.P. Feynman. The Feynman Lectures on Physics. Addison-Wesley Publishing Company, chapters 12-14, Bombay, India, 1963.

Gill, Thomas P., The Doppler effect: an introduction to the theory of the effect, London: Logos, 1965.

T. Ishida, L. Gasser, and M. Yokoo, "Organization self design of production systems," *IEEE Transactions on Knowledge and Data Engineering,* 4(2):123-134, 1992.

Pöss, Christian Doppler in Banska Stiavnica, in *The Phenomenon of Doppler* (Prague, 1992), 55-62.

R. Scott, "*Organizations: Rational, Natural and Open*," Prentice-Hall, 1992.

O.Shehory, S. Kraus and O. Yadgar, "Emergent cooperative goal-satisfaction in large scale automated-agent systems," *Artificial Intelligence Journal,* 110(1), pages 1-55, 1999.

Young-pa So and Edmund Durfee , "A distributed problem-solving infrastructure for computer network management, International Jornal of Intelligent and Cooperative Information Systems, 1992.

So, Y., and Durfee, E.H., "Designing Tree-Structured Organizations for Computational Agents," *Computational and Mathematical Organization Theory, 2(3), pages 219-246, 1996.*

Leen-Kiat Soh and Costas Tsatsoulis. Reflective Negotiation Agents for Real-Time Multisensor Target Tracking, IJCAI-01, Vol. 2, 2001.

T. Wagner and V. Lesser, "Relating Quantified Motivations for Organizationally Situated Agents," *ATAL 2000.*

Lesser, V.R., Corkhill, D.D., and Durfee, E.H. *An update on the Distributed Vehicle Monitoring Testbed,* CS Technical Report 87-111, University of Massachusetts, Amherst, 1987.

Proceedings of the AAAI Fall Symposium on Negotiation Methods for Cooperative Systems, AAAI Press, 2001.

Vincent, Regis, Horling, Bryan, Lesser, Victor and Wagner, Thomas. "Implementing Soft Real-Time Agent Control." In Autonomous Agent 2001, Montreal, June, 2001, AAAI.