Chapter 9

SCALING-UP DISTRIBUTED SENSOR NETWORKS: COOPERATIVE LARGE-SCALE MOBILE-AGENT ORGANIZATIONS

Osher Yadgar[1], Sarit Kraus[1,2], and Charles L. Ortiz, Jr.[3]

[1] *Department of Computer Science*
Bar Ilan University
Ramat Gan, 52900 Israel
{yadgar,sarit}@macs.ac.il

[2] *Institute for Advanced Computer Studies*
University of Maryland
College Park, MD 20742

[3] *Artificial Intelligence Center*
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
ortiz@ai.sri.com

Abstract We present a system called the Distributed Dispatcher Manager (DDM) for effectively managing very large-scale networks of thousands of sensor agents and thousands of objects. DDM makes use of a hierarchical team organization in which the solution process is distributed into smaller fragments of problems that can be solved partially by simple agents. We present extensive experimental results which indicate that problems involving hundreds and thousands of Dopplers and targets cannot be solved in a traditional flat architecture. We also present a new sensor tracking algorithm through which a single agent can track an object by taking multiple sequential measurements and combining them. We then suggest ways to combine partial solutions to form a global solution. We show that the number of levels in the hierarchy influences the accuracy of results. As the number of levels increases the number of tracked targets drops, even though this drop is moderate. However, as the number of levels increases

the time every agent needs to complete its mission drops exponentially. By combining these two results DDM can achieve a balance between these two properties.

In this chapter we consider the complexities that arise when one scales up distributed agent networks to thousands of sensor agents and thousands of objects. We describe a system for effectively managing such networks, called the Distributed Dispatcher Manager (DDM). DDM differs in a number of important ways from the systems which have been discussed so far.

Mobility: We focus on mobile sensor agents; agents remain relatively simple and lightweight.

Organization: The complexity of the distributed control problem for such massive agent systems is managed through a hierarchical organization in which teams of agents are associated with sectors; teams themselves can represent elements of other teams.

Tracking: We have extended the tracking algorithms discussed so far in this book so that a *single* agent can track an object by taking multiple sequential measurements and combining them. We assume that multiple objects can be discriminated within the field of a sensor. Finally, we do not focus on the tracking of a *particular* object, but rather on adequate coverage of given areas.

Task synchronization: One consequence of the above extensions to the tracking algorithm is that the communication requirements between agents are lessened and, in particular, synchronization between agents is not necessary.

Sensors: DDM, in its current state, does not manage the usage of certain resources, such as sensor power utilization. In addition, our treatment of sensor noise is a bit different from the systems described so far in that, in DDM, some measurements are lost, but the ones that are not lost are accurate.

Simulation: In order to experiment with DDM in domains of the above sort, we have developed a simulation which reflects the above assumptions. In some cases, the set of environments constructed for testing the system have been complicated to reflect such complexities; in other cases, a number of simplifying assumptions have been made so as to be able to focus on the scale-up issues (for example, we focus on objects which move in a straight lines).

DDM organizes the Dopplers in teams, each with a distinguished team leader. A team is assigned a specific geographic sector of interest. Each

Doppler can act autonomously within its assigned area while processing local data. Teams are themselves grouped into larger teams. Communication is restricted to flow only between an agent (or team) and its team leader. Each team leader is provided with an algorithm to integrate information obtained from its team members.

We present results from experiments that involved hundreds of agents and more than a thousand objects. These results support our hypothesis that DDM is successful in large-scale environments. The experiments also allow us to examine the question of how to determine the number of levels of the DDM hierarchy in a large-scale system. Our results show that as the number of levels of the hierarchy increases the quality of the results slightly decreases. However, the time complexity of the system decreases exponentially. Consequently, we found that using too few levels may not suffi ce to solve the global problem.

In the next section we describe the large scale ANTS problem and present the main elements of DDM. Subsequently, we detail the comprehensive study we conducted to evaluate the hierarchical solution. We conclude by discussing the major contribution of our solution to the large-scale agent system challenges in terms of capability, accuracy, effi ciency, cost-effectiveness, robustness and fault tolerance.

1. The large scale ANTS challenge problem and the DDM

We consider a large-scale environment where there are many mobile targets and many mobile Dopplers moving in a specifi ed geographic area. The goal of the DDM system is to track the targets. Each target moves in a steady velocity along a straight line. Targets differ from each other by their motion properties. Motion properties defi ne the target state, location and velocity, at any given time. Both location and velocity are vectors. The location vector is referred to in the physics literature as the radius vector, the vector from the axis origin (0,0) to a target. The velocity vector describes the change in a target's location every second. A steady motion equation may look like the following [Feynman 1963]:

$$f(t) = \langle \overline{r}_0 + \overline{v} \cdot t, \overline{v} \rangle$$

where $\overline{r}_0$ is the location of the target at time $t = 0$ and $\overline{v}$ is the velocity vector. The goal of the DDM system is to identify the motion equation of each target in the area.

DDM uses agents to fi nd the set of motion equations that represent the targets. We will refer to this set as the global information map, denoted by *InfoMap*. The base level of the DDM consists of mobile sampling agents. Mobile sampling agents are agents that use simple Doppler sensors to sense targets. Each of these sampling agents moves autonomously according to a predefi ned

movement algorithm. Each agent periodically stops briefly to take measurements. When an agent takes measurements we refer to its state as the *viewpoint* from which a particular object state was measured. The measurements alone are not sufficient to identify the exact state of the observed target. The sampler agent can only determine two possible states of the observed target based on four consecutive measurements. Only one of them is the correct state of the target at the observation time. Using such estimates, a sampler agent then produces what we call a *capsule*. A capsule consists of (a) two possible states of a target, (b) the time associated with the measurements that were used to infer the possible target states, and (c) the state of the agent, i.e., its location and orientation during the measurements. Later on we will show how sampling agents transform measurements into capsules.

The DDM's goal is to estimate the motion equations of the targets using capsules. The equations can be deduced from a sequence of target states. The main problem faced by the DDM with respect to a capsule is how to choose the correct state. To resolve this problem, the DDM makes use of the fact that two states may be the correct states of the same target if they are instances of the same motion equation. We introduce a relation called *ResBy* that holds if the state of one target comes about from another. DDM tries to match states of capsules using this relation and makes linked lists of target states. Each linked list represents a potential target movement. We refer to this linked list of target states as a *path*. DDM also attempts to determine the accuracy of potential paths. It assumes that if two or more target states in a path were recorded by different agents, then the path represents target motion accurately.

In a large-scale environment, DDM would have to link many capsules from the entire area of interest. Applying the relation *ResBy* many times is time consuming. However, there is a low probability that capsules created based on measurements taken far away from one another will fit. Therefore, the solution is distributed. 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 location. Each group has a leader. The sampling agents create capsules and send them to their group leaders. The second level of the hierarchy consists of the sampler group leaders. Each sampler group leader obtains capsules only from the sampling agents in its group. This limits the time that it needs to process the capsules, but may reduce its ability to link between states since it obtains only a portion of the capsules.

The sampler leaders are also grouped according to their areas of responsibility. Each such group of sampler leaders is associated with a zone group leader. A sampler leader sends its zone leader its estimates of target motion equations in its area and capsules that it was not able to use in the estimation process. The third and higher levels of the hierarchy consist of zone group leaders, which in

turn, are also grouped according to their areas of interest. Zone leader agents are responsible for retrieving and combining information from their group of agents. That is, they try to estimate the motion equations based on the estimates they received from agents in their zone. All communication is restricted to exchanges between a group member and its leader. The information unit sent by leaders to their higher-level leaders is called a *local information map*. A local information map, which we refer to as *localInfo*, is a triple consisting of: (i) an accurate solution component consisting of a set of motion equations that with a high probability represent targets; (ii) a semi-accurate solution component consisting of a set of paths; and (iii) a set of capsules that were not used for the formation of any motion equation of (i) or any path of (ii). That is, each zone leader obtains local information maps of all its agents and combines them into an information map of its area. Thus, the top-level leader agent forms a local information map of the entire area.

To conclude, 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 their capsules and integrates them into its *localInfo*.
- Every dT seconds each zone group leader obtains from all its subordinate 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. In the next section we present those algorithms.

2. Descriptions of algorithms

The first algorithm describes the method for constructing a capsule from raw sensed data. This algorithm is activated by each sampler agent and uses consecutive raw sensed data. The second and the third algorithms describe the way in which every group leader processes incoming local information maps of the sub-areas of its zone to generate a more comprehensive local information map of its entire area.

First, we will describe the main data structures that the agents use. In these data structures specifications and in the algorithm descriptions, we will use a dot notation to describe a field in a structure, e.g., *c.sa* is the sampling agent field of the capsule *c*.

Target state: $s = \langle \overline{D}, \overline{V} \rangle$ where D is the location of the target and $\overline{V}$ is its velocity at a given time. For example: $\langle \langle 100, 100 \rangle, \langle 2, -1 \rangle \rangle$ is a target

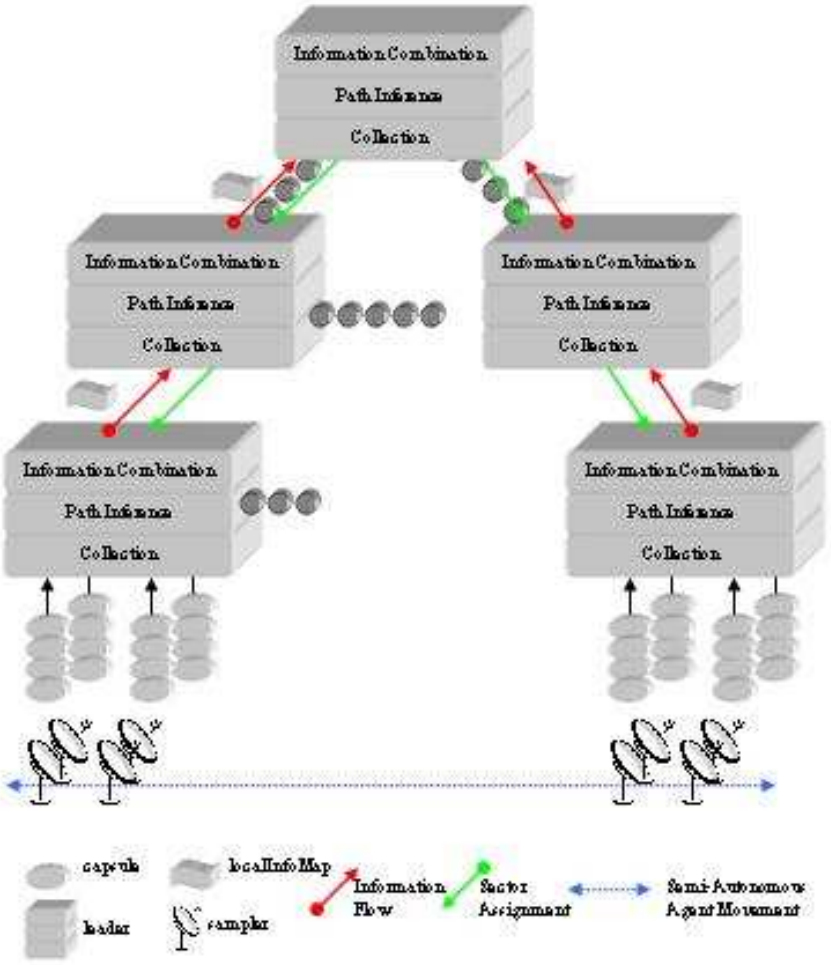

Figure 9.1. DDM hierarchy information fbw diagram.

state where the target was at location $X = 100$ and $Y = 100$ and its velocity was $V_x = 2$ and $V_y = -1$.

Sensing agent state: $sa = \langle \overline{D}, O \rangle$ where $\overline{D}$ is the location of the sensor and O is the orientation of the sensor. For example: $\langle \langle 150, 150 \rangle, \pi/2 \rangle$ is an agent's state where the sensing agent was at location $X = 150$ and $Y = 150$ and had an orientation of $\pi/2$.

Capsule: $c = \langle t, sa, \{s_1, s_2\} \rangle$ where t is the time of the sampling, *sa* is the sensing agent's state during the time the measurements that were used for the formation of the capsule's target states were taken, and s_1, s_2 are two possible target states computed by the sampler agent. An example is, $\langle 30, \langle \langle 150, 150 \rangle, \pi/2 \rangle, \{ \langle \langle 100, 100 \rangle, \langle 2, -1 \rangle \rangle, \langle \langle 200, 200 \rangle, \langle -2, -3 \rangle \rangle \} \rangle$ where the time of the sampling was 30 and the sensing agent state was $\langle \langle 150, 150 \rangle, \pi/2 \rangle$ where the two possible target states were $\langle \langle 100, 100 \rangle, \langle 2, -1 \rangle \rangle$, and $\langle \langle 200, 200 \rangle, \langle -2, -3 \rangle \rangle$.

Path point: $\pi_i = \langle t_i, sa_i, s_i \rangle$ where t_i is the time of the point, sa_i is the sensing agent's state during the time the measurements that were used to compute the point's state were taken, and s_i is the target state of the point. For example: $\langle 30, \langle \langle 150, 150 \rangle, \pi/2 \rangle, \langle \langle 100, 100 \rangle, \langle 2, -1 \rangle \rangle \rangle$ where while the time of the path point was 30, the sensing agent state was $\langle \langle 150, 150 \rangle, \pi/2 \rangle$ and the target state was $\langle \langle 100, 100 \rangle, \langle 2, -1 \rangle \rangle$

That is, while in a capsule there are two possible states associated with measurements, in a path point there is only one. The goal of the agents is to choose the correct one.

Path: $p = \langle \pi_1 \dots \pi_n \rangle$ where π_1 and π_n are the first and the last path points. Every pair of path points in a path satisfies the ResBy relation.

Target state function: $f_{\pi_s, \pi_e}(t) = \langle \pi_s.s_s.\overline{D} + \pi_s.s_s.\overline{V} \cdot (t - \pi_s.t, t), \pi_s.s_s.\overline{V} \rangle$ valid in the range $\pi_s.t..\pi_e.t$. For example: if

$\pi_s = \langle 30, \langle \langle 150, 150 \rangle, /pi/2 \rangle, \langle \langle 100, 100 \rangle, \langle 2, -1 \rangle \rangle \rangle$ and

$\pi_e = \langle 40, \langle \langle 450, 25 \rangle, /pi/4 \rangle, \langle \langle 120, 90 \rangle, \langle 2, -1 \rangle \rangle \rangle$ then $f_{\pi_s, \pi_e}(t) = \langle \langle 100, 100 \rangle + \langle 2, -1 \rangle \cdot (t - 30), \langle 2, -1 \rangle \rangle$ and at t=30 we have that $f_{\pi_s, \pi_e}(30) = \langle \langle 100, 100 \rangle, \langle 2, -1 \rangle \rangle$ and at t=40 we will have that, $f_{\pi_s, \pi_e}(40) = \langle \langle 120, 90 \rangle, \langle 2, -1 \rangle \rangle$. For simplicity, we will refer to this function as $f(t) = \langle \overline{D}(t), \overline{V} \rangle$ and to its properties: $f.\overline{D}(t), f.\overline{V}(t), f.t_s$ and $f.t_e$.

Local information map: $\langle \langle f^1, \dots, f^h \rangle, \langle p_i, \dots, p_l \rangle, \langle c_i, \dots, c_m \rangle \rangle$. To form a local information map out of raw sensed data agents should follow a set of steps corresponding to the following stages of data evolution: (i) measurements, (ii) capsules, (iii) path, (iv) target state function and (v) local information map.

2.1 The raw data transformation and capsule generation algorithm

The process proceeds as follows. A sampling agent deduces a set of possible target states at a given time to form a capsule. A sampling agent accomplishes this by first taking four consecutive measurements and then creating a new capsule, c, such that the time associated with that capsule corresponds to the time of the last measurement. The state of the sampling agent while taking the measurements is given by *c.sa*. The target states resulting from the application of the function *rawDataTransformation* to the four consecutive measurements is assigned to *c.states*. The agent stores the capsules until it is time to send them to its group leader. After delivering the capsules to the group leader the sampler agent deletes them. We will now show now how an agent transforms four consecutive measurements into a capsule.

A measurement is a pair of amplitude, η, and radial velocity, v_r, values for each sensed target. A radial velocity is the velocity of a target towards the measuring Doppler. Given a measurement from a Doppler radar the target location can be computed using the following equation:

$$R_i^2 = \frac{k \cdot e^{\frac{-(\vartheta_i - \beta)^2}{\sigma}}}{\eta_i} \tag{9.1}$$

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; θ_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 . It is possible to infer the exact location of a target by intersecting three different measurements taken at the same time by three different Dopplers. Using the intersection method is very problematic in large scale systems as it requires full synchronization and cooperation between groups of three Dopplers. Thus, the DDM uses measurements from only one Doppler to deduce a possible target state.

It is known that if the location of an object at time 0 is $\overline{D}_o$ and its velocity is $\overline{V}$ then the next location, $\overline{D}_1$, at time 1 of the object is given by:

$$\overline{D}_1 = \overline{D}_0 + \int_{t_0}^{t_1} \overline{V}\, dt \tag{9.2}$$

where $\overline{D}_t$ is the displacement of the object in time t. If we consider the distance from the center of the Doppler we have that

$$R_1 = R_0 + \int_{t_0}^{t_1} V_r\, dt \tag{9.3}$$

where R_t is the displacement from the center of the sensor at time t and V_r is the relative velocity between the Doppler and the target in the direction of the Doppler's center. We assume that the acceleration of a target over a short

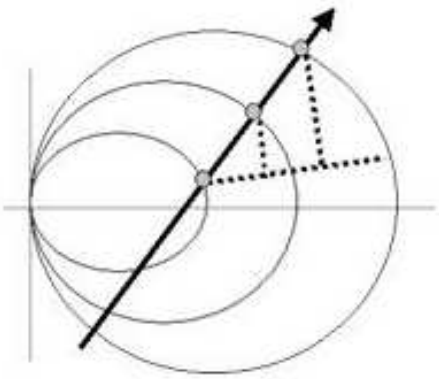

Figure 9.2. Target sampling by one Doppler.

period of time is zero. The next target location is therefore:

$$R_1 = R_0 + V_r \cdot (t_1 - t_0) \tag{9.4}$$

We denote $(t_1 - t_0)$ by $t_{1,0}$. From the relation between R, θ and η, given by equation 9.1, we can find the next angle as a function of the former.

In Figure 9.2 the dark arrow represents a target movement vector, the small circles along the target movement represent target locations, (R_0, θ_0),(R_1, θ_1) and (R_2, θ_2), at time t_0, t_1, and t_2, respectively, as sensed by the Doppler. Following the projection of R_1 and R_2 over R_0, and R_{1,R_0} and R_{2,R_0} respectively, as shown by the dotted line, we have the following:

$$\frac{R_{1,R_o} - R_0}{R_1 \cdot sin(\theta_1 - \theta_0)} = \frac{R_{2,R_0} - R_0}{R_2 \cdot sin(\theta_2 - \theta_0)} \tag{9.5}$$

Trigonometrically, we may write R_{1,R_0} and R_{2,R_0} as

$$R_{1,R_0} = R_1 \cdot cos(\theta_1 - \theta_0)$$

and

$$R_{2,R_0} = R_2 \cdot cos(\theta_2 - \theta_0)$$

by equation 9.1, θ_i can be written as

$$\theta_i = \beta \pm \sqrt{-\sigma \cdot ln(\frac{n_1}{k} \cdot R_i^2)} \tag{9.6}$$

By substituting R_1 as given by equation 9.4 into equation 9.6 we can deduce that the location, (R_1, θ_1), at t_1 of a sensed target may be written as a function of θ_0 as specified in the next proposition.

THEOREM 9.1 *Assuming that the acceleration of a target in a short time period, $t_{1,0}$, is zero, the next location of the target is then given by*

$$\theta_1(\theta_0) = \beta \pm \sqrt{-\sigma \cdot ln(\frac{\eta_1}{k} \cdot (R_0 + V_{r0} \cdot t_{1,0})^2)} \tag{9.7}$$

while

$$R_0 = \sqrt{\frac{k \cdot e^{\frac{-(\vartheta_0 - \beta)^2}{\sigma}}}{\eta_0}}$$

and

$$R_1 = \sqrt{\frac{k \cdot e^{\frac{-(\vartheta_1 - \beta)^2}{\sigma}}}{\eta_1}} \tag{9.8}$$

Where $R_0, \theta_0, \eta_0, V_{r0}$ represent values of the target at time $t = 0$ and θ_1, η_1 represent values of the target at time $t = 1$. The same holds for the next angle, θ_2.

An agent can use the relationship given by equation 9.7 for θ_0, θ_1 and θ_2 together with equation 9.5 to find θ_1 and θ_2 from θ_0. However, the value of θ_0 is not known and thus can't be used in equations 9.7 and 9.5. Therefore the algorithm examines the range of $0, ..., 2\pi$ to determine which value of θ_0 solves these two equations.

Note that, given a specific value of θ_0, the result of equation 9.7 may lead to two valid solutions. This is the reason for the use of capsules: the sampling agent will leave the decision of determining the correct target states to the higher levels.

Equation 9.5 cannot be solved symbolically and therefore the sampler agent uses computational methods. The sampler agent explores the range of θ_0 and looks for suitable locations corresponding to θ_0. Only certain angles will fit the above equation. To be more precise, the sampler agent uses one more sample and applies the same mechanism to θ_1, θ_2 and θ_3. Comparing the results from both cases improves the accuracy of the results. The calculated angles are used to form a set of possible pairs of location and velocity of a target (i.e., a capsule). In the algorithms of figures 9.3, 9.4 and 9.5 we will use the notation $sample_i$ to represent a measurement $\langle \eta, V_r \rangle$ at $t = i$.

THEOREM 9.2 *The time complexity of the* capsule generation algorithm *is O(1).*

Find θ_0 function
Input: *sa,*$sample_0, sample_1, sample_2$
Output: θ_0
minimum_diff=ϵ
min_θ_0=-1

For $\theta_0 = 0$ to 2π in δ steps
    calculate θ_1 using θ_0 by equation 9.7
    calculate θ_2 using θ_0 by equation 9.7
    diff = the difference between the left side the right side of
    equation 9.5 using θ_1 and θ_2
    if (diff < minimum_diff)
            minimium_diff=diff
            min_θ_0=θ_0

Return min_θ_0

Figure 9.3. Finding a value of θ_0.

rawDataTransformation function
Input: *sa,*$sample_0, sample_1, sample_2, sample_3$
Output: *target states*
θ_0 = Find $\theta_0(sa, sample_0, sample_1, sample_2)$
θ_1 = Find $\theta_0(sa, sample_1, sample_2, sample_3)$

if ($\theta_0 \neq -1$ && $\theta_1 \neq -1$)
  θ_3 = calculate θ_3 using θ_0 by equation 9.7
  θ_3^* = calculate θ_3^* using θ_1 by equation 9.7
  if (the difference between θ_3 and θ_3^* < epsilon)
            Return $\langle \overline{D}(\theta_3), \overline{V}(\theta_3) \rangle, \langle \overline{D}(-\theta_3), \overline{V}(-\theta_3) \rangle$
else
  Return null

Figure 9.4. The rawDataTransformation function

Capsule generation algorithm
Input: *sa*,$sample_0, sample_1, sample_2, sample_3$
Output: *capsule*

targetStateSet=rawDataTransformation(sa,$sample_0, sample_1, sample_2, sample_3$*)*

If *(targetStateSet* $\neq$ *null)*
  capsule=new Capsule()
  capsule.sa=sa
  capsule.states = targetStateSet
else
  capsule= null

Return *capsule*

Figure 9.5. Capsule generation algorithm.

Proof: While generating a capsule, the *rawDataTransformation* function uses the *Find* function twice. The time complexity of considering the range of all angles from 0 to 2π in *Find* is O(1) as it does the same simple assignments 2π $/\delta$ times. Therefore, the time complexity of the whole algorithm is O(1).

However, despite the low order, this algorithm can be CPU intensive. A sampler agent applies this algorithm every four consecutive measurements. Thus, it may have to apply it many times if it acquired many measurements or if many targets passed through its sector. A sampler agent may sometimes not have suffi cient resources to execute this algorithm many times in real time. In such cases, one can consider using simpler sampling agents, i.e., with smaller detection sectors, which will reduce the computation load on a single agent. One may also consider taking fewer samples.

Example: We will now present an example of how a sampler agent forms a capsule from four consecutive measurements. Consider a case of a sampler agent located at the coordinates $\overline{D}_a = \langle 200, 200 \rangle$ with orientation of 0 degrees. The sampler uses a Doppler with the characteristic $k = 1$ and $\sigma = 1$ and maximum detection range of 100 meters (see Figure 9.1). Consider the following measurements taken by the Doppler.

Time	η	V_r
0	1.08E-04	0.014141
1	1.11E-04	0.042405
2	1.15E-04	0.070619
3	1.18E-04	0.098748

Scanning the range of $0, ..., 2\pi$ for the value of θ_0 in *Find* θ_0 function, the algorithm computes $\theta_1(\theta_0)$ and $\theta_2(\theta_0)$ using equation 9.7. At the end of the scanning loop the algorithm finds out that while θ_0 had the value of -0.7854 the difference between the left side and the right side of equation 9.5 was minimal. Doing the same in the case of θ_1, the algorithm finds that when θ_1 was -0.7654 the difference between the left side and the right side of equation 9.5 was minimal.

The algorithm then uses equation 9.5 to find that $\theta_3(\theta_0)$ is -0.72547 and that $\theta_3^*(\theta_1)$ is also -0.72547. Realizing that the values of the calculated $\theta_3(\theta_0)$ and $\theta_3^*(\theta_1)$ are equal, the algorithm constructs two targets states. The first is $\langle \overline{D}(\theta_3), \overline{V}(\theta_3) \rangle$ and the second is $\langle \overline{D}(-\theta_3), \overline{V}(-\theta_3) \rangle$. The values of $\overline{D}(-\theta_3)$ and $\overline{V}(-\theta_3)$ are given by the following equations:

- $\overline{D}(\theta_3) = \langle R(\theta) \cdot sin(\theta) + \overline{D}_a.x, R(\theta) \cdot cos(\theta) + \overline{D}_a.y \rangle$
- $\overline{V}(\theta) = \langle \overline{V}_r \cdot sin(\theta), \overline{V}_r \cdot cos(\theta) \rangle$

where $R(\theta)$ is given by equation 9.8 and, in our case, $R(\theta_3) = 70.8378$. According to our example the two target states will be $\langle \langle 153, 253 \rangle \langle 1, 1 \rangle \rangle$ and $\langle \langle 247, 253 \rangle \langle -1, 1 \rangle \rangle$.

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

Figure 9.6 describes 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): this is needed since, as we noted earlier, target 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 target and therefore it may be that different functions coming from different agents will refer to the same target. 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.

Step 1 - Obtaining new information algorithm
In: LocalInfo
Out: updated localInfo

```
if activated as Samplergroup leader
  for each subordinate sampler, sampler
        additionalCapsules =obtain set of capsules from each sampler
        localInfo.unusedCapsules= localInfo.unusedCapsules ∪
              additionalCapsules
else % activated in Zone group leader
  for each subordinate leader, leader
        %in this part we identify identical functions and leave only one of them
        additionalLocalInfo = ask each leader for its local info
        additionalCapsules=additionalLocalInfo.unusedCapsules
        additionalInfoMap=additionalLocalInfo.infoMap
        localInfo.unusedCapsules=localInfo.unusedCapsules ∪
              additionalCapsules
        mergedFunctions(localInfo.infoMap,additionalInfoMap);

return infoMap, unusedCapsules
```

Figure 9.6. Obtaining new information algorithm.

mergeFunctions algorithm
In: target function sets: F, F'
Out: updated target function: F

for eachstatefunction, f^i, in F'
  merged = false
  for each state function, f^j, $f^i \neq f^j$, in F && not merged
    if ($f^i.\overline{D}(0) - f^j.\overline{D}(0) < \epsilon\overline{D}$ && $f^i.\overline{V} - f^j.\overline{V} < \epsilon\overline{V}$)
      $f^j.t_s = min(f^i.t_s, f^j.t_s)$
      $f^j.t_e = max(f^i.t_e, f^j.t_e)$
      merged=true
    if (not merged)
      $F = F \cup \{f^i\}$

Return F

Figure 9.7. mergeFunctions algorithm.

LEMMA 9.3 *Let* $p^1 = \langle \pi_s^1, \ldots, \pi_e^1 \rangle, p^2 = \langle \pi_s^2, \ldots, \pi_e^2 \rangle$ *be two paths, where* $\pi_j^i = \langle t_j^i, sa_j^i, s_j^i \rangle$ *and*

$$f^1_{\pi_s^1, \pi_e^1}(t) = \langle \pi_s^1.s_s.\overline{D} - \pi_s^1.s_s.\overline{V} \cdot (t - \pi_s^1.t), \pi_s^1.s_s.\overline{V} \rangle$$

$$f^2_{\pi_s^2, \pi_e^2}(t) = \langle \pi_s^2.s_s.\overline{D} - \pi_s^2.s_s.\overline{V} \cdot (t - \pi_s^2.t), \pi_s^2.s_s.\overline{V} \rangle$$

If $ResBy(\langle t_s^1, s_s^1 \rangle, \langle t_s^2, s_s^2 \rangle)$ *then for any* $f^1_{\pi_s^1, \pi_e^1}(t), f^2_{\pi_s^2, \pi_e^2}(t)$, $f^1_{\pi_s^1, \pi_e^1}(t) = f^2_{\pi^2, \pi_e^2}(t)$

The *mergeFunctions* algorithm shown in Figure 9.6 is based on lemma 9.3. In that algorithm, the leader uses the *ResBy* relation to check whether the first state of the target state function results from the first state of a different target state function. If one of the states results from the other, the leader changes the minimum and the maximum triplets of the target 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 corresponds to the largest range given the found points. In case that a leader cannot find any target state function to meet the subordinate's function, the leader will add it as a new function to its *infoMap*.

PROPOSITION 9.1 *The time complexity of the* obtaining new information *algorithm is* $O(T^2)$ *where T is the number of targets in the* τ *seconds window of time in which target information is kept by an agent.*

Proof: While obtaining new information, agents implementing the algorithm query each subordinate agent for information. The number of subordinate agents is predefined and therefore constant. In the case of the sampler leader the algorithm combines all capsules. The number of capsules depends on the number of targets up to a constant factor. The constant factor depends on predefined constant values, such as, the number of agents and time period for sampling. Therefore the time complexity for the sampler leader component is O(T). However, for each subordinate leader, the zone group leader also performs the mergeFunctions algorithm. The time complexity of the mergeFunctions algorithm is $O(T^2)$ as it runs over a set of task state functions for every other task state function in another set.

The second step, as shown in Figure 9.8 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 target state out of each capsule. This constraint is based on the following lemma. According to this lemma one state of one capsule cannot be in a *ResBy* relation with two different states in another capsule, with respect to the capsule's time.

LEMMA 9.4 *Let* $C^1 = \langle t^1, sa^1, \langle s_1^1, s_2^1 \rangle \rangle, C^2 = \langle t^2, sa^2, \langle s_1^2, s_2^2 \rangle \rangle$ *then if* $ResBy(\langle t^1, s_1^1 \rangle, \langle t^2, s_1^2 \rangle)$ *then*

- *(i)* $ResBy(\langle t^1, s_1^1\rangle, \langle t^2, s_2^2\rangle)$ *is false and*
- *(ii)* $ResBy(\langle t^1, s_2^1\rangle, \langle t^2, s_1^2\rangle)$ *is false.*

Proof: If a capsule could be in such a relationship with both target states, the two targets states would stand in a ResBy relation between themselves. Given that the two target states have the same creation time and that a target cannot be at the same time in two places, that is not possible.

For the algorithm in Figure 9.8 that creates new paths we add two temporary fields to two of the structures only for the purposes of the algorithm. The first is a boolean flag named *mark* that will be added to the capsule structure. The second is a pointer to the original capsule that will be added to every triple stored in a path. In the first phase, every agent tries to fit every state in unused capsules to an existing path. If the state does not fit, a new path will be created, starting at that state. The second phase the agent separates the paths into accurate and semi-accurate paths according to the number of sampling agents generating them.

Example: Consider a case in which Finding_new_paths algorithm receives the following set of capsules as unusedCapsules:

Sensing Agent State				*Target State A*		*Target State B*	
Capsule	Time	Location	Orientation	Location	Velocity	Location	Velocity
0	0	0,0	0	100,100	5,5	60,60	3,-1
1	0	0,0	0	50,50	2,3	30,40	-2,-2
2	1	0,0	0	105,105	5,5	63,59	3,-1
3	1	0,0	0	70,80	2,3	52,53	2,3
4	2	0,0	0	66,58	-1,3	110,110	5,5
5	3	10,10	0	56,59	2,3	30,30	3,3

Let us assume that at the beginning of the algorithm shown in Figure 9.8, allPaths does not contain a path (line 2). Considering each target state in each capsule (lines 3 to 5), we start with TargetStateA of capsule 0. Because allPaths does not contain a path, a new path will be created with TargetStateA of capsule 0 at its head (lines 16 to 18). The next state, TargetStateB of capsule 0, will be tested to see if it is in a ResBy relation with any of the tails of the paths, stored in allPaths (line 10). It does not stand in a ResBy relation with the only tail that exists: TargetStateA of capsule 0. Therefore, a new path will be created with TargetStateB of capsule 0 at its head (lines 16 to 18). TargetStateA and TargetStateB of capsule 1 will result in a new path as well. However, TargetStateA of capsule 2 is in a ResBy relation with TargetStateA of capsule 0 and will join its path as a new tail (line 14). TargetStateB of capsule 2 will do the same with TargetStateB of capsule 0. At the end of the first phase 6 paths will be formed, 3 of them are made from more than one capsule.

In Figure 9.9 we summarize the outcome of phase 1. Each arrow corresponds to a ResBy relation between two path points. Every capsule contains

Step 2 - Finding new paths algorithm
In: unusedCapsules
Out: updated unusedCapsules, *accurateFunctions, mediocrePaths*
% phase 1: make links
(1) sort(unusedCapsules) % by time stamp
(2) *allPaths* = {}

(3) for each capsule, $c = \langle t, sa, \{s_1, s_2\}\rangle$, in unusedCapsules
(4) cap.mark=false % marking for phase 2
(5) for each target state, si, in cap states
(6) linked=false
(7) % because of the above assumption and given that the path
(8) % elements came from capsules there will be only one suitable
(9) %path. Therefore, we exit the loop after finding such a path
(10) for every last triplet, $\langle t\_last, sa\_last, s\_last\rangle$, in each path, p,
(11) in *allPaths* && not linked
(12) if $(ResBy(\langle t\_last, s\_last\rangle, \langle t, si\rangle)$
(13) or $(t\_last = t \&\& sa\_last \neq sa))$
(14) $p = p * \langle\langle t, sa, si\rangle\rangle$
(15) linked = true
(16) if (not linked)
(17) $p = \langle\langle t, sa, si\rangle\rangle$
(18) allPaths=allPaths $\cup$ {p}
(19)% phase 2: collect target representing paths that has no common capsules
(20)% when giving a greater priority to paths with more viewpoints
(21) sort($allPaths$) % by number of viewpoints
(22) paths={}
(23)for each path, p, in *allPaths*
(24) if (not isAnyCapsuleMarked(p) & & numberOfViewpoint(p) > 1)
(25) markAllCapsules(p)
(26) unusedCapsules.=unusedCapsules - allCapsules(p)
(27) accurateFunction=accurateFunction
(28) accurateFunctions= accurateFunctions $\cup$ pathToFunc(p)
(29)if activated as top-level leader
(30) mediocrePaths=collectMediocrePaths(*allPaths*)
(31)else
(32) mediocrePaths={}
(33) Return unusedCapsules, *accurateFunctions,* mediocrePaths

Figure 9.8. Finding new paths algorithm.

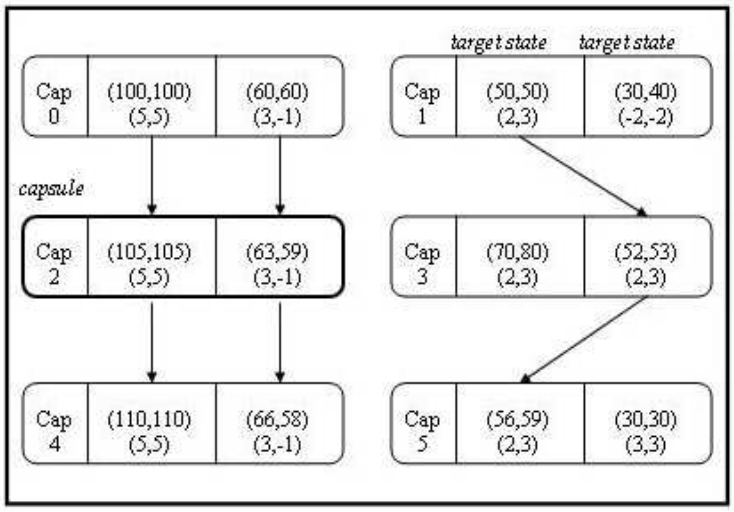

Figure 9.9. An example of an outcome of phase 1

the original sampler state, which is not shown in the figure due to space limitations.

In phase 2, the algorithm finds that one of the paths in *allPaths* is formed by capsules generated by agents with different states. This path is an accurate path and will be added to the accurate paths set. The algorithm removes all target states that share the same capsule with accurate paths' target states. At the end of the second path, the algorithm looks for semi-accurate paths. A semi-accurate path is a path of target states sensed by the same agent at the same agent state.

The function *pathToFunc* receives a path and returns a function based on it. In our case it will receive the paths shown in the left side of the figure and return:

$f_{\pi_s,\pi_e}(t) = \langle\langle 50, 50\rangle + \langle 2, 3\rangle \cdot t, \langle 2, 3\rangle\rangle$ with respect to the first and the last path points: $\pi_s = \langle 0, \langle\langle 0, 0\rangle, 0\rangle, \langle\langle 50, 50\rangle, \langle 2, 3\rangle\rangle\rangle$ and $\pi_e = \langle 3, \langle\langle 0, 10\rangle, 0\rangle, \langle\langle 56, 59\rangle, \langle 2, 3\rangle\rangle\rangle$.

PROPOSITION 9.2 *The finding new paths algorithm time complexity is* $O(T^2)$ *where T is the number of targets in the* τ *seconds window of time passing through the controlled area.*

Proof: In this algorithm, every leader runs over a set of paths for every capsule in its set of capsules. The paths and the capsule sets are correlated

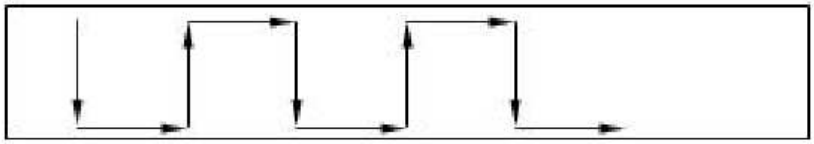

Figure 9.10. Patrol movement pattern

with the number of targets in the time period of τ, and therefore results in a time complexity of $O(T^2)$.

2.3 The movement of a sampler agent

While the integration algorithms play an important role in producing an accurate infoMap, ultimately, the accuracy of the infoMap fundamentally depends on the accuracy of the observations made by the sampling agents. There are several degrees of freedom associated with the movements of sampler agents. At this point in our research we wanted the sampler agents to move autonomously according to a predefined algorithm without making any assumptions regarding target location. We hypothesize that the following criteria should be considered when determining the sampler agent's behavior: (i) the union of all the sensed area at time *t* should be maximized and (ii) the intersection of the areas sensed by sampling agent *s* at time *t* and at *t*+1 should be minimized. One of the ways to achieve this is to move in the pattern demonstrated in figure 9.10. We refer to this pattern as the *Patrol movement pattern*.

We compared the patrol movement pattern with a steady random movement that was used by the agents in [Yadgar 2002; Yadgar 2003]. A steady random movement is defined as a movement in a random direction and velocity. Upon reaching the end of the controlled zone, the velocity and the direction is changed and re-directed into the zone. We found out that most of the time the patrol movement pattern is more efficient than the random one. Hence, we will present our simulation results using the former.

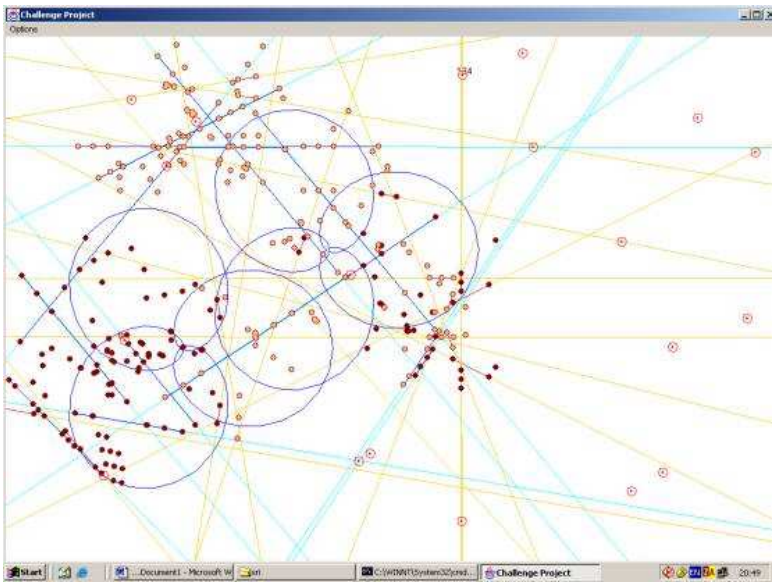

Figure 9.11. Simulating 2 Doppler radars tracking 30 targets. The dots represent sampled target states. The shades of lines represent 100% and 50% tracked targets.

3. Simulation, experiments and results

3.1 Simulation environment

We developed a simulation of the Doppler domain to study the problems associated with the application of the DDM in a large-scale environment.[1] The simulation consists of an area of a fixed size in which Dopplers attempt to extract the object state functions of moving targets. Each target has an initial random location along the border of the area and an initial random velocity of up to 50 km. per hour in a direction that leads inwards. Targets leave the area when reaching the boundaries of the zone. Each target that leaves the area causes a new target to appear at a random location along the border and with a random velocity in a direction that leads inwards. Therefore, each target may remain in the area for a random time period. Figures 9.11 and 9.12 are screendumps of a simulation in progress.

To run our simulations we used a Pentium 4 computer with Windows 2000 as an operating system and 1GB RAM.

[1] We did not use the RadSim simulator.

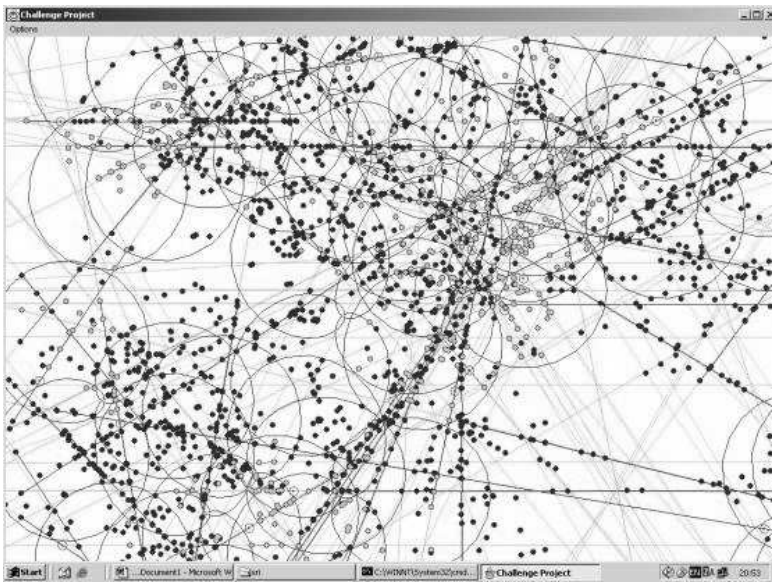

Figure 9.12. Simulating 20 Doppler radars tracking 30 targets. The dots represent sampled target states and the lines represent tracked targets.

3.2 Evaluation methods

We collected the state functions produced by 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 tracked if the path identified by the agent satisfied the following: (a) the maximum distance between the calculated location and the real location of the target did not exceed 1 centimeter, and (b) the maximum difference between the calculated v(t) vector and the real v(t) vector was less than 1 centimeter per second.

In addition, the identified object state functions could be divided into two categories: one in which 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 object state function and denoted as accurate tracking. The other in which two possible object state functions were associated with a target. Each was assigned a 50% probability of corresponding to the actual function. We refer to these functions as "semi-accurate." We will say that one set of agents *did better* than another if it reached a higher tracking percentage and a lower tracking time with respect to the accurate 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 in tracking, according to the

above definitions, by the actual number of targets during the simulated hour. We considered only targets that exited the controlled zone. The *tracking time* was defined as the time that the agents needed to find the object 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.

Basic settings. The *basic setting* for the environment corresponded to an area of 10,000 by 10,000 meters. In each experiment, we varied one of the parameters of the environment, keeping the other values of the environment parameters as in the basic settings.

Each Doppler moved one second and stopped for 5 seconds to take 5 measurements. The maximum detection range of a Doppler in the basic setting was 100 meters; the number of Dopplers was 1,000. The controlled area was divided into 1,000 equal rectangles, each 400x250 squared meters. Each patrolling Doppler was assigned to such an area and executed the patrol movement pattern. 1,000 Dopplers with a detection range of 100 meters each, can cover together up to approximately 8,000,000 squared meters, which is 8% of the controlled area.

The number of targets at a given time point was 1,500. In total, during one hour 5,700 targets entered the controlled area and 4,200 of them exited the area.

In the basic setting we used a hierarchy of 4 levels: three levels of zone group leaders and one of sampler group leaders. Each of the zone group leaders divided its zone into 4 areas and assigned a sub-leader to each one of them. Therefore there was one leader at the top level, 4 at the second level, 16 at the third and 64 at the fourth. Each Doppler sensor communicated with one of the fourth-level leaders.

3.3 Results

We conducted three sets of tests: (i) evaluating the basic settings, (ii) investigating the influences of the number of levels in the hierarchy, and (iii) studying the tolerance towards faulty sensing agents, leaders and sensing noises. At this state of our research, samplers and leaders do not react to the changes in the functioning agent community.

Basic settings results. Our hypothesis was that by applying the DDM hierarchy model we would be able to very quickly track many targets. We also hypothesized that the tracking period for each target would be significant. We ran the simulation using the basic settings and evaluated the results.

Figure 9.13 shows the percentage of tracked targets as a function of the time each target remained in a zone. To put this histogram in context we added

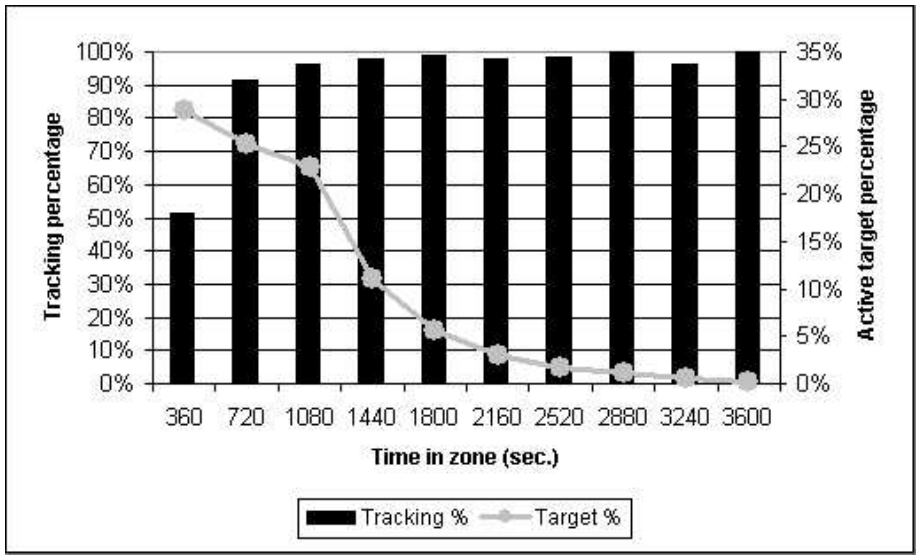

Figure 9.13. Tracking percentage by time in zone (Sec.)

the gray graph that corresponds to the right legend. This graph reflects target distribution with respect to the time spent in the zone.

We can see that the system accurately tracked 83% of the targets. This was achieved with Dopplers covering only 8% of the area. A little more than 50% of the targets that stayed in the controlled zone less than 360 seconds were tracked. Note that most of the targets passing through the simulated area remained in the area less than 720 seconds. During that time the patrol method tracked many targets and therefore achieved rapid tracking.

Figure 9.14 shows the number of targets that were tracked upon entering a zone. Most of the tracking was achieved in less than 2 minutes from the time of a target's entrance into the zone. The system tracked 71% of its tracked targets in this period.

Figure 9.15 plots the tracking duration, which is the period of time between the first and the last time a target was detected. The figure indicates that the system tracks more targets for less duration. However, it tracks most of the tracked targets for more than 6 minutes.

Level comparison. We investigated the influence of the number of levels in the hierarchy. Our hypothesis was that too few levels would overload the leader agents so they would not have enough time to process the information. We also anticipated that, as more leader agents were involved in generating the global solution, a less accurate solution would result.

Figure 9.16 presents the tracking performance of the system as a function of the number of levels in the hierarchy. As we hypothesized, the system tracked

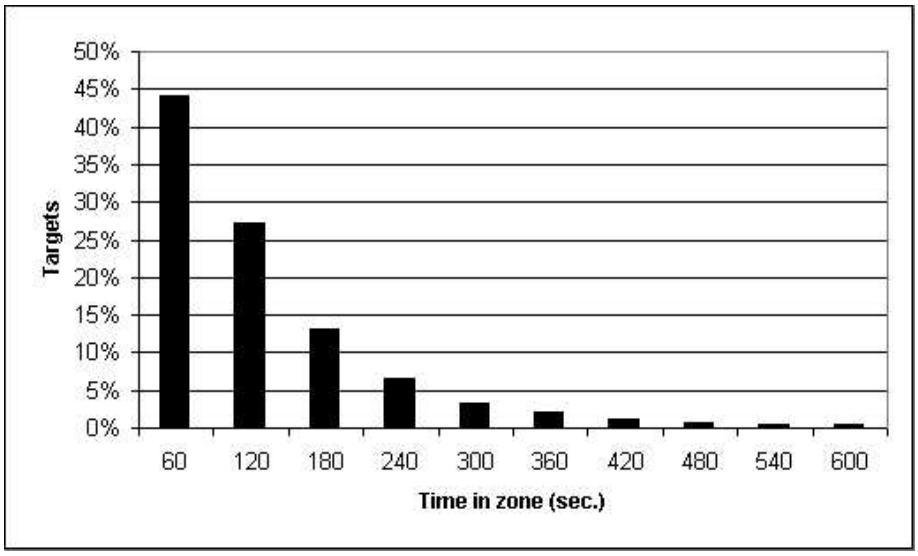

Figure 9.14. Time to track distribution (Sec.)

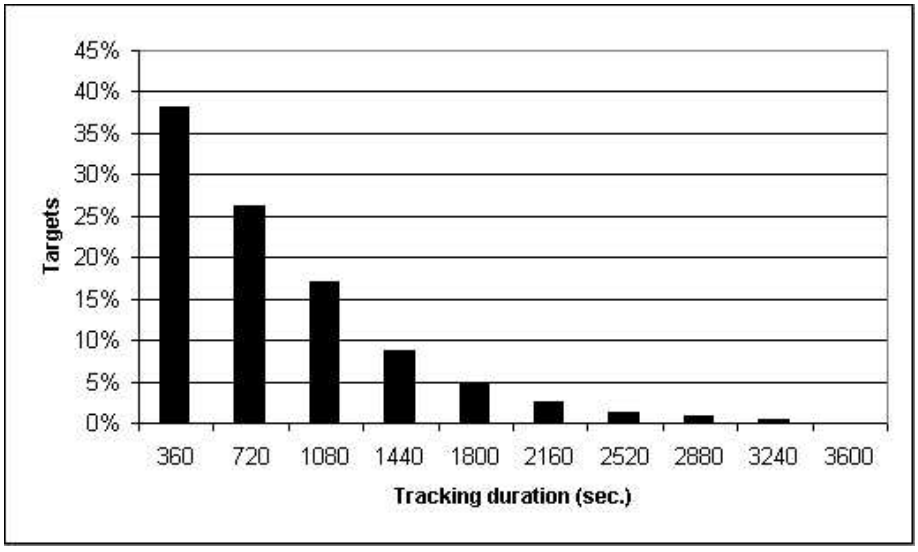

Figure 9.15. Tracking duration distribution (Sec.)

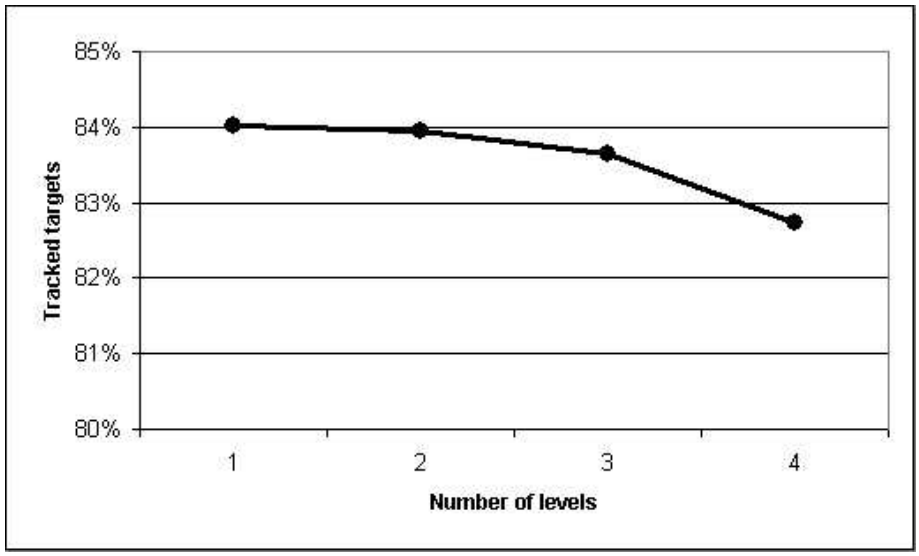

Figure 9.16. Accurate tracked target percentage as a function of the number of levels

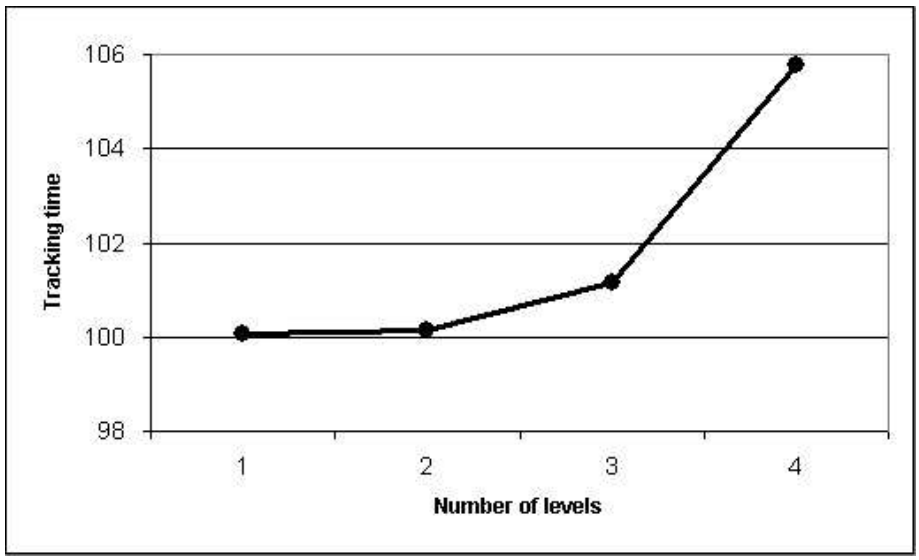

Figure 9.17. Accurate tracking time (Sec.) as a function of the number of levels

less targets as the number of levels increased. This can be explained by a greater fragmentation of the zone, i.e. 4 quarters in 2 levels versus 64 in 4 levels. The fi gure shows that the decrement is narrow.

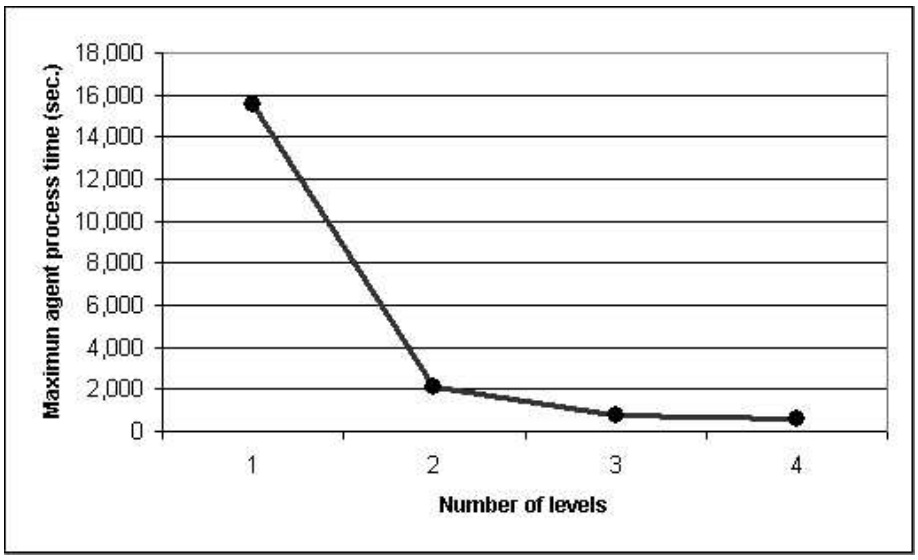

Figure 9.18. Maximum agent process time (Sec.) as a function of the number of levels

As shown in Figure 9.17, the average time to track a target increases as the number of levels increases. However, it increases only from 100 seconds to 106 seconds while the number of levels increases from 1 to 4.

Figure 9.18 presents the duration it took an agent to perform its task. In this figure we present the maximum time when using the computer capabilities as detailed above. The maximum time is very close to the average time; therefore we do not present the latter here. As we predicted, while using only one level the agent will need more time than it has. In our case an agent needed 16,000 seconds (about 5 hours) to process data gathered during 1 hour. That means that in the case of one level the system will not converge. Using 2 levels enabled the system to solve the problem in only 35 minutes. Using 4 levels decreased the maximum time that an agent needed to process data collected in an hour to only 10 minutes.

In Figure 9.19 we show the total number of bytes transferred between agents during one hour, relative to the number of levels in the hierarchy. The capsules generated by samplers and sent to sampler group leaders resulted in a transfer of 4Mb. Having a massive communication load may cause a bottleneck in the receiving agent that may lead to delays. Moreover, such a bottleneck may result in a loss of important information in case of agents' faults. When adding more levels to the hierarchy, more agents transfer information upwards and therefore the total number of bytes transferred is increased. On the other hand, adding more levels decreases the average number of bytes every agent receives. In figure 9.20 we can see the significance of the reduction of the av-

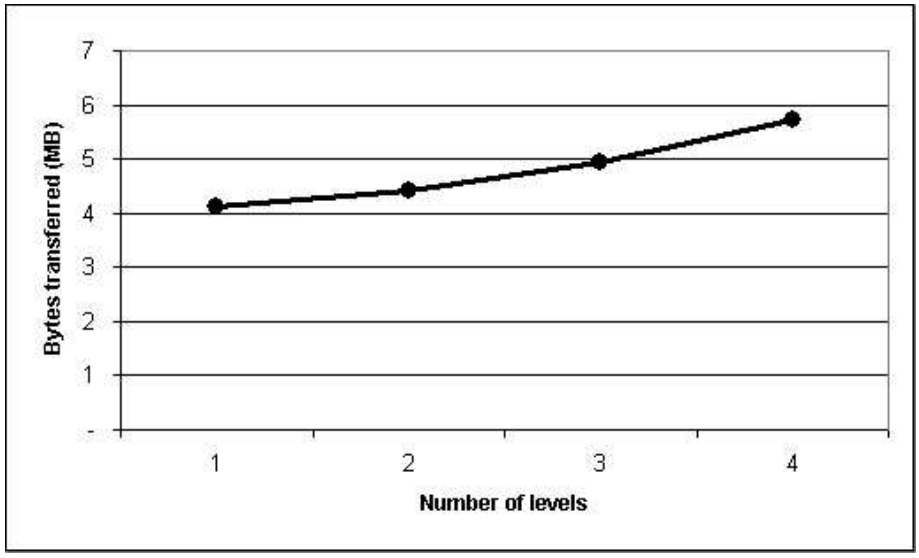

Figure 9.19. Bytes transferred as a function of number of levels.

erage communication load on the receiving agent when increasing the number of the levels.

We used a hierarchy formation such that every level has four times more agents then its leader's level. Therefore, the total number of leader agents receiving information in the hierarchy is 1, 5, 21 and 85 when using hierarchy of 1,2,3 and 4 levels. Figure 9.20 show the number of bytes transferred according to the number of agents.

Dysfunctional sampling agents. To investigate the fault tolerance property of the hierarchy model in a large-scale environment we disabled some of the sampling agents. We increased the number of disabled sampling agents from 0% as in the basic settings to 90%, leaving only 10% active agents. We hypothesized that by increasing the number of faulty sampling agents the system would not perform as well as in the basic settings. Our goal was to place a bound on the number of dysfunctional sampling agents that the system could tolerate while still performing well.

Figure 9.21 shows the accurate tracked targets percentage as a function of the number of samplers which stopped functioning. We found that increasing the number of disabled sampling agents also increases the time it takes to track targets. By increasing the number of disabled sampling agents by 5% the average time it takes to track a target increased by 6%.

Dysfunctional leader agents. A second aspect of the system's fault tolerance is its response to dysfunctional leaders. In contrast to dysfunctional samplers, a dysfunctional leader will result in a difference in the coverage of

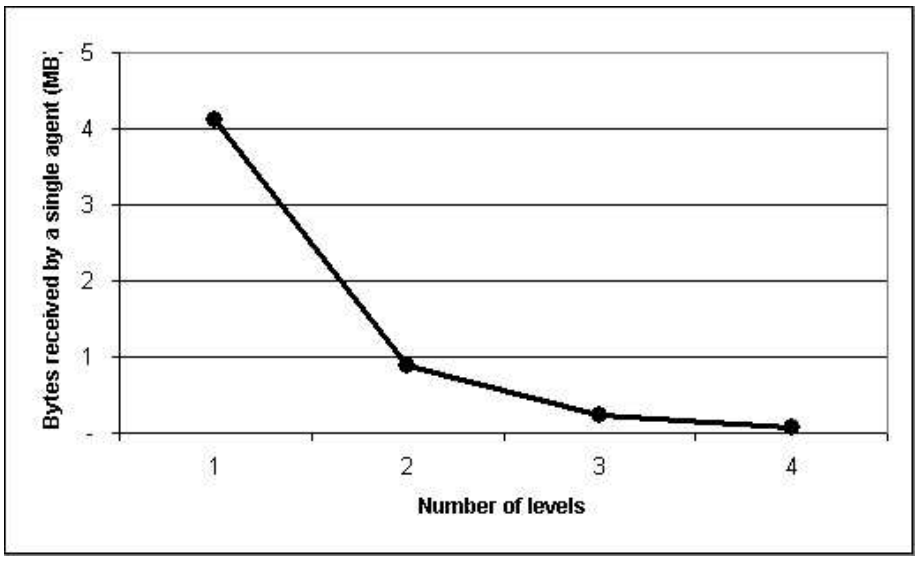

Figure 9.20. Average number of bytes received by a single agent as a function of the number of levels.

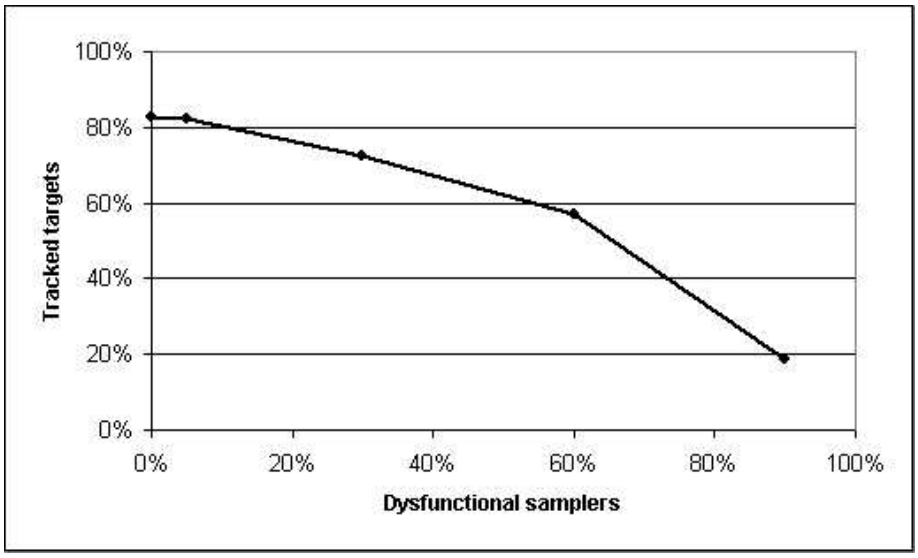

Figure 9.21. Accurate tracked target percentage as a function of dysfunctional samplers

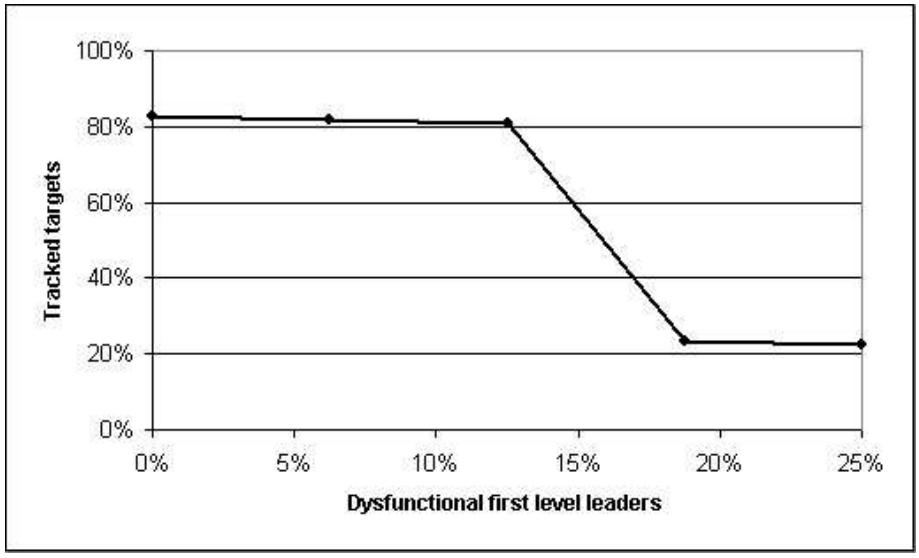

Figure 9.22. Accurate tracked target percentage as a function of dysfunctional first level leaders

the system. For example, consider a case in which a leader responsible for half of the controlled area stops functioning. Using the patrol Doppler movement pattern will result in a loss of information from half of the samplers. We hypothesized that performance would be significantly influenced by this factor. To validate this hypothesis we conducted several simulations in which we varied the number of dysfunctional sampler leaders.

Figure 9.22 confirms our hypothesis. It shows that the system could tolerate a reduction of up to 13% in the number of functioning sampler leaders. A reduction of 18% or more resulted in a very low performance level. However, despite the fact that the system demonstrated poor tracking percentage for high-rate dysfunctional sampler leaders, we discovered that it still tracked targets quickly. We hypothesize that adopting a reactive approach that will enforce division of the area among the active agents will overcome this problem. We plan to report the results of our investigation of this hypothesis in a future document.

Noisy communication. As we stated, we would like to show that using simple and cheap sensors may be beneficial even if they tend to malfunction or if communication with their leaders degrades. We conducted a thoughtful simulation testing the system while using faulty communication between samplers and leaders. We predicted that the system would be tolerant towards such noise.

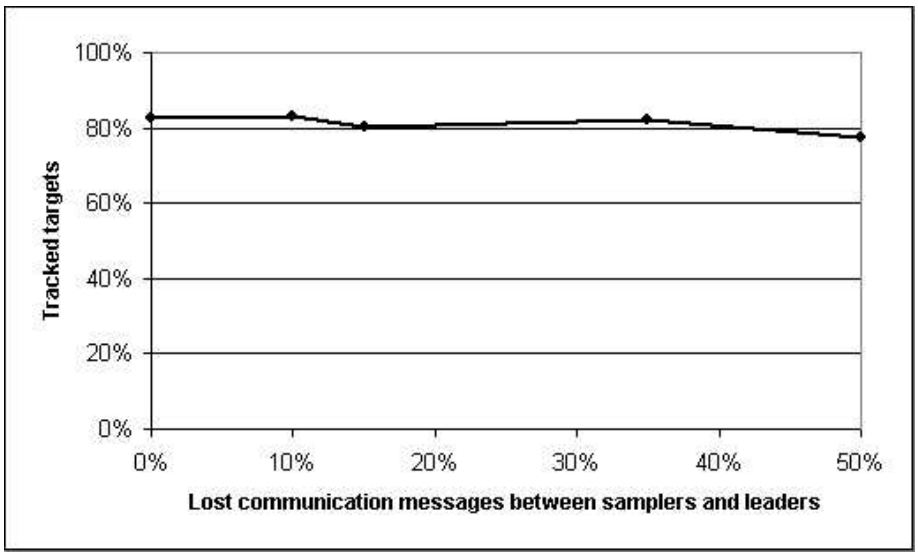

Figure 9.23. Accurate tracked target percentage of patrol as a function of lost communication messages between samplers and leaders

We found, as shown in Figure 9.23, that even if 50% of the messages did not reach their destination (either because of faulty communication or faulty samplers), the system still performed well. Losing 50% of the messages resulted in a reduction of only 5% of the tracked targets and increased the tracking time by 20 seconds.

4. Related work

The benefi ts of hierarchical organizations have been argued by many. So and Durfee draw on contingency theory to examine the benefi ts of a variety of hierarchical organizations; they describe a hierarchically organized network monitoring system for task decomposition and they also consider organizational self-design [So 1996]. 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 testbeds (DVMT) [Lesser 1987]. DVMT also operated in a domain of distributed sensors that tracked objects. However, the algorithms for support of mobile sensors and for the actual specifi cs 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* [Corkill 1983] which partitioned

problem solving nodes according to roles and communication, but they were not initially hierarchically organized [Ishida 1992; Scott 1992]. Wagner and Lesser examined the role that knowledge of organizational structure can play in control decisions [Wagner 2000].

All of the other approaches discussed in this volume assume that agents are stationary. Those approaches make use of measurements from three Doppler sensors, taken at the same time, and intersect the arcs corresponding to each sensor. The intersection method depends on the coordinated action of three Doppler sensors to simultaneously sample the target. Such coordination requires good synchronization of the clocks of the sensors and therefore communication among the Doppler agents to gain that synchronization. In addition, communication is required for scheduling agent measurements. We have descried an alternative that can make use of uncertain measurements; we focus on the combination of partial and local information. Note, that even though our agents associate a time stamp with each capsules, DDM does not require that the sensors are fully time synchronized. The ResBy relation may allow small deviation of the time. For example: $ResBy\langle\langle t1, s1\rangle, \langle t2, s2\rangle\rangle$ may be $s1.v == s2.v \& (s1.x - t1 * s1.v) - (s2.x - t2 * s2.v) \leq \epsilon$ Using a large ϵ may indicate high tolerance towards non synchronized clocks. However, increasing the value of epsilon increases the probability to identify two different targets as the same one.

In their work, Yu and Cysneiros [Yu 2002] describe challenges related to large-scale information systems. They claim that large-scale systems have the potential to support greater diversity, offering more flexibility and better robustness as well as more powerful functionalities compared to traditional software technologies. In our work we address these challenges and propose an effi cient solution.

Silva *et al*, have developed the Reflective Blackboard architectural pattern for large-scale systems [Silva 2002]. This is the result of the composition of two other well-known architectural patterns: the Blackboard pattern and the Reflection pattern. They separate control strategies from the logic and data. In our work we use independent agents that act autonomously. Such a loose coupling is benefi cial in terms of simplicity, robustness and fault tolerance.

Tel has studied the performance of a network tree with n processors providing communication between every pair of processors with a minimal number of links (n-1) [Tel 1991]. The communication complexity in a tree topology is influenced by the diameter of the number of levels in the tree. Therefore a tree with fewer levels will have a better communication complexity. However each node has more computations to perform and can therefore become a bottleneck. A failure of a node will split the tree into a larger number of unconnected subsets. In the work we have described, we have investigated the relation between the number of levels in a hierarchical structure and perfor-

mance; we have presented suggestions of how to choose the right number of levels.

5. Conclusions

We have introduced a solution to a modified large-scale ANTs problem. We have shown that problems involving hundreds and thousands of Dopplers and targets cannot be solved in the traditional flat architecture. Distributing the solution into smaller problems that can be solved partially by simple agents is the approach we adopted. Using many simple and cheap agents instead of a much smaller number of sophisticated and expensive ones may also be cost-effective: it is often more affordable to replace and maintain many simple agents than to depend on a few sophisticated ones. We also suggested ways to combine partial solutions to form a global solution. We established an autonomous movement algorithm to be implemented by each sampling agent. We have also shown that the capabilities of the hierarchical model are greater than those of the flat one. In particular, we found that the flat model could not solve the problem we addressed.

We have shown that the number of levels in a hierarchy influences the accuracy of results. As the number of levels increases the number of tracked targets drops, even though this drop is moderate. However, as the number of levels increased, the time every agent needed to complete its mission dropped exponentially. By combining these two results we are able to balance these two properties. Choosing the right number of levels should also take into consideration the time it takes to track targets. As we have shown, it takes more time to track targets as the number of levels in the hierarchy is increased.

To conclude, we have shown that the large-scale ANTs system can perform well even if agents are very simple and inaccurate. We have shown how partial information can be combined and how the existence of dysfunctional participants can be overcome.

6. Acknowledgments

This work was supported by the DARPA Autonomous Negotiating Teams Program under contract F30602-99-C-0169 and NSF grant number IIS9907482. We would like to thank the reviewers for comments on earlier drafts of this paper as well as the many individuals from the ANTS program who we interacted with while conducting this research.

References

D. Corkill and V. Lesser “The use of meta-level control for coordination in a distributed problem solving network,” in proceedings of the International Joint Conference on Artificial Intelligence, 1983, pages 748–756.

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

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

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

C. L. Ortiz, E. Hsu, M. desJardins, T. Rauenbusch, B. Grosz, O. Yadgar, and S. Kraus. "Incremental negotiation and coalition formation for resource-bounded agents," in Proceedings of the AAAI Fall Symposium, 2001.

Pöss, Christian Doppler in Banska Stiavnica, in *The Phenomenon of Doppler*, Prague, 1992.

W. Richard Scott, *Organizations: Rational, Natural and Open*, Prentice-Hall, 1992.

Silva, O.; Garcia, A; Lucena, C. J. "The Reflective Blackboard Architectural Pattern for Developing Large Scale Multi-Agent Systems". To appear in the Proceedings of the 1st International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2002) at ICSE 2002, Orlando, USA, May 2002.

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

Y. So and E. H. Durfee, "A Distributed Problem-Solving Infrastructure for Computer Network Management." *International Journal of Intelligent and Cooperative Information Systems*, 1(2):363-392,1992.

G. Tel, *Topics in distributed algorithms*. Cambridge University press, pp. 27-31, 1991.

T. Wagner and V. Lesser. "Relating Quantifi ed Motivations for Organizationally Situated Agents." in Proceedings of ATAL 2000.

O. Yadgar, S. Kraus and C. L. Ortiz, "Hierarchical organizations for real-time large-scale task and team environments", in Proceedings of AAMAS, 2002.

O. Yadgar, S. Kraus and C. L. Ortiz, "Information Integration Approach for Large-Scale Multiagent R.M.", in *Communication in Multiagent Systems* (to appear), 2003.

E. Yu and L. M. Cysneiros, "Large-Scale Agent Systems: A World Modeling Perspective," in SELMAS02, 2002.