Correspondence

Multiresolution Data Integration Using Mobile Agents in Distributed Sensor Networks

Hairong Qi, S. Sitharama Iyengar, and Krishnendu Chakrabarty

***Abstract*—We describe the use of the mobile agent paradigm to design an improved infrastructure for data integration in distributed sensor network (DSN). We use the acronym MADSN to denote the proposed mobile-agent-based DSN. Instead of moving data to processing elements for data integration, as is typical of a client/server paradigm, MADSN moves the processing code to the data locations. This saves network bandwidth and provides an effective means for overcoming network latency, since large data transfers are avoided. Our major contributions are the use of mobile agent in DSN for distributed data integration and the evaluation of performance between DSN and MADSN approaches. We develop an enhanced multiresolution integration (MRI) algorithm where multiresolution analysis is applied at local node before accumulating the overlap function by mobile agent. Compared to the MRI implementation in DSN, the enhanced integration algorithm saves up to 90% of the data transfer time. We develop objective functions to evaluation the performance between DSN and MADSN approaches. For a given set of network parameters, we analyze the conditions under which MADSN performs better than DSN and determine the condition under which MADSN reaches its optimum performance level.**

***Index Terms*—Distributed sensor network (DSN), mobile agent, multiresolution integration (MRI).**

I. Introduction

Distributed sensor networks (DSNs) have recently emerged as an important research area [1]–[5]. This development has been spurred by advances in sensor technology and computer networking. Even though it is economically feasible to implement DSNs, there are several technical challenges that must be overcome before DSNs can be used for today’s increasingly complex information gathering tasks. These tasks, such as battlefield surveillance, remote sensing, global awareness, etc., are usually time-critical, cover a large geographical area, and require reliable delivery of accurate information for their completion.

Wesson *et al.* [5] were among the first to propose the design of DSNs. Since then, several efficient DSN architectures have been presented in the literature, including the deBruijn based network [1], the flat tree network [2], [4], the multi-agent fusion network [3], and the hierarchical and committee organization [5]. While improving the performance of DSNs in different aspects, all these approaches use a common network computing model: the client/server model, which supports many distributed systems, such as remote procedure calling (RPC) [6], common object request broker architecture (CORBA) [7], [8], etc. In client/server model, the client (individual sensor) sends data to the server (processing element) where data processing tasks are carried out. However, the client/server model is not appropriate for data integration in DSNs. First, the data integration at the server requires data transfer from local sensor nodes. When the size of data file is large and the number of sensor node is big, the network traffic can be extremely heavy, resulting in poor performance of the system. Second, suppose connection-oriented service is used (e.g., ftp application uses protocol), the client/server model requires the network connection to be alive and healthy the entire time a data transfer is taking place. If the connection goes down, both the client and the server have to wait until the connection is recovered to finish the data transfer and do further analysis, which will affect the system performance as well. Third, the client/server-based DSN cannot respond to the load changing in real time. When more sensors are deployed, it cannot perform load balancing without changing the structure of the network.

Recent advances in sensor technology allow better, cheaper, and smaller sensors to be used in both military and civilian applications, especially when the environment is harsh, unreliable, or even adversarial. A large number of sensors are usually deployed in order to achieve quality through quantity. On the other hand, sensors typically communicate through wireless networks where the network bandwidth is much lower than for wired communication. These issues bring new challenges to the design of DSNs. First, data volumes being integrated are much larger. Second, the communication bandwidth for wireless network is much lower. Third, the environment is more unreliable, causing unreliable network connection, noisy background, and increasing the likelihood of input data to be in faulty.

In this paper, we design an improved DSN architecture using mobile agents—we refer to this as mobile-agent-based DSN (MADSN). In traditional DSNs, data are collected by individual sensors, and then transmitted to a higher-level processing element which performs sensor fusion. During this process, large amounts of data are moved around the network, as is the typical scenario in the client/server paradigm. MADSN adopts a new computation paradigm: data stay at the local site, while the integration process (code) is moved to the data sites. By transmitting the computation engine instead of data, MADSN offers the following important benefits.

- Network bandwidth requirement is reduced. Instead of passing large amounts of raw data over the network through several round trips, only the agent with small size is sent. This is especially important for real-time applications and where the communication is through low-bandwidth wireless connections.
- Better network scalability. The performance of the network is not affected when the number of sensor is increased. Agent architectures that support adaptive network load balancing could do much of a redesign automatically [9].
- Extensibility. Mobile agents can be programmed to carry task-adaptive fusion processes which extends the capability of the system.
- Stability. Mobile agents can be sent when the network connection is alive and return results when the connection is re-established. Therefore, the performance of MADSN is not much affected by the reliability of the network.

Fig. 1 provides a comparison between DSN and MADSN from both feature and architectural points of view.

The organization of this paper is as follows. Section II discusses the definition of mobile agents and application examples that benefit from

Manuscript received October 16, 2000; revised July 16, 2001. This work was supported in part by DARPA under Grant N660010018946 and in part by the Office of Naval Research under Grant N000140110712. This paper was recommended by Associate Editor R. Popp.

H. Qi is with the Department of Electrical and Computer Engineering, University of Tennessee, Knoxville, TN 37996 USA (e-mail: hqi@utk.edu).

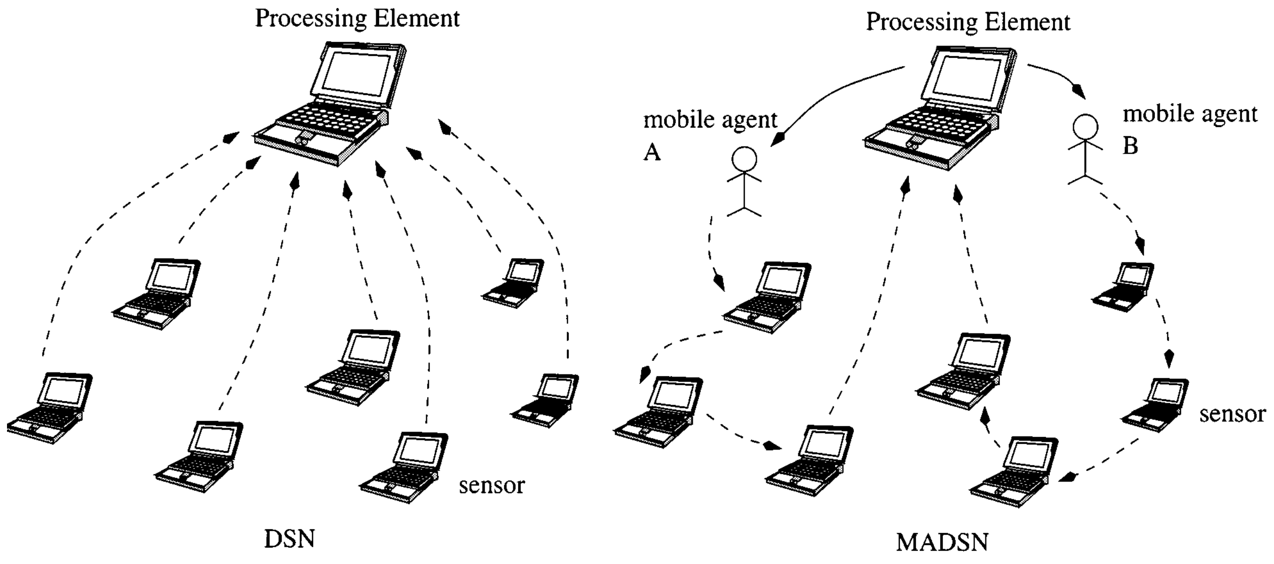
S. S. Iyengar is with the Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803 USA (e-mail: iyengar@bit.csc.lsu.edu).

K. Chakrabarty is with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (e-mail: krish@ee.duke.edu).

Publisher Item Identifier S 1094-6977(01)08913-1.

Features	DSN	MADSN
Element moving on the network	Data	Computation
Bandwidth consumption	High	Low
Scalable?	No	Yes
Extensible?	No	Yes
Affected by network reliability?	Yes	No
Fault-tolerable?	Yes	Yes

(a)



(b)

Fig. 1. Comparison between DSN and MADSN. (a) Feature. (b) Architecture.

using mobile agents. It also defines the two problems studied in the design of MADSN. Section III first reviews the multiresolution data integration algorithm implemented in traditional DSN, then describes its implementation using mobile agents. A case study is provided. Section IV compares the performance of DSN and MADSN. For a given set of parameters, it derives the condition under which MADSN performs better than DSN, also the condition under which MADSN reaches its optimum performance level. Section V presents the conclusions.

II. Background and Problem Statement

This section reviews the basic DSN architecture and the key characteristics of mobile agents. The problems studied in this paper are formally defined at the end of the section.

A general DSN (Fig. 2) consists of a set of *sensor nodes,* a set of *processing elements* (PEs), and a *communication network* interconnecting the various PE's [1]. One or more sensors is associated with one PE. One sensor can report to more than one PE. A PE and its associated sensors are referred to as a *cluster*. Data are transferred from sensors to their associated PE(s) where the data integration takes place. PEs can also coordinate with each other to achieve a better estimation of the environment and report to higher level PEs. Notice that only the lowest-level PEs are connected to the sensor nodes. Higher-level PEs only connect to lower-level PEs, but not the sensor nodes. In the context of this paper, we assume that the sensor field is a two-dimensional (2-D) surface and the sensor nodes are fixed.

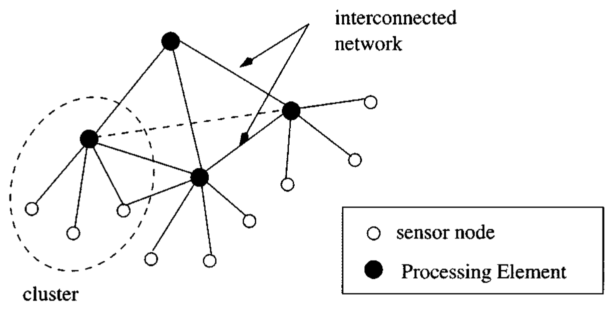


Fig. 2. Architecture of a general DSN.

A. Mobile Agents

Generally speaking, mobile agent is a special kind of software which can execute autonomously. Once dispatched, it can migrate from node to node performing data processing autonomously, while software can typically only execute when being called upon by other routines. Franklin and Graesser provided a formal definition of agent in [10].

Definition 1: An **autonomous agent** is a system situated within a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.

A good example to describe the difference between an agent and an ordinary computer program is also given in [10]. A payroll program in a real world environment is not an agent because its output would not normally affect what it senses later. A payroll program also fails the "over time" test of temporal continuity. It runs once and then goes into a coma, waiting to be called again. A mobile agent is an agent with the ability to migrate. Note that Java "applets" as used by web browsers are not mobile agents because they do not satisfy the autonomous and migration criteria. Java applets only run on the local machine once being requested and downloaded.

Lange listed seven good reasons to use mobile agents [11], including reducing network load, overcoming network latency, robust and fault-tolerant performance, etc. Although the role of mobile agents in distributed computing is still being debated mainly because of the security concern [12], [13], several applications have shown clear evidence of benefiting from the use of mobile agents. For example, mobile agents are used in networked electronic trading [14] where they are dispatched by the buyer to the various suppliers to negotiate orders and deliveries, and then return to the buyer with their best deals for approval. Instead of having the buyer contact the suppliers, the mobile agents behave like representatives, interacting with other representatives on the buyer's behalf, and alert the buyer when something happens in the network that is important to the buyer. Another successful example of using mobile agents is distributed information retrieval and information dissemination [15]–[18]. Agents are dispatched to heterogeneous and geographically distributed databases to retrieve information and return the query results to the end-users. Mobile agents are also used to realize network awareness [19] and global awareness [20]. Network-robust applications are of great interest in military situations today. Mobile agents are used to be aware of and reactive to the continuously changing network conditions to guarantee successful performance of the application tasks.

In this paper, we use mobile agent in distributed sensor networks to perform multiresolution data integration. Problems to be studied are defined in the following section.

B. Problem Statement

We define the mobile agent as an entity of the following five attributes.

1) Identification.
2) Itinerary.
3) Data.
4) Method.
5) Interface.

These attributes are defined as follows.

- *Identification* is in the format of 2-tuple $(i,\ j)$, where i indicates the identification number of its dispatcher and j the serial number assigned by its dispatcher. Each mobile agent can be uniquely identified by its identification. We use $MA_{i,\,j}$ to indicate different mobile agents.
- *Itinerary* includes information about migration route assigned by processing element before dispatched.
- *Data* is an agent's private data buffer which mainly carries integration results.
- *Method* is the implementation of algorithms. In MADSN, the key method is the multiresolution data integration algorithm.
- *Interface* provides interface functions for agent and processing element to communicate with each other, and for processing element to access agent's private data buffer.

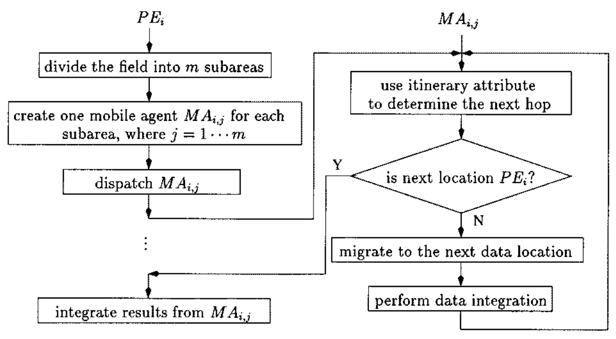

PE_i
divide the field into m subareas
create one mobile agent $MA_{i,j}$ for each subarea, where $j = 1 \cdots m$
dispatch $MA_{i,j}$
integrate results from $MA_{i,j}$
$MA_{i,j}$
use itinerary attribute to determine the next hop
is next location PE_i?
Y
N
migrate to the next data location
perform data integration

Fig. 3. Flowchart for agent creation, dispatch, and migration with single processing element.

Let PE_i represent a certain processing element that is in charge of the surveillance of a certain area. Let $\{MA_{i,\,1},\ \cdots,\ MA_{i,\,m}\}$ represent a group of m mobile agents dispatched by PE_i. Without loss of generality, we assume that each $MA_{i,\,j}$ $(j = 1,\ \cdots,\ m)$ visits the same number of sensor nodes, denoted by n. The problems studied in this paper are formally defined as follows:

Data Integration Problem: At each sensor site, what kind of data processing should be conducted and what integration results should be carried with the mobile agent?

Optimum Performance Problem: How to balance the value of m and n, such that the performance of MADSN is superior to DSN?

Fig. 3 outlines the life cycle of a mobile agent $MA_{i,\,j}$ and its relationship with its dispatcher PE_i. Details are explained in following sections.

III. Multiresolution Integration Algorithm

As mentioned in Section I, MADSN must respond to the challenges of a larger amount of sensor nodes and higher probability of faulty sensor readouts due to both environmental noise and physical damage. More sensor nodes can increase the computation load, while more faulty sensors can cause the integration results to be more unreliable. Algorithms are therefore sought which should not be significantly affected by network scaling, and yet provide better performance and higher fault tolerance. This section first reviews the original multiresolution integration (MRI) algorithm proposed for DSNs [21]. Enhancements to the basic MRI algorithm are then described in order to take advantage of mobile agents to achieve better network scalability and fault tolerance. The enhancements involve a multiresolution analysis of individual sensor readout to generate a simple function (the overlap function) at the sensor site, followed by an integration of the simple functions at the processing element. Compared to the MRI implementation in DSNs, where the integration of individual sensor readout (carried out at the processing element) is followed by the multiresolution analysis of the integrated simple function, the mobile agent implementation of MRI algorithm reduces the data transfer time by as much as 90%.

A. Original MRI Algorithm in DSNs

The original MRI algorithm was proposed by Prasad, Iyengar, and Rao in 1994 [21]. The idea essentially consists of constructing a simple function (the overlap function) from the outputs of the sensors in a cluster and resolving this function at various successively finer scales of resolution to isolate the region over which the correct sensors lie. Each sensor in a cluster measures the same parameters. It is possible that some of them are faulty. Hence it is desirable to make use of this redundancy of the readings in the cluster to obtain a correct estimate

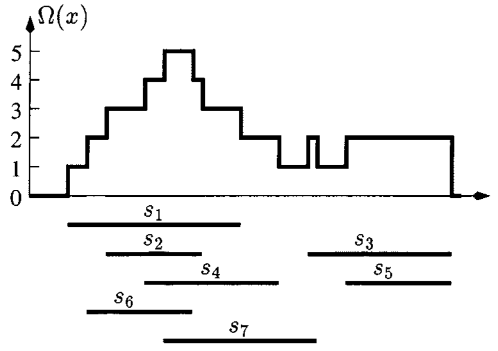


Fig. 4. Overlap function for a set of seven sensors.

of the parameters being observed. Before detailed discussion, we first review several relevant definitions.

Definition 2: An **abstract sensor** is defined as a sensor that reads a physical parameter and gives out an abstract interval estimate which is a bounded and connected subset of the real number of a certain dimension. We classify abstract sensors into two categories: **correct sensors** and **faulty sensors**.

Definition 3: A **correct sensor** is an abstract sensor whose interval estimate contains the actual value of the parameter being measured. Otherwise, it is a **faulty sensor**.

Definition 4: A faulty sensor is **tamely faulty** if it overlaps with a correct sensor, and is **wildly faulty** if it does not overlap with any correct sensors.

Definition 5: Let sensors $S_1, \ldots, S_N$ feed into a processor P. Let the abstract interval estimate of S_j be I_j $(1 \leq j \leq N)$, the closed interval $[a_j, b_j]$ with end points a_j and b_j. The characteristic function χ_j of the jth sensor S_j is defined as follows:

$$\chi_j(x) = \begin{cases} 1, & \text{if } a_j \leq x \leq b_j \\ 0, & \text{if } x > b_j \text{ or } x < a_j. \end{cases}$$

Definition 6: Let $\Omega(x) = \sum_{j=1}^{N} \chi_j(x)$ be the "overlap function" of the N abstract sensors. For each $x \in R$ (R is the set of the real number of 1-dimension), $\Omega(x)$ gives the number of sensor intervals in which x lies; that is, the number of intervals overlapping at x.

Definition 7: Crest is a region in the overlap function with the highest peak and the widest spread.

Fig. 4 illustrates the overlap function for a set of seven sensors calculated from their characteristic functions. We can observe several key characteristics from the profile which is common to all overlap functions.

- Tamely faulty sensors cluster around correct sensors and create high and wide (maximal) peaks in the profile of $\Omega(x)$.
- Wildly faulty sensors on the other hand do not overlap with correct sensors, and therefore contribute to smaller and narrower peaks.

Therefore, the actual value of the parameter being measured lies within regions over which the maximal peaks of $\Omega(x)$ occur with the widest spread.

1) Multiresolution Analysis of the Overlap Function: Multiresolution analysis provides a hierarchical framework for interpreting the overlap function. It is natural and more efficient to first analyze details at a coarse resolution and then increase the resolution for only the region of interest.

Given a sequence of increasing resolutions $(2^{-c}, 2^{-c+1}, \ldots, 2^0)$, where c is a positive integer, we define the difference of function $f(x)$ at resolution 2^{-c+1} and resolution 2^{-c} as the details of $f(x)$ at resolution 2^{-c+1}. The algorithm is described as follows.

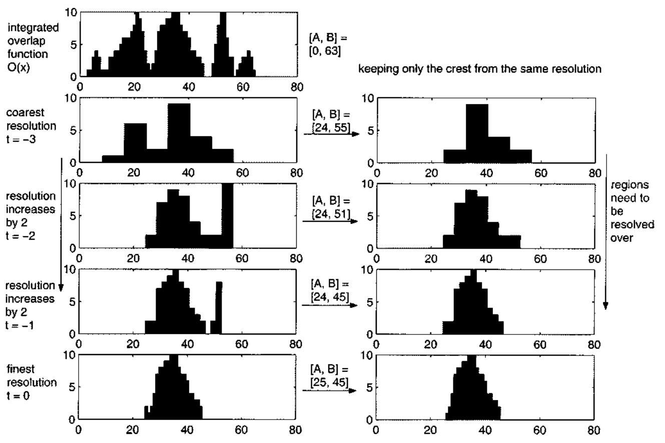


Fig. 5. Overlap function $\Omega(x)$ and its appearance at different resolutions. (The shaded region indicates the region needs to be resolved over.)

```
Algorithm 1: Multiresolution analysis of the overlap function in DSN.
  Data $\Omega(x)$, $2^k$, $(-c \leq k \leq 0)$, assuming the coarsest resolution is
  :  $2^{-c}$, the highest resolution is $2^0$, $[A, B]$ is the interval of the
  :  overlap function $\Omega(x)$
  Result the final crest $[\gamma_l, \gamma_h]$ under resolution $2^k$, where $\gamma_l$ and $\gamma_h$
  :  are the lower and higher bounds of the crest respectively
  $t = -c$;
    While $t <= k$ do
    resolve $\Omega(x)$ at resolution $2^t$ by sampling it over the interval $[A, B]$ at
    points $n2^{-t}$, $(\lfloor A/2^{-t} \rfloor \leq n \leq \lfloor B/2^{-t} \rfloor)$, to obtain $\Omega_t(x)$;
    select the highest peaks from $\Omega_t(x)$;
    choose from these peaks the one with the widest spread $[A_t, B_t]$, which
    is a crest;
    $\Omega(x) = \Omega_t([A_t, B_t])$;
    $A = A_t$, $B = B_t$;
    $t = t + 1$;
    end

  $\gamma_l = A$, $\gamma_h = B$;
```

This procedure results in the isolation of those regions over which the overlap function $\Omega(x)$ has a maximum value, corresponding to high degree of overlapping of individual sensor readouts. The algorithm is optimal, since the overall time required is $O(n \log n)$, which is the time required to maintain $\Omega(x)$. This algorithm is also robust, satisfies the Lipschitz condition [22], which ensures that minor changes in the input intervals cause only minor changes in the integrated result. Fig. 5 illustrates the multiresolution analysis procedure.

B. MRI Implementation Using Mobile Agents

In a DSN, all readouts from the sensor nodes are sent to their corresponding processing elements, where the overlap function at the *finest* resolution is first generated, and the multiresolution analysis procedure is then applied to find the crest at the *desired* resolution.

In a MADSN, the mobile agents migrate among the sensor nodes and collect readouts. Therefore, $MA_{i,j}$ always carries a *partially* integrated overlap function which is accumulated into a final version at PE_i after all the mobile agents return. During this process, if MADSN applies the multiresolution analysis method in the same way as DSN does, that is, letting $MA_{i,j}$ carry the partially integrated overlap function in its finest resolution and then use multiresolution analysis (MRA) to find the crest at desired resolution at PE_i, the advantages of mobile agents will be nullified because of heavy data migration.

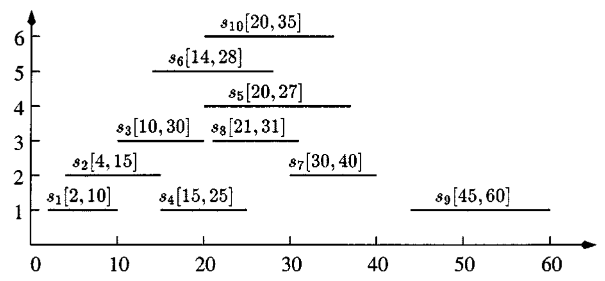


Fig. 6. Readouts from ten sensor nodes at time t.

TABLE I
TRACING THE CHANGE OF $\omega_{i,1}$ GENERATED BY $MA_{i,1}$

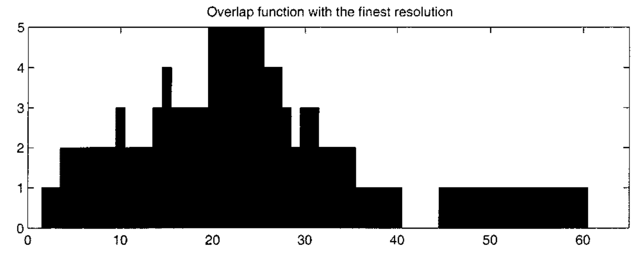
$s_j[a,b]$	d_{min}	d_{max}	$\frac{d_{min}}{2^{-k}}$	$\frac{d_{max}}{2^{-k}}$	$\omega_{i,1}$
$s_1[2,10]$	8	8	1	1	[0,1,0,0,0,0,0,0]
$s_2[4,15]$	8	8	1	1	[0,2,0,0,0,0,0,0]
$s_3[10,20]$	16	16	2	2	[0,2,1,0,0,0,0,0]
$s_4[15,25]$	16	24	2	3	[0,2,2,1,0,0,0,0]
$s_5[20,27]$	24	24	3	3	[0,2,2,2,0,0,0,0]

TABLE II
TRACING THE CHANGE OF $\omega_{i,2}$ GENERATED BY $MA_{i,2}$

$s_j[a,b]$	d_{min}	d_{max}	$\frac{d_{min}}{2^{-k}}$	$\frac{d_{max}}{2^{-k}}$	$\omega_{i,2}$
$s_6[14,28]$	16	24	2	3	[0,0,1,1,0,0,0,0]
$s_7[30,40]$	32	40	4	5	[0,0,1,1,1,1,0,0]
$s_8[21,31]$	24	24	3	3	[0,0,1,2,1,1,0,0]
$s_9[45,60]$	48	56	6	7	[0,0,1,2,1,1,1,1]
$s_{10}[20,35]$	24	32	3	4	[0,0,1,3,2,1,1,1]

We enhance the basic MRI algorithm for MADSNs and present a more efficient implementation. The key concept underlying the enhanced algorithm is that MRI is applied *before* accumulating the overlap function. A one-dimensional (1-D) array, $\omega_{i,j}$, can serve as an appropriate data structure to represent the partially-integrated overlap function carried by $MA_{i,j}$. If the size of $\omega_{i,j}$ is s_{20} at resolution 2^0, then at resolution 2^k (2^{-k} times coarser than 2^0), the size of $\omega_{i,j}$ is $s_{20}/2^{-k}$, that is, 2^{-k} times less than s_{20}. The following algorithms describe the procedure in detail.

Let PE_i be the processing element of interest, m the number of mobile agents dispatched, $MA_{i,j}$ the mobile agent dispatched by PE_i ($1 \leq j \leq m$), and $[A_i, B_i]$ the interval that covers readouts from all the sensors migrated by $MA_{i,j}$. Algorithm 2 creates the 1-D array $\omega_{i,j}$ based on the desired resolution. Algorithm 3 accumulates the sensor readouts to $\omega_{i,j}$ and forms the partially integrated overlap function at the desired resolution. Algorithm 4 integrates the partial overlap functions from all $MA_{i,j}$ dispatched by PE_i. The final integration is carried out at the processing element. A case study is provided for better illustration.



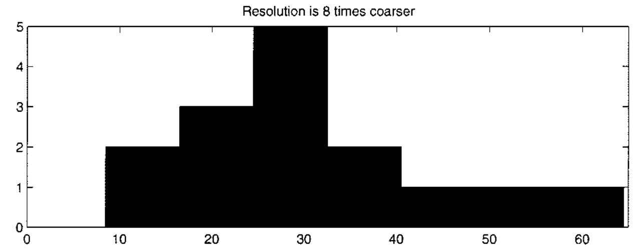


Fig. 7. Overlap function at its finest resolution and the version with 8 times coarser resolution obtained by modified MRI using mobile agent implementation.

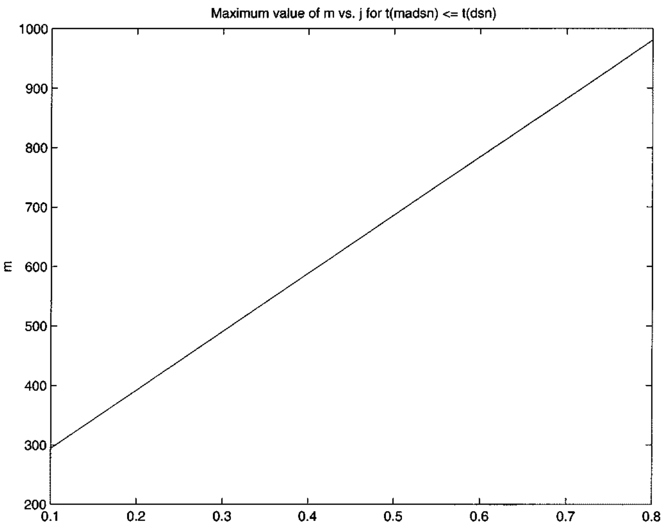


Fig. 8. Performance evaluation between DSN and MADSN: m vs. j.

Algorithm 2: Modified MRI algorithm for MADSN—before $MA_{i,j}$ leaves PE_i

```
Data integration interval $[A_i, B_i]$, highest resolution $2^0$, desired
    resolution $2^k$ $(-c \leq : k \leq 0)$
Result array $\omega_{i,j}$ to hold partially-integrated overlap function
$s = (B_i - A_i + 1)/2^{-k}$;
initialize $\omega_{i,j}$ as a zero vector with $s$ elements;
```

Algorithm 3: Modified MRI algorithm for MADSN—$MA_{i,j}$ at sensor node

```
Data $\omega_{i,j}$, $2^k$, readout interval from the abstract sensor $[a, b]$ (a bounded
connected: set of real numbers)
Result $\omega_{i,j}$
:
find the smallest multiple of $2^{-k}$, $d_{\min}$, such that $d_{\min} \geq a$;
find the largest multiple of $2^{-k}$, $d_{\max}$, such that $d_{\max} \leq b$;
 if $d_{\min} \leq d_{\max}$ then
 increase elements $\omega_{i,j}[d_{\min}/2^{-k} : d_{\max}/2^{-k}]$ by 1, ($d_{\min}/2^{-k}$ and
 $d_{\max}/2^{-k}$ are indices of the array)
 end
```

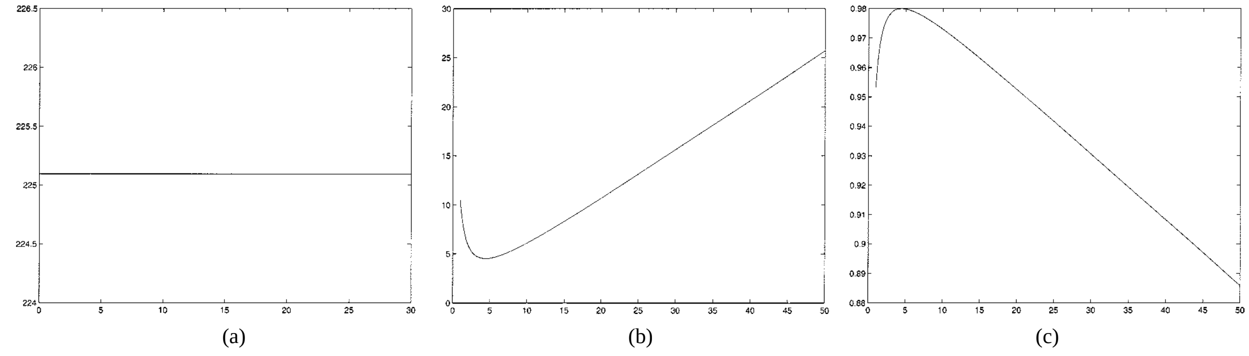


Fig. 9. Execution time for DSN and MADSN with respect to m with $p = 1000$, $v_n = 100$ Kbps, and $j = 0.25$.

```
Algorithm 4: Modified MRI algorithm for MADSN—$MA_{i,j}$ back to $PE_i$
  Data $\omega_{i,j}$, $2^k$, $m$ (total number of agents dispatched by $PE_i$), integration interval: $[A_i, B_i]$
  Result final crest $[\gamma_l, \gamma_h]$ at resolution $2^k$
  :
  create a zero vector $\psi_i$ with $(B_i - A_i + 1)$ elements;
  $j = 2$;
  while $j \le m$ do
    accumulate $\omega_{i,j}$ to $\omega_{i,1}$;
    $j = j + 1$;
  end
  $index = 0$;
  while $index < (B_i - A_i + 1)/2^{-k}$ do
    $\psi_i[index : index + 2^{-k} - 1] = \omega_{i,1}[index/2^{-k}]$;
    $index = index + 2^{-k}$;
  end
  select the highest peak of $\psi_i$. If there are more than one peak with the same height, then all the peaks should be selected;
  choose from these peaks the one with the widest spread $[\gamma_l, \gamma_h]$, which is a crest;
```

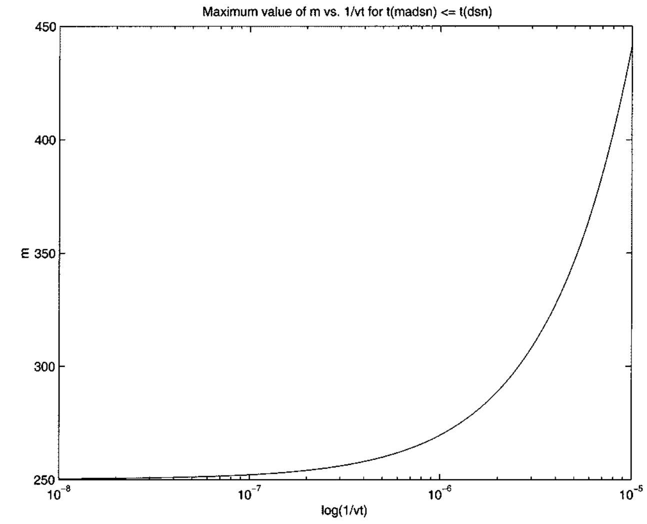


Fig. 10. Performance evaluation between DSN and MADSN: m vs. $1/v_n$.

1) Case Study: In this section, we present a case study to illustrate the MADSN-based MRI algorithm. Suppose PE_i has ten sensor nodes ($s_1, \ldots, s_{10}$), migrated by $m = 2$ mobile agents with $MA_{i,1}$ covering s_1 to s_5, and $MA_{i,2}$ covering s_6 to s_{10}. The readouts of sensors at time t are listed in Fig. 6. The integration interval $[A_i, B_i]$ is [0, 63]. The overlap function at its highest resolution then has 64 elements.

If the desired resolution is 2^{-3} (or eight times coarser than the finest resolution), according to Algorithm 2, an array $\omega_{i,j}$ with $8 = 64/8$ elements will be created and initialized by each mobile agent. Table I and II list the step-by-step execution result for each agent according to Algorithm 3.

According to Algorithm 4, $\omega_{i,1}$ and $\omega_{i,2}$ are summed up to generate [0, 2, 3, 5, 2, 1, 1, 1], which is then extended to ψ_i as

$$[\underbrace{0, \ldots, 0}_{8}, \underbrace{2, \ldots, 2}_{8}, \underbrace{3, \ldots, 3}_{8}, \ldots, \underbrace{1, \ldots, 1}_{8}, \underbrace{1, \ldots, 1}_{8}]$$

Compared to the results from DSN as shown in Fig. 7, they are exactly the same. If we define the unit data transfer time as the time spent for one $MA_{i,j}$ migrating from one node to another, carrying a one-element array, then MADSN spends $8 \times 5 + 8 \times 2 = 56$ units of time (assuming $MA_{i,1}$ and $MA_{i,2}$ are executed in parallel when migrating from node to node or from PE_i to node which costs 8×5 units of time, and in serial when returning to PE_i which costs 8×2 units of time), while DSN spends $64 \times 10 = 640$ units of time. Hence, MADSN offers a save of up to 91.25% of data transfer time in this case. Here, we assume the size of mobile agent is very small and thus can be ignored. Notice that in this case study, the performance gain is actually due to the parallel fusion carried out by mobile agents.

IV. Performance Comparison Between DSN and MADSN

The case study from Section III-B-1 shows that while obtaining the same integration results, MADSN saves 91.25% of data transfer time compared to DSN. However, this does not necessarily mean that MADSN is always better than DSN since MADSN also introduces overhead, such as the extra time spent for agent creation and dispatch. On the other hand, DSN needs to transfer data files to PE_i which also causes overhead due to file accesses. In this section, we analyze the relative performances of DSN and MADSN, and determine conditions under which an MADSN is more efficient than a DSN. These conditions are determined by a set of parameters, including the network transfer rate v_n, the data processing rate v_d, the data file size s_f, the mobile agent data buffer size s_a, overhead of agent o_a, overhead of file access o_f, the number of sensor nodes p, and the balance between the number of agents m and the number of sensor nodes n that each agent migrates (Notice that $p = m \times n$). Equations (1) and (2) are two formulas estimating the execution time for MADSN (t_{madsn}) and DSN (t_{dsn}). In both equations, the three components calculate the data transfer time, the overhead, and the data processing/integration time respectively

$$t_{madsn} = \frac{(m+n)s_a}{v_n} + mo_a + \frac{(m+n-1)s_a}{v_d} \tag{1}$$

$$t_{dsn} = \frac{mns_f}{v_n} + mno_f + \frac{(mn-1)s_f}{v_d}. \tag{2}$$

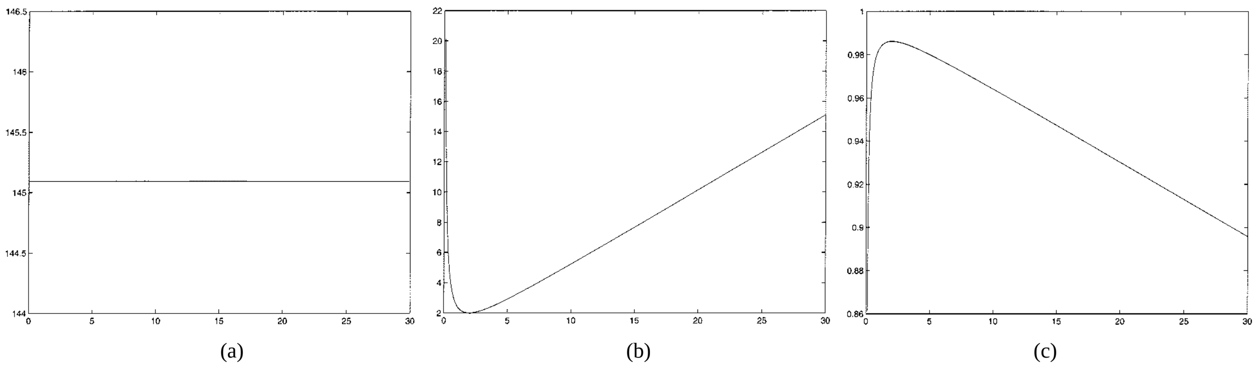


Fig. 11. Execution time for DSN and MADSN with respect to m with $j = 0.25$, $p = 1000$, and $v_n = 500$ Kbps.

We use m as the variable. Assume k and j are positive scalars, and $s_f = ks_a$, $o_f = jo_a$, $v'_n = 1/v_n$, $v'_d = 1/v_d$, in order to ensure that $t_{madsn} \le t_{dsn}$, (3) must be satisfied, that is, m must be chosen within a certain interval

$$\alpha m^2 - \beta m + \gamma \le 0 \tag{3}$$

where

$$\begin{aligned}\alpha &= s_a v'_n + o_a + s_a v'_d\\ \beta &= s_a v'_d - ks_a v'_d + kps_a v'_n + kps_a v'_d + jpo_a\\ \gamma &= ps_a v'_n + ps_a v'_d.\end{aligned}$$

In the following sections, we evaluate the performance variation of MADSN with respect to relationships between m and j, m and v'_n, and m and p. m is the number of nodes migrated by each mobile agent. j is the overhead ratio between DSN and MADSN. v'_n is the reciprocal of network transfer rate. p is the total number of sensor nodes. These parameters play a more important role than others.

A. Performance Evaluation: m vs. j

Suppose the size of agent is 1 KB, the overhead of agent is 0.5 s (including agent creation time), the network transfer rate is 100 Kbps, data processing rate is 100 Mbps, the number of sensor nodes is 1000, and the data size is 10 KB. Fig. 8 is a profile of the maximum value of m satisfying (3) when changing the overhead ratio between MADSN and DSN.

We then fix j at 0.25, that is, the overhead of file access is one fourth of the overhead of mobile agent, where the corresponding maximum m satisfying (3) is 441. By changing m from 1 to 441, we generate the performance curves for MADSN and DSN using the execution time: t_{madsn} and t_{dsn}.

Fig. 9(a) shows the variation of t_{dsn} with respect to the number of mobile agents m. It is a straight line since t_{dsn} is independent of the number of mobile agents and the total number of sensor nodes is a constant. Fig. 9(b) illustrates the variation of t_{madsn} with respect to m. The execution time t_{madsn} reaches its minimum when m is 4. Note that even though in the range of $m \in [1, 441]$, t_{madsn} is always less than t_{dsn}, after a decreasing segment at the very beginning, and reaching a minimum when $m = 4$, t_{madsn} starts to increase. This is because of the overhead from mobile agent: the more agents used, the heavier the overhead, the longer execution time needed; on the other hand, the less the agents, the lighter the overhead, but the longer the migration time. In order to investigate this further, we define the relative difference rate between t_{dsn} and t_{madsn} as $(t_{dsn} - t_{madsn})/t_{dsn}$. Fig. 9(c) shows that the relative difference rate is maximum when m is chosen to be 4.

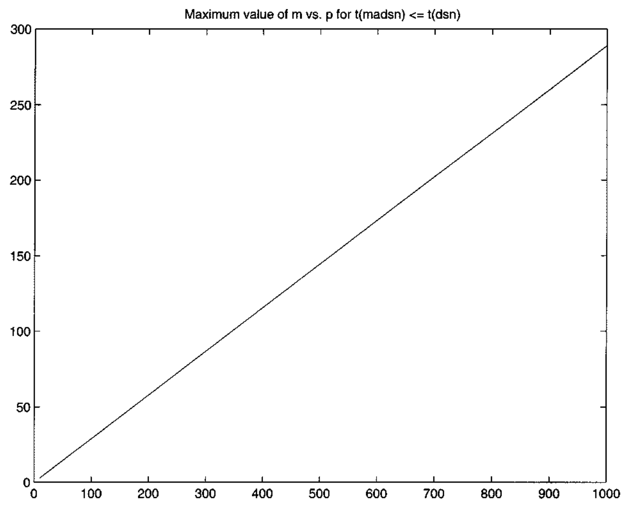


Fig. 12. Performance evaluation between DSN and MADSN: m versus p.

B. Performance Evaluation: m vs. v'_n

In this set of experiments, we fix j at 0.25, but vary the network transfer rate from 100 Kbps to 100 Mbps. Fig. 10 shows the variation of m with respect to $\log(1/v_n)$.

We then fix $1/v_n$ at 5×10^{-5}, that is, the network transfer rate is 500 Kbps, where the corresponding maximum m satisfying (3) is 269. We again generate the performance curves (Fig. 11) for MADSN and DSN using the execution time, t_{madsn} and t_{dsn}, with respect to m. Notice that we generate three similar profiles as those in Fig. 9, except that the optimal m is close to three instead of four since the network transfer rate has been increased from 100 Kbps to 1 Mbps.

C. Performance Evaluation: m vs. p

In this set of experiments, we first keep v_n at 500 Kbps, and change p (the total number of sensor nodes) from 10 to 1000. The variation of m with respect to p is shown in Fig. 12.

We then fix p at 3000, where the corresponding maximum m satisfying (3) is 867. We generate the performance curves for MADSN and DSN using the execution time, t_{madsn} and t_{dsn}, with respect to m. Again, we generate three similar profiles (Fig. 13) as those in Fig. 9, except that the optimal m is close to four since the number of sensor nodes has been increased three times.

Table III summarizes some typical parameter values and the corresponding performance. From the last row of Table III, we can see that based on the parameter value we choose, when MADSN reaches its optimum performance, it can save more than 98% of execution time than DSN which mainly contributes from the less data transfer time spent.

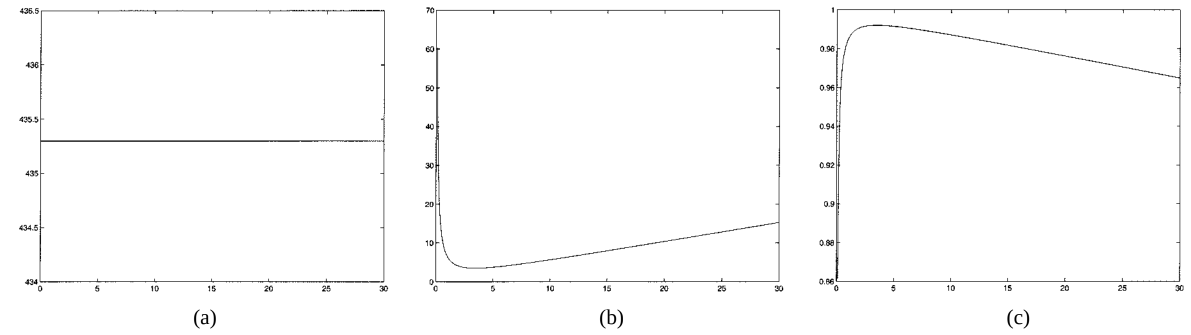


Fig. 13. Execution time for DSN and MADSN with respect to m with $j = 0.25$, $v_n = 500$ Kbps, and $p = 3000$.

TABLE III
SUMMARY OF PERFORMANCE COMPARISON BETWEEN DSN AND MADSN

Parameters	m vs. j	m vs. v'_n	m vs. p
size of agent (s_a)	1K	1K	1K
ratio ($k = \frac{s_f}{s_a}$)	10	10	10
data processing rate (v_n)	100Mbps	100Mbps	100Mbps
overhead of agent (o_a)	0.5s	0.5s	0.5s
ratio ($j = \frac{o_f}{o_a}$)	0.25	0.25	0.25
network transfer rate (v_n)	100Kbps	500 Kbps	500Kbps
total number of sensor nodes (p)	1000	1000	3000
optimal number of agents (m)	4	3	4
execution time in DSN (t_{dsn})	225.1s	145.2s	435.3s
execution time in MADSN (t_{madsn})	4.5s	2s	3.5s
execution time saved ($\frac{t_{dsn}-t_{madsn}}{t_{dsn}} \times 100\%$)	98%	98.6%	99.2%

V. CONCLUSION

This paper describes the use of the mobile agent paradigm to design an improved infrastructure for data integration in distributed sensor network (DSN). We use the acronym MADSN to denote the proposed mobile-agent-based DSN. Compared to the traditional client/server paradigm used in DSN, where data are moved from the client to the processing center, MADSN moves the processing code to the data locations. This saves network bandwidth and provides an effective means for overcoming network latency, since large data transfers are avoided. We studied two important problems related to MADSN design: the distributed data integration problem, and the optimum performance problem.

We show that by applying multiresolution analysis at each sensor node instead of processing element, MADSN saves up to 90% of data transfer time. However, MADSN is not always better than DSN, since the involvement of mobile agents also adds overhead. We analyze the conditions under which MADSN performs better than DSN and the conditions under which MADSN achieves its optimum performance. The conditions are determined by a set of parameters, and the most important ones include the network transfer rate, the overhead ratio between DSN and MADSN, and the total number of sensor nodes. The evaluation shows that when MADSN reaches its optimum performance, it can save more than 98% of execution time (mainly contributed from the less data transfer time spent). We conclude that mobile agent paradigm is a promising approach for distributed computing, especially when the amount of data transfer is very huge which is the typical case in distributed sensor networks.

REFERENCES

[1] S. S. Iyengar, D. N. Jayasimha, and D. Nadig, "A versatile architecture for the distributed sensor integration problem," *IEEE Trans. Comput.*, vol. 32, pp. 175–185, Feb. 1994.

[2] D. N. Jayasimha, S. S. Iyengar, and R. L. Kashyap, "Information integration and synchronization in distributed sensor networks," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, pp. 1032–1043, Sept./Oct. 1991.

[3] A. Knoll and J. Meinkoehn, "Data fusion using large multi-agent networks: An analysis of network structure and performance," in *Proceedings of the International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*. Piscataway, NJ: IEEE Press, 1994, pp. 113–120.

[4] L. Prasad *et al.*, "Functional characterization of sensor integration in distributed sensor networks," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, pp. 1082–1087, Sept./Oct. 1991.

[5] R. Wesson *et al.*, "Network structures for distributed situation assessment," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-11, pp. 5–23, Jan. 1981.

[6] A. D. Birrell and B. J. Nelson, "Implementing remote procedure calls," *ACM Trans. Comput. Syst.*, vol. 2, pp. 39–59, Feb. 1984.

[7] S. Baker, "CORBA implementation issues," in *Proc. IEE Colloq. (Dig.) on Distributed Object Manage.*, London, U.K., January 14, 1994, no. 007, pp. 5/1–5/3.

[8] A. Watson, "OMG (object management group) architecture and CORBA (common object request broker architecture) specification," in *Proc. IEE Colloquium (Digest) Distributed Object Manage.*, January 14, 1994, no. 007, p. 4/1.

[9] T. Sundsted. (1998, June) An introduction to agents. *Java World* [Online] http://www.javaworld.com/javaworld/jw-06-1998/jw-06-howto.html.

[10] S. Franklin and A. Graesser, "Is it an agent, or just a program?: A taxonomy for autonomous agents," in *Proceedings Third International Workshop on Agent Theories, Architectures, and Languages*, J. G. Carbonell and J. Siekmann, Eds. New York: Springer-Verlag, 1996, vol. 1193. [Online]. Available: http://www.msci.memphis.edu/franklin/AgentProg.html.

[11] D. B. Lange and M. Oshima, "Seven good reasons for mobile agents," *Commun. ACM*, vol. 42, no. 3, pp. 88–89, Mar. 1999.

[12] C. G. Harrison, D. M. Chess, and A. Kershenbaum. (1995, March) Mobile agents: Are they a good idea?. Tech. Rep. RC 19887, IBM Thomas J. Watson Research Center, Yorktown, NY. [Online]. Available: http://www.research.ibm.com/massive/mobag.ps

[13] D. Milojicic, "Trend wars—Mobile agent applications," *IEEE Concurrency*, vol. 7, pp. 80–90, July–Sept. 1999.

[14] P. Dasgupta *et al.*, "Magnet: Mobile agents for networked electronic trading," *IEEE Trans. Knowl. Data Eng.*, vol. 11, pp. 509–525, July/Aug. 1999.

[15] M. Hattori *et al.*, "Agent-based driver's information assistance system," *New Generation Comput.*, vol. 17, no. 4, pp. 359–367, 1999.

[16] J. Kay *et al.*, "Atl postmaster: A system for agent collaboration and information dissemination," in *Proceedings of the Second International Conference on Autonomous Agents*. Minneapolis, MN: ACM, 1998, pp. 338–342.

[17] T. Oates, M. V. N. Prasad, and V. R. Lesser, "Cooperative information gathering: A distributed problem-solving approach," *Inst. Elect. Eng. Proc. Softw. Eng.*, vol. 144, no. 1, pp. 72–88, Feb. 1997.

[18] J. S. Wong and A. R. Mikler, "Intelligent mobile agents in large distributed autonomous cooperative systems," *J. Syst. Softw.*, vol. 47, no. 2, pp. 75–87, 1999.

[19] W. Caripe *et al.*, "Network awareness and mobile agent systems," *IEEE Commun. Mag.*, vol. 36, pp. 44–49, July 1998.

[20] K. N. Ross *et al.*, "Mobile agents in adaptive hierarchical Bayesian networks for global awareness," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*. Piscataway, NJ: IEEE Press, 1998, pp. 2207–2212.

[21] L. Prasad, S. S. Iyengar, and R. L. Rao, "Fault-tolerant sensor integration using multiresolution decomposition," *Phys. Rev. E*, vol. 49, no. 4, pp. 3452–3461, Apr. 1994.

[22] L. Lamport, "Synchronizing time servers," Digital System Res. Center, Palo Alto, CA, Tech. Rep. 18, 1987.

Fuzzy Temporal Rules for Mobile Robot Guidance in Dynamic Environments

M. Mucientes, R. Iglesias, C. V. Regueiro, A. Bugarín, P. Cariñena, and S. Barro

***Abstract*—This paper describes a fuzzy control system for the avoidance of moving objects by a robot. The objects move with no type of restriction, varying their velocity and making turns. Due to the complex nature of this movement, it is necessary to realize temporal reasoning with the aim of estimating the trend of the moving object. A new paradigm of fuzzy temporal reasoning, which we call fuzzy temporal rules (FTRs), is used for this control task. The control system has over 117 rules, which reflects the complexity of the problem to be tackled. The controller has been subjected to an exhaustive validation process and examples are shown of the results obtained.**

***Index Terms*—Avoidance of moving obstacles, fuzzy control, fuzzy temporal rules (FTRs), robot guidance.**

I. Introduction

One of the principal fields of research in robotics is the development of techniques for the guidance of autonomous robots. There are many complex problems in this field, mainly due to the nature of the real world (environments which are difficult to model) and the great uncertainty in these environments: the knowledge about an environment is often incomplete, uncertain and approximated, the information usually supplied by the robot sensors is limited and not totally reliable and the environment in which the robot is located usually has a dynamism which cannot be predicted. For all these reasons, fuzzy logic is a useful tool in the field of robotics [1], as has also been demonstrated in numerous studies carried out for guidance in real environments [2], [3], obstacle avoidance [4], route planning [5], etc.

A number of approaches for tackling the problem of robot navigation in the presence of a moving obstacle have been presented. Some studies deal with estimating the moving object's future positions using either an autoregressive model [6] or neural networks [7]. Reference [8] describes a method based on attractive and repulsive forces. On the other hand, in [9], an approach based on the concept of a collision cone is presented. In [10], a system for the monitoring of trajectories to be followed is described. The trajectories of the robot as well as of the moving objects are made up of linear segments along which they move at a constant speed. In [11] and [12], the avoidance of a moving obstacle is solved in a geometrical manner. Finally, in [13] and [14] the avoidance of moving obstacles is done using a fuzzy control system.

With respect to these solutions, a number of aspects should be pointed out. First, the fact that in some approaches the moving objects have restrictions in their movements. On the other hand, a robot usually acts according to the position of the moving object in the immediate past. In certain cases, this may lead to carrying out precipitated and inadequate actions. For instance, given two identical situations at present time, if one of them has been produced due to a hard brake of the moving object and the other one due to an acceleration of this object, they should be solved in a different way, although at present time both situations may look exactly the same.

Our approach to the problem aims to solve this by taking into account the history of more or less recent values of determined variables, which enable us to reflect the different scenarios through which the obstacle has been passing and, thus, verify what its trend is. In this way, one can deduce what the behavior of the robot should be, and take corresponding actions (modification of its speed and/or turning the robot) in order to obtain a behavior pattern in tune with the recent situations. This system is robust in its working, as it permits the avoidance of collisions even when the moving object behaves in a totally unexpected manner. The need to evaluate past situations and previous values of the variables (which in many cases are fuzzy) and principally, to reason them out, has led us to incorporate a temporal reasoning model which we call fuzzy temporal rules (FTRs). The use of conventional fuzzy rules would not permit the direct treatment of this knowledge, since use of average values of variables, would not reflect sharp variations of a variable in a cycle, or it would take a long time to detect a gentle and constant change in a variable. Use of derivatives of variables is even less valid, since it does not permit reasoning with values from the past.

This paper describes a knowledge-based control system for the avoidance of a free-moving mobile object by a robot [16] in a limited environment.[1] The moving objects move varying their speed or turning with no restriction. The system operates in real time (sending the robot three orders/s), it is robust, it enables the robot to operate with imprecise knowledge and takes into account the physical limitations of the environment in which the robot moves, obtaining satisfactory responses for a large number of different situations analyzed by means of the simulation software.

In the following section the problem is posed. In Section III the control system is described in detail, along with the presentation of the temporal reasoning model that is used. Section IV analyzes the results obtained for the simulations carried out and conclusions are given in Section V.

II. Posing of the Problem

As has already been mentioned, the movement of a robot in a dynamic environment is an extraordinarily complex problem. Besides avoiding the collision with the moving object, the robot must move

Manuscript received October 14, 1999; revised June 10, 2001. This work was supported by the Secretaría Xeral de I+D of the Xunta de Galicia under Grant PGIDT99PXI20603A and by the European Commission and Spanish Comisiòn Interministerial de Ciencia y Tecnologìa (CICYT) under Grant 1FD97 0183.

M. Mucientes, R. Iglesias, A. Bugarín, P. Cariñena, and S. Barro are with the Department of Electronics and Computer Science, University of Santiago de Compostela, Santiago de Compostela, Spain (e-mail: manuel@dec.usc.es; rober@dec.usc.es; alberto@dec.usc.es; puri@dec.usc.es; senen@dec.usc.es).

C. V. Regueiro is with the Department of Electronics and Systems, University of A Coruña, A Coruña, Spain (e-mail: cvazquez@udc.es).

Publisher Item Identifier S 1094-6977(01)08914-3.

[1]The robot used is a Nomad 200 by Nomadic Technologies [15].