Distributed Mobile Robotics by the Method of Dynamic Teams

James Jennings and Chris Kirkwood-Watts*

**Department of Electrical Engineering and Computer Science, Tulane University, New Orleans, Louisiana (LA) USA 70118,* {jennings|kirkwood}@eecs.tulane.edu

Abstract. *Distributed teams of agents (robots and workstations) hold great promise for solving complex tasks efficiently, reliably, and automatically. But automatic coordination of the actions of the autonomous agents within a team remains a difficult problem. We suggest that, in order to engender cooperation and to avoid interference among the agents in a team, the general organization of the team reflect the structure of the task. Moreover, the team itself may be a dynamic, fluid entity whose logical structure is maintained by a run-time system, even in the face of attrition, substitution of team members, and explicit recruitment of new members. The Method of Dynamic Teams is based on this principle of fluidity.*

Key Words. Distributed autonomous robots; cooperative tasks; agent teams; dynamic teams; programming systems

1 Introduction

A *distributed team (of agents)* is a logical association of autonomous agents. For our purposes, an agent is a robot or a general-purpose workstation (computer). Distributed teams (of robots and workstations) hold great promise for solving complex tasks efficiently, reliably, and automatically. But automatic coordination of the actions of the agents in a team of autonomous agents remains a difficult problem. For some tasks, such as "foraging", the team may be very loosely coupled, executing an algorithm which does not even require communication among the agents. For other tasks, such as "search and rescue", communication appears to be required, and for cooperative manipulation tasks, close coordination of the actions of agents in the team is essential, e.g. in the Pusher/Steerer manipulation system of [5]. Working cooperatively, our small collection of mobile robots (two of which are shown in Figure 1) are able to explore and map large areas [7], intelligently resolve robot-robot collisions [14], and perform complex large-scale manipulation tasks, such as locating and moving furniture [8]. They do so by forming *dynamic teams* which grow, shrink, and change in membership automatically during task execution in response to conditions in the environment and within the team itself.

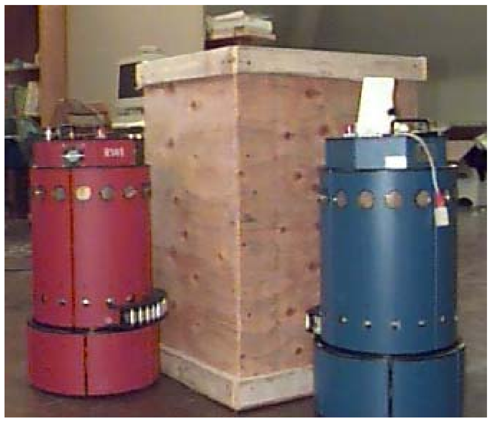

Figure 1 *Two of our mobile robots, Ernst and Moseley, near a large box which might be the object for which they are searching and which they would then retrieve. There are two other mobile robots in our laboratory, Elvis and Stella.*

1.1 Problem Statement

We wish to allow teams of small mobile robots (*mobots*) to cooperate to solve tasks too difficult for a single robot to achieve or to solve tasks more efficiently than could a single robot working alone.

Cooperation implies at least non-interference; that is, agents should not interfere with other agents' abilities to accomplish their goals. Cooperation also means that agents can (and may be required to) help each other by collaborating on a task, i.e. coordinating their actions. Because robot tasks vary so widely (e.g. from exploration to map-making to assembly operations to large-scale manipulation) and robot teams vary so widely (e.g. from an array of thousands of nano-robots to a heterogenous collection of a just a few large robots), it appears impossible to design a set of enforceable constraints on robot programs which promotes collaboration and avoids interference *for all tasks*.

We believe the key to enabling cooperation lies in the structure of the task itself: the specification of the robots' (generally, agents') program should encode, however loosely, the desired form of cooperation *for that particular task*. The role of the agent architecture, or programming environment, is then to support as wide a variety of team organizations and activities as possible. The Mover system is an attempt at constructing such an architecture.

1.2 Dynamic Teams

In our architecture MOVER, enhanced since [8], tasks are specified procedurally, but at a high level of abstraction. *Selection* of the agents which participate in a team is automatic, as is *substitution* for agents in the team by agents outside of the team. For example, if the battery voltage of a robot participating in a team task falls too low, that robot can migrate its task (its computational state) to another robot whose batteries are fully charged. This is accomplished without human intervention. Similarly, due to failures or the intervention of higher priority tasks, teams may experience *attrition*. Finally, any agent in the collective may be a member of an arbitrary number of teams simultaneously.

2 Tasks and Teams to Solve Them

2.1 Properties of Association

In the field of autonomous mobile robotics, many tasks involve one or more forms of association of agents. Key issues in cooperative mobile robotics may be termed "properties of association" because they describe the ways in which agents are associated with one another. Questions about association properties include:

1. Can the agents sense each other?
2. Can the agents sense the effects of the actions of other agents?
3. Can the agents communicate with each other?
4. Can several agents act in synchrony?
5. How the agents organized?

Questions 4 and 5 are the subject of this report. In our current work, we answer the other questions as follows: Our mobots sense each other and the world only through sonar and bumper contact, but they can communicate via wireless ethernet.

Answering question 4 invariably exposes limitations in robotic hardware and software systems. Two robots which are slaves to a central controller can clearly act in synchrony, but at the expense of their autonomy. But can two autonomous robots perform synchronous actions? The answer appears to be yes, up to the limits of their ability to communicate. In this paper, we use the following definition:

Definition 2.1 *A set of agents perform action B* synchronously *in the sequential task $\{A; B; C\}$ if (i) no agent starts B until all agents have finished A, and (ii) no agent starts C until all agents have completed B.*

Finally, question 5, "How are the agents organized?", is the chief concern of our current work. First we note that many types of organization are possible, including *a single set of peers*, *distinct groups of peers*, *groups with leaders*, *hierarchies*, etc. Which is best? One cannot answer this question without knowledge of the problem that the agents are trying to solve. *Foraging*, for example, appears to be performed very successfully by a single group of peers [2]. *Following*, on the other hand, appears to require a leader, but the leader may be elected dynamically and may change frequently. Cooperative *manipulation* often requires very close coordination, e.g. between a leader and a follower, and still other tasks may be naturally specified in terms of hierarchies.

Varying the answers to questions 1–5 above yields teams of agents with different abilities, from reticent, asynchronous, and unorganized teams, to extremely communicative, highly synchronized teams, organized in complex ways. A framework for programming teams of agents should embrace a wide variety of association properties, and thus enable more creative, powerful, and efficient solutions to many tasks. The Method of Dynamic Teams is an attempt at forging such a framework.

2.2 The Method of Dynamic Teams

We begin by defining *team* and *dynamic team* more precisely.

Definition 2.2 *A* team *of agents is a logical association of members of a (possibly larger, possibly unorganized) set of agents.*

Definition 2.3 *A* dynamic team *of agents is a temporary and fluid team whose association properties are allowed to vary over time. That is, teams dynamically and automatically grow and shrink, and members may be substituted. Also, an agent may be a member of more than one team at a time.*

Definition 2.4 The Method of Dynamic Teams (MDT) *is a programming model which addresses the mapping of a task into dynamic teams of agents such that the structure of the teams is consistent with that of the task.*

A team τ of agents is created for the purpose of performing some task. In this way, the team is "temporary", for its dissolution will be contemporaneous with the completion of the task. If τ has the duration of some task T, then the decomposition of T into two less complex subtasks, T_1 and T_2, begs the ability to form subteams τ_1 and τ_2 from τ.

The team τ is said to be "fluid" with respect to the task T. Members of the team may be added (*recruited*), swapped with members of other teams (*substituted*), or the team may lose members (*attrition*), without the need for another team to be explicitly formed. In this manner, teams of agents may be adjusted dynamically and automatically to increase performance or to account for unforseen changes in the task structure.

Another critical aspect of MDT is that an agent may belong to more than one team at once. However, some care must be taken to ensure that the non-sharable nature of some resources is not compromised (the wheelbase motors of a mobile robot, for example). Certain resources (such as a mobot's wheelbase) are managed automatically by the MOVER system.

2.3 Specifying Dynamic Team Structure

In the MDT approach, the desired type of agent organization is part of the task description given to the agents. Whether a task solution (program) is designed by a user, by a planning system, or by another agent, the general type of organization dervies from the task solution itself. For example, in a *search and rescue* task (see Figure 2), a team of robots might search a building in parallel for some object (as a group of peers). Upon finding the object, a subset of those robots might bring the object back (as a separate group from the group of robots whose services are no longer needed). This "rescue" of the object may require coordinated manipulation (implemented, e.g. using a leader and a follower).

Which agents participate in the task and in what capacities is determined at run-time and is mostly automatic. That is, how the desired form of organization is achieved is separate from its specification. For example, in the MOVER system, teams are assembled automatically from a pool of available agents using the `with-team` construct.[1] New agents are explicitly recruited when needed using `recruit`. Agents which experience a hardware failure, low battery voltage, or some other condition may allow another agent to substitute for them using `substitute`. In the event no substitute can be found, or is not desired, an agent can bail out of a task using `bailout`, and the team experiences attrition. If attrition is too great, a *team exception* is raised, the entire distributed task is stopped, and an operator is alerted. (See Section 4.2.)

2.4 Implementation

The MOVER system provides `with-team`, `recruit`, and the other dynamic team constructs within a novel distributed implementation of the Scheme programming language called Kali-Scheme [6]. Because of MOVER's architecture, `with-team` programs can and do re-use pre-existing single-agent code without modification.

Using a collection of Sun SPARC workstations running Solaris, Intel x86-based computers running Linux, and RWI B13 and B14 mobile robots (also running Linux), MOVER is used in our laboratory for multi-agent tasks [8] [7].

[1]The `with-team` construct and the others mentioned here are explained in Section 4.2.

3 Previous Work

Due to space limitations, we are forced to focus our attention narrowly on architectures for multi-robot cooperation. We start with behavior-based approaches.

Behavior-based robot programming lends itself naturally to multi-robot tasks because each robot is programmed (given a set of behaviors) and then set out in the world, and from the interaction of the robots a group behavior *emerges*. If the emergent behavior helps accomplish the task, the behavior is said to be cooperative. A great deal of effort is expended designing cooperative behaviors, e.g. [4] [11] [10], and classifying them [2]. In this work there is typically no formal notion of a team, nor of subteams, contemporaneous teams, etc. Even with direct communication, it is difficult to envision how one might craft sets of behaviors from which would emerge teams (logical associations) that would be capable of attaining structured goals. (This may largely be due to the fact that structured tasks are often not the goal of such work.)

Some recent work [12] [9] models reactive robot behaviors using dynamical systems, in an attempt to define and then synthesize cooperation at the systems level. The idea is appealing, but it is uncertain how it might be applied to real robots in complex environments which need to perform structured tasks.

Other recent work presents a variety of multi-agent protocols designed to engender cooperation, e.g. using plan-merging [1]. In a system in which individual robots are controlled by planning systems which are amenable to the approach, the idea of cooperative plan merging is appealing. In a more general scenario, in which individual robots are controlled using a variety of paradigms, the negotiation method of [3] shares more with our approach.

Similarly, the architecture of [13] allows hierarchical organizations to form as needed in support of a "global mission plan." Dynamic teams for accomplishing cooperative tasks are similar in spirit to the presence of a "mission plan" which is global only to the robots involved in the team. The idea of organizations of agents emerging dynamically as needed is common to this work and ours.

4 An Example Task: Search and Rescue

We now examine an implemented distributed robot task with several steps, each requiring a slightly different organization of a set of mobile robot agents.[2] The goal of a *search and rescue* task is to search for an object, and upon finding it, to retrieve it. We desire the search phase to exploit parallelism, with many robots searching at once. On the other hand, our manipulation

[2]In our implementation, three of our mobile robots search the lab in parallel, using a random walk, looking for a large box of a particular shape. When one robot finds it, two robots manipulate it to a specified goal in the room.

```
(with-team (all-available 'mobots)
  (let* ((location
           (on ((all)
                (collect or-collector)
                (on-error substitute-if-possible))
              (lambda () (search-for *object*))))
          (first-to-arrive
            (on ((all)
                 (collect (make-n-collector 2))
                 (on-error substitute-if-possible))
               (navigate-to (success-data location)))))
     (subteam (success-agents first-to-arrive)
       (on ((all))
         (lambda () (push-to *goal*))
         (lambda () (steer-to *goal*))))))
```

Figure 2 *The search and rescue program. In Scheme, the keyword* `lambda` *indicates a procedure. The* `with-team` *construct assembles a team and then distributes its body (which begins with* `let*`*) to each member of the team for autonomous execution.*

primitives require exactly two robots to move a large object in the retrieval ("rescue") phase. A task solution, shown in Figure 2, demonstrates the following attributes of the system:

- All available mobile robots are selected to participate in the search phase of the task. A simple classification mechanism serves to remove workstations and non-mobile robots from the search team.
- A step marked `on` is a synchronous step. The result of an `on` step is a structure containing the values returned from each robot participating in the step. The keyword `(all)` selects for participation *all* members of the current team.
- The results of the first `on` step (the *search step*) are collected using the `or-collector`, which terminates the `on` step when the first successful result is returned from any participating robot.
- During the search step, `substitute-if-possible` is the error handling procedure which attempts to migrate the task of a robot which experiences an error to another robot if possible.
- When the search step is done, two robots `navigate-to` the object in order to manipulate it. Because it does not matter to the operator which robots perform the manipulation, all robots are instructed to navigate to the object synchronously, with `(make-n-collector 2)`, the result collector, terminating the step when two robots have completed it. Thus, the first two robots to arrive will perform the manipulation.

- The `subteam` construct narrows the team. Here, a subteam composed of the first two members of the search team to arrive at the object will perform the manipulation task. Note that the default collector of results is the `and-collector`, which terminates the synchronous step when *all* agents have completed their task.

The meaning of other forms appearing in Figure 2 may be summarized as follows. The `success-data` operator selects the data returned by a successful agent in a synchronous sub-task. (Unsuccessful agents return errors, or indicate that they are leaving the team.) The `success-agents` operator performs a similar extraction by selecting the agents which succeeded from the result of a synchronous sub-task. Finally, `push-to` and `steer-to` are manipulation primitives designed to work together (when each is executed by a different robot) to move a large object to a specified goal location [5].

4.1 Some Features of `with-team`

The search and rescue example illustrates many features of the `with-team` construct. The body of the `with-team` form is a sequence of Scheme forms which are sub-tasks, executed in order by all agents in the team. Every subtask which is wrapped in an `on` form is executed synchronously, and sub-tasks outside `on` forms are performed asynchronously by all agents.[3]

At times it is desirable to specify restrictions on how MOVER selects agents for team tasks. In Figure 2, the keyword `all` is used in every step to select all robots in the current team. The `on` construct permits options such as `(arbitrary 2)` and `(fixed (list ernst))` in place of `(all)`. The `(arbitrary` *n*`)` option automatically and arbitrarily chooses *n* agents, and the `(fixed` *list*`)` option selects exactly the agents in *list*.

Finally, a task's structure may allow opportunities to recruit new members into existing teams. A `recruit` construct is provided in MOVER for this purpose, and it is a synchronous operation much like `on`, but which has the sole effect of possibly enlarging the team. A forthcoming report will describe recruitment, attrition, and substitution in more detail than space permits in this report.

4.2 Specification of Concurrency

It is beyond the scope of this paper to do more than state the following properties of our implementation of MDT.

MOVER's `with-team` is a high-level concurrency specification mechanism which abstracts away from: *choosing among equivalent agents; selecting the "closest", "strongest", or "least-loaded" agent; innocuous substitution of agents; and lower level issues such as synchronization and communication.*

[3]Note that there is no restriction on the tasks which might appear inside `with-team`, or on their number. For example, `with-team` tasks may be nested.

The agents' operator retains control over: *the amount of parallelism used in the task; the level of coordination required among team members; which errors require aborting the task; and identifying opportunities for recruiting new members smoothly into the task.*

Substitution in MOVER is implemented with true task migration. The thread (a form of lightweight process) which executes on agent A is actually migrated to agent B when B substitutes for A. This disassociates agent A completely from the task. Note, however, that for a robot, an important part of its state is not computational but physical. In some tasks, such as general navigation (implemented in MOVER as `navigate-to`), it may be possible for a robot to substitute for another without physically changing places, because the task goal is simply for a robot to arrive at a location.

In addition to error handling at the individual agent level, MOVER has *team exceptions* which are processed synchronously by every agent in the team. This allows the team to respond as a whole to unexpected events.

5 Conclusion and Future Work

We have presented a general framework for distributed robot (agent) programming by the Method of Dynamic Teams, which suggests that the general structure of a team of agents should mirror the structure of the task they must perform. When teams are dynamic, they can automatically respond to conditions in the environment and within the agents themselves. In other words, a system implementing MDT gives the robot "operator" the ability to combine reactive agent primitives with adaptive execution strategies, in the context of a user-specified task structure. Future work includes the design of algorithms for mapping well-defined classes of tasks directly and automatically into dynamic teams.

6 Acknowledgements

We are grateful to the developers of Kali-Scheme, and especially to Richard Kelsey for much assistance. We thank Eric Beuscher, Matt Bosworth, and Craig Tanis for much coding and decoding.

This paper describes research done in the Robotics Laboratory of the Department of Electrical Engineering and Computer Science at Tulane University. Support for our robotics research was provided in part by the Louisiana Education Quality Support Fund under Contract Number LEQSF-RD-A-27.

7 REFERENCES

[1] R. Alami, F. Ingrand, and S. Qutub. Planning coordination and execution in multi-robots environment. In 8^{th} *International Conference on Advanced Robotics*, Monterey, CA, 1997.

[2] R. Arkin, T. Balch, and E. Nitz. Communication of behavioral state in multi-agent retrieval tasks. In *Proc. of the 1993 IEEE International Conference on Robotics and Automation*, volume 2, pages 588–594, Atlanta, Ga, 1993.

[3] K. Azarm and G. Schmidt. Conflict-free motion of multiple mobile robots based on decentralized motion planning and negotiation. In *Proc. IEEE Int. Conf. on Robotics and Automation*, Albuquerque, NM, 1997.

[4] D. Barnes, R. Aylett, A. Coddington, and R. Ghanea-Hercock. A hybrid approach to supervising multiple co-operant autonomous mobile robots. In 8^{th} *International Conference on Advanced Robotics*, Monterey, CA, 1997.

[5] R. Brown and J. Jennings. Manipulation by a pusher/steerer. In *Proc. of IEEE Conf. on Intelligent Robot Systems*, Pittsburgh, PA, August 1995.

[6] H. Cejtin, S. Jagannathan, and R. Kelsey. Higher-order distributed objects. *ACM Transactions on Programming Languages and Systems*, September 1995.

[7] J. Jennings, C. Kirkwood-Watts, and C. Tanis. Distributed map-making using online generalized voronoi graphs. In **submitted to** *IEEE ICRA*, Brussels, Belgium, 1998.

[8] J. Jennings, G. Whelan, and W. Evans. Cooperative search and rescue with mobile robots. In *IEEE International Conference on Advanced Robotics*, Monterey, CA, 1997.

[9] E. Large, H. Christensen, and R. Bajcsy. Dynamic robot planning: Cooperation through competition. In *Proc. IEEE Int. Conf. on Robotics and Automation*, Albuquerque, NM, 1997.

[10] M.J. Mataric, M. Nilsson, and K.T. Simsarian. Cooperative multi-robot box-pushing. In *Proc. of IEEE Conf. on Intelligent Robot Systems*, Pittsburgh, PA, 1995.

[11] L. Parker and B. Emmons. Cooperative multi-robot observation of multiple moving targets. In *Proc. IEEE Int. Conf. on Robotics and Automation*, Albuquerque, NM, 1997.

[12] K. Sekiyama and T. Fukuda. Modeling and controlling of group behaviour based on self-organizing principle. In *Proc. IEEE Int. Conf. on Robotics and Automation*, Minneapolis, MN, 1996.

[13] J. Sousa and F. Pereira. A general control architecture for multiple vehicles. In *Proc. IEEE Int. Conf. on Robotics and Automation*, Minneapolis, MN, 1996.

[14] C. Tanis. Cooperative localization and mapmaking for mobile robots. Department of electrical engineering and computer science technical report, Tulane University, 1997.