# A Distributed Layered Architecture for Mobile Robot Coordination: Application to Space Exploration

**Dani Goldberg, Vincent Cicirello, M. Bernardine Dias**
**Reid Simmons, Stephen Smith, Trey Smith, Anthony Stentz**
Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213
{danig,cicirello,mbdias,reids,sfs,trey,axs}@ri.cmu.edu

### Abstract

This paper presents an architecture that enables multiple robots to explicitly coordinate actions at multiple levels of abstraction. In particular, we are developing an extension to the traditional three-layered robot architecture that enables robots to interact directly at each layer – at the behavioral level, the robots create distributed control loops; at the executive level, they synchronize task execution; at the planning level, they use market-based techniques to assign tasks, form teams, and allocate resources. We illustrate these ideas in the context of a Mars exploration scenario.

**Keywords:** Multi-robot coordination, robot architecture, task-level control, space exploration.

## Introduction

An architecture for multi-robot coordination must be able to accommodate issues of synchronization and cooperation under a wide range of conditions and at various levels of granularity and timescales. At the concrete, physical level of sensors and actuators, the robots need to respond quickly to dynamic events (such as imminent collisions), while at the same time reasoning about and executing long-term strategies for achieving goals. Executing such strategies is likely to involve establishing and managing a variety of synchronization constraints between robots. For some tasks, such as distributed search, coordination may be loose and asynchronous; for others, such as cooperative manipulation of a heavy object, coordination must be tightly synchronized.

In designing a multi-robot architecture that allows flexibility in synchronization and granularity, one must inevitably negotiate the tension between centralized and distributed approaches. A centralized system can make optimal decisions about subtle issues involving many robots and many tasks. In contrast, a highly distributed system can quickly respond to problems involving one, or a few, robots and is more robust to point failures.

We are developing a multi-robot coordination architecture that addresses these issues, providing flexibility in granularity and synchronization, while also attempting to accommodate the strengths of both distributed and centralized approaches. The architecture is an extension of the traditional three-layered approach, which provides event handling at different levels of abstraction through the use of behavioral, executive, and planning layers. Our approach extends the architecture to multiple robots by allowing robots to interact directly at each layer (see Figure 2). This provides several benefits, including: (1) plans can be constructed and shared between multiple robots, using a market-based approach, providing for various degrees of optimization; (2) executive-level, inter-robot synchronization constraints can be established and maintained explicitly; and (3) distributed behavior-level feedback loops can be established to provide for both loosely- and tightly-coupled coordination.

Each layer is implemented using representations and algorithms that are tuned to the granularity, speed, and types of interactions typically encountered at each level (symbolic/global, hybrid/reactive, numeric/reflexive). The strengths of this architecture are its flexibility in establishing interactions between robots at different levels and its ability to handle tasks of varying degrees of complexity while maintaining reactivity to changes and uncertainty in the environment.

This paper presents the major components of the architecture together with a space exploration application illustrating their use. An initial implementation of the architecture is nearly complete, with detailed experiments scheduled to begin soon. The status of the major components of the architecture is also discussed in the following sections.

## Related Work

Our approach blends the advantages of both the centralized and distributed approaches to multi-robot systems. In the centralized approach, a centralized planner plans out detailed actions for each robot. For example, a planner might treat two 6 DOF arms as a single 12 DOF system for the purpose of generating detailed trajectories that enable the arms to work together in moving some object, without bumping into each other (Khatib 1995). While this approach provides for close coordination, it does so at the expense of local robot autonomy. In particular, this approach usually employs centralized monitoring and, if anything goes wrong, the planner is invoked to re-plan everything. Thus, this approach suffers from single point failure and lack of local reactivity.

At the other end of the spectrum, in the distributed approach (Arkin 1992; Balch & Arkin 1994; Mataric 1992; Parker 1998), each agent is autonomous, but there is usually no explicit synchronization among the robots. Coordination (or, more accurately, cooperation) occurs fortuitously, depending on how the behaviors of the robots interact with the environment. For instance, in the ALLIANCE architecture (Parker 1998), robots decide which tasks to perform in a behavior-based fashion: They have "motivations" that rise and fall as they notice that tasks are available or not. While ALLIANCE can handle heterogeneous robots (robots can have different motivations for different tasks), it does not deal with the problem of explicit coordination.

(Jennings & Kirkwood-Watts 1998) have developed a distributed executive for multi-robot coordination. The executive, based on a distributed dialect of Scheme, is similar to our executive language in the types of synchronization constructs it supports. As with our work, this enables robots to solve local coordination problems without having to invoke a high-level planner.

Several researchers have investigated economy-based architectures applied to multi-agents systems (Sandholm & Lesser 1995; Sycara & Zeng 1996; Wellman & Wurman 1998), beginning with work on the Contract Net (Smith 1980). (Golfarelli, Maio, & Rizzi 1997) proposed a negotiation protocol for multi-robot coordination that restricted negotiations to task-swaps. (Stentz & Dias 1999) proposed a more capable market-based approach that aims to opportunistically introduce pockets of centralized planning into a distributed system, thereby exploiting the desirable properties of both distributed and centralized approaches. (Thayer *et al.* 2000), (Gerkey & Matarić 2001), and (Zlot *et al.* 2002) have since presented market-based multi-robot coordination results. (Musliner & Krebsbach 2001) described MASA-CIRCA, a distributed Contract Net-based architecture with an emphasis on hard real-time execution.

Before presenting the details of our approach in Section , we first motivate the discussion with a description of our experimental scenario.

## Mars Exploration Scenario

Our Mars exploration scenario is premised on the notion of scientific return, i.e., that a group of robots would be sent to Mars for the (potentially) valuable information they gather and return to Earth. We envision a scenario where a colony of heterogeneous robots is deployed on Mars. Scientists on Earth communicate high-level task descriptions to the colony (e.g., "find and gather data on several carbonate rocks"). We assume that the tasks given to robots may far exceed what can be accomplished during the lifetime of the mission. In addition, communications limitations (bandwidth, delays, blackouts) necessitate highly autonomous robots, and preclude effective tele-operation of the robots or micro-managing of task execution by the scientists. The robots are therefore responsible for deciding which/how tasks are to be accomplished, based on, among other things, the tasks' relative priorities. The goal for the robots is to utilize their time, resources, and capabilities efficiently so as to provide the highest possible scientific return on the tasks they are given. Our distributed, layered architecture is designed to facilitate this goal.

In terms of the development and testing of our current system, we have focussed on a characterize-region task that will fit within a broader exploration scenario. In this task, a user/scientist specifies a region on the Mars surface, indicating that rocks within that region are to be characterized with an appropriate sensing instrument. The scientist may also specify the locations and types of rocks, if known. In our current implementation of the task, there are three types of rocks (`RockA`, `RockB`, and `RockAB`) and three types of rock characterizing sensors (`SensorA`, able to fully characterize `RockA` and partially characterize `RockAB`; `SensorB`, able to fully characterize `RockB` and partially characterize `RockAB`; and `SensorAB`, able to fully characterize all three rock types). Each robot has at most one rock characterizing sensor, requiring, for example, a robot with `SensorA` and one with `SensorB` to coordinate in order to fully characterize a `RockAB`-type rock, but allowing a robot with `SensorAB` to fully characterize the rock by itself. While relatively simple conceptually, this scenario has many possible variations that make it a nice testing ground for our architecture. With respect to testing, a 3D graphical simulator developed for the project currently provides the "physical" robots and environment required (Figure 1),

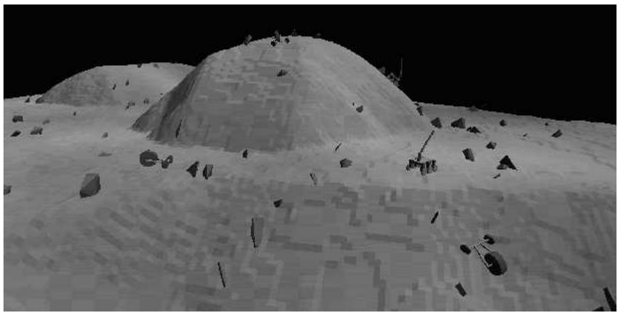

Figure 1: Simulated Mars environment with multiple rovers and rocks.

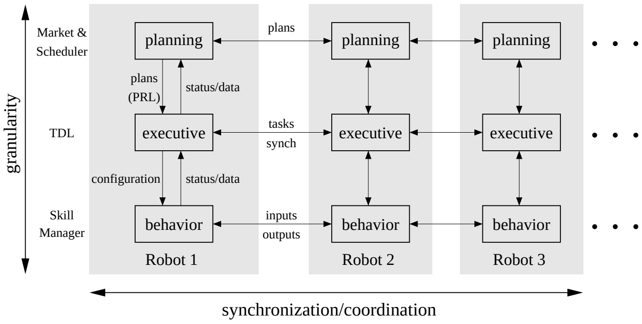


Figure 2: Layered multi-robot architecture

though, in the future, we plan to use real robots as well.

## Approach

Our multi-robot architecture is based on the layered approach that has been adopted for many single-agent autonomous systems (Bonasso *et al.* 1997; Muscettola *et al.* 1998; Simmons *et al.* 1997). These architectures typically consist of a planning layer that decides how to achieve high-level goals, an executive layer that sequences tasks and monitors task execution, and a behavioral layer that interfaces to the robot's sensors and effectors.

Typically, information and control flows up and down between layers. The planning layer sends plans to the executive, which further decomposes tasks into subtasks and dispatches them based on the temporal constraints imposed by the plan. Dispatching a task often involves enabling or disabling various behaviors. The behaviors interact to control the robot, sending back sensor data and status information. The executive informs the planner when tasks are completed, and may further abstract sensor data for use by the planner. The executive also monitors task execution: In case of failure, it can try to recover or it can terminate the task and request a new plan from the planner.

We extend this architectural concept to multiple robots in a relatively straightforward way. Each robot is composed of a complete three-layered architecture. In addition, each of the three layers can interact directly with the same layer of other robots (Figure 2). Thus, each robot can act autonomously at all times, but can coordinate (at multiple levels) with other agents, when needed. By allowing each layer to interact directly with its peers, we can form distributed feedback loops, operating at different levels of abstraction and at different timescales. In particular, the behavioral layer coordinates behaviors, the executive layer coordinates tasks, and the planning layer coordinates/schedules resources. In this way, problems arising can be dealt with at the appropriate level, without having to involve higher layers. This decreases latency and may increase robustness (since lower layers typically operate with higher-fidelity models).

The following sections describe each of the layers of the architecture in more detail, and how they have been applied to our exploration scenario.

### Coordinated Behaviors

The behavioral layer consists of real-time sensor/effector feedback loops. By connecting the sensor behaviors of one robot to the effector behaviors of another, we can create efficient distributed servo loops (Simmons *et al.* 2000b). Similarly, by connecting effector behaviors together, we can create tightly coordinated controllers. For instance, two robots could coordinate their arm and navigation behaviors to jointly carry a heavy piece of equipment (Pirjanian, Huntsberger, & Barrett 2001).

A multi-robot behavioral layer that can support such capabilities needs several critical features. First, it must be possible to connect behaviors to one another, enabling sensor data and status information to flow between behaviors on different robots. The architecture should not place restrictions on the type of data that can pass between distributed behaviors. Also, it should be possible to connect behaviors transparently on different robots, in the same way that one connects them on the same robot. Finally, high-bandwidth, low-latency communications is needed to achieve good performance in interacting, multi-robot behaviors.

Our implementation extends the Skill Manager of (Bonasso *et al.* 1997) to provide for both intra- and inter-robot connections. Skills are connected via input/output ports and operate in a data-flow fashion: When a new input value arrives on a port, the skill runs an action code that (optionally) produces outputs. For skills on the same robot, the connection is via function call; for inter-robot connections, data flows using a transparent message-passing protocol. Certain aspects of the skills, such as the action code and number and types of ports, are statically determined at compile time. Most aspects, however, can be dynamically configured at run time (either from the executive layer, or from a skill's action code). These include the ability to enable and disable a behavior, set the value of an input, set parameters of the skill, and create or destroy the connections between ports. The executive layer can also subscribe to skill outputs.

These ideas have been demonstrated in the context of distributed visual servoing for large-scale assembly using multiple, heterogeneous robots (Simmons *et al.* 2000b). The task, which is to move the end of a large beam into a vertical stanchion, uses three robots (Figure 3) – an observer (a mobile robot with stereo vision) and two controllers (an automated crane and a mobile manipulator). A similar form of tightly-coupled visual servoing could be applied to our Mars rock characterization task if, for example, the characterization sensor required extremely precise placement not possible without an observer robot. In our current implementation, tightly-coupled behavior-level interactions are not required, and consequently neither are intra-robot connections, though future scenarios will make use of both.

The executive layer does make extensive use the dynamic configuration capability of the behavioral layer. Suppose the executive layer wishes to execute the task of characterizing a rock at a give location. It first establishes the proper inter-robot behavioral connections for navigating to the rock and awaits the skill output signaling completion of the task, at which point it destroys the current connections and establishes those required for characterizing the rock. Behavior connections are "established" and "destroyed" using a publish/subscribe mechanism provided by our interprocess communications package.

We have developed a flexible, efficient inter-process communications package, called IPC (Simmons & Whelan 1997), that is based on TCP/IP and runs under most operating systems in C, C++, Java, and Lisp. It features both publish/subscribe and client/server types of message passing. In particular, publishers can broadcast messages and all processes that subscribe to that message will receive a copy. When messages are received, callbacks are automatically invoked. IPC includes features for automatically marshaling and unmarshaling arbitrary types of data. Programs that use IPC indicate the message format using a string-based language, and IPC parses the format string and uses it to convert user data into byte streams, and *vice versa*, taking into account network byte ordering and structure packing conventions.

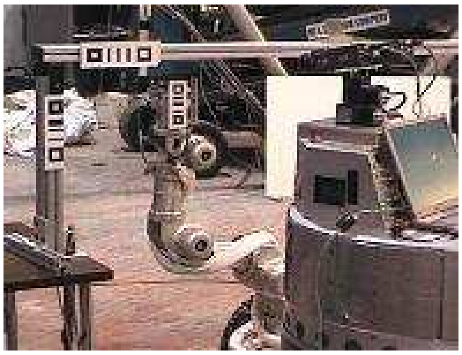

Figure 3: Heterogeneous robots for large-scale assembly

Figure 4: Coordinated deployment of heterogeneous robots

In our architecture, IPC is used both for communicating between layers in a single agent and between layers in different agents. Processes can subscribe, and unsubscribe, to messages at any time. In the behavioral layer, this capability is used for making and breaking intra-robot and inter-robot skill connections. Making a connection simply requires subscribing to the correct message; breaking the connection requires unsubscribing. The next section describes the executive layer of the architecture.

## Coordinated Task Execution

The executive layer has responsibility for hierarchically decomposing tasks into subtasks, enforcing synchronization constraints between tasks (both those imposed by the planner and those added during task decomposition), monitoring task execution, and recovering from exceptions (Simmons 1994).

For coordinated multi-robot task execution, it should be possible to synchronize tasks transparently on two different agents, in the same way as if they were performed by a single agent. For instance, we may want to state that a robot should not start analyzing a rock until two other rovers have moved into place to provide assistance. In addition, a distributed executive should facilitate one robot monitoring the execution of another robot and helping it recover from faults. Finally, it is desirable for one robot to be directly able to request that another robot perform a task (such as assisting it to perform visual servoing), but care must be taken to prevent the robot from illegitimately executing a task on the other robot.

Our interface between planner and executive is based on PRL, a plan representation language that was developed for an earlier project (Simmons *et al.* 2000a). PRL represents plans as a hierarchy of tasks, with each task defined in terms of parameters, subtasks, and temporal constraints between the subtasks. We have recently extended PRL in two ways: (1) to enable specification of the resource utilization of a task and which agent should be executing it; (2) to provide automatic status reports on executing tasks (indicating, for example, when each task has successfully completed).

PRL's automatic status reports provide an important connection between the executive and planning layers in our architecture. When, for example, a robot completes the task of characterizing a particular rock, the status report reflecting this is used by the planning layer to update the schedule and determine which new task(s) should be sent to the executive.

The executive is implemented using the Task Description Language (TDL). TDL is an extension of C++ that contains explicit syntax to support hierarchical task decomposition, task synchronization, execution monitoring, and exception handling (Simmons & Apfelbaum 1998). Recently, we have extended TDL to handle task-level coordination between robots and to enable one robot to spawn or terminate a task on another. TDL transparently handles passing task data (function parameters) and synchronization signals between robots, using message passing based on our IPC package. These capabilities have been used to perform coordinated deployment (Figure 4) of multiple, heterogeneous robots (Simmons *et al.* 2000a).

We have demonstrated TDL's distributed task-level coordination capabilities in a scripted, proof-of-concept scenario meant to encompass the characterize-region task. This scenario has a rover climbing a hill (Figure 1), doing a panoramic "survey" of the surrounding area, and deploying other rovers to areas of potential interest where they might, for example, perform the characterize-region task. While TDL's distributed features are available in our current system, all of the necessary protocols for using them are not in place. Among these is a distributed planning-layer mechanism that would establish the parameters and constraints for executive-level distributed tasks. This would prevent the undesirable possibility of executing a distributed task that has not been agreed upon. For example, using the distributed features of TDL, one rover can execute a task on a second rover. Unless the second rover has "agreed" (at the planning level) to execute this task, the results could be disastrous. The second rover, for example, could be in the process of placing a delicate scientific instrument on a rock. The unplanned movement caused by the distributed task might break the instrument.

The next section presents details of the planning layer.

## Coordinated Planning: Market and Scheduler

Our approach to task allocation and planning is based on a market economy. An *economy* is essentially a population of agents coordinating with each other to produce an aggregate set of goods. *Market economies* are generally unencumbered by centralized planning, instead leaving individuals free to exchange goods and services and enter into contracts as they see fit. Despite the fact that individuals in the economy act only to advance their own self-interests, the aggregate effect is a highly productive society.

We have developed a market-based architecture in which tasks are allocated based on exchanges of single tasks between pairs of robots (Dias & Stentz 2001). A robot that needs a task performed announces that it will auction off the task as a buyer. Each seller capable of performing the task for a cost $c$, bids to do so for $c + \epsilon$. The buyer accepts the lowest bid, as long as it is cheaper than doing the task itself. If a bid is accepted, the seller performs the task and pockets $\epsilon$ as profit.

In addition to the market component, or *trader*, of each robot, the market also contains *operator* traders that are similar in function except that: they are not associated with a robot, they provide a user interface to the system, and they act on behalf of the scientists/users. When a scientist introduces a characterize-region task into the system, it is the operator trader that decomposes the task into its components, if necessary, and auctions off the task(s) while trying to minimize cost.

(Zlot *et al.* 2002) reports on multi-robot exploration of an unknown environment using a similar market architecture. Each robot generates a list of target points to visit, and orders them into a tour (using an approximate TSP algorithm). Our characterize-region task can be similarly solved with a distributed approximation to an optimal TSP tour.

The market works closely with the other major component of the planning layer: the scheduler. Following the market economy framework, a scheduler is associated with each robot and it is responsible for maintaining the robot's current agenda of accepted and pending tasks. The scheduler plays a critical role both in the formation of bids and in the interaction between the planning and executive layers. Before a given robot trader can bid on a new task, it must first ascertain from the robot's scheduler whether the task can in fact be feasibly undertaken (given resource/timing constraints and the other tasks already in the schedule) and, if so, what is the cost. For our initial characterize-region task scenario, we are assuming robot travel time as a baseline measure of cost. Later versions will incorporate more sophisticated metrics such as expected value of information. Once a robot trader is awarded a task, it is added to the schedule and now further constrains decisions to bid on other tasks. The scheduler is responsible for subsequently sending the task to the executive so that it is executed appropriately with respect to the other tasks that the robot must accomplish.

The scheduler component of the market architecture is implemented using a general stochastic search framework called WHISTLING (Cicirello & Smith 2001; 2002). The WHISTLING framework is inspired by the self-organizing behavior of wasp colonies (hence the name Wasp beHavior-Inspired STochastic sampLING). But extracted from this context, it provides a basic mechanism for randomizing a given search heuristic in a way that concentrates random activity (and search) in those neighborhoods of the search space where the heuristic is not well-informed. Presuming a reasonable domain heuristic (which is the case for many scheduling objectives), the approach provides good solutions fast with subsequent anytime improvement properties if decision time permits. The procedure also provides the flexibility to incorporate a range of different search heuristics (e.g., as a function of the perceived drivers of task cost).

We are currently in the process of completing the major components of the planning layer and beginning testing of the characterize-region task of our Mars exploration scenario, as well as examining market/scheduler parameters that might affect performance. We have demonstrated all layers of our architecture functioning in a limited test of the characterize-region task. In this test, the operator trader is given several rocks to auction among the rovers. The auctioning takes place, the rovers appropriately placing bids, scheduling the awarded tasks, and executing them through the executive and behavioral layers. The major component that is currently lacking, though approaching completion, is the facility allowing the robot traders to auction tasks among themselves, rather than only having the operator trader auctioning to the rovers.

In addition to these enhancements, there are a number of other issues that will require attention. One of these relates to the timing between the various processes of the system. Currently, each agent consists of four processes (behavioral layer, executive, trader, and scheduler) in addition to the single simulator process. In order for planning, task coordination, and behavior execution to work properly, all processes must function using the same rate for time. Unfortunately, this is complicated by the fact that the flow of time in the simulator may be sped up or slowed down on purpose or due to CPU load. Since the simulator provides the grounding for the other layers, timing mismatches can easily break components such as low-level controller code in the behavioral layer. As a solution, we are considering the possibility of adjusting the timing-dependent functionality of each layer to the rate of time in the simulator. While this might help, other factors, such as the latency of IPC messages, would be unaffected and may have exacerbating effects.

The previous sections described the major components of control flow in the architecture. The next section describes a mechanism by which essential capability data flows through the architecture.

## Capability Registration and the ACS

A mechanism for providing up-to-date information on robots and their capabilities is necessary for highly dynamic multi-robot systems, where robot attrition, accession, and modification are common. The Agent Capability Server (ACS) of our architecture is designed to handle this by providing a distributed facility for automatically disseminating agent information. The ACS is not associated with one particular layer, but rather permeates all layers of the architecture. Each robot layer in the system has its own local ACS that it can quickly query for the capabilities and status of its peer layers on other robots. This information can then be used in determining which robots can perform which tasks, planning efficient multi-robot strategies for a task, establishing the appropriate executive-level and behavior-level coordination mechanisms between robots, and accommodating non-responsive (possibly failed) robots. Our ACS is similar in purpose, though specialized by comparison, to middle agents that map capabilities to particular agents, and agent naming services that map agents to locations (Sycara *et al.* 2001).

The ACS participates in a larger capability registration and propagation process that is important to the proper functioning of our architecture. On startup, each layer of a robot registers the capabilities that it can provide (i.e., the behavioral layer register Skill Manager blocks, the executive layer registers TDL tasks, etc.). The capabilities of each layer are reconciled with those of the layer beneath, beginning with the behavioral layer. In our scenario, for example, in order for the planning layer to consider the task of characterizing a rock, the executive must have the proper TDL code, the behavioral layer must have the necessary Skill Manager code, and the "physical" characterization sensor/instrument (e.g., `SensorA`) must exist. The ACS is used to reconcile these requirements and provisions between the layers of the robot.

## Summary and Future Work

This paper described a multi-robot extension to the traditional three-layered architecture, where each layer can communicate directly with its peer layers on other robots. This gives the robots the ability to coordinate at multiple levels of abstraction with minimal overhead in terms of inter- and intra-agent communication. We have described criteria and design decisions for each layer, and have presented the implications of the layer-to-layer interactions on our ongoing Mars exploration scenario.

The architecture is a work in progress, and many refinements are required. These include: integrating the planning and executive layers more fully; generalizing the market-based planning framework, adding more sophisticated bidding structures and cost estimation approaches; having all levels of the architecture deal more fully with the loss of agents; expanding and generalizing capability registration; demonstrating the architecture in rich domains, including space exploration.

Multi-robot coordination promises huge benefits in terms of increased capability and reliability. The price, however, is often added complexity. We believe that a well-structured, flexible architecture will facilitate the development of such systems, at reasonable cost.

### Acknowledgments

This work has been supported by several grants, including NASA NCC2-1243, NASA NAG9-1226, DARPA N66001-99-1-892, and DARPA DAAE07-98-C-L032. Thanks go to Jeff Schneider and Drew Bagnell for their comments and advice on the architecture. David Apfelbaum implemented and helped design TDL. Stuart Anderson implemented the simulator. Thanks also to David Hershberger and Robert Zlot.

## References

Arkin, R. 1992. Cooperation without communication: Multiagent schema-based robot navigation. *Journal of Robotic Systems* 9(3):351–364.

Balch, T., and Arkin, R. C. 1994. Communication in reactive multiagent robotic systems. *Autonomous Robots* 1(1):27–52.

Bonasso, R.; Kortenkamp, D.; Miller, D.; and Slack, M. 1997. Experiences with an architecture for intelligent, reactive agents. *Journal of Artificial Intelligence Research* 9(1).

Cicirello, V. A., and Smith, S. F. 2001. Randomizing dispatch scheduling policies. In *Using Uncertainty Within Computation: Papers from the 2001 AAAI Fall Symposium*, 30–37. AAAI Press.

Cicirello, V. A., and Smith, S. F. 2002. Amplification of search performance through randomization of heuristics. Submitted to the Eighth International Conference on Principles and Practice of Constraint Programming.

Dias, M. B., and Stentz, A. 2001. A market approach to multirobot coordination. Technical Report CMU-RI-TR-01-26, Robotics Institute, Carnegie Mellon University.

Gerkey, B. P., and Matarić, M. J. 2001. Sold!: Market methods for multi-robot control. In *IEEE Transactions on Robotics and Automation Special Issue on Multi-Robot Systems*.

Golfarelli, M.; Maio, D.; and Rizzi, S. 1997. A task-swap negotiation protocol based on the contract net paradigm. Technical Report CSITE, 005-97, University of Bologna.

Jennings, J., and Kirkwood-Watts, C. 1998. Distributed mobile robotics by the method of dynamic teams. In *Proc. Conference on Distributed Autonomous Robot Systems*.

Khatib, O. 1995. Force strategies for cooperative tasks in multiple mobile manipulation systems. In *Proc. International Symposium of Robotics Research*.

Mataric, M. 1992. Distributed approaches to behavior control. In *Proc. SPIE Sensor Fusion V*, 373–382.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. 1998. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence* 103(1–2):5–48.

Musliner, D. J., and Krebsbach, K. D. 2001. Multi-agent mission coordination via negotiation. In *Working Notes of the AAAI Fall Symposium on Negotiation Methods for Autonomous Cooperative Systems*.

Parker, L. 1998. Alliance: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation* 14(2):220–240.

Pirjanian, P.; Huntsberger, T.; and Barrett, A. 2001. Representation and execution of plan sequences for distributed multi-agent systems. In *Proc. International Conference on Intelligent Robots and Systems.*

Sandholm, T., and Lesser, V. 1995. Issues in automated negotiation and electronic commerce: Extending the contract net framework. In *Proc. International Conference on Multiagent Systems*, 328–335.

Simmons, R., and Apfelbaum, D. 1998. A task description language for robot control. In *Proc. International Conference on Intelligent Robots and Systems.*

Simmons, R., and Whelan, G. 1997. Visualization tools for validating software of autonomous spacecraft. In *Proc. of i-SAIRAS.* See also http://www.cs.cmu.edu/ IPC.

Simmons, R.; Goodwin, R.; Haigh, K.; Koenig, S.; and O'Sullivan, J. 1997. A layered architecture for office delivery robots. In *Proc. First International Conference on Autonomous Agents.*

Simmons, R.; Apfelbaum, D.; Fox, D.; Goldman, R.; Haigh, K. Z.; Musliner, D.; Pelican, M.; and Thrun, S. 2000a. Coordinated deployment of multiple, heterogeneous robots. In *Proc. International Conference on Intelligent Robots and Systems.*

Simmons, R.; Singh, S.; Hershberger, D.; Ramos, J.; and Smith, T. 2000b. First results in the coordination of heterogeneous robots for large-scale assembly. In *Proc. International Symposium on Experimental Robotics.*

Simmons, R. 1994. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation* 10(1):34–43.

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

Stentz, A., and Dias, M. B. 1999. A free market architecture for coordinating multiple robots. Technical Report CMU-RI-TR-99-42, Robotics Institute, Carnegie Mellon University.

Sycara, K., and Zeng, D. 1996. Coordination of multiple intelligent software agents. *International Journal of Cooperative Information Systems* 5(2–3).

Sycara, K.; Paolucci, M.; van Velsen, M.; and Giampapa, J. 2001. The retsina mas infrastructure. Technical Report CMU-RI-TR-01-05, Robotics Institute, Carnegie Mellon University.

Thayer, S.; Digney, B.; Dias, M. B.; Stentz, A.; Nabbe, B.; and Hebert, M. 2000. Distributed robotic mapping of extreme environments. In *Proceedings of SPIE: Mobile Robots XV and Telemanipulator and Telepresence Technologies VII.*

Wellman, M., and Wurman, P. 1998. Market-aware agents for a multiagent world. *Robotics and Autonomous Systems* 115–125.

Zlot, R.; Stentz, A.; Dias, M. B.; and Thayer, S. 2002. Multi-robot exploration controlled by a market economy. In *Proc. International Conference on Robotics and Automation.*