AN OPEN ARCHITECTURE AND MIDDLEWARE FOR COLLECTIVE ROBOT TEAMS

Micah Lesmeister (student), Theodore Elhourani (student)
Michael Marefat (faculty advisor), and John Reagan (faculty advisor)
Department of Electrical and Computer Engineering
The University of Arizona, Tucson AZ 85721
Email contact:{marefat,reagan}@ece.arizona.edu

ABSTRACT

In this paper we propose an open multi-robot architecture that dramatically reduces the time to deployment and increases the utility value to the mainstream non-technical user. We describe a multi-robot behavior-based coordination architecture and argue its suitability in the context of general-purpose robot teams operating in dynamic and unpredictable environments. We then formalize and describe a command fusion module for the coordination of high-level behaviors of the system. The command fusion module is interfaced to our middle-ware/compiler that generates behavior selection tips from a user specified abstract description of a scenario. Finally, we utilize an example search and rescue scenario to illustrate the overall process and give preliminary results of the experiments performed on actual robots.

KEYWORDS

Multirobot Systems, Behavior-Based Control, High-Level Language Decomposition, Behavior Coordination Mechanisms.

INTRODUCTION

Robotic systems will eventually replace humans in hazardous areas to perform tasks such as search and rescue, fire-fighting, de-mining, space exploration, etc. Robots have previously been used to perform relatively simple and repetitive tasks in manufacturing plants and warehouses. However, robot control in unstructured and unpredictable environments like the rough terrain of the surface of a planet or a minefield has proven to be hard and complex. The advantages of a robot team over a single robot have frequently been addressed in various research papers [4,7,8]. Multi-robot systems are inherently robust. In the event of a failure of one of the robots other team members can pursue and complete the mission. Moreover, robots sensing and acting in parallel usually accomplish tasks more efficiently, which is a major requirement in time critical missions. In addition, a single robot with advanced capabilities is usually very expensive to build. In a poorly represented environment various robotic capabilities may be required (sensing and actuation). Evenly distributing these capabilities on relatively simpler robots will certainly

lower the costs. On the other hand, multi-robot systems are significantly harder to develop, implement and test. Coordination between the robots in matters of task allocation and task execution is required and it adds to the complexity of such systems. Three main multi-robot control architectures have previously been considered. Fully decentralized reactive architectures avoid all kinds of explicit communication between robots. A robot does not need to coordinate its actions with other robots; nevertheless it can implicitly (through sensors) learn about the state of other team members and then minimally try to avoid conflicts. Such architectures are typically used for tasks where actions need only be loosely coordinated. On the other extreme, in deliberative architectures, all the major processing (planning, action synchronization, etc) is undertaken in a central location. Robots in the team transmit all relevant information to the leader robot. The lead robot then does all the required planning and retransmits subplans respectively to team members. In addition, the leader is required to synchronize the execution of subplans and to resolve conflicts that may emerge during execution. Obviously, such architectures suffer single point failures. Hybrid architectures lie in the middle between fully decentralized reactive architectures and deliberative architectures. Such systems make use of coordination protocols to achieve coherence among team members. The robots explicitly communicate among each other in order to resolve conflicts. In this paper we propose a hybrid behavior-based architecture, which allows smooth integration with our high-level description language and its associated compiler/middleware.

Typically, teams of robots require vast amounts of time to design and implement. In this paper we propose a high-level description language that seeks to dramatically reduce this development time. In addition to reduced implementation time, the high-level language has other benefits: individuals with limited technical knowledge can now easily deploy the robots, and the high-level descriptions allow users to build upon previously generated descriptions. High-level language commands are broken down into high-level behaviors using a decomposition process. Previous work on high level behaviors has, for the most part, been limited to a bottom up framework, where the user specifies low level commands that when grouped together form some new high-level behavior. Our proposed solution, referred to as the High Level Robot Team Command Language, is to do the contrary and command at the high-level and let the robot system decide which behaviors it needs to execute the task. A major problem in behavior-based systems is action selection, which is also referred to as behavior coordination. Given a set of *behaviors*: Avoid Obstacle, Push Object, Find Object, and a set of *goals*, a behavior coordination mechanism is supposed to resolve conflicts between behaviors and to schedule their execution. Consequently, the conflict resolution and scheduling will result in an emergent overall strategy that will direct the robot to reach its goal(s). We divide the behavior coordination task into two main stages. In the first stage our compiler/middleware recognizes the major goals of the mission and selects the appropriate behaviors from a precompiled repertoire of behaviors. In the second stage, during execution of the mission, a command fusion behavior coordination mechanism assumes control of the behavior selection and coordination process.

In the rest of this paper we start by describing previous work in the areas of behavior-based multirobot control, behavior selection and high-level language decomposition. Then we introduce our behavior-based architecture and formally define the two-stage behavior coordination/selection mechanism. In the following sections we provide an overview of the high-

level description language syntax language and the decomposition process. Then we describe our experimental setting and give preliminary results followed by a brief conclusion.

PREVIOUS WORK

Many behavior-based multirobot systems have been designed and implemented over the last decade. In completely decentralized systems the robots act largely independently and try to implicitly communicate through the use of their sensors. Two illustrative examples of such systems are [8] and [7]. In [8] a behavior-based multirobot system utilizing *motor schemas* [3] is developed. The goal of the robots team is to navigate a previously unknown terrain while avoiding obstacles and maintaining a formation. Parker [7] proposes ALLIANCE, a multirobot control system based on the *subsumption architecture*. The main goal of ALLIANCE is the design of a fault-tolerant and adaptive system. The developed implicit cooperation techniques are tested in a hazardous waste cleanup scenario. Both systems assume missions composed of loosely coupled subtasks, where explicit negotiation and synchronization between robots is not required. In a more recent related research [4] suggests a behavior-based multirobot architecture, which addresses the problem of tight coordination for the task of cooperative transport of extended objects over a rough planetary terrain. In order to support tight coordination CAMPOUT [4] implements the concept of distributed coordinating behaviors. All three examples implement a restrictive set of behaviors which makes the system highly domain dependant. We do not claim that a completely domain independent autonomous system is currently achievable, however we believe that in order to make such systems useful for the mainstream audience a reasonable set of scenario achieving behaviors needs to be available to the user.

A major concern in behavior-based systems is behavior coordination mechanisms (*BCM*). This problem has been extensively studied in virtually all related previous research. Pirjanian [9] presents a thorough assessment of the major approaches for solving this problem. He places all the previous approaches into two major categories: *Arbitration BCMs* [5,7] and *Command Fusion BCMs* [8]. Arbitration is better suited to multiobjective systems where the goals change over time. The arbitration module switches between behaviors according to the current goal of the mission. ALLIANCE illustrates the use of such a paradigm. Parker devises an arbitration mechanism for behavior selection based on the *impatience* and *acquiescence* motivational behaviors. The motivational behaviors modules provide a rough measurement of the applicability of a given behavior at a given moment according to available sensor data and the internal state of the robot. The most applicable behavior is then activated. In contrast, command fusion mechanisms combine the incoming data from various behaviors and sensors and then select the most applicable one or just linearly superpose the actions of all behaviors. This method is usually employed in systems where all the resources are continuously allocated to only one task that requires very high attention (e.g. tightly coupled coordination). An example is the *motor schema* architecture, which was used in [8] where the only goal is to navigate a terrain while avoiding obstacles and maintaining a formation. CAMPOUT [4] is the most closely related to our work in that it employs both arbitration and command fusion based *BCMs*.

Previous work in controlling a team of robots through the use of some high-level command interface is relatively new. One such example is the Mission Lab project at the Georgia Institute of Technology, whose theoretical architecture is covered in [1]. The Mission Lab software suite includes programs to handle a variety of robot deployment tasks. Our proposed robot system will have similar interfaces with the user: robot simulator, high-level behavior editor, system status monitor and a compilation program for generating low level behaviors. However, our high-level behavioral generator environment will be completely unique in our implementation. The Mission Lab implementation makes use of a graphical tool for building complex FSA structures for team state transitions. A simple text interface, much like a simple word processor, will be used in our case with the list of available high level commands listed in a sidebar for easy dragging and dropping of commands into the editor environment. Previous work regarding capturing higher-level behaviors has been discussed thoroughly in Ishida's paper [12]. Ishida uses a scenario description language called Q to describe interactions between software agents and the external users. Q has been applied to several scenarios including a web user interface and disaster simulations. Ishida's research regarding agent interaction is important background information because of the need of our robots to be able to effectively pass information between each other. This is especially important in scenarios where the robots must coordinate with each other to collectively move objects in a tightly coupled fashion like in applications discussed in [4].

THE MULTI-ROBOT CONTROL ARCHITECTURE

As our testbed we consider a *search and rescue* scenario reduced to the simple task of collaboratively finding an object and then pushing that object to a given location. We define two classes of behaviors: *Single Robot Behaviors* and *Multi-robot Behaviors*. For instance, in the search and rescue scenario, the Task Allocation behavior is a collaborative behavior. Robots need to collaborate in order to find a reasonable task allocation that takes into consideration the current state of each of the robots in a given environment. As figure 1 shows Collaborative Pushing is another collaborative behavior. Again, robots need to coordinate their actions while collaboratively transporting an extended object. Others are Single Robot behaviors, like the atomic behaviors: move forward, turn, etc. In addition some composed behaviors are also considered single robot behaviors, for instance Push Object, Avoid Obstacle. We also classify the behaviors into three categories according to the abstraction level. First, high-level behaviors consist of composite behaviors such as Task Allocation, Search Location, Collaborative Pushing (see fig. 1). Intermediate level behaviors, which are also composed of lower level behaviors (in Figure 1; Communication, Systematic Area Scan, Obstacle Avoidance, Push Object), and finally the primitive behaviors layer consisting of simple actions such as "turn", "forwards", "backwards".

All the atomic behaviors are modeled as *motor schemas*. Motor schemas are associated with perception schemas. In the case of collaborative behaviors messages received from other robots implement the perception schemas. While single robot behaviors employ vision feedback as their perception schemas. Performing tightly coupled tasks requires a close spatial and temporal coordination of behaviors of different robots. In [4] this problem was approached through the introduction of collaborative compliant control. Due to real-time constraints the compliant

control was designed such that no time consuming explicit negotiation takes place between robots. Instead robots try to coordinate implicitly through the use of sensors. We allow the robots to explicitly communicate with each other to achieve coherence. The collaborative behaviors of different robots in the team communicate with each other. As mentioned before these behaviors are treated as motor schemas and the communication taking place between them represents the corresponding perception schemas.

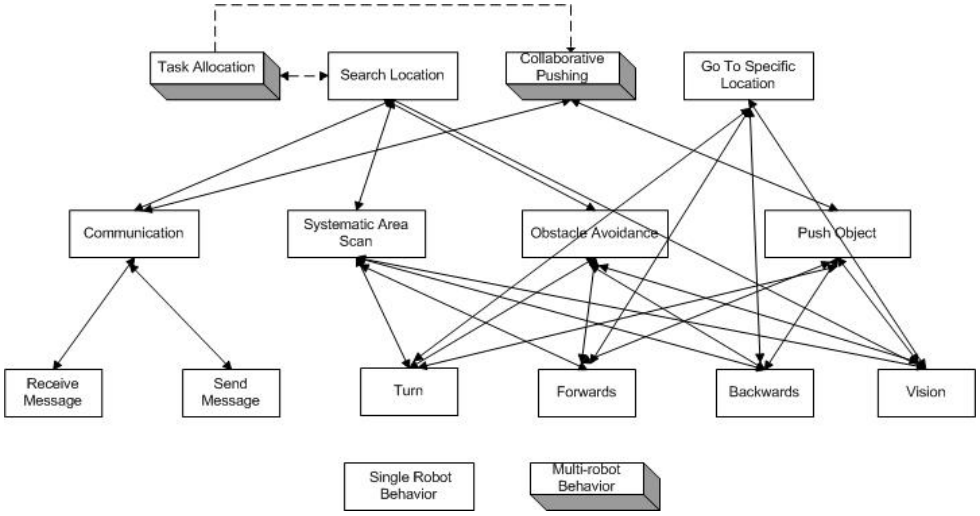

Fig.1. Required Behaviors For The Search and Rescue Scenario, arrows represent selection of actions

BEHAVIOR SELECTION AND BEHAVIOR COORDINATION

Given a repertoire of behaviors (Fig.1.) the next task is to decompose the high-level language and to proceed with the action selection mechanism. The decomposition of the high-level language results in a set S of prioritized atomic behaviors subsets $b \subseteq B$, where B is the complete repertoire of behaviors. Each set of atomic behaviors represents a subgoal of the mission. A *Behavior Selection* module schedules the execution of the sets in S in an order consistent with the priorities (the order of execution will be specified in the following sections).

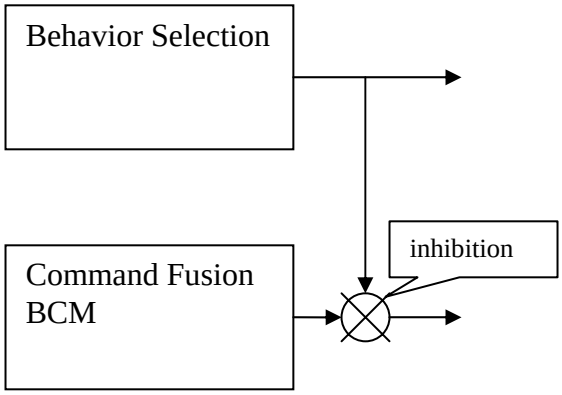

Fig.2. Behavior Selection Module inhibiting the Command Fusion BCM

Scheduling refers to the activation of the individual behaviors belonging to subsets b ($b \in S$). After activation of the behaviors of a subset b the control is passed to the *Command Fusion* module. The command fusion module effectuates a superposition (linear combination) of the outputs of behaviors (motor schemas) belonging to the same control layer. The result of the superposition of the outputs of these behaviors is then fed to the next layer of behaviors. Figure 3 shows the different stages of control. For more details about the *motor schemas* superposition mechanism consult [3].

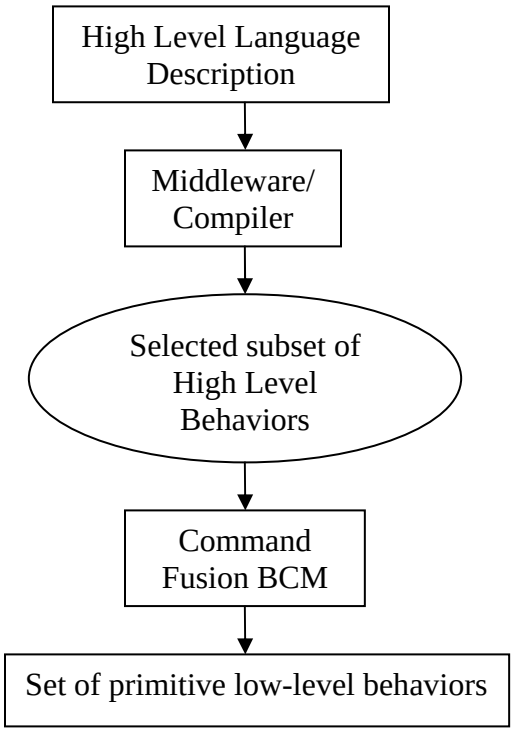

Fig. 3. System Overview

HIGH LEVEL LANGUAGE SYNTAX

The High Level Robot Team Command Language syntax is inspired by a test equipment programming language, Standard Commands for Programmable Instrumentation (SCPI). A complete description of the SCPI standards can be found in [11] (note: the syntax form was used, but the standards are not adhered to – they are provided for background information on SCPI only). Programming commands are in an English-like form allowing even the most novice user the ability to implement complex programs. Command parameters are passed to a particular function using a (:) – each parameter provides more information used in the decomposition process to further define the necessary behavior set. An example of a command is the find command. The find command's specification, find:length:width:height:color, is a relative example of a typical high-level command. This find command specifies to the robots to collaboratively look for an object of the given length, width, height and color. In most cases some or all of the description parameters might not be known, in this case unknown variables are simply skipped.

The use of a high-level language is apparent – now users can control the team of robots as a whole instead of each robot individually. In addition to the ease of programming, the command structure allows for maximum portability. The same commands can now be used across any robot platform as long as the platform's creator provides support for the command. Consider an example where a team of robots is asked to search a space for a cube with the dimensions of .25 meters and colored red. Once the object is found the robots are then to move the object (collectively if the object is too heavy to be moved by one robot) and bring it back to the point the robots started from. Using the high level language to define the procedure the user instructs the team using the following commands:

```
find:.25:.25:.25:red=>cube    (1)
move:cube:0:0                 (2)
```

Another syntax feature shown above is the naming mechanism. The naming mechanism is used for capturing the results of a particular action, which can be used in either another command such as the case in (2) above or to repeat the same command later in the program without having to retype the command. This device is especially useful if the robots are to interact with the same object on multiple occasions or if there are multiple objects with similar qualities.

DECOMPOSITION PROCESS

Two important pieces of information are generated through the compilation of high-level language commands into the mid-level behaviors – the behavior list and the behavior ordering. Decomposing a robot system is very similar to the compilation process that all computer programmers are used to – the high level language such as C is broken down until it reaches a machine level, which is then understood by the processor. In our robot compilation the high level commands are broken down further and further until we arrive at the lowest levels of behaviors – the basic units that the robots can use to execute in their world. Typical compilations are broken down into four categories: tokenizing, parsing, code generation and optimizing. The optimizing problem is not the focus of our research and is left as future work. The first step tokenizing the high-level syntax points are recognized such as the colons and the commands themselves, which are combined into usable statements in the parsing step. The final step in compilation is the code generation, which in our case is the generation of the behavior set.

The execution of the high-level behaviors is managed through a decomposition process. Decomposition takes the high-level commands and uses this set to decide which lower level behaviors to use and the role each behavior will take in the system. This process is determined and generated without any user intervention. The formalization can be realized with the details below. Where C represents the collection of possible high-level commands that can entered by the user, B is the set of mid level behaviors known to exist on each robot and B_x represents the minimum set of behaviors that the decomposition process decides is needed for any given command x:

$$C = \langle c_1, c_2, \ldots, c_n \rangle$$
$$B = \langle b_1, b_2, \ldots, b_n \rangle$$

$B_x \subseteq B$, where x ranges from 1 to the number of commands entered by the user

The process of decomposition has several stages of computation starting with the user describing their scenario using the high-level commands. Using the high-level commands the scenario is then broken down into the mid and low-level behaviors that the robot then uses to operate in its environment.

Timing at the top level is derived from the order by which the user enters the commands, where the order of the commands determines the order of execution of the behavior set. Most temporal issues are handled at the lower levels where timing is vital for successful robot operation. The high-level timing is simply formalized where B_x represents the minimum set of behavior for a command x, as defined above and O, which represents the ordering at the high-level. Mathematically expressed it is defined as:

$O = <B_1, B_2, \ldots, B_n>$, for x = 1 to n-the number of commands entered by the user

In other words, a behavior set B_x is executed in the order in which the command was entered into the system. A change in the behavior set currently being executed is triggered by the successful completion of a particular command.

The decomposition procedure is not limited to just making decisions on what behaviors are to be used – it must also make decisions on the time sequencing of the behaviors and the amount of collaboration that goes on between the behaviors. With dynamic environments it is extremely necessary to have some sort of behavior conflict resolution. The importance of which is demonstrated with the following example. Consider two behaviors from Figure 1 – Obstacle avoidance and Search location and while the search location behavior is being executed, the obstacle avoidance behavior detects an object in its immediate path. In this case the search location behavior, if allowed to continue, would cause a collision with that object in its path. The behaviors need a mechanism for a behavior to communicate to another behavior that its decision is incorrect. This overruling is handled during the decomposition process using a priority scheme. A value in the range of zero to three is assigned to the behavior with the higher valued behaviors the ability to override any decisions a lower level behavior makes. The priority value is not static, however, consider our search and rescue example from above. The object avoidance behavior will change from a value of three in equation (1) to a lower priority in equation (2).

Decomposition is best described through an example – consider a simple search and rescue scenario where the robots are to search an area in a distributive asynchronous manner and find a red cube with .25 meters dimensions and finally move that cube to the location the robots started from denoted as the point (0,0). At the high level, the commands would simply be the same as the find and move commands mentioned above. These commands would produce a minimum set of mid level behaviors similar to the set seen in Figure 1. Also shown in the figure is the composition properties of the mid level commands – each mid level command encompasses several low level behaviors such as sensor input and motor output that collectively work together to complete the specified behavior.

HARDWARE TESTBED AND PRELIMINARY RESULTS

The variety and number of sensors available for connection directly to a pc was the primary reason for using off the shelf pc components to provide the onboard computing power on each one of the robots in the team. The current configuration for the developed team of robots uses a desktop motherboard powered by a DC-to-DC power supply through two sealed lead acid batteries for each individual robot. Inter-robot communication is managed through the use of wireless Ethernet adaptors utilizing the 802.11b protocol. Low-level motor control and odometry is implemented using a commercially available microcontroller that communicates with the robot via RS232. The future configuration's aim is to reduce the dimensions and weight of the previous testbed. Laptop computers with built in power supply will now be the source for the onboard computing power. 802.11b will again be used via built in wireless Ethernet provided by the laptop.

Our initial experiments using the testbed described above have been limited to a small world that contains very few objects. Preliminarily we have limited the size of the team of robots to only two robots and limited the area in which they have to operate in. These few scenarios provide one of the basic building blocks that will be used to test more complex high-level behaviors. In the example that has a starting position seen in Figure 2 – the robots are instructed to move a yellow box from the location it finds it at to a location between the two chairs seen in Figure 4. The image of significant interest however is Figure 3, which shows the midpoint of the scenario's completion. This image is taken a few moments after the first robot has pushed the box a certain distance between the two chairs, it then signals the other robot to push the box the rest of the way into the goal location and moves itself out of the way.

This simple example demonstrates all aspects of the proposed system. Using a slightly modified set of commands from above:

```
find::::yellow=>box    (3)
move:box:-.5:.5        (4)
```

the scenario can be described and executed with the correct set of behaviors generated in the correct sequence.

Figure 4 – Box Pushing Start Point

Figure 5 – Box Pushing Mid Point

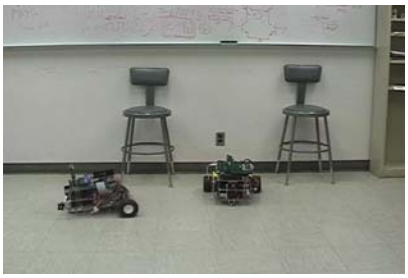

Figure 6 – Box Pushing End Point

CONCLUSION AND FUTURE WORK

The contributions of this work include general high-level command language for a team of robots along with a specially tailored behavior-based architecture with its associated action selection mechanisms. As a result the development costs and time to deployment for complex tasks would be dramatically reduced. Experimentation will be expanded upon to include more robots executing more complex tasks in a world that contains many objects that the robots will have to decipher the role, if any, the object has. Unlike the example experiment above, the future test will be performed in dynamic environments and situations where the robots must look for and avoid objects.

REFERENCES

[1] D. Mackenzie, R. Arkin, and J. Cameron, "Multiagent mission specification and execution," Autonomous Robots, 4(1), 1997: 29-57.

[2] Y. Nakauchi and R. Simmons, "A Social Robot that Stands in Line," Proceedings of the International Conference on Intelligent Robots and Systems, Vol.12, No.3, 2002: 313-324.

[3] R. C. Arkin, Behavior-Based Robotics, Cambridge, MA: MIT Press, 1998.

[4] T. L. Huntsberger, P. Pirjanian, A. Trebi-Ollennu, H. Nayar, H. Aghazarian, A. Ganino, M. Garrett, S. S. Joshi and P. S. Schenker, "CAMPOUT: a control architecture for tightly coupled coordination of multirobot systems for planetary surface exploration," IEEE Transactions on Systems, Man, and Cybernetics, Part A 33(5), 2003: 550-559.

[5] T. L. Huntsberger, H. Aghazarian, E. Baumgartner and P. S. Schenker, "Behavior-based control systems for planetary autonomous robot outposts," Proceedings IEEE Aerospace, 2000: 679-686.

[6] R. Simmons, S. Singh, D. Hershberger, J. Ramos, and T. Smith, "First Results in the Coordination of Heterogeneous Robots for Large-Scale Assembly," Proc. of the ISER '00 Seventh International Symposium on Experimental Robotics, December, 2000.

[7] L. E. Parker, “ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation,” IEEE Transactions on Robotics and Automation, Vol. 14, No. 2, April 1998: 220-24.

[8] T. Balch and R. C. Arkin, “Behavior-Based Formation Control for Multirobot Teams,” IEEE Transactions on Robotics and Automation, Vol. 14, No. 6, December 1998: 926-939 .

[9] P. Pirjanian, "Behavior Coordination Mechanisms - State-of-the-art", Tech-report IRIS-99-375, Institute for Robotics and Intelligent Systems, School of Engineering, University of Southern California, October, 1999.

[10] P. Pirjanian and M. J. Mataric´, "Multi-Robot Target Acquisition Using Multiple Objective Behavior Coordination", Proceedings, International Conference on Robotics and Automation (ICRA 2000), Apr 25-28, 2000: 2696-2702.

[11] SCPI Consortium, “Standard Commands for Programmable Instruments,” Version 1999.0, Volumes 1-4, May 1999.

[12] Toru Ishida, “Q: A Scenario Description Language for Interactive Agents”, *IEEE Computer*, Vol.35, No.10, pp.54-59, 2002.