In the Proceedings of the 1996 IEEE Symposium on Autonomous Underwater Vehicle Technology.

Organization and Reorganization of Autonomous Oceanographic Sampling Networks*

Roy M. Turner
Department of Computer Science
University of Maine
Orono, ME 04469 USA
rmt@umcs.maine.edu

Elise H. Turner
Department of Computer Science
University of Maine
Orono, ME 04469 USA
eht@umcs.maine.edu

D. Richard Blidberg
Marine Systems Engineering Laboratory
The Autonomous Undersea Systems Institute
86 Old Concord Turnpike
Lee, NH 03824 USA
drb@bluefin.net

Abstract

Systems such as autonomous oceanographic sampling networks (AOSNs) that have multiple autonomous or semi-autonomous components must have an organization which specifies the interactions between the components to allow them to distribute and accomplish the system's tasks. AOSNs present a special challenge. They will be deployed for long periods of time, and they are *open systems* whose composition will change over time. Such systems require the ability to autonomously organize and reorganize in response to changes in its composition, the environment, or the mission.

In this paper, we present preliminary results from a project whose goal is to develop mechanisms to allow AOSNs to self-organize and reorganize. We discuss characteristics of AOSNs which impact their organization and give an overview of an approach which addresses their special requirements. We discuss a simulation methodology designed to simulate the aggregate properties of the protocols developed, and we describe preliminary results obtained using that simulator.

*This work was funded in part by contract N0001–14–96–1–5009 from the Office of Naval Research and grant number NSF BES-9222146 from the National Science Foundation. We thank Steve Chappell for helpful comments on an earlier draft of this paper. © 1996 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

I Introduction

In the near future, networks of vehicles and non-mobile instrument platforms (VIPs) will be deployed for extended periods to perform a variety of tasks. Autonomous ocean sampling networks (AOSNs) [Curtin *et al.*, 1993] are examples of such networks. To take full advantage of the variety of VIPs available, AOSNs should be developed that are not constrained to use only specific types of component VIPs. To allow the networks to work efficiently for long periods, AOSNs must allow VIPs to enter the system as they become available and leave the system when they fail or are needed elsewhere; that is, AOSNs are *open systems* [Hewitt, 1986]. In addition, AOSNs must be able to take on a variety of tasks while they are deployed. The system may be given several tasks to complete at the time of its deployment, and the current task may change or be interrupted by a higher priority task. A new task may also be added during the mission by a human operator or in response to unanticipated features of the environment.

This paper describes a project aimed at developing intelligent control mechanisms for AOSNs and other multi-component systems. Our approach is to test design decisions in a simulator which demonstrates the aggregate behavior of the design. This allows us to evaluate the overall design before incurring the expense of a complete implementation in a more traditional simulator or on deployed vehicles. In this paper, we present preliminary results for our project. In the following section, we discuss some properties of the AOSN control task. Section III describes our approach to the problem. Section IV discusses our simulation method and Section V describes our initial simulation of protocols for organization and reorganization of AOSNs. Finally, the lessons learned and future directions are presented in Section VI.

II Organization and AOSNs

One important aspect of intelligent control for multi-component systems is providing the *organization* which allows the components to work together to complete their task. The organization specifies which components control which others, the pathways for communication, and the information which should be communicated. There are organizational structures, or types, such as hierarchies, that specify the relationships in a general way. However, each assignment of components to roles within an organizational structure should be seen as a distinct organization. *Protocols* can be specified for a particular type of organization to describe what information should be communicated and how agents (i.e., VIPs) should act in specific situations. In addition to specifying the details of the organization's structure, we must also specify how to achieve the mission goals within that organization.

Long-duration, flexible systems such as AOSNs have some special requirements for organizations that have not been addressed in previous systems:

The system must be able to organize and reorganize itself autonomously. As VIPs enter or leave the system, or the task changes, the existing organization may no longer be effective. To allow the system to work without human intervention, the system itself must be able to recognize the need for change and to create a new organization.

It is also advantageous for the system to be able to create its initial organization autonomously. Because each deployment of an AOSN may have different tasks and component VIPs, forming the initial organization is a non-trivial task requiring knowledge about which organizational structures are best for the current situation, which roles in the organization require which capabilities, and which VIPs have these capabilities. In order to re-organize, the AOSN must have this information and the ability to use it. A human operator, for example, an ocean scientist, should not have to have this expertise. By allowing the AOSN to create its own initial organization, the human operator is relieved of this responsibility.

A single organization may not be equally useful for both organization and mission-related tasks. During organization/reorganization, the AOSN should not necessarily deal with the detailed knowledge of subtasks necessary for accomplishing the mission, but rather with the overall properties of agents and how best to assign them to tasks; communication channels should allow agents to exchange information with any other agent that might have strategic knowledge or abilities important to setting up an organization to accomplish the mission. During actual work on the mission, overall strategic knowledge may be less important than knowledge about how that task should be performed; communication should be more local and should encourage focused information exchange about tasks that are underway.

Some, but not all, of the VIPs have the ability to reason independently. Organization of multiagent systems has been addressed in the distributed artificial intelligence (DAI) literature [e.g., Fox, 1981; Smith, 1980; Durfee & Lesser, 1987]. However, most DAI approaches explicitly or tacitly assume that all of the agents involved have the same level of cognitive capabilities: either they are all full-fledged problem-solving systems, or they are all simpler systems. In contrast, an AOSN is a heterogeneous system in which some VIPs are "intelligent", some less so, and some are little more than simple instruments with some communication mechanism attached.

The distinction is important for two reasons. First, the ability to reason is important for performing many tasks in many different organizations. Specifically, it is needed for an agent to reason about the organization of the system. Second, new kinds of organizational structures, or new takes on existing structures, will have to be developed if both types of agents are to be integrated into a single organization. For example, we may form the non-reasoning VIPs into distributed systems that function as a single VIP within the broader organization. Alternatively, we may create only organizations which ensure that each non-reasoning VIP is directed by a VIP which can reason.

III Meta-Level and Task-Level Organizations

We take a two-level approach that tries to match the organizational style to the task at hand. The actual work performed in accomplishing a mission is conducted in the context of a *task-level organization*, or TLO. Problem solving aimed at creating this organization or reorganizing it when there is a change is conducted in a different organization, the *meta-level organization*, or MLO. By explicitly splitting the work needed for organization/reorganization of the AOSN from that needed to accomplish its mission, the AOSN is free to use whichever organizational style is most appropriate for each type of work.

Meta-level organization. We make a minimally constraining assumption: the AOSN is deployed as an unorganized collection of VIPs. Some within this group will have the ability to participate in an MLO, some will not. Those that can (called the MLO-agents) will self-organize to form an MLO. Self-organization occurs based on simple protocols designed to allow MLO-capable agents to discover each other's existence, as discussed in more detail below.

Once formed, the MLO's job is to devise an appropriate TLO for the current task. This involves two major tasks: discovery of resources and matching those to the demands of the task to create the TLO. The first of these involves identifying the capabilities of all agents in the AOSN. Some knowledge may already exist about some VIPs. For others, messages will have to be broadcast to first discover their existence.

Armed with the knowledge about the AOSN, the MLO can then decide on an appropriate TLO. There are many ways this can be done: a single agent can act as planner, the entire set can collaboratively create the TLO, etc. Below, we discuss one way we have simulated so far. There are also many task-allocation methods based on the capabilities discovered by the MLO: first-fit of VIP to task, best-fit, minimizing slack resources, etc. As discussed below, we have implemented a simple first-fit approach to assign VIPs to tasks.

When the TLO has been designed, messages are sent assigning roles to VIPs. The MLO is then dissolved and work begins on the mission by the TLO.

Task-level organization. There are myriad possible task-level organizations. We are primarily interested in organization and reorganization of AOSNs at this stage of our work. Consequently, we have chosen to use simple hierarchical TLOs.

TLOs in our approach are hierarchies of two or more layers. Roles within the hierarchy are assigned based on VIP capabilities, including (e.g.) the ability to function as a manager. Ultimately, we will experiment with allowing peers to communicate with one another while working on tasks for information exchange and to utilize slack resources; this should reduce message traffic while increasing overall TLO performance.

We recognize that a hierarchical organization is prone to particular kinds of failures, including those related to weak links, cognitive overload, and other single-points of failure. A conjecture to be tested by our research is that such failures can be overcome in a two-layer approach such as ours by the reformation of the MLO when there is a failure.

Reorganization. Changes in the situation can often be handled without disrupting the TLO. For example, a failed VIP—or one that must leave the system for some other reason—might be noticed by its manager, which would reassign its tasks to other VIPs under its control with sufficient slack resources. Sharing some information across levels, such as a manager knowing who its controlled VIPs control, can even protect against some failures of managers. New VIPs can sometimes be handled similarly, with existing managers either assigning the VIP to tasks it has or passing the VIP off to its manager for assignment elsewhere. Some changes, however, necessitate changing the TLO organization. For example, should the top-level manager fail, the TLO would essentially be left "headless", with no coordination for the integration of the mission's major tasks. Other changes can also indicate that the TLO in its current form is simply not working, or not working as well as it could.

The first task facing a TLO and its VIPs is recognizing when a reorganization is necessary. Obviously the top-level agent is in a position to notice this; however, it may not be the only one.

Once it has been noticed that something is wrong, the next question is what to do about it. In our approach, the answer is to initiate the reformation of the MLO, which will then reorganize the AOSN to create a new TLO.

Representing capabilities: generic behaviors. In our approach, all VIPs are reasoned about primarily by considering only their capabilities. To an MLO-agent, another agent is simply a collection of capabilities, or behaviors, that can be matched up to the requirements of the mission at hand. This provides us with an abstraction that allows ignoring the physical idiosyncrasies of the particular VIP.

To do this, a representation scheme for behaviors is needed. We make use of work on *generic behaviors*, or GBs, resulting from another of our projects [Turner *et al.*, 1993]. A generic behavior describes a particular behavior of a VIP, including the preconditions for that behavior to occur, the effects of it having occurred, its time course, and resources required. For example, an AUV might have GBs representing "goto location", "dock", "hover", various sensor capabilities, etc. Though the implementation details are not important for this paper, GBs can be implemented as data structures similar to frames.

IV The Simulator

To evaluate our approach, we are developing a simulator that simulates the aggregate properties of the protocols we develop. Systems which implement reasoning capabilities for agents are notoriously large, taking many person-years to program. Waiting until a prototype system is built before evaluating the approach runs the risk of wasting these resources. In addition, these complex systems are often difficult to evaluate. Because of the interactions within the system, it is hard to identify the exact effect of any individual design decision. Because of the cost of programming, it is hard to experiment with alternatives.

Our simulator steps back from the traditional prototype implementation to focus on the aggregate properties of the system. For example, suppose we are evaluating protocols for communication used to set up the initial MLO. In a traditional prototype implementation, we would have to implement the algorithm which selects the information to fill the protocols. In the aggregate simulator, we can simulate the *effects* of the algorithm having run, simply by giving VIPs the information that they should receive from the protocol.

The simulator is a rule-based system implemented in CLIPS [Giarratano, 1993]. CLIPS is a widely-used expert system shell, so, after the simulator has been developed, it can be made available to other researchers. The rule-based system implementation also allows us to use a top-down approach to evaluation. For example, we may begin with a very abstract rule for communication. This rule can then be replaced with rules which specify when and how a VIP makes the decision to communicate. The more detailed communication rules can be replaced again with rules that reflect the possibility of lost messages. In this way, we first test the need for the information that will be communicated, then the method for selecting the information, and next the value of the communication method given real world constraints. Finally, by linking C to CLIPS, we can implement and test the algorithm for communication that will be used by actual VIPs.

V Simulation of Protocols

In this section, we describe the AOSN protocols embodied in an initial version of our simulator and the results we have so far obtained. The protocols we have implemented are at a fairly high level of abstraction, which allows us to get a sense of how our overall approach works and where weaknesses may lie.

The activity of an AOSN falls naturally into several phases, including: (1) formation of the MLO; (2) discovery by the MLO of the capabilities of the AOSN (or updating the MLO's information since the last time it was in existence); (3) decision by the MLO of what TLO to use to achieve the mission; (4) creation of the TLO; and (5) work on the mission by the TLO. In addition, we can think of phases corresponding to handling errors or other changes in the situation or AOSN (e.g., addition or failure of a VIP).

CLIPS, like other forward-chaining rule-based systems, lends itself to structuring by *contexts* in which rules fire, implemented by adding a clause to rule antecedents checking for the appropriate context in which the rules should fire. We have therefore created contexts corresponding to each of the AOSN's phases of operation (`MLO-formation`, `MLO-discovery`, `MLO-decision`, `TLO-formation`, `TLO-work`, and `error`) as well as some (e.g., `MLO-formed`) that are used for housekeeping functions.

After we briefly touch on our representation (e.g., of VIPs), the remainder of this section is structured around these phases/contexts. For each, we discuss the protocols simulated, briefly describe the rules used to simulate them, and show some representative output from the simulator.

Representation. All knowledge in CLIPS is represented as either rules or facts in working memory. Facts can either be predicate calculus-like assertions or instances of "templates", which are frame-like slot-filler structures. VIPs, for instance, are represented as a template that looks like:

```
(vip (name vip1) (caps a c f)
     (location 1 1 1))
```

That is, a VIP named "vip1", with capabilities (generic behaviors) "a", "c", and "f", and that is currently at location (1,1,1).

For ease of generating experiments, capabilities are represented generically as symbols rather than as actual generic behavior names. There is one distinguished behavior, "C", that means the agent is capable of participating in an MLO.[1]

Tasks are also represented as template instances. Tasks can have several alternative ways of being accomplished, each of which requires a particular set of capabilities. VIPs are assigned to tasks based on this information.

Other facts in memory include such things as a representation of a fact an agent knows (e.g., "AUV1 knows VIP1 has capabilities a g f") and various kinds of information needed for the implementation of the protocols as rules.

MLO formation phase. In this phase, the individual MLO-capable agents are concerned with determining if there is an existing AOSN organization and, if not, creating one. One can think of either the entire AOSN or an individual agent being in this phase, the former when there is no existing organization and the latter when there is one, but the agent does not know about it.

The general protocol followed by an agent in this phase is as follows. When an MLO-agent finds that it does not know of the existence of an organization, it broadcasts[2] a "who's there" message tailored to elicit responses from other MLO-agents. (Since non-MLO-agents cannot participate in the MLO, they are not involved in this phase.) If there is an existing TLO, then members hearing the message will respond; the agent will select one of the ones responding and contact it to join the TLO. If there is an existing MLO, then one of the MLO members will

[1] "C" stands for "CDPS-capable", meaning that it is capable of cooperative distributed problem solving—i.e., it is an MLO-agent.

[2] We use the term "broadcast" loosely to mean "tries to get a message to all other agents" of whatever type desired; the implementation will vary depending on the assumptions about the underlying communication network.

```
(SIM: switching context -> MLO-formation)
   [...]
(MLO) AUV1 broadcasting 'who's there' message.
(MLO) AUV2 broadcasting 'who's there' message.
(MLO) Convention chooses AUV1 to initiate MLO.
(MLO) AUV1 broadcasts 'mlo-exists' message;
      others update their knowledge.
(MLO) MLO contains members (AUV1 AUV2).
```

Figure 1: MLO formation phase.

respond; at that point, the agent can send a message saying in essence: "here's my location and I want to join the MLO". The MLO will then update its knowledge (see below) and the new agent will be in.

The more interesting situation is when there are no existing organizations. In that case, all MLO-agents will be trying to discover who else is "out there". After some time, when there is no reply by an agent that is part of an existing organization, the agents assume that there is no MLO and that they know about all other MLO-agents. One of them (determined by conventions to be developed) initiates MLO creation by sending a message to the others saying it exists.

In the simulator, several rules simulate the aggregate behavior of this protocol. When there is no MLO, a rule fires to add agents to a blank MLO structure (i.e., a fact in CLIPS' working memory) and another asserts that the MLO has been formed. At that point, another rule moves the simulator into the next phase.

Figure 1 shows an example from the simulator during this phase.

MLO discovery phase. This phase is concerned with discovering the AOSN's total capabilities in terms of the collection of all the VIPs' generic behaviors. We currently simulate two different protocols during this phase. One involves no hierarchical structure: all non-MLO-agents communicate freely with all MLO-agents. This is called the "flat" form of the MLO. The other is one in which an MLO-agent controls those non-MLO-capable VIPs that are closest to it; this is the "hierarchical" MLO form. Rules for both are in the simulator; the user asserts an "`MLO-form`" fact to switch between the two forms.

Regardless of the form of the MLO being simulated, each MLO-agent tells its peers (via broadcast or a set of messages) about its location and capabilities; this establishes some common knowledge among the members of the MLO. Each MLO agent also broadcasts a "who's there" message aimed at eliciting responses from non-MLO-agents. It is important for all of them to do this, since not all VIPs may be reachable by all MLO-agents due to properties of the underlying communication medium.

In the flat version of the MLO, each VIP responds with its location and its capability list. Any MLO-agent that believes that it has unique knowledge of the VIP tells the others about it. When the message traffic ceases, all MLO-agents have common knowledge about the AOSN's capabilities.

```
(MLO) Agents are attempting to discover other
      VIPs.
(MLO) AUV1 broadcasting 'who's there' message.
(MLO) AUV2 broadcasting 'who's there' message.
(MLO) AUV2 --> AUV1: I have capability(ies)
      (b e C f), and I'm at location (10 10 10).
(MLO) AUV1 --> AUV2: I have capability(ies)
      (a d C e), and I'm at location (0 0 0).
(MLO) vip1 --> AUV1: I am at (1 1 1).
(MLO) vip1 --> AUV2: I am at (1 1 1).
(MLO) vip2 --> AUV1: I am at (5 5 5).
(MLO) vip2 --> AUV2: I am at (5 5 5).
(MLO) Closest MLO agent AUV1 now controls vip1
(MLO) AUV1 --> vip1: tell me your capabilities.
(MLO) vip1 --> AUV1: I have capabilities (a g f).
(MLO) Closest MLO agent AUV1 now controls vip2
(MLO) AUV1 --> vip2: tell me your capabilities.
(MLO) vip2 --> AUV1: I have capabilities (a b d).
   [...]
(MLO) MLO formation complete.
```

Figure 2: MLO discovery phase.

In the hierarchical form as currently simulated, VIPs respond with their location to the "who's there" messages. If an MLO-agent believes that it has sole knowledge of the VIP, or that it is the closest to the VIP (based on common knowledge of other MLO members' locations), then it becomes the controller of that VIP. This is done via shared conventions rather than message-passing between the MLO members. At that point, it sends a message to the VIP requesting the VIP's capabilities, and the VIP responds to it.

There are many rules for this phase in the simulator, for example, rules to simulate the broadcast "who's there" message and an MLO-agent telling about itself.

Figure 2 shows the simulation of discovery in a hierarchical MLO.

MLO decision and TLO formation phases. These are the phases in which the MLO agents determine the form of the TLO that will accomplish the mission. The protocol we are currently simulating has the MLO select one of its members as a planning agent based on a very simple convention: the first one appearing in working memory is selected. This simulates conventions that a real MLO might use, such as choosing an agent based on an alphabetical ordering of the agents' names. Other mechanisms could be used; in particular, in the near future we will be looking at those suggested by Cammarata *et al.* [1983], such as "select least constrained agent".

```
(MLO) Selecting AUV1 as planner (convention:
      first in MLO).
(MLO) Planner AUV1 querying others about
      capabilities they may contribute for
      tasks: (task2 task1).
(MLO) AUV2 --> AUV1: these agents may work:
      (AUV2).
(MLO) Planner AUV1 directly knows that
      agents (vip2 vip1 AUV1) may work.
(MLO) AUV1 deciding on  TLO.
(SIM: switching context -> TLO-formation)
(MLO) AUV1: informing other agents about
      VIP <-> task assignment.
   [...]
(MLO) Vips assigned to task1: (vip2 AUV1)
(MLO) Vips assigned to task2: (vip1)
(MLO) AUV1: dissolving the meta-level
      organization.
```

Figure 3: MLO decision and TLO formation phases.

The planner then decides which capabilities it needs for its tasks. If the MLO is flat, then the planner has all the knowledge it needs. If hierarchical, then the planner asks each other MLO member if it has any of the capabilities needed for any of the tasks, or if any VIP it controls does. If so, the member reports which VIPs (including itself) might be useful. The planner then assigns VIPs to tasks, and assigns managerial roles, again based on the capabilities of the agents. This is currently done by C functions that implement a first-fit of capabilities to tasks and return the results to CLIPS; the simulation of managerial role assignment is not yet implemented.

The last thing that happens in the TLO formation phase is that the MLO is dissolved until needed again.

Rules for these phases select the planner, create representations of capabilities of VIPs and those that are needed for the tasks, and call the C functions. The simulator has complete knowledge of VIP capabilities, so the "messages" sent by MLO-agents in the hierarchical form contain only the names of VIPs.

```
(TLO) Assignment and acknowledgment messages
      exchanged.
(TLO) Beginning work on task task1.
(TLO) Assignment and acknowledgment messages
      exchanged.
(TLO) Beginning work on task task2.
>> Enter an assertion (or 'go' or 'quit'):
(status task1 done)
   [...]
>> Enter an assertion (or 'go' or 'quit'):
(status task2 done)
   [...]
(TLO) All tasks are done.
(SIM: switching context -> MLO-formation.)
```

Figure 4: TLO work done.

```
(TLO) ** Error in task task3 detected **
(SIM: switching context --> error)
(TLO) Can't handle error in task3 within TLO.
(TLO) Unrecoverable error in task task3;
      trying to reinvoke the MLO.
(SIM: leaving error context --> MLO-formation)
   [...]
(MLO) MLO contains members (AUV1 AUV2).
   [...]
(MLO) MLO taking error into account, work
      continuing on task3.
   [...]
(SIM: switching context -> TLO-work.)
```

Figure 5: Simulation of error handling.

Figure 3 shows a simulation of these phases for a hierarchical MLO.

Other phases. At the time of writing, we have not yet completed simulation of the TLO work phase, the phase in which the mission is actually accomplished by the AOSN. We simulate this now by having CLIPS ask for facts the user wishes to enter to change the state of the simulation, including changing the status of a task (e.g., to "done" or "error").

When all tasks are done, the TLO is dissolved and a new MLO forms, as shown in Figure 4.

At this point, the MLO formation process occurs again, shortened this time due to prior knowledge the MLO-agents have about the AOSN. If there is nothing left to work on, then the simulator accepts input from the user within the context of the MLO being present; this simulates the MLO waiting for something to do.

When an error occurs during the TLO work phase, the TLO attempts to handle it itself. If it can, then work continues. If not, then the MLO is re-formed to handle the task from its more global, less constrained perspective. It may be able to patch the existing TLO, or it may have to create a new TLO for the changed situation.

Figure 5 shows the skeleton of how errors will be simulated; the rules currently present are essentially placeholders for more concrete rules to be developed in the future.

VI Discussion

In this paper, we have argued for two things, a particular approach to the problem of controlling an AOSN and a general simulation methodology for such approaches. We have begun the task of designing the protocols for the two-level organizational structure and implementing them in the simulator.

With respect to the simulation methodology, our experience so far leads us to believe that it will be quite useful. The learning curve for CLIPS is not very steep, and it was relatively easy to code the abstract protocols existing at this stage. The interface between CLIPS and C is straightforward, though adding custom C functions does require recompiling part of CLIPS, and C cannot access all the information that one might want. However, this tends to encourage the implementor to use CLIPS rules for as much as possible, which is good at this point in the project: it keeps the knowledge (e.g., protocols, etc.) explicit, hence easier to examine and change.

The if–then structure of production rules seems very natural for capturing the flavor of the protocols, at least at the current level of detail. Most of our descriptions of what agents and organizations do are either already in such rules or are easily put in that form.

Simulating the aggregate properties of a set of protocols worked well, too, rather than directly implementing the protocols in code, then running that code in a simulation testbed. Those of us in AI have had much experience with the latter kind of simulation. It tends to be time-consuming, and it tends to lock one into design decisions because of the substantial commitment of time and coding effort before the design can be tested. Consequently, one often finds oneself patching implementations rather than designs. The approach taken here has already allowed us to test some design decisions with relatively little effort. For example, we are trying both a flat and a hierarchical MLO to see which performs best from the standpoint of message traffic. To try a new design alternative such as this requires little more than writing a handful of rules.

Although this project is less than two months old at the time of writing, using the simulation methodology we describe has already allowed us to progress to the point where we can simulate the overall flow of the approach and begin to make concrete statements about the relative merits of one protocol choice versus another. We would have been hard-pressed to have progressed so far in so short a time otherwise.

In the near future, we will complete the first draft design of the protocols for our approach and test them in the simulator. Our goal is to run a set of simulation experiments to evaluate our design in mid-autumn of this year. In the second year of the project, the protocols will be made more concrete by specifying such details as message types and content, and design flaws will be fixed that may have surfaced during the first year. Future phases of the project will implement the protocol, with evaluation both in simulation testbeds such as SMART [Turner *et al.*, 1991] and in the water using AUVs and other instrument platforms.

References

Cammarata, S., McArthur, D., & Steeb, R. (1983). Strategies of cooperation in distributed problem solving. In *Proceedings of the 1983 International Joint Conference on Artificial Intelligence*, pages 767–770.

Curtin, T., Bellingham, J., Catipovic, J., & Webb, D. (1993). Autonomous oceanographic sampling networks. *Oceanography*, 6(3).

Durfee, E. H. & Lesser, V. R. (1987). Using partial global plans to coordinate distributed problem solvers. In *Proceedings of the 1987 International Joint Conference on Artificial Intelligence*, pages 875–883.

Fox, M. S. (1981). An organizational view of distributed systems. *IEEE Transactions on Systems, Man and Cybernetics*, 11:70–80.

Giarratano, J. C. (1993). *CLIPS User's Guide*. NASA, Information Systems Directorate, Software Technology Branch, Lyndon B. Johnson Space Center, Houston, TX.

Hewitt, C. (1986). Offices are open systems. *Communications of the ACM*, 4(3):271–287.

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.

Turner, R. M., Blidberg, D. R., Chappell, S. G., & Jalbert, J. C. (1993). Generic behaviors: An approach to modularity in intelligent systems control. In *Proceedings of the 8th International Symposium on Unmanned Untethered Submersible Technology (AUV'93)*, Durham, New Hampshire.

Turner, R. M., Fox, J. S., Turner, E. H., & Blidberg, D. R. (1991). Multiple autonomous vehicle imaging system (MAVIS). In *Proceedings of the 7th International Symposium on Unmanned Untethered Submersible Technology (AUV '91)*.