N93-11924

Distributed Environmental Control

Gary A. Cleveland
McDonnell Douglas Space Systems Company - Space Station Division
5301 Bolsa Avenue, Huntingon Beach, CA 92647
(714) 896-3311 x7-0311
cleveland%ssdvx1.decnet@mdcgwy.mdc.com

1.0 INTRODUCTION

We present an architecture of distributed, independent control agents designed to work with the Computer Aided System Engineering and Analysis (CASE/A) simulation tool. CASE/A simulates behavior of Environmental Control and Life Support Systems (ECLSS). We describe a lattice of agents capable of distributed sensing and overcoming certain sensor and effector failures. We address how the architecture can achieve the coordinating functions of a hierarchical command structure while maintaining the robustness and flexibility of independent agents. These agents work between the time steps of the CASE/A simulation tool to arrive at command decisions based on the state variables maintained by CASE/A. Control is evaluated according to both effectiveness (e.g., how well temperature was maintained) and resource utilization (the amount of power and materials used).

1.1 Motivations and Criteria

We employ five criteria in designing and building this control system.

1) The controller is to work with the CASE/A simulation system.

2) The architecture should introduce as little new vocabulary as possible to describe the systems being simulated by CASE/A and their control. We wish to keep the usability of the CASE/A system high, especially among its current user community.

3) The controller must coordinate diverse and conflicting functions among sensors and effectors that are spatially remote, work in a system with significant time delays, and are subject to certain types of faults.

4) Control of the system must degrade gracefully in the face of several types of faults.

5) The control architecture should be modular so that new controllers can be constructed to match new simulations without changing any large part of the basic scheme.

6) The control mechanism should make use of a parallel processing model of computation. We do not want to eliminate the option of a parallel implementation.

1.2 General Description of this Work

It is fairly straightforward to control a given device at about the level at which one would describe a thermostat or proportional control. One uses rules such as, "When the temperature gets above a certain point, turn on the air conditioner," and, "As the vibration increases, decrease the rate of spin by an amount depending on the severity of the vibration." These examples and our viewpoint are meant to be consistent with the control found in the field of robotics; sensors provide input, effectors receive output, controllers map the former to the latter.

This effort takes a more generalized view of sensors and effectors, introducing explicit levels of control and allowing control to be described uniformly at all levels. Communication between the control levels is carried out using the same set of constructs as are the sensing and effecting at the lowest level. In effect, a higher level controller "senses" the information that the lower level controllers make available. The introduction of multiple levels of control is the key factor in being able to address control issues that

span several sensors, and in being able to produce coordinated control over a number of effectors. This abstraction also provides an ability to step away from the hardware a bit and deal with the control problem in an intuitive manner. For example, suppose a pair of sensors measure air pressure and the partial pressure of oxygen, respectively. If our control problem is built around the percent of oxygen in the air, then the control algorithm will be made less clear by the extra computation. We are proposing that this extra translation be extracted from the basic control algorithm. Similarly, levels of control allow sensors such as "AirLock Nominal" to be constructed. Such a "virtual" sensor might look at a half-dozen other sensors (both virtual and hard sensors) using a complex algorithm before actually determining that the airlock status is nominal. In both cases, the translational and analytical work is separated from the control algorithm that uses the analysis.

1.3 Relationships to Other Work

The official description of the CASE/A simulation system is found in the CASE/A User's Manual [CASE/Aa] with additional insights and details provided by the Programmer's Manual [CASE/Ab]. Of special value in understanding our work is the detailed description of how CASE/A handles time steps.

More to the point of our effort, the reader should consult the work on subsumption architectures at MIT ([Brooks], [Connell]) from which we have adopted much of our communication model and protocol. This communication scheme has been modified to address some of the same issues as Henderson's work ([Henderson84], [Henderson90]). We perceive Henderson's work as an attempt to rise above limitations in the MIT work coming from the low level at which those systems are built. We have adopted Henderson's scheme as far as our simple sensors and effectors made desirable. We contrast the complexity of Henderson's vision sensors to that of our thermometers and air pressure gauges.

One may note the similarity in some aspects to neural networks [Hopfield]. Three comments pertain. First, our graph of agents does employ a communication protocol similar to that used between neurons. Second, the model of computation carried out by our agents leans more towards the symbold than that found in neurons (making use of stored state variables, for example.) Third, it is not our intention to endow our agents with any learning potential.

Mention should also be made of experiments in the area of reactive intelligent control such as performed by Agre and Chapman [Agre]. By using the building blocks described below, our long term goal is to be able to construct reactive controllers which use knowledge at a level similar to that found in Agre and Chapman's Pengi system.

1.4 Organization of the paper.

Section 2.0 presents a low level view of communication with the CASE/A simulation tool. Section 3.0 describes the lattice of controlling agents and their functions. Section 4.0 describes the evaluations to be performed on the simulation runs controlled using this architecture.

2.0 COMMUNICATING WITH THE CASE/A SIMULATION SOFTWARE

2.1 The CASE/A Simulation Software.

The Computer Aided System Engineering and Analysis (CASE/A) modeling package is an Environmental Control and Life Support System (ECLSS) and Active Thermal Control System (ATCS) analysis and trade study tool. The package is written in FORTRAN and supports the construction and analysis of ECLSS and ATCS models by offering primitive units such as pumps, heat exchangers, etc., that can be linked together to form the desired models. The primitive units are referred to as *components* and the links between them, for the purposes of this presentation, are called *streams*. Streams themselves are discussed in terms of the *constituents* that flow through them. Oxygen, water, and other materials are examples of constituents. The *properties* of streams are also of interest. The most often discussed properties are temperature and pressure.

Each component in a CASE/A model has a function that computes, at every time step, the values for the output streams given the values found at the input streams. The CASE/A package visits the various components in a model, finding new values for the streams. Naturally, the output

streams of some components are the input streams for others. Since there are cycles in the models, CASE/A will visit some streams and components repeatedly. While so doing, CASE/A is attempting to find a solution that satisfies all of the interconnected components. Some components may be visited numerous times on a given time step before convergence is achieved.

CASE/A also supports its own version of controllers that can be linked among the streams and components and which support a language similar to a zero-register (stack-based) assembly language for forming and carrying out control decisions. Unfortunately, these controllers can only sense one value at a time and can only affect one value at a time. This eliminates the possibility of any straightforward scheme for coordination among sensors or effectors. These controllers do, however, provide the basics of the model of communication between CASE/A and the controller we are building.

2.2 Communicating with CASE/A

As was hinted in the previous section, the streams (constituents) in CASE/A provide a natural correspondence to the sensors that we desire to create. The components provide our vocabulary of effectors as well as providing more sensors of interest (e.g., pump flow rate). As effectors, we may set a pump flow rate based on the temperature of a water line. It remains only to create an import/export mechanism enabling the values to be moved between the existing CASE/A package and the newly created controller. It turns out that for the VAX/VMS system where CASE/A resides, communication between FORTRAN and the controller's home language of Ada is straightforward.

The timing of the communication is also straightforward but still bears discussion. To control a physical system, we would be faced with accepting and reacting to asynchronous sensor signals. To the degree possible, we wish to work with this model even though the CASE/A system works in discrete time steps. CASE/A will transfer program control to the controller only between those time steps. The resulting communication scheme has CASE/A passing a number of sensor signals to the controller all at the same time but in no particular order. Because the controller processes these signals using an asynchronous model, it sometimes happens that control (effector) signals are generated and then overridden before reaching a final form. Overrides, as discussed below, are one form of communication between control agents. After all the control signals have been generated and have stabilized, the packet of signals is sent back to CASE/A for another time-step iteration.

3.0 THE AGENT ARCHITECTURE

3.1 Control Agent Description

The primary unit of control in our architecture is the *agent* as depicted in Figure 1. Each agent has ports to accept sensor input and to produce effector commands. Command mappings from the sensors to the effectors are produced by one of a number of algorithms available to the agent. Each such algorithm may be viewed as one part of a production (rule-based) system. These control algorithms are local to the agent and operate independently from those of other agents. Each agent can also maintain memory of past sensor values and effector commands so that trends may be noticed and complex actions requiring a schedule of sub-actions may be effected. The set of sensor ports and the set of effector ports are redundant in that there may be several means of deriving the same information or issuing the same command from different subsets of the available ports. Coordinated behavior is achieved spatially by using connections between the ports as communication lines and temporally by utilizing the internal memory of the agents.

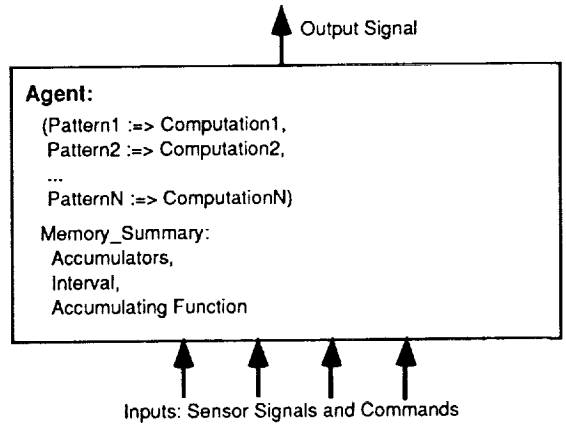

Figure 1: Structure of Agents (with Memory)

3.2 Coordination of Agents

All agents are arranged in a tangled hierarchy (directed acyclic graph) with the sensor and control signals traveling up through the graph. Topologically, this is identical to Brooks' networks of subsuming agents and still similar to Henderson's hierarchy of logical sensors. All the control agents below a given agent are treated as sensors while all the agents above a given agent are treated as effectors. The leaves of the graph represent the actual sensors (as found in the CASE/A simulation) while the roots of the graph represent the actual effectors. Functionally, this arrangement is also similar to Henderson's work because the higher level agents have control (through overrides and subsumption) over those at lower levels. The difference lies in the arrangement of connections in the graph.

The network of agents is static during a simulation run. Dynamic behavior is obtained from this static net when a given agent chooses to invoke a different algorithm for sensor analysis and effector control signal generation. This parallels Henderson's work with the important distinction that sensor and command signals are combined in one communication channel.

Generally, the leaves of the graph directly relate to sensors whose signals are transmitted through the interface from the CASE/A simulation. Control is accomplished by manipulating these sensor signals as they pass up through the graph. Signal manipulation takes the form of computing new signals to pass on from those received. Some of the signals received will correspond closely to physical values produced by the simulation while others will be better interpreted as control or context signals. The two types of signals are treated alike.

While these "control signals" are treated the same as the "sensor signals," one may view their treatment from several viewpoints. The first is that the controlling agents reside in the agent network "above" those agents that they control. The agents higher in the graph produce control signals that override the signals generated by the lower agents. This view corresponds to subsumption as put forth by Brooks *et al.* The other view is that the controlling agents reside "below" the controlled agents. By providing different inputs, the lower level agents can influence the behavior of the upper level agents. As with Brooks, we prefer that the higher level agents might have knowledge of the graph below them but not that any lower level agent should ever have knowledge of the graph above.

Depending on current and previous sensor (and control) values, various computations may be used to create the signals that will continue up the graph. The choice of algorithm may also depend upon estimates of confidence in the signals being passed in. Actual algorithm selection is performed by a simple pattern match against current input and stored values.

3.2.1 Sensing

Agents sense declared numeric values within the CASE/A simulation. All sensors are defined in terms of the structures (usually constituents of streams) that CASE/A already maintains although communication among agents depends on slightly different streams. Using this scheme, three more complex sensing behaviors can be constructed.

3.2.1.1 Grouping

Homogeneous groups of identical sensors in parallel can be used as the simplest means of obtaining fault tolerance. Most of the time a group of such sensors is viewed as a single sensor, producing a single reading derived from the combination (usually the average) of the readings of the individual sensors. A complex sensor of this type needs to have some facility for dealing with failed individuals. The reading from the combination of sensors can usually be assigned a higher confidence value than that of any of the individuals.

3.2.1.2 Fusion or Virtual Sensing

Heterogeneous groups of sensors may also be constructed and represented as a single, combined sensor as shown in Figure 2. This type of complex sensor can produce a "sensed" value derived from but not directly related to any physical measurement. Most of the "control" agents take this form.

3.2.1.3 Integration, Trends, and Time Averages

Sensor agents with memory can store past readings in order to produce values for totals, trends, and averages over time. The outputs

from these sensors are referred to as control signals for the sake of uniformity. In fact, these signals are usually piped straight to the input of another agent that treats the signal as another type of sensor.

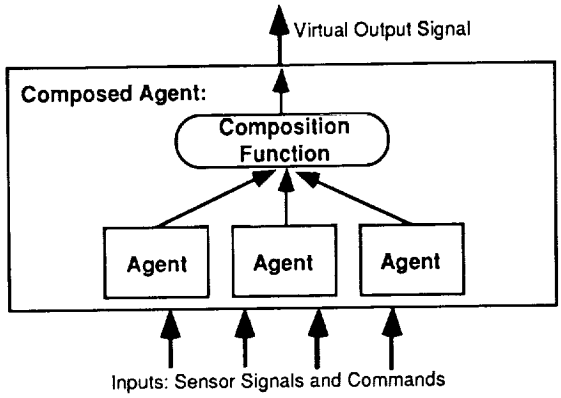

Figure 2: Agent Composition for Coordinated Control and Virtual Sensing

3.2.2 Effecting

Carrying out the control decisions is carried out by communicating those decisions to CASE/A, in the form of numeric values corresponding to desired settings for component attributes (e.g., flow rate of a pump). Most of the important pieces needed for coordinated control have already been introduced with the discussion of the controllers above. It remains to show how those controllers can be used to carry out complicated behaviors.

3.2.2.1 Coordinated actions

3.2.2.1.1 Among Agents

Coordinating actions among agents is straightforward given the arrangement of controllers in the graph as described above. A controlling agent merely outputs a value that signals a certain context has been entered. The agents being coordinated must have this context preprogrammed as one of the patterns to which they respond. We are again relying upon experience to show whether the number of such patterns is prohibitive and whether coordinated actions will need to show more flexibility than can be achieved with the scheme just outlined.

An alternative means of rendering such coordination would be to have agents watch each others' command streams and act accordingly. This second scheme eliminates the need for the (extra) coordinating agent but even more strongly begs the question of pattern complexity. Probably, the domain and the specific behavior being programmed will determine which method is used.

3.2.2.1.2 Through Time

For coordinating actions through time we rely upon the sensor agents' capability for collecting accumulated data such as averages and trends. Such sensors can "count time" as well as watch the behavior of sensed values. Agents already have the ability to respond to changes in the sensed environment and can thus respond to other agents' actions (e.g., after Agent1 starts the motor, Agent2 should open the valve). The agents with the time memory will allow scripted behaviors among one or more agents. A scripted behaviors resides within a single agent and is carried out when that agent sends appropriate signals (just as all others) to other agents. For example, Agent1 and Agent2 should both turn on motors for five minutes. The beginning and end of the five minutes is called out by Agent3, possibly by claiming to sense an imbalance in a hydraulic line that goes away after five minutes.

3.2.2.2 Virtual Effectors

Agents that are somewhat removed from the actual control of the components accessible from CASE/A may be referred to as "virtual effectors." An example is a controller that is programmed to "raise the temperature" but does not have direct access to the CASE/A program. Such a controller should be in a position of communicating to some set of heaters, fans, lights, etc that can be coordinated according to the current set of tradeoffs. In fact, this is a complementary view of the coordination of sensors mentioned above. In one, the system tells the sensors what to be sensitive to and in the other, we view this from the coordinating agent's vantage of working towards some purpose.

3.3 Fault Tolerance

Fault tolerance is becoming one of the most important issues in controlling complex systems. For large systems, it is not practical to try to eliminate all faults. For this reason, nearly all fault tolerance arises from the introduction of redundancy into the system. The basic scheme is to have one part of the system stand in for another part when it fails. This leaves us with the two questions concerning what parts may fail such that the

system still functions reasonably and how is it that this "reasonable" or "acceptable" behavior is achieved.

For the current version of the control system, we address faults in the CASE/A sensors and effectors only. When a sensor fails, it stops broadcasting signals of any kind. When an effector fails, it does so by acting as though it were receiving random commands or by "sticking" to either a maximum or minimum command value. We assume the control system components themselves to be above suspicion. As a further simplification in the first implementation, we assume that there exists some diagnostic program for the sensors that can tell us which sensor is failing. This allows us to concentrate on the actions the controller should take given a (single) failure rather than revisiting the subject of automated diagnosis. With this amount of information, we are also capable of quickly inferring faults in the effectors when they occur. Again, our emphasis is on the treatment of these faults more than the location. It may be that in some cases we can not specifically locate a fault but can take steps to work around it.

3.3.1 Sources of Redundancy

Two of the forms of redundancy that the system uses have already been mentioned. The first is the parallel replication of identical sensors. Assuming that only one sensor fails at a time, we can always achieve a reliable reading. We still lose information, of course, with each loss of a parallel sensor. The second form of redundancy is buried within the coordinating controllers. The coordinating agent must shift the control commands from a failed effector to those still working. For example, if a heater fails, one might turn a fan down in order to conserve more of the available heat.

Other forms of redundancy are also exploitable. Given that the controllers are able to carry out sequences of actions through time, one may rely upon the inertia of the system to achieve what an effector normally would. When a water pump fails, one may use the water stored in a holding tank for a fixed amount of time.

3.3.2 Sensors

Because of the way sensor signals are processed through a pattern match and then computation, it is possible to invoke different processing algorithms based on the availability of a sensor signal as well as based on the signal's value. When a sensor fails, those controllers that rely on the signal switch to contingency algorithms or get overridden by controllers designed to watch for just such an occurrence. In many ways, the lack of signal from a sensor is treated exactly as if the signal values were out of some range; new action is triggered.

The one truly unusual way that sensors may be used in this whole architecture is to make use of the "predictive" properties of those controllers that have memory for charting averages or trends. With the assumption that an average or trend will continue, a controller may issue commands for some time based on predicting what the important signals ought to be. This behavior is an example of using the system inertia for sensing purposes. In fact, this behavior may be invoked even without loss of a sensor.

3.3.3 Effectors

The one comment that remains to be made with respect to the effector is in addressing diagnosis. It will sometimes be the case that a sensor will begin to report unbelievable values while still checking out as operative. In this case, the system must identify the effector that is malfunctioning. This will be a non-trivial chore as several effectors are sure to affect any given sensor. Possibly by modulating the control values sent to the effectors, the site of the malfunction can be deduced. Another promising technique is that of set covering. Even if we can only pare down the list of possible failures, that may be enough to allow the controllers to use the operational effectors to offset to ill effects of the failed effector.

3.4 An Example

The example control network that we return to repeatedly is that of a thermostat. Despite its simplicity, the thermostat can be used to demonstrate most of the interactions we face.

Supposing we have a thermostat connected to a thermometer for a sensor (Thermostat_Sensor1) and a heater for an effector as shown in Figure 3.

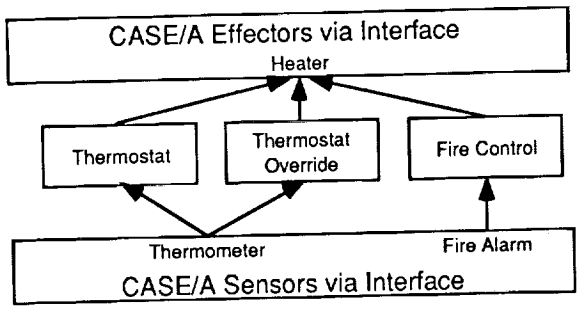

Figure 3: Successive Overrides of Control Signal

The behavior of the thermometer is to hold the temperature at approximately 25°C. The basic controller for the thermostat works under a control law expressed in patterns and commands as shown in Table I. Granted, this control law may result in an undesirable oscillation from one output command to the other.

Pattern:		Output
Thermostat_Sensor1 < 25	⇒	ON(50%)
Thermostat_Sensor1 >= 25	⇒	ON(0%)

Table I. Simple Control Law

Assuming that 50% of the heater capacity is sometimes insufficient to keep the area warm, we see that our design is not finished. We decide to add a second control agent which provides no output signal until the area we are warming becomes thoroughly cold. Under these conditions, the second agent overrides the signal of the first. The second agent has a control law similar to the first as is shown in Table II.

Pattern:		Output
Override_Sensor1 < 5	⇒	ON(100%)
Override_Sensor1 >= 5	⇒	Nil

Table II. Override Control Law

Obviously, this capability could have been included in the control law of the first agent. That it was not is due to our notions of modularity. When circumstances change significantly (e.g., water in the room is about to freeze), it is appropriate that another controller watch for the change and take appropriate action. An example of a more significant change would be the presence of a fire in the area. An additional agent is introduced to monitor the fire alarm. Alarm_Sensor1 referenced in the control law shown refers to the Fire Control agent's first sensor. When a fire is spotted, the heater is turned off as any electrical appliance should be. This law is shown in Table III.

Pattern:		Output
Alarm_Sensor1 = OFF	⇒	Nil
Alarm_Sensor1 = ON	⇒	On(0%)

Table III. Fire Alarm Control Law

We note that the actual implementation of the override function is carried out by the Heater agent which communicates through the interface to issue commands directly to CASE/A. The Heater agent contains a control law that prefers the override signals (if present) to the basic command signal. This agent's control law is shown in Table IV.

Pattern:		Output
Heater_Sensor3	⇒	Heater_Sensor3
Heater_Sensor2	⇒	Heater_Sensor2
Heater_Sensor1	⇒	Heater_Sensor1

Table IV. The presence and absence of signals determines behavior.

As a last note, we comment that this entire complex controller for the Heater can be encapsulated by another agent with two sensor inputs and one effector output. Although we may not have landed on the exact control laws that we need, we have built the structure needed to support the behavior necessary. We have also built this structure incrementally and in modular form that can be repeated and further built upon. If modifications are necessary, they should be also well-contained.

4.0 EVALUATION OF THE ARCHITECTURE

4.1 Effectiveness of Control

Part of any experiment must be an evaluation of the results of the experiment. In our case, this may be roughly voiced as, "Have we provided a sufficiently robust and realistic control that the users of the CASE/A system have benefited significantly?" We may also ask if the results of this experiment might apply to other domains as well. It has been our design goal to ensure that they do. Since all of the values and control parameters are numeric, the effectiveness may be measured using standard statistical measure. Due to the unavailability of these measures for other control systems and architectures, comparisons may not be possible.

4.1.1 Setpoint Accuracy

The primary criterion of our control system behavior must be, "Does it do what we told it to?" Given a list of setpoints for the values in the system, are we able to control the system so that those values are at or near those setpoints? Does this remain true over time?

4.1.2 Prediction and Anticipation

Is there a means of informing the system of a major change of context or of operating mode such as harvesting a crop in a plant chamber? How well does the system cope with this? How large a disturbance can it handle?

4.2 Resource Utilization

Given that the system is behaving properly or at least acceptably (i.e., within some limits), we wish to observe the cost of the behavior in terms of resource consumption.

4.2.1 Resource Usage and Local Optimization

Once an acceptable behavior has been achieved it should be possible to make small changes to the system's behavior and measure the change in terms of resource utilization. Typically, resources will include power, crew time, and materials (oxygen, water, etc). Both types of resource trade-offs can be addressed using statistical decision making tools.

4.2.2 Resource Trade-Offs

Some importance or relative cost for the various resources must be assigned since there will be a number of control strategies meeting the system requirements but using different amounts of the various resources. In this case, time must also be considered a resource as some tasks may be carried out with less material commitment if done more slowly. Within the limits of parameter perturbation, the control system can be used to investigate resource trade-offs. We are not attempting to automate the more involved notion of trying out entirely different control strategies.

If the behavior of the system is specified within relatively large intervals, it should also be possible to trade the system effectiveness for resource conservation. For example, if a temperature can be held on the low end of its acceptable range, we may be able to avoid using a heater. This might be accomplished by pumping waste heat from the living quarters and saving electrical power both in avoiding use of the heater and in avoiding excessive use of coolers elsewhere in the environment.

BIBLIOGRAPHY

[Agre] Agre, Phillip E. and David Chapman, "Pengi: An Implementation of a Theory of Activity", *Proceedings of the Sixth National Conference on Artificial Intelligence*, Morgan Kaufman Publishers, 1987.

[Brooks] Brooks, Rodney, "A Robust Layered Control System for a Mobile Robot, *Journal of Robotics and Automation*, March 1986.

[CASE/Aa] Integration Analysis CASE/A User's Manual, Sept. 1989, McDonnell Douglas Report MDC W5074-4.

[CASE/Ab] Integration Analysis CASE/A Programmer's Manual, Aug. 1990, McDonnell Douglas Report MDC W5146-3.

[Connell] Connell, Jonathon, "A Colony Architecture for an Artificial Creature," MIT Tech. Report AI-TR 1151, 1990.

[Henderson84] Henderson, Thomas and Esther Shilcraft, "Logical Sensor Systems," in *Journal of Robotic Systems* 1(2), pp 169-193 (1984), John Wiley & Sons, Inc.

[Henderson90] Henderson, Thomas and Rod Grupen, "Logical Behaviors," in *Journal of Robotic Systems* 7(3), 1990, John Wiley & Sons, Inc.

[Hopfield] Hopfield, J.J. and D.W. Tank, "Computing with Neural Circuits: A Model," *Science* Vol. 233, Aug. 1986.