The Language of Dynamic Planning: Inter-Agent Communications in ALP

Marshall Brinn
BBN Technologies
10 Moulton St.
Cambridge, MA 02138
mbrinn@bbn.com

Todd M. Carrico
Defense Advanced Research Projects Agency
3701 N. Fairfax Dr.
Arlington, VA 22043
tcarrico@darpa.mil

ABSTRACT

The application of distributed agent techniques to large-scale problems like global logistic planning raise interesting challenges regarding the communication and management of information between the agents. The distributed nature of the problem often makes for wide gaps of knowledge that need to be bridged through an expressive yet efficient language for expressing requirements and service results. The Advanced Logistics Program (ALP) has developed an infrastructure for representing extremely large problems in a distributed manner. The use of an expressive communication language along with techniques for state encapsulation and decision space tradeoff analysis have provided for efficient coordination among large numbers of medium complexity planning agents towards solving large, complex problems. This article describes in detail the language and structures developed under ALP to effect the inter-agent communications that enable successful distributed coordination.

Keywords

Agent Negotiations, Agent Communication Language, Distributed Coordination

1. INTRODUCTION

There are many benefits associated with using distributed agent techniques to model a complex dynamic system. First, of course, one can take advantage of the efficiency of parallelism in execution. Second, one can achieve significant modularity in software development by breaking up a problem into explicit, separable pieces and cleanly encapsulating those in individual software agents. Most significantly, a complex problem that is modeled by a series of small agents, each representing some small localized view of a problem, can achieve significantly better fidelity over a large monolithic model. The emergent behavior of the society of cooperative, interoperating agents, each with its own parochial view of a given domain problem, can be much more complex and realistic than any attempt to model the entire society in all its complexities.

However, there are additional challenges associated with such a distributed agent approach. A single set of requirements and results is now broken up into a complex chain of delegations and negotiations. Two parties may need to solve a problem together, but only one understands the requirements, while the other understands the domain constraints in which those requirements can be solved. In order for there to be an efficient distribution of complex requirements and an efficient negotiation of solution among agents, there must be an expressive and efficient shared language among all agents.

DARPA's Advanced Logistics Program (ALP) has been addressing these challenges in its work to model the complexities of military logistics in a distributed agent-based framework. The challenge of the military logistics domain is daunting: there are thousands of organizations formulating extremely complex plans for the transportation, supply and support requirements for global military operations. Millions of asset objects of different sorts must be scheduled or requisitioned, each by organizations with their own constraints of business rules, policies and most significantly, reality.

The challenging operating paradigm of ALP is continuous planning, execution monitoring and dynamic replanning. Each organization must model real-world/real-time events with as much fidelity as possible. As such, at any time the system must be able to express and handle a commitment that was made and must now be broken (a plane has gone into maintenance, or a part arrived late, e.g.). Each organization must make its plans, and continually monitor reality as it unfolds, replanning as best it can in light of these changes.

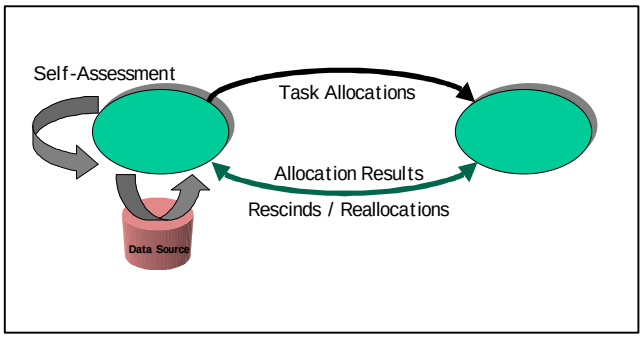

Figure 1. ALP Execution Monitoring and Dynamic Replanning Model: Continually reevaluate plan against requirements, business-rules and real-time resource state.

To tackle the complexities of this domain, ALP has modeled each organization in the logistics pipeline as an agent. This ranges from the front line combat units requiring logistics support to support, supply and transportation organizations providing that support. These agents, called ***clusters*** in ALP terminology, interoperate with one another through a rich series of structured interactions that allows each organization to express

its requirements, and to receive dynamic responses in adequate detail to respond appropriately.

The domain in which ALP works is the satisfaction of complex logistics requirements, often by decomposing and distributing these requirements among other organizations. For example, a supply depot knows how much it has in stock and how long it will take to replenish, a truck company knows how many drivers and trucks are available to transport material and what kinds of material they can handle. However, only the organization generating the requirement knows what constitutes a satisfactory plan to meet the set of requirements given the constraints and conditions of the situation. For example, the traveler, and not the travel agent, knows how to manage tradeoffs between cost and schedule constraints. We have of necessity a knowledge gap over which the negotiation of requirements in a distributed environment will occur: one party knows what is desirable, another knows what is possible.

This article describes the structures and interfaces developed in ALP to bridge this 'knowledge gap'. These structures have been developed to express complex requirements and the resulting solution space in a manner that is sufficient to allow successful inter-agent negotiations, while supporting the real-time requirements of dynamic replanning and execution monitoring.

2. ALP EXECUTION MODEL

The operating concept of ALP is to take requirements and generate the best plan to satisfy these requirements relative to currently available resources and situational conditions. The top-level requirements are typically expressed in terms of a high-level objectives and requirements. ALP must iteratively transform these high-level requirements into detailed logistics plans.

ALP models all organizations in the logistics chain as agents called ***clusters***. Each cluster maintains its own local sense of the world, representing the state of the resources it controls, and the portion of the global logistics plan that it manages and contributes to. This cluster-internal information is called the local ***PLAN*** of the cluster, and is not shared or visible to other clusters other than as allowed by the cluster itself. We thus have a notion of a distributed plan that constitutes the solution being generated. This plan is not in any one place, but is the composite of all plans being generated in the local PLAN of each cluster.

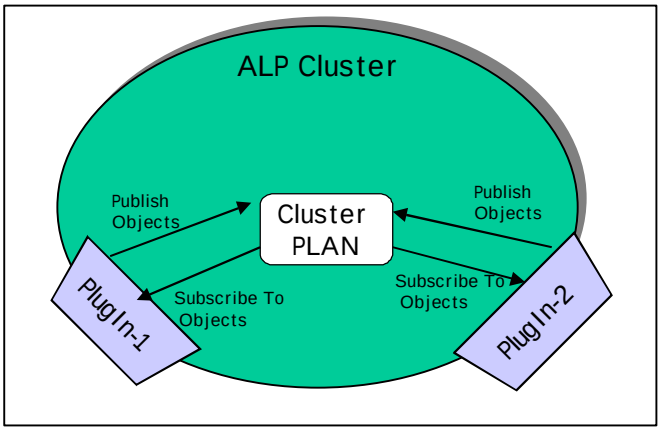

Figure 2. Sample ALP Cluster with PlugIns publishing objects to and subscribing to objects from the Cluster PLAN.

Each cluster is imbued with particular behaviors and capabilities through software modules called ***PlugIns***. The aggregate behavior of all PlugIns of a cluster makes up the expressed behavior of that cluster. PlugIns work in the context of the local PLAN of the cluster in a publish/subscribe model. Each PlugIn subscribes to particular kinds of objects of interest in the PLAN and publishes additions, changes, and removals to the PLAN in response to these subscriptions.

2.1 PlugIn Behaviors and the Human Cognitive Model

While a great deal of work has been in the areas of Artificial Intelligence and Cognitive Psychology [7], there is no definitive way to represent human cognition in software. While the concept intelligent agents implies the presence of this cognitive layer, little real work is being done on trying to achieve it. Rather, significant emphasis has been placed on modeling the appearance of intelligence through hand crafted control layers, often using a subsumption [1] or hybrid subsumption-deliberative planner approach.

While we are not explicitly attempting to achieve intelligence in ALP agents, we have developed an underlying cognitive model to ground the methodology of building complex agent systems. This underlying cognitive model serves as a natural roadmap to constructing components and assembling those components to perform complex business processes. Though drawing heavily from previous work in the area of cognitive model [2], we believe one of the major contributions to agent design comes from our work in this area.

The cognitive model the ALP architecture employs is modeled after the way human planners make decisions today. It is composed of the following basic elements, not all of which are required for a given planning action:

1- *Decomposition* : Take a problem and break it into smaller problems

2- *Delegation* : Take a problem and assign a resource to handle it, which may, in turn cause further decomposition and delegation

3- *Consolidation* : Taking a set of separate problems and merging them as a single problem

4- *Monitoring* : Continually reassessing a solution to make sure it is consistent with changing reality

5- *Gathering* : Gain more information from external entities

6- *Reporting* : Report information and results to external entities

7- *Acting* : Interacting with real world entities to implement the plan in real-time.

Each element would be manifested in a system component called a PlugIn. This component performs the function of that element by employing specific domain information and rules, encapsulated in the component, through a structured interface appropriate for that cognitive element.

The underlying ALP methodology expects problems to be analyzed and represented in this fashion. In fact, ALP PlugIns typically fall into one of the template patterns above:

1- *Expander* (Decomposition): A PlugIn that takes a problem and decomposes it into sub-problems.

2- *Allocator* (Delegation): A PlugIn that takes a problem and allocates it to a resource. This resource may be a physical asset which can satisfy the problem (e.g. a plane can satisfy a mission requirement), or another organization or cluster which will continue the cycle of expansion and allocation

3- *Aggregator* (Consolidation): A PlugIn that takes a series of separate problems, recognizes their commonality, and merges them into a single subsuming problem.

4- *Assessor* (Monitoring): A PlugIn that continually monitors the state of previous allocations, expansions and aggregations produced in the Cluster's PLAN for consistency with the changing state of the assets and resources managed and represented in the PLAN, taking corrective action as necessary

5- *Data* (Gathering): A PlugIn to read information from external data sources (sensors, databases, human interfaces) to populate the PLAN with detailed real-time information about relevant real-world entities

6- *UI* (Report): A PlugIn that allows for external user interfaces to receive detailed information (either on a pull or push basis) on the internal state of a cluster's PLAN.

7- *Execution* (Acting)*:* A PlugIn that executes the plan in real time by interacting and making commitments with real-world entities.

When an agent receives a request to perform some service, its PlugIns will typically invoke a logical business process of the following general form:

Step 1: Receive a new task

Step 2: Expand task into lower level subtasks

Step 3: Allocate each lower level task to resources. Leaf subtasks are allocated to operations, non-leaf subtasks are allocated to other agents

Step 4: Periodically assess progress toward achieving main task via execution status of subtasks

Step 5: When problems and inconsistencies are detected, reinvoke step 1 for plan repair

At any point in this general process, decision support information can be retrieved through the data PlugIns or provided to the users through the UI. Also, in some cases it is appropriate to aggregate activities, merging multiple sub-tasks into larger tasks - the exact opposite of the expansion function. This is done by the aggregators and is useful for activities like packaging units in a box or combining people into an airplane flight passenger list.

Since each of the PlugIns is required to be stateless, the critical results of each processing step are maintained in the shared ALP Plan. The next section presents the unique features of these elements design and briefly reviews how that plan is maintained across multiple agents.

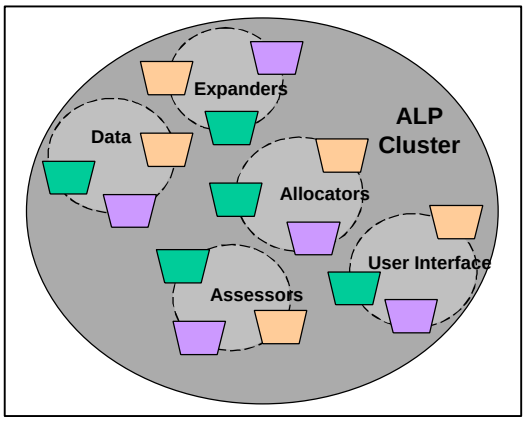

Figure 3. Sample ALP Cluster consisting of composition of multiple PlugIns of different standard cognitive patterns.

2.2 Fundamental PLAN Structures

The fundamental objects contained in the ALP PLAN are ***assets*** and ***tasks***, which one can think of as the nouns and verbs of the planning operations. Assets come in two varieties: *PhysicalAssets*, which represent the real-time state of a real-world physical entity, and *OrganizationAssets*, which represent proxies to another agent in the society within the PLAN of an agent. Tasks represent a requirement to perform or plan some operation, for example, to transport a set of materials between two points, or to order spare parts, or to fix a flat tire.

ALP seeks to allocate all tasks to assets. By allocating a task to a PhysicalAsset, the task is considered completely dispositioned: the requirement has been satisfied. By allocating a task to an OrganizationAsset, the task is handed off to the associated underlying cluster for further operations. Tasks may be expanded into sub-tasks or aggregated with other tasks into a single task from which point the cycle of allocation continues.

The ALP PLAN contains structures called ***PlanElements***, which represent the disposition of a task, by ALP. PlanElements can be of various forms:

- ***Allocations***, representing that a task was allocated to an asset (Physical or Organization)
- ***Expansions***, representing that a task was decomposed into a given set of subtasks
- ***Aggregations***, representing that a task was merged with other tasks to form a given unified task.

All of the information of the chain of events involved in the attempt to satisfy a given task is captured in the ALP PLAN. Of course, the PLAN of a given cluster only contains the information about that cluster's association with that task. As with any

question in ALP, the whole picture is contained in the union of all PLANs across all ALP clusters in an interoperating society.

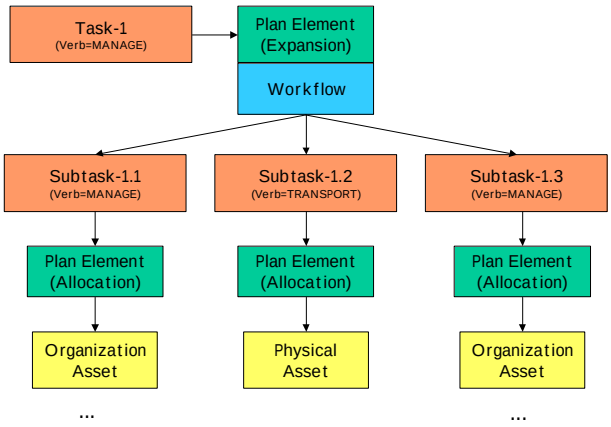

Figure 4. Sample ALP PLAN Element Structure, representing expansion of task to subtasks, and allocation of tasks to physical and organizational assets

3. ALP Inter-Clusters Dialog

The entirety of the operations of an ALP society of clusters can be described as the aggregate behavior of all pair-wise interactions among clusters negotiating the disposition of a particular task. The allocation of a task by one cluster to another is the fundamental operation of delegation, and requires a capability to express requirements and preferences of various sorts by the 'tasking' cluster, and the capability to pass back the results determined solution space from the 'tasked' cluster.

It is important to keep in mind that each cluster is, at all times, acting in multiple roles in these dialogs. It may be tasking one cluster in an attempt to satisfy the requirements and requests that have been made of it; it may be a consumer of one service as part of the process of providing services to another.

3.1 Expressing Requirements and Preferences

The task is the fundamental structure of request portion of the agent negotiation dialog. The structure of the task is a sentence in the language of discourse for ALP agents. The operational parameters of a task are contained as essential immutable contents of the task itself. In ALP, a task consists of the following data:

- ***Verb*** : A mnemonic verb name indicating the type of service requested (e.g. TRANSPORT, SUPPLY, MANAGE, SUPPORT)
- ***PrepositionalPhrases*** : A set of prepositional phrases, associating mnemonic preposition (e.g. FROM, TO, WITH, FOR, USING, CONTAINING) with a particular object from the sending cluster's PLAN (e.g. FROM <Source Location>, TO <Destination Location>, USING <Particular Policy Guidelines>)

In addition to these strict expressions of requirements, a tasking organization can specify a series of preferences. A preference consists of the following data:

- ***Aspect*** : A dimension of measurement about which a preference is being expressed. ALP contains a predefined set of these, including START_TIME, END_TIME, COST, QUANTITY, READINESS, and CUSTOMER_SATISFACTION.
- ***ScoringFunction*** : A mapping from the aspect measurement space into a dimensionless score space. This function allows for specification of best and worst points in the space of solutions in a given aspect dimension, preferred values, and, by convention, unacceptable values.
- ***Weight*** : A scalar allowing for inter-preference comparison and aggregating scores from individual preferences

The intended semantics of a preference is that the receiving cluster, given a series of alternative dispositions for a given task, should use the one 'preferred' (as indicated by the preferences) by the requesting cluster. The cluster should determine through the ScoringFunction, the score for each alternative solution in that aspect space and recommend the solution with the best score. Where there are multiple preferences provided, the servicing cluster is expected to select the course of action that minimizes the weighted sum of all scores over all aspects.

Note that the preferences to not require, nor do they enable, the servicing cluster to find a solution that is optimal in the eyes of the customer. The service provider only has knowledge of its own business rules to determine what is possible: the preferences merely direct the cluster to select the best alternative.

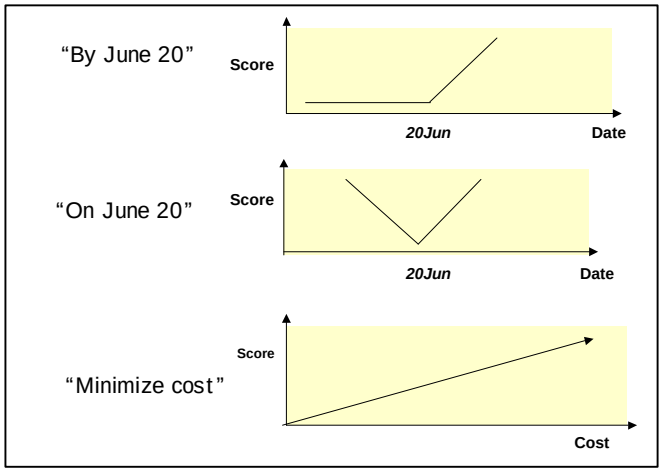

Figure 5. Sample ALP Scoring Functions used in Preferences.

3.2 Expressing Results

When a cluster has received a task and has dispositioned that task (allocated it to some resource), it must report these results back to the cluster that tasked it (the customer). The structure provided in ALP for this report is through an AllocationResult. The AllocationResult contains the details of the disposition of the task in each of the aspect dimensions for which preferences were expressed on the original task.

There are times that the results call for a phased expression of one aspect with respect to another. For example, a task may request receiving 20 objects by June 20 (QUANTITY required at 20, END_TIME required by June 20). In fact, fulfilling this request may have 10 objects arriving by June 10, 5 by June 15 and 5 by June 20. The AllocationResult contains summary information (QUANTITY = 20, END_TIME = June 20), as well as detailed phased information ([QUANTITY = 10, END_TIME = June 10], [QUANTITY = 5, END_TIME = June 15], [QUANTITY = 5, END_TIME = June 20]).

3.3 Dynamic Negotiation

Any PlanElement contains a pair of AllocationResults, labeled 'reported' and 'estimated'. The 'reported' AllocationResult represents the information received from 'below' (that is, reported from PhysicalAssets or OrganizationAssets to which the tasks have been allocated). The 'estimated' AllocationResult represents the information reported from the cluster 'upwards' (that is, to the cluster that allocated the task to you). This forms a natural reporting chain for providing the status and results through a complex processing path.

There are several points of note in this reporting chain. First, the 'reported' AllocationResult represents the abstraction of the state of a given task based, potentially, on synthesizing details from lower-level processing. The 'reported' AllocationResult represents the particular information and granularity that the reporting cluster chooses to report to its customer. In addition, the 'estimated' AllocationResult received from the service provider represents the best estimation of the ability to satisfy the request at any point in time and is subject to change over time as higher fidelity planning processes have time to complete. This last notion is especially important in that most planning processes are multi-pass, reporting 'quick-and-dirty' estimates initially and refining those estimates over time. This not only increases overall performance, but also allows a fast negotiation process to occur in the absence of complete plans. Through machine learning, these estimates can get grow increasingly accurate for routine operations.

The ALP infrastructure provides an automated mechanism to copy 'estimated' results from below (the 'tasked' cluster) to the 'reported' results above (the 'tasking' cluster). Whenever new significant information about the state of an allocation is available, it can and should be entered into the 'estimated' result. This filling in of the 'estimated' is the responsibility of the clusters and their PlugIns themselves to determine what to report upwards, what is a significant change and what is an unnecessary perturbation.

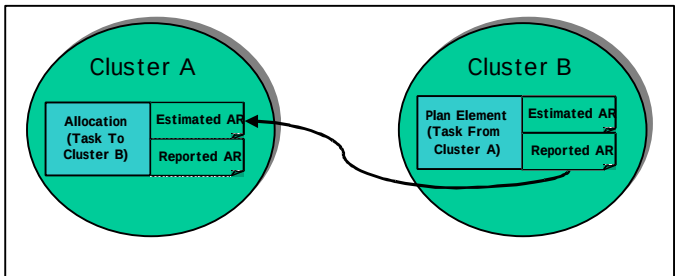

Figure 6. ALP copies 'reported' AllocationResult to 'estimated' AllocationResult of previous Allocation Plan Element

If we consider the full downward flowing chain of expansion and allocation that decomposes and dispositions a task, then, we see another backward flowing chain of allocation results flowing back up though the allocation results of the plan elements, from 'reported' to 'estimated'. Clusters should subscribe to the changes reported from below, and determine how and whether to report these changes up the chain.

There are some automated mechanisms provided by ALP to support this backflow. On any Expansion, there is an 'aggregation' function that takes the 'estimated' results on all subtasks, and merges them into a 'reported' result on the Expansion itself. Similarly, on any Aggregation, there is a 'distribution' function that takes the 'estimated' result on the supertask and breaks them into 'reported' results on all the input tasks.

3.4 Assessment

The ordinary context of 'reported' estimates is that of reports coming in from another cluster about the results of that clusters allocation of the original task request. However, any cluster that has allocated a task should be monitoring the 'reported' results and reacting accordingly. This behavior allows for ALP to implement a powerful yet efficient assessment feature from the same infrastructure as supports the inter-cluster dialogs. There are PlugIns whose role it is to monitor the internal consistency of PlanElements with respect to the state of asset resources. If some change happens in reality to a modeled resource, the state of that asset should change within the Cluster. The Assessor PlugIn should then question the validity of Allocations made against that asset. It doesn't have the 'smarts' at hand to understand whether this allocation is still valid, but it does have the mechanism to force this re-evaluation by the Allocator itself. By filling in the 'reported' result on the allocation of a task to an asset, the Assessor is triggering that it is reporting that this allocation is in question and should be re-evaluated. It is up to the Allocator to re-evaluate this allocation just as it would had it received this information from another cluster.

In other words, allocation is an act that needs continual monitoring; delegation to others can be unreliable, and so can relying on reality happening as planned. Either can produce results counter to objectives of the plan, and any well-written Allocation algorithm must anticipate and react to such contingencies.

3.5 Future Directions and Summary

There are many areas that the ALP is continuing to investigate alternate solutions in order to make its infrastructure more robust, efficient and expressive. A few issues under consideration for future ALP development may provide additional power in the expressiveness or efficiency of the negotiation process.

- **Auxiliary Queries**: The capacity to augment the Allocation/AllocationResult dialog with information about additional information about the results of an allocation that is not tied to a preference, nor quantifiable in an aspect. For example, I want to know what route you'll be taking, or what port you'll be using, or the name of the ship's captain, even if I have no preference in the matter.
- **Dynamic Aspects**: The capacity to manage arbitrary new aspects that can be negotiated between two clusters without prior knowledge of the aspect in the core aspect ontology of the ALP infrastructure.
- **Multiple Concurrent Independent Scenarios**: The capacity to handle parallel 'what-if' scenarios on top of the base 'reality' scenario and have the family of realities negotiated in a single dialog.
- **Traceability:** The enhancement of the dialog between clusters to contain 'intent' and 'expectation' semantics. That is, a customer can provide details to provide the context of why the request is being made, so the supplier can identify reasonable alternative paths. Further, the supplier can respond with accountability tracing, defining why a given result (success or failure) was given to a given request.
- **Confidence:** The enhancement of the infrastructure to specify levels of fidelity requested and provided for a given transaction. In this way, clusters can select different operating modes depending on operational requirements and available resources.

The ALP architecture is a general-purpose agent architecture for the construction of large-scale distributed agent systems. The development of logistics unique PlugIns has afforded the construction of a large military logistics command and control prototype for experimentation and demonstration. The architecture itself, without the sensitive military specific PlugIns, is available under OpenSource at: www.cougaar.org. In the OpenSource realm, ALP uses the name COUGAAR (Cognitive Agent Architecture) to emphasize the modeling of the human cognitive process inherent in the decomposition of organization processes into component functions as PlugIns.

The Advanced Logistics Program continues to enhance the depth and breadth of its model of various aspects of military logistics. In parallel with and in support of this effort, ALP continues to refine and extend the infrastructure for cluster interoperability to allow for more successful interoperations between clusters.

4. ACKNOWLEDGEMENTS

This work is sponsored by the Defense Advanced Research Projects Agency, Information Systems Office.

5. REFERENCES

[1] R. A. Brooks. "A Robust Layered Control System For A Mobile Robot", IEEE Journal of Robotics and Automation, Vol. RA-2, No. 1, March 86.

[2] M.G. Dyer. "Connectionism versus Symbolism in high-level cognition", Connectionism and the Philosophy of Mind, 1990.

[3] J. Fodor, Z. Pylyshyn. "Connectionism and cognitive architecture: A Critical Analysis", Cognition, 1988.

[4] S. Harnad. "Symbols and Nets: Cooperation vs. Competition", Connections and Symbol Connection Science, 1990.

[5] S. Harnad. "Connecting Object to Symbol in Modeling Cognition", Connectionism in Context, 1992.

[6] S. Koeing, R. Simmons. "Risk-Sensitive Planning", AAAI 1994 SSS Workshop on Decision-Theoretic Planning, AAAI Press, TR SS-94-06.

[7] A. Newell, "SOAR - State, Operator and Result". The SOAR Papers, P.S. Rosenbloom et. al. eds., MIT Press 1993.