A Hybrid Hierarchical Schema-Based Architecture for Distributed Autonomous Agents

Matthew Gillen, Arvind Lakshmikumar, David Chelberg, Cynthia Marling, Mark Tomko and Lonnie Welch
School of Electrical Engineering and Computer Science
Ohio University, Athens, OH 45701 USA

Abstract

The advent of inexpensive computers has spurred interest in distributed architectures, in which a cluster of low-cost computers can achieve performance on the same scale as expensive super-computers. The goal of research in distributed systems is to take a highly decomposable problem solution, and put that solution into a framework that allows the system to take full advantage of the solution's decomposability. An architecture that allows additional processors to be taken advantage of with little or no manual reconfiguration would be ideal.

Artificial Intelligence (AI) algorithms for planning in non-trivial domains are typically resource-intensive. We believe that a framework for developing planning algorithms that allows an arbitrary level of decomposition and provides the means for distributing the computation would be a valuable contribution to the AI community.

We propose a general, flexible, and scalable hierarchical architecture for designing multi-agent distributed systems. We developed this architecture to facilitate the development of software for soccer-playing robots. These robots will compete in the international robot soccer competition RoboCup. All of our software, from the high-level team strategy planning down to the control loop for the motors on our robots fits into this architecture.

We build on the notion of deliberative and reactive agents, forming a hierarchy of hybrid agents. The hierarchy covers the entire continuum of hybrid agents: agents near the root of the hierarchy are mostly deliberative, while agents near the leaves are almost purely reactive.

Introduction to Multi-Agent Systems

A multi-agent system can be thought of as a group of interacting agents working together to achieve a set of goals. To maximize the efficiency of the system, each agent must be able to reason about other agents' actions in addition to its own. A dynamic and unpredictable environment creates a need for an agent to employ flexible strategies. The more flexible the strategies however, the more difficult it becomes to predict what the other agents are going to do. For this reason, coordination mechanisms have been developed to help the agents interact when performing complex actions requiring teamwork. These mechanisms must ensure that the plans of individual agents do not conflict, while guiding the agents in pursuit of the goals of the system.

Agents themselves have traditionally been categorized into one of the following types (Woolridge and Jennings 1995):

- Deliberative
- Reactive
- Hybrid

Deliberative Agents

The key component of a deliberative agent is a central reasoning system (Ginsberg 1989) that constitutes the intelligence of the agent. Deliberative agents generate plans to accomplish their goals. A world model may be used in a deliberative agent, increasing the agent's ability to generate a plan that is successful in achieving its goals even in unforeseen situations. This ability to adapt is desirable in a dynamic environment.

The main problem with a purely deliberative agent when dealing with real-time systems is reaction time. For simple, well known situations, reasoning may not be required at all. In some real-time domains, such as robotic soccer, minimizing the latency between changes in world state and reactions is important.

Reactive Agents

Reactive agents maintain no internal model of how to predict future states of the world. They choose actions by using the current world state as an index into a table of actions, where the indexing function's purpose is to map known situations to appropriate actions. These types of agents are sufficient for limited environments where every possible situation can be mapped to an action or set of actions.

The purely reactive agent's major drawback is its lack of adaptability. This type of agent cannot generate an appropriate plan if the current world state was not considered a

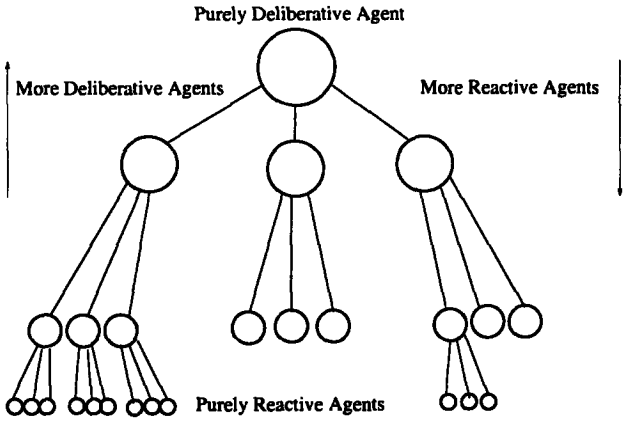

Figure 1: Agent Hierarchy

priori. In domains that cannot be completely mapped, using reactive agents can be too restrictive (Mataric 1995).

Hybrid Agents

Hybrid agents, when designed correctly, use both approaches to get the best properties of each (Bensaid and Mathieu 1997). Specifically, hybrid agents aim to have the quick response time of reactive agents for well known situations, yet also have the ability to generate new plans for unforeseen situations.

We propose to have a hierarchy of agents spanning a continuum of deliberative and reactive components. At the root of the hierarchy are agents that are mostly deliberative, while at the leaf nodes are agents that are completely reactive. Figure 1 shows a visual representation of the hierarchy.

A Hybrid Hierarchical Architecture

The Test Domain for the Architecture

This architecture was developed for the Robobcats (Chelberg et al. 2000; 2001) to cope with the demands of competing in the small-size RoboCup robot soccer league (Robocup Federation 2001). As such, examples and analogies will relate to that domain. The key features in the domain of RoboCup as it relates to this architecture are as follows:

- One or more overhead cameras provide the world view
- All robots on the field must be untethered (e.g., use wireless communication)
- Any number of computers off the field may provide computing power
- Once the game starts, no human interaction may help or guide any robot
- The exact state of the world is not obtainable, nor is it possible to entirely predict the opponent's future actions, making long-term planning difficult

We believe that this domain incorporates many of the issues present in real world planning problems, and therefore is an ideal testbed for our architecture.

A domain like RoboCup requires coordination to achieve the full benefit of applying multiple agents to the problem. In addition to the regular hardware/software problems faced by mobile robotic systems, soccer playing robots also need to cope with communication delays, and vision system imperfections and inconsistencies. Within such systems, each agent has a responsibility to carry out tasks that must benefit the whole team. When a team fails to perform well, it becomes difficult to assess and analyze the source of the problem. The agents might have incorrect individual strategies or the overall team strategy may not be suitable. (Mataric 1998) and (Boutilier 1996) have suggested communication as a possible solution to resolve these problems in certain multi-agent environments. For our domain, extensive communication would result in unacceptable delays between changes in the world state and appropriate responses. Our architecture is designed to minimize communication among agents to keep the required bandwidth low.

The Schema

We use the term *goal* to mean a desired world state. A goal could imply a partial world state that is desired, for example: some goal may be accomplished if the ball reaches some position on the field irrespective of where all the players are. According to Webster's dictionary, a *schema* is "a mental image produced in response to a stimulus, that becomes a framework or basis for analyzing or responding to other related stimuli". We will use the term *schema* in the same spirit, but define it more concretely by saying that it is a strategy for achieving one or more goals. An ideal schema generates a plan, based on the current world state, that has the highest desirability for achieving some subset of the goals the schema was designed to accomplish. Desirability is a function of the utility of the goals and the probability of success of the plan. We care about the probability of success, because even an ideal strategy producing an optimal plan cannot guarantee success in a world that cannot be completely modeled.

The Hierarchy of Schemas

The plan a schema generates consists of either a simple action or a sequential execution of other schemas' plans. Schemas are designed hierarchically; the lowest level schemas perform simple actions such as moving a robot to a specified destination. A higher level schema may use any schema that falls strictly below it in the schema hierarchy in order to generate a plan. For example, a higher level schema might use the 'move to destination' schema several times sequentially to maneuver a robot around an obstacle.

When a parent schema tells a child schema (that is part of the parent's plan) to start executing its plan, a new thread

of control is started for the execution of that plan. The parent's thread of execution does not stop, however. The parent must continue to monitor the world state, in case the plan it began executing becomes non-optimal. Thus, the higher level 'avoid obstacle' schema may interrupt the execution of 'move to destination' if the obstacle moves unexpectedly. Continual monitoring of a plan's optimality at all levels of the hierarchy is a key part of this architecture.

Given this hierarchy of schemas (strategies), each schema is only responsible for optimality with respect to its own goals. The goals of the parent schema need not be considered, since it is assumed that the parent is optimizing its own goals. The only goals any given schema needs to consider are its own goals and the goals of its children.

This encapsulation helps to make the system manageable. If any one schema A seems to do too much, its goals could be split into smaller pieces, and the schema divided into two levels: a high level A_H that keeps the same parents of A and manages the more abstract goals, and a lower level A_L that keeps the same children as A, but has A_H as a parent. A sanity check of A_H's parents to ensure that A_H is still being used correctly should not be necessary, since A_H provides the same functionality that A did, and A_H's parents didn't depend on anything that A used in its implementation (e.g., A's children).

An *Agent* can be composed of one or more schemas. What constitutes the boundaries of an agent in this system is still an open question. We use the agent to represent a process, an atomic unit that may be distributed among the available resources. How many schemas a given agent contains is up to the designer. For maximum flexibility, a designer may decide that each schema is an agent. Or, if distributing the computation is not as important, the designer may group the top half of the hierarchy into one agent and the rest of the hierarchy into a set of agents.

The first implementation of our RoboCup system provides a concrete example. We grouped all schemas considering multiple robots into a single Meta-agent (e.g., formations, passing). Schemas that only dealt with one robot (e.g., roles in a formation) were grouped into a Player-agent. Finally, all schemas for interfacing with the hardware on the robots composed the Robot-agent. There was an instantiation of Player-agent and Robot-agent for each of our five physical robots.

Decision Making and its Role in the Architecture

When schemas have to make decisions in a dynamic environment, they have to to take into account the fact that their actions may have several outcomes, some of which may be more desirable than others. They must balance the potential of a plan achieving a goal state against the risk of producing an undesirable state and against the cost of performing the plan. Decision theory (French 1988) provides an attractive framework for weighing the strengths and weaknesses of a particular course of action, with roots in probability theory and utility theory. Probabilistic methods provide coherent prescriptions for choosing actions and meaningful guarantees of the quality of these choices. While judgments about the likelihood of events are quantified by probabilities, judgments about the desirability of action consequences are quantified by utilities.

Given a probability distribution over the possible outcomes of an action in any state, and a reasonable utility (preference) function over outcomes, we can compute the expected utility of each action. The task of the schema then seems straightforward – to find the plan with the maximum expected utility (MEU).

Desirability Analysis: An Example

Intuitively, the goals of a schema can be thought of as having a context-free utility that is the same across all schemas with that goal, and some context-dependent utility that is specific to the particular schema implementation. The context-free utility is used to imply that in general, pursuing some goal with a high context-free utility is better than some other goal (with a lower context-free utility). It can also be interpreted as a way for expressing heuristics about the relationships between goals. For example, if we take the old adage that "the best defense is a good offense" as a general heuristic for accomplishing the goal of winning, then the goal of **scoring** would have a higher context-free utility compared to the goal of **preventing opponent scores**.

The context-dependent utility refers to the utility of the goal in the context of the strategy that the schema was designed to implement. A strategy may be implemented to achieve some set of goals H. As a by product of achieving those goals, it may achieve some other set of goals L. However, since the main focus of the strategy is H and not L, the goals within H will have a context-dependent utility much greater than those goals which are a part of L. There are also other ways of ordering the relative context-free utilities of the set of all goals that a schema achieves. This is just one example.

The important notion is that the ordering of context-dependent utilities does not matter outside of the context of the schema. All context-dependent utilities are local to the schema that defines them, and are never propagated to parents or children. So how does a parent decide whether a child schema is an appropriate strategy for accomplishing its goals?

An example is instructive. Assume a schema, A_S, is at the root node in the hierarchy, and that A_S's children are represented by the set $C_{SS} = \{C_{S1}, C_{S2}\}$ ($_S$ denotes a schema, $_{SS}$ denotes a set of schemas). $G(A_S)$ will represent A_S's goals, and $G(C_{SS})$ will represent the union of the goals of the elements of C_{SS}. Each child's goals represent some subset of the possible ways of achieving one or more elements of $G(A_S)$. Assume $G(A_S)$ has one element: **win** the game of soccer. Then $G(C_{SS})$ could consist of **score** and **prevent opponent score** , which could be thought of as components

of the goal win. For A_S to determine the best way of achieving $G(A_S)$, it must decide which child will be permitted to execute its plan. A_S ranks its children based on the *desirability* of each strategy. Desirability is a function of the *utility* of achieving elements of $G(C_{SS})$ with respect to $G(A_S)$ and the *feasibility* of the instantiated plan achieving $G(A_S)$.

We've discussed what the utility of a goal represents, so we'll move on to the feasibility. Feasibility is a measure of how achievable a goal is given the current situation, irrespective of how useful accomplishing that goal would be. To estimate how achievable a goal is in the current situation, it is necessary to have a plan already instantiated. The feasibility is dependent on three things: a plan, a goal, and a current world state.

So, to rank the elements of C_{SS} in A_S, A_S asks each element of C_{SS} the feasibility of its accomplishing each element of $G(C_{SS})$. For any child to compute the feasibility of achieving a given goal, the child must have a plan. Each child in turn develops a plan by asking its children about the feasibility of accomplishing their goals. This recursive structure allows accurate prediction of feasibility. Once the feasibility of a child's plan achieving each goal ($F(c_j(p), g)$) is determined, a ranking of the children according to desirability is possible.

We will now describe one way of computing desirability, although other algorithms are certainly possible. Given the utility of each goal in $G(C_{SS})$, $U(g_n)$, and a feasibility for each $\{child, goal\}$ pair $F(c_j, g_i)$, we could define desirability of a given child c_j as

$$D(c_j) = \sum_{i=0}^{n} U(g_i) * F(c_j, g_i) \quad (1)$$

We will call the most desirable child c_D. If we now assume that A_S is not the root node, when asked about the feasibility of each element k of $G(A_S)$ by one of A_S's parents, A_S would return a value computed by considering the feasibility of c_D accomplishing g_k.

Related Work

Choosing the best plan for a given strategy entails selecting child schemas (actions) that are appropriate to the current situation. Many action selection mechanisms for robot control have been discussed in the literature. At the highest level, these mechanisms could be divided into two main categories, *arbitration* (Ishiguro et al. 1995) and *command fusion* (Rosenblatt 1995). Arbitration mechanisms can be divided into: *priority-based*, *state-based* and *winner-take-all*. The subsumption architecture (Brooks 1986), is a priority-based arbitration mechanism, where behaviors with higher priorities are allowed to subsume the output of behaviors with lower priority. State-based arbitration mechanisms include *Discrete Event Systems (DES)* (Kosecka and Bajcsy 1993), a similar method called *temporal sequencing* (Arkin 1992), and action selection based on *Bayesian decision theory* (Hager 1990). The advantage of the DES and the temporal sequencing approaches is that they are based on a finite-state machine formalism that makes it easy to predict future states from current ones. However, this modeling scheme is known to be NP-hard. Finally, in winner-take-all mechanisms, action selection results from the interaction of a set of distributed behaviors that compete until one behavior wins the competition and takes control of the robot. An example is the activation network approach (Maes 1990), where it is shown that no central bureaucratic module is required.

Command fusion mechanisms can be divided into *voting*, *fuzzy* and *superposition* approaches. Voting techniques interpret the output of each behavior as votes for or against possible actions. The action with the maximum weighted sum of votes is selected. Fuzzy command fusion methods are similar to voting. However, they use fuzzy inferencing methods to formalize the voting approach. Superposition techniques combine behavior recommendations using linear combinations.

Conclusions

We have presented an architecture employing a hierarchy of hybrid agents. The advantages of this architecture are many. It allows the planning process to be distributed across any available computing platforms. Additionally, the required communication among agents is minimized by communicating feasibilities of plans, not plan details. This architecture allows an arbitrary decomposition of the planning problem, while introducing minimal overhead. This architecture facilitates the integration of middle-ware based resource managers, like DeSiDeRaTa (Welch et al. 1998), to ensure optimal use of resources in dynamic environments.

We will implement this architecture as part of our RoboCup effort this year. We expect to achieve low communications latencies while utilizing several off-field platforms to control a group of five robots playing soccer. This uniform architecture provides an ideal framework to build planning agents at different levels of abstraction, from team-level strategy to robot-level motor control. The uniformity of our approach to distributed hierarchical planning lends itself to rapid software development, and the ability to more easily integrate the efforts of our team of programmers.

Acknowledgments

This work has been partially supported by the NASA CETDP grant "Resource Management for Real-Time Adaptive Agents" and by the Ohio University 1804 Research Fund. The authors would like to thank the entire Robobcats team without whose support and contributions this work would not be possible.

References

Arkin, R. 1992. Integrating behavioral, perceptual, and world knowledge in reactive navigation. *Robotics and Au-*

tonomous Systems 6(1):105–122.

Bensaid, N., and Mathieu, P. 1997. A hybrid architecture for hierarchical agents.

Boutilier, C. 1996. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*, 195–210. San Mateo, CA: Morgan Kaufmann.

Brooks, R. A. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* RA-2(1):14–23.

Chelberg, D. M.; Gillen, M.; Zhou, Q.; and Lakshmikumar, A. 2000. 3D-VDBM: 3D visual debug monitor for RoboCup. In *IASTED International Conference on Computer Graphics and Imaging*, 14–19.

Chelberg, D. M.; Welch, L.; Lakshmikumar, A.; Gillen, M.; and Zhou, Q. 2001. Meta-reasoning for a distributed agent architecture. In *Proceedings of the Southeastern Symposium on System Theory*, 377–381.

Cooper, G. 1990. The computational complexity of probabilistic inference using bayesian networks. *Artificial Intelligence* 42(2-3):393–405.

French, S. 1988. *Decision Theory*. Ellis Horwood, Chichester, West Sussex, England.

Ginsberg, M. 1989. Universal planning: An (almost) universally bad idea. *AI Magazine* 10(4):40–44.

Hager, G. D. 1990. *Task directed sensor fusion and planning: A computational approach*. The Kluwer International Series in Engineering and Computer Science. Boston: Kluwer Academic Publishers.

Ishiguro, A.; Kondo, T.; Watanabe, Y.; and Uchikawa, Y. 1995. Dynamic behavior arbitration of autonomous mobile robots using immune networks. In *IEEE International Conference on Evolutionary Computing(ICEC)*, 722–727.

Kosecka, J., and Bajcsy, R. 1993. Discrete event systems for autonomous mobile agents. In *Workshop on Intelligent Robot Control*, 21–31.

Maes, P. 1990. How to do the right thing. *Connection Science Journal, Special Issue on Hybrid Systems* 1(3):291–323.

Mataric, M. 1995. Issues and approaches in the design of collective autonomous agents. *Robotics and Autonomous Systems* 16(2-4):321–331.

Mataric, M. 1998. Using communication to reduce locality in distributed multiagent learning. *Journal of Experimental and Theoretical Artificial Intelligence* 10(3):357–369.

Noda, I., and Matsubara, H. 1996. Learning of cooperative actions in multi-agent systems: A case study of pass play in soccer. In *Working notes for the AAAI Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, 63–67.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, California: Morgan Kaufmann.

Robocup Federation. 2001. Robocup Overview. World Wide Web site, http://www.robocup.org/overview/2.html.

Rosenblatt, J. 1995. Damn: A distributed architecture for mobile navigation. In *AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents*.

Welch, L.; Shirazi, B.; Ravindran, B.; and Bruggeman, C. 1998. DeSiDeRaTa: QoS management technology for dynamic, scalable, dependable real-time systems. In *Proceedings of The 15th IFAC Workshop on Distributed Computer Control Systems*.

Woolridge, M., and Jennings, N. 1995. Intelligent agents: Theory and practice. *Knowledge Engineering Review* 10(2):115–152.