

# Meta-Reasoning For a Distributed Agent Architecture

Dr. David Chelberg, Dr. Lonnie Welch, Arvind Lakshmikummar, Matthew Gillen, Qiang Zhou  
Laboratory for Intelligent, Real-Time and Secure Systems  
Department of Electrical Engineering and Computer Science  
Ohio University, School of Engineering and Technology.  
Athens, OH 45701

*Abstract*— Agent based computing offers the ability to decentralize computing solutions by incorporating autonomy and intelligence into cooperating, distributed applications. It provides an effective medium for expressing solutions to problems that involve interaction with real-world environments and allows modelling of the world state and its dynamics. This model can be then used to determine how candidate actions affect the world, and how to choose the best from a set of actions. Most agent paradigms overlook real-time requirements and computing resource constraints. In this paper, we discuss the application of agent based computing to RoboCup and examine methods to improve it. In particular, we discuss the incorporation of a meta-level reasoning mechanism that handles individual agent organization, plan generation, task allocation, integration and plan execution. We also propose an architecture where a meta-agent is further enhanced by combining it with system-level resource allocation and optimization. The approach adopted by us unifies agent based computing with adaptive resource management for dynamic real-time systems. The goal is to build and implement a distributed, intelligent, agent based system for dynamic real-time applications.

## I. INTRODUCTION

In this paper, we propose an architecture that allows cooperation of multiple, intelligent agents to solve a problem under dynamic, real-time conditions. In particular, we are targeting the RoboCup Challenge as a test-bed for our research. The RoboCup challenge represents a standard problem on fast-moving multiple robots, which collaborate together to play a soccer game [7]. Past approaches have been monolithic in their design, wherein an agent (an individual soccer player) is a local process or a group of processes on a single machine. During a game of soccer, computing needs may increase depending on the state of the game. If these processes cannot migrate to a less loaded processor, a single machine could get overloaded causing an agent to fail to complete its computations in the desired time. In general, without some means of dynamic resource management, there is an inefficient allocation of resources. Our approach to achieve this objective can be categorized into two main areas, distributed agent based computing and resource management under dynamic, real-time constraints. This paper discusses the development of agents that are cognizant of the real-time constraints and resources. To ease the management of these agents, we propose the development of meta-agents that manage the time and resources for these agents.

## II. REAL-TIME SYSTEMS

A real-time system may be characterized as being static or dynamic. A static system is one whose resource requirements do not change with changes in its environment. A dynamic

system [15] is one whose resource requirements change unpredictably at runtime. This characterization is based on the temporal properties and execution behavior of the system. The majority of real-time computing research has focused on systems whose requirements are evaluated statically. However, RoboCup is a highly dynamic environment. This precludes the accurate characterization of the environment and its properties by static models. In such contexts, temporal and execution characteristics can only be determined accurately by empirical observation or experience (i.e., a posteriori). In most real-time computing models, the execution time of a job is used to characterize workload statically as an integer worst-case execution time [3], [9], [6]. This is often difficult and sometimes impossible. The DeSiDeRaTa (Dynamic, Scalable, Dependable Real-Time System)[15] middleware was designed to address these shortcomings. One of the fundamental innovations of DeSiDeRaTa is the dynamic path paradigm, that is employed for modeling and resource management of distributed, real-time systems. The primary metric used in DeSiDeRaTa is the Quality of Service (QoS) that is a measure of the path latency (end-to-end). The application programs of real-time control paths send time-stamped events to the QoS metrics component. This component is responsible for computing the path-level QoS metrics (like temporal properties and execution behavior) and sends them to the QoS monitor. The monitor checks the deviation of observed QoS from the required QoS, and notifies the QoS diagnosis component when a violation occurs. The diagnoser notifies the action selection component of the cause(s) of poor QoS and recommends actions (e.g., move a program to a different host or LAN, shed a program, or replicate a program) to meet QoS requirements. Action selection ranks the recommended actions, identifies redundant actions, and forwards the results to the allocation analysis component; this component consults resource discovery for host and LAN load index metrics, and determines an efficient method for allocating the hardware resources to perform the actions, and requests that the actions be performed by the allocation enactment component. This entire module serves as a Resource Manager (RM) for a distributed application. In this paper, we discuss the integration of one such resource manager with an intelligent agent.

## III. AN IDEAL, RATIONAL, REAL-TIME AGENT

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon the environment

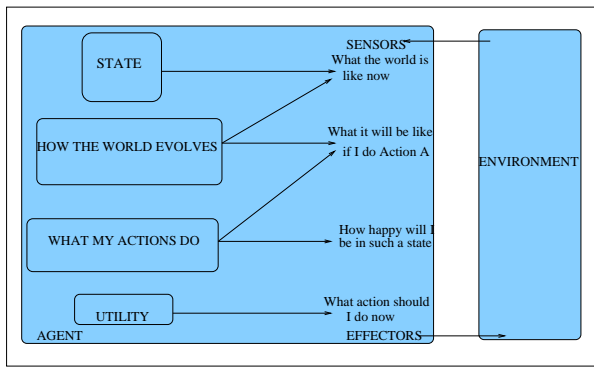


Fig. 1. A complete utility-based agent (from [1])

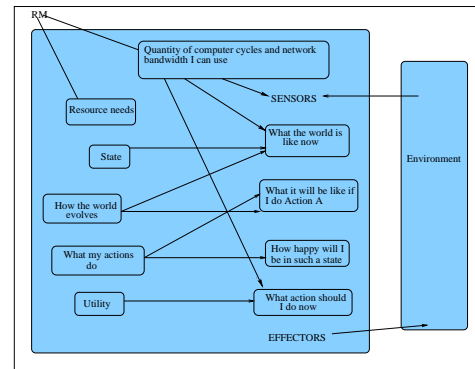


Fig. 2. Structure of an ideal, real-time agent

through effectors; and a rational agent is one that "does the right thing" [1]. Figure 1 shows the overall structure of a rational agent that determines its course of action by applying utility theory [6]. In [1] an ideal rational agent is defined as one that does whatever is expected to maximize a performance measure. Practical considerations such as computational complexity and space complexity of "ideal" algorithms are often prohibitive. Real-time requirements, acting in an environment and computing resource constraints often require that absolute perfection be abandoned for reasonable, timely responses. Unfortunately, no framework exists for reasoning about how "good" a solution an agent can obtain, subject to such constraints. An ideal, rational, real-time agent is one that is cognizant of when it must select an action. Its determination of what the world is like, and what it could be like given actions by its effectors, is bounded by timing requirements (e.g., when a response must be made to a condition in the environment in order to be effective). Another aspect of which the agents should be aware is computing resources. To determine if a set of analyses is feasible, it is necessary to know computing resource needs and computing resource availability. Figure 3 illustrates the integration of resource-awareness into an agent. The resource needs of an agent performing in a dynamically changing environment (like a soccer field) change constantly. As needs change, an agent can gracefully adapt by acquiring additional resources from a RM. In addition to this, if the resources of a machine are over worked, a RM moves applications to machines that are underutilized.

#### IV. MULTI-AGENT SYSTEMS

A multi-agent system may be regarded as a group of ideal, rational, real-time agents interacting with one another to collectively achieve their goals. To achieve their goals, each one of these individual agents needs to be equipped to reason not only about the environment, but also about the behavior of other agents. Based on this reasoning, agents need to generate a sequence of actions and execute them. In a heterogeneous environment like soccer, each agent needs to have a strategy to solve a particular problem. In this context, a strategy refers

to a decision-making mechanism that provides a long term consideration for selecting actions to achieve specific goals. Each strategy differs in the way it tackles the solution space of the problem. The presence of multiple agents necessitates the need for a different treatment. We need a coordination mechanism that handles the interaction between the agents. This mechanism is responsible for the implementation of the agents' actions and also the planning process that goes into the implementation. Traditionally, depending on their approach to solving a problem of this nature, agents have been divided into three main categories [12]

- Deliberative
- Reactive
- Hybrid

##### A. Deliberative Agents

In traditional AI approaches, an agent is made up of three parts: sensors, a central symbolic reasoning and planning system and effectors. The central processing system uses symbolic reasoning to decide on an appropriate sequence of actions, or plan. The plan is then executed, step by step, by the effectors. The problem with planning systems is that they don't scale very well when the complexity of the problem increases and they can't react well in real time. One reason for this is that that world may change while the system is doing its, often slow, reasoning or during plan execution. Another is that plans rely on being able to predict the outcome of a series of actions, but the world may not always behave as predicted. The main backbone of this design is a central reasoning system [11] that accounts for the reactivity of the agent. This is not appropriate for a soccer game. If we relied on a central reasoning system, our reasoning to action time would not be short enough. Normally, if we were to design a soccer playing system with this architecture, we would need to assign some roles to each agent along with its associated plans. For example, a plan for a player in the offense (role) would be to move to a position in front of a goal, turn towards the ball until he gets close to it and then kick it. There are a lot of problems with this approach [13]. For instance, if the player on offense

is unable to kick the ball, the entire team strategy fails. Also, if the ball is kicked towards this player, he has no corresponding plan to handle it. Even if he did have a backup plan to handle a situation like that, he does not have the time to react to it. This leads us to consider a second kind of architecture, where the agent has no predefined plans and is completely reactive.

### B. Reactive Agents

These agents maintain no internal model of the current situation. An action is chosen by referencing a lookup table of situation-action pairs. These kind of agents have been found to be very effective for well defined problems. However incorporating them in a system that requires run-time flexibility or goal directed behavior is very difficult [10] Also, this approach makes sense when the environment does not change very much. In a soccer game, a forward does not wait for the ball to roll upto him. Instead, he is proactive and runs towards the ball to kick it. To draw a parallel with the analogy for the deliberative agent let us consider what would happen if a ball were to land up at a player's feet, when he least expects it. If this were handled by a situation-action pair, then the player would most likely kick the ball. This may not be beneficial in the long run.

### C. Hybrid Agents

To overcome the weaknesses described above, a combination of reactive and deliberative agents is used [4]. Here, a system is divided into two layers, one layer does high-level reasoning, the other handles the interaction with the real world. The upper level is normally a deliberate planning system while the lower level is more reactive or behavior based. The kind of agent that would be appropriate for a soccer playing agent would be a hybrid agent whose reactivity is determined by its behavior. An important feature of the hybrid agent used by us is resource awareness. Every individual agent is fully cognizant of the resources available to them and their individual decisions are bounded by that constraint.

## V. META-AGENTS

To be able to design a system to handle the challenges discussed above, we need to do the following things

- Design a representation for individual agent actions and have a method to evaluate them
- Have an agent that is superior to the existing agents and is capable of reasoning about them
- Enable adaptability to improve the decision making required to select a strategy

To achieve these objectives, and allow coexistence of multiple agents, we propose the development of a reusable meta-agent that manages these agents. The goals of this meta-agent are

- Perform reasoning (to reason about agent autonomy): This is going to decide how much cooperation there needs to be between the agents
- Planning: Plan actions

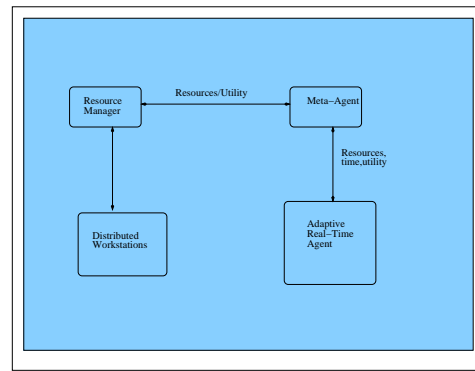


Fig. 3. A Meta-Agent framework

- Individual Agent modeller: Maintain individual agent state information
- Other Agent modeller: Maintain information on other agents and attempt to predict future behavior
- Conflict Manager: Classify conflicts and resolve them

Meta-agents are a way to help agents observe the environment, evaluate alternatives and prescribe and schedule actions. In addition, strategies can be formulated and implemented not only within an agent but also among a group of agents. For any given problem, various strategies may be available. The main role of the meta-agent is to sift through these strategies and make intelligent decisions. Each agent will consult the meta-agent prior to performing analyses; the agent will state the desired analyses and the utility of performing them. Subsequently, the meta-agent will determine the feasibility of performing the actions by considering timing requirements and resource constraints. In addition, extra resources may be requested from the resource manager.

For the meta-agent to make decisions about the individual agents, we need some way to assess the capabilities of the agents and thereby classify them. The following agent attributes allow us to classify them uniquely

- Data/Knowledge contained within an agent
- Agent dependencies
- Agent behavior: Set of states, events and transitions

So far, we have established the necessity for a meta-agent and enumerated its responsibilities. The next section deals with the actual design of one such meta-agent.

### A. Meta-agent design

The meta-agent decides on a strategy [14]. It determines what behaviors of the agents would achieve this strategy and accordingly triggers those behaviors. By triggering only the behaviors appropriate to the current strategy, the meta-agent also reduces behavior-behavior interactions. This is also in accordance with the layered architecture we are trying to build. Higher level layers implement more abstract behaviors by selecting and activating the appropriate behaviors from the next level. The higher levels provide the strategic reasoning while

the lower level provides reactivity to the system. This leads us to the two most important questions the meta-agent needs to answer

- How to choose an appropriate strategy?
- How to characterize agent behaviors?

### B. Behavior mechanism of a single agent

A behavior [8] may be thought of as a unit that controls an action. A group of behaviors combined in the right manner achieve a strategy. Each layer in our system implements a complete behavior based module. There can be a number of generic behaviors (each with its associated global information), a set of active behaviors and processes that control these behaviors. Generic behaviors are fixed but active behaviors change. A particular behavior is created by instantiating it from a generic behavior with a specific parameter. The best possible behavior is achieved by combining the applicability of a behavior and its priority. Priority of a behavior is a measure of its importance or desirability. There is no central, complete or symbolic description of the world. The information extraction process takes incoming percepts, extracts relevant information for each generic behavior's world information (performing some simple consistency checking to ensure the reliability of the stored data), then passes the information to the behavior. The behavior instantiation process receives a command from the layer above indicating a new set of behaviors, with associated priorities, that are to be made active. The behavior instantiation process destroys the current active behaviors and creates a new active behavior set, whenever the upper layer active behavior changes. The active behaviors in the topmost layer are constant for the life of the agent.

A generic behavior provides two functions, a control function, for issuing a command, and an applicability function through which the behavior indicates its relevance in light of the current state of the world. A generic behavior is designed to control a very specific action without regard to any other actions the agent may execute. Each generic behavior has associated with it some simple information about the world which has been extracted from the incoming percepts. Information for higher level behaviors is at a more abstract level than for low level behaviors. For example defense, an upper level behavior, may record that the ball is in the opposite half of the field, while Ball-intercept, a lower level behavior, may record that the ball is 45 degrees to the left. The use of parameters means that the same generic behavioral code can be used many times with varying effects. For example, move may be used with a number of different positions. One generic behavior may be part of several behaviors simultaneously, each using different parameters.

A behavior issues commands and calculates its applicability without regard to the applicability or priority of other behaviors in the system. The meta-agent ensures that the resulting interactions between behaviors yield the best required. In a nutshell, the three main factors the meta-agent uses to choose

a particular behavior are applicability of the behavior, its priority and the duration of the behavior's existence.

High level behaviors issue messages to the next layer describing the low level behaviors required to achieve the high level strategy. For instance, a high level behavior could be *Launch offense along right wing*. These would lead low level behaviors like *move to ball*, *move to right wing*, *pass along right wing* and *kick ball* to be instantiated with corresponding priorities. Sometimes adequate time and/or resources are not available to perform the desired analyses. In these situations, the meta-agent and the individual agents negotiate to find a satisfactory compromise. This kind of negotiation and strategic decision-making takes into consideration the following constraints

- Strategy imposed requirements
- Cost of executing a particular strategy
- The quality of the solution provided by the strategy

Each agent has multiple objectives. The difficulty of the problem lies in the fact that there is no single objective solution that can be obtained for all the objectives put together. So, any kind of reasoning is at best a trade-off. This could lead to circumstance-dependent solutions, which are generally a compromise. RoboCup provides an ideal test-bed to implement these agent models. It is a complex system where there are multiple clients competing for resources. The next section describes the architecture adopted by us to ensure the optimal distribution of resources among the various competing clients.

## VI. OUR ROBOCUP ARCHITECTURE

The architecture proposed by us is applicable to both the simulator league and the small sized robot league. The existing RoboCup client models do not adequately take into account the real-time constraints faced by the soccer agents on the field. This leads to an overworked and sub-optimal system. To perform these activities under real-time conditions, an architecture is proposed that puts the entire RoboCup subsystem under the control of a resource manager. This manager monitors the performance of various players on the field and balances the processor load based on their capacities. In the proposed model, we introduce the concept of a *Client manager* that resides in a level between the server and the clients. This client manager, based on the input from the resource manager decides the resources needed by every client. The client manager communicates with the server using the libraries native to the server and with the clients using DeSiDeRaTa's communication libraries.

The architecture proposed by us is a layered one. This is similar to a subsumption architecture [5], [2]. Figure 4 illustrates this architecture and the various layers. Level 0 has only two functions - detect/avoid obstacles and gather statistics. These statistics indicate the number of collisions that occurred and the number of exceptions sent back to level 1. If there are no obstacles, the player continues on his current path, else he throws an exception to level 1. Level 1 deals with the

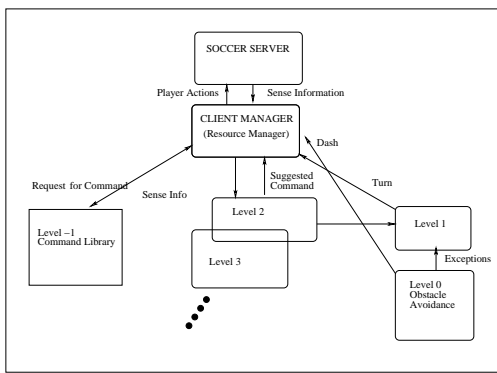


Fig. 4. Layered Architecture

factors that affect the action and performance of the client. This includes object tracking and exception handling from level 0. Level 2 pertains to the individual planning adopted by the clients. Some of the actions we identified under this category were the actual strategy planning (offense, defense and goalie), player communication, utility of reasoning and prediction. In level 3, we perform the action evaluation of the clients. This includes modeling the opponent, sharing the knowledge learned and computing the resources needed to reason. All the levels following level 3 would follow the same communication path.

The main advantage of this scheme is that the clients may be distributed across different machines. The resource manager would still be managing both the CPU and the network loads. At an implementation level, our system has three main modules, the meta-agent, the player agent and a database of the world. Table 1 summarizes the tasks performed by each of these.

<b>Meta-Agent</b>
Achieves high-level goals like <i>win</i> or <i>score</i> Analyzes game state Coordinates team oriented skills Performs opponent modelling
<b>Autonomous player agents</b>
Fill a particular role in a formation Execute instructions from the meta-agent Recieve updates from the vision system and update plans
<b>World View database</b>
Provides current vision information Keeps a history Predict opponent movements

## VII. CONCLUSIONS

The main highlight of our architecture is its uniqueness. Resource management is vital to dynamic systems operating in real-time. The system developed by us balances the load between the different machines running the clients so as to en-

sure real-time constraints and achieve the most efficient allocation of computational resources. The agents developed by us are fully cognizant of the resources available to them and the constraints of operation. These agents are efficiently controlled by the meta-agent which manages their time and resources.

## REFERENCES

- [1] *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.
- [2] Eyal Amir and Pedrito Maynard-Reid II. Logic based subsumption architecture. In *Sixteenth International Joint Conference on Artificial Intelligence*, pages 27–36, June 1999.
- [3] T.P. Baker. Stack-based scheduling of real-time processes. *Journal of Real-time Systems*, 3, March 1991.
- [4] Nourredine Bensaid and Philippe Mathieu. A hybrid architecture for hierarchical agents. pages 91–95. GRIFFITH UNIVERSITY, Gold-Coast, Australia, February 1997.
- [5] Rodney A. Brooks. A robust layered control system for a mobile robot. Technical Report 864, MIT, September 1985.
- [6] J.S. Breese E.J. Horvitz and M. Henrion. Decision theory in expert systems and artificial intelligence. *International Journal of Approximate Reasoning*, 2:247–302, Special Issue on Uncertainty in Artificial Intelligence.
- [7] Yasuo Kuniyoshi Hiroaki Kitano, Minoru Asada and Itsuki Noda. *RoboCup: A Challenge Problem for AI and Robotics*, volume I of *RoboCup 97: Robot Soccer World Cup*, chapter 1, pages 1–20. Springer-Verlag, 1998.
- [8] J.P.Muller. The design of autonomous agents - a layered approach. *Lecture Notes in Artificial Intelligence*, 1177, 1996.
- [9] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20:46–61, February 1973.
- [10] Maja Mataric. Issues and approaches in the design of collective autonomous agents. *Robotics and Autonomous Systems*, 16, 1995.
- [11] M.Ginsberg. Universal planning: An (almost) universally bad idea. *AI magazine*, Winter 1989.
- [12] N.Jennings M.Woolridge. Intelligent agents: Theory and practice. In *Knowledge Engineering Review*, volume 10, chapter 2. Cambridge University Press, 1995.
- [13] Itsuki Noda and Hitoshi Matsubara. Learning of cooperative actions in multi-agent systems: a case study of pass play in soccer. In *Proceedings of AAAI Symposium*, September 1996.
- [14] Peter Stone and Manuela Veloso. Using decision tree confidence factors for multiagent control. In H.Kitano, editor, *RoboCup-97: The First Robot World Cup*. 1997.
- [15] Lonnie R. Welch and Behrooz A. Shirazi. Dynamic real-time benchmark for assessment of qos and resource management technology. In *Proceedings of the IEEE Real-time Technology and Applications Symposium*, pages 36–45, June 1999.