2nd International Conference on Autonomous Robots and Agents
December 13-15, 2004 Palmerston North, New Zealand

An Intuitive and Flexible Architecture for Intelligent Mobile Robots

Xiao-Wen Terry Liu and Jacky Baltes
Department of Computer Science,
University of Manitoba, Winnipeg, Manitoba, Canada
tliu@cs.umanitoba.ca, jacky@cs.umanitoba.ca

Abstract

The goal of this research is to develop an intuitive, adaptive, and flexible architecture for intelligent mobile robots. We propose a hybrid architecture that uses behaviour trees and finite state machines. A task manager selects behaviours based on approximations of their applicability and the expected reward of a behaviour. One major feature of this architecture is that important information of the perception, reasoning, and execution parts of the system are made explicit. This information includes parameters (e.g., colour definitions), structural information (e.g., the behaviour tree), and the ability to represent prototypical scenarios. We use robotic soccer as a sample domain to ground our work on agent architectures.

Keywords: agent architecture, XML, hybrid architecture, behaviour architecture

1 Introduction

This paper describes our research into the design of an intelligent agent architecture for mobile robot teams. In contrast to previous approaches, the architecture makes important information explicit, which makes it easier to modify, adapt, extend this architecture.

An architecture is a unifying, coherent form or method of construction, which provides the foundation for creating powerful intelligent systems. Developing an architecture for intelligent mobile robot is a difficult task. Just like other intelligent systems, mobile robots must select correct actions out of a huge set of possibilities; to make matters worse mobile robots need to act under constraints imposed by the real world. One obvious example is that the robot has to react fast enough to avoid certain obstacles. Furthermore, sensors and actuators are noisy and inaccurate.

The following aspects are common to all intelligent mobile robots: perception, reasoning, and execution (see Fig. 1). Sensors such as light, sonar, or touch, gather information about the environment such as the robot's approximate location, location of nearby obstacles or other robots, and information about itself such as battery power levels. The raw sensor data must often be filtered, correlated and/or interpreted to form perceptions. For example, a group of pixels are smoothed, grouped, and colour classified to generate the position of a ball. The set of all perceptions forms a model of the world in which the agent reasons. Reasoning often involves the agent trying to generate and satisfy multiple and possibly competing goals. Commonly, the output of the reasoning stage is an abstract plan (e.g., a sequence of actions or behaviors) to achieve the agent's goals. In the execution stage, the abstract plan is implemented. Abstract operators (e.g., drive through the door) are converted into lower level commands for motors and other actuators.

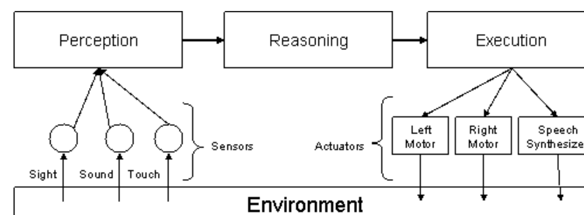

Figure 1: Generic Architecture

Developing, maintaining, and modifying systems to control intelligent mobile robots in the real world can be a daunting task. The problem with most systems is that they are often limited by the initial design of the original developer or specifications. These systems cannot cope with error, and are not flexible enough to change certain minor aspects of the program. For example, transforming a soccer playing robot into a garbage collecting robot may require going through the control program and to modify perceptions (e.g., targets and obstacles), reasoning (e.g., approach a trash can from any direction vs. approaching a ball in only certain directions), and execution (e.g.,add an additional actuator). The problem is that much of the necessary information is only available implicitly in the code. We believe a system with an architecture that utilizes an agent-based approach *and* makes relevant information explicit will be intuitive, flexible and extensible.

Although the need for an intuitive architecture is inherent to all intelligent mobile robot applications, we will use examples from the domain of robotic soccer. In section 2, architectural issues, which may seem abstract and vague, are made concrete through examples gathered during our participation at previous robotic soccer competitions. Section 3 describes previous approaches. Section 4 shows the design of our new architecture. Section 5 describes how we evaluated our new architecture. Section 6 concludes the paper.

2 Motivation

Robotic soccer is a great testbed for robotic research into intelligent mobile robotics. Playing soccer well requires solutions to many problems that are currently actively investigated by researchers. For example, a player has to have a set of skills: real-time control (e.g. able to kick the ball and accept passes), perception (e.g. able to see the ball), awareness (e.g. localization), strategy, coordination and communication (e.g. set up plays).

In the last two years, the University of Manitoba has fielded a robotic soccer team ([1]) at these events. Our participation at these events has made the need for powerful architectures for mobile robots blatantly apparent. To be successful in the competition, the system must be adapted to different environments and different opponents.

For example, less reliable actions need to be removed from a robots set of possible actions to allow a robot to be more predictable and reliable. In 2003, we would need to sift through many lines of C++ code to figure out exactly where to remove a behaviour. Extra time was needed to test the effect of this change on the whole system because of the interaction of different behaviours. Even in the C++ code, it was difficult to determine what other behaviours depended on a given behaviour.

Furthermore, the developer would need to be aware of the subtle nuances of the program such as if two behaviours were both applicable, then the first behaviour that was loaded would be used. This knowledge is implicit in the C++ code and is not very intuitive for new developers.

From our experiences at RoboCup, it was clear that an architecture was needed and the requirements for this new architecture are as follows:

1. Designing behaviours needs to be simple and intuitive.

2. The goals, assumptions and effects of behaviours need to be made explicit.

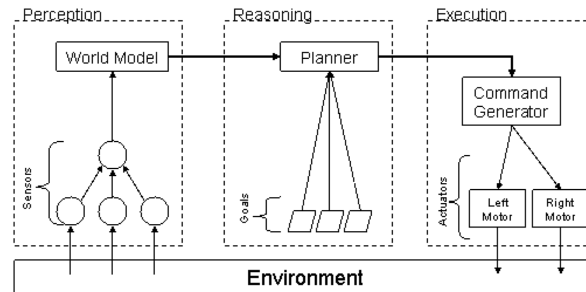

Figure 2: Generic Top-down Architecture

3. Implementation aspects of the decision-making mechanism that chooses and switches between behaviours also needs to be made explicit.

4. The process to create and remove behaviours needs to be simple and flexible.

Even though robotic soccer provided us with real-world problems and highlighted the need for powerful architectures, the resulting research is applicable to all intelligent mobile robot domains.

The following section provides background information on robotics, agent-base approaches, behavioral control architectures, and other relevant information.

3 Related Work

There are three main approaches to implementing the previous architectures: top-down, bottom-up, and hybrid approaches.

3.1 Top-Down Architectures

A top-down architecture (see Fig. 2) approach uses abstraction to decompose the perception, reasoning, and execution cycle. The motivation for the top-down architecture is that abstraction can hide details of the lower levels from the higher levels.

Pure top-down architectures have an explicit world model and focus on devising one strategy and carrying it through to the end. These systems are good at planning and higher level reasoning, but are not reactive enough for dynamic environments. To overcome this problem, researchers have developed extensions to pure top-down architectures, such as behavior tress. A behaviour tree is a collection of behaviours organized in a tree. It maps complex behaviours by branching them into smaller simpler behaviours. The depth of the tree depends on the complexity of the most complex behaviour, while the breadth of the tree depends on the number of behaviours. Behaviour trees are very useful to help manage the complexity of one branch from another on the same level. However, it is difficult to jump from

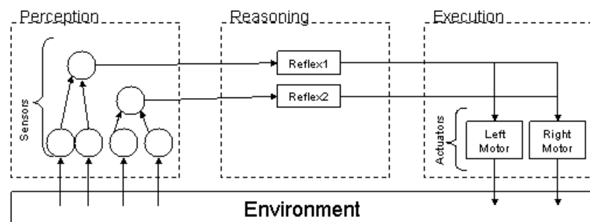

Figure 3: Generic Bottom-Up Architecture

one state in a certain level in a branch to a different state in another level of a different branch.

3.2 Bottom-up architectures

Bottom-up architectures (see Fig. 3) are the opposite of the top-down architectures. Instead of multiple levels of abstraction in the perception, reasoning, and execution stage, bottom up architecture include simple behaviors that map perceptions directly to actuator commands (similar to reflexes). More complex behaviors are created by combining simpler ones. Bottom- up architectures are able to react quickly to the environment because of the direct links between sensors and actuators (e.g., avoid an obstacle). The disadvantage of bottom-up architectures is that it is often difficult to know what lower level behaviors are needed and to predict the interaction of multiple behaviors.

Pure reactive architectures have a set of low-level behaviours, which can react to certain situations when they arise. The main advantage of using these systems is that they are computationally efficient. The main disadvantage is that these systems usually have no mechanism for higher level planning or reasoning.

Brooks subsumption architecture is one well-known architecture for controlling robots and their behaviours [2]. It does not use an explicit world model like pure top-down systems. The main concept of Brooks subsumption architecture is that the robots behaviours are designed in a layered approach. Each layer is an asynchronous module and higher-level layers have the ability to *subsume* (i.e., override) the lower layers. The higher layer subsumes the lower layers by either inhibiting the inputs to or suppressing the outputs of the lower layers. This makes the architecture robust when new or additional behaviours are required. One disadvantage of Brooks' subsumption architecture is that the complexity of designing the system increases greatly as more complex higher level layers are added. Nevertheless, subsumption architectures have been used successfully in robotic soccer and other tasks [3, 4].

3.3 Hybrid Architectures

Hybrid architecture architectures are a mix between top-down and bottom- up architectures. These types of architectures are the most popular because they take advantage of the strengths of top-down and bottom-up architectures [5, 6, 7, 8]. However, because they also inherit the weaknesses of those architectures, the difficulty lies in finding a reasonable balance between the two architectures.

In hybrid architecture, some sensors and perceptions are directly connected to the actuators, whereas others are processed more extensively. Instead of a complete world model, the perceptions place the system in a finite set of states. In other words, the environment can be mapped to certain states in the system. The reasoning system moves the agent into desired states. For example, the *kick to goal* behavior includes three states (e.g. position behind ball, facing the ball, and kicking the ball into the goal). One main advantage of using such architecture is that it is easy to transfer from one state to another on the same level. For example, if after approaching the ball the robot is already facing the ball, then the system will kick the ball immediately.

A popular hybrid architecture is the belief-desire-intention (BDI) architecture which is being used by several robotic soccer teams [9]. Belief refers to the facts that an agent holds to be true about itself, and its environment. Desire refers to the goals of an agent. Finally, intention refers to the steps the agent plans to take to reach its goals or desires. The belief and desires of a robot help drive its long term planning strategies. The intention and desires of a robot help with its ability to improvise.

Decugis and Ferber described another autonomous-agent hierarchical distributed reactive planning architecture [10]. Their architecture uses a network of behaviours with a flexible activation function to switch behaviours.

Many teams in robotic soccer competitions have used hybrid approaches [11, 12, 13]. Nonetheless, most of the crucial information about the agent's play is only implicit in the code. One exception is the "German Team" in the Four Legged League [14]. They developed a different architecture for their Sony Aibo robot dogs based on Extensible Agent Behavior Specification Language (XABSL), to help describe behaviours (see Fig. 4 for a sample behaviour definition) [15].

The use of XML means that this information is made explicit to the developer. For example, it is easy to change the XABSL example above so that the strikers stay further behind the ball.

4 Design

Our proposed architecture uses some basic object-oriented (OO) design principles. Behaviours that share similar qualities with each other are grouped together and common information or functions can

```
<option xsi:schemaLocation="http://www.ki.informatik.hu-berlin.de/XABSL2.1
../../../Xabsl2/xabsl-2.1/xabsl-2.1.option.xsd" name="striker" initial-state="initial">
 - <state name="initial">
   - <subsequent-basic-behavior ref="go-to">
     - <set-parameter ref="go-to.x">
       - <minus>
           <decimal-input-symbol-ref ref="ball.x"/>
           <decimal-value value="8"/>
         </minus>
       </set-parameter>
     - <set-parameter ref="go-to.y">
         <decimal-value value="11"/>
       </set-parameter>
     </subsequent-basic-behavior>
   - <decision-tree>
       <transition-to-state ref="initial"/>
     </decision-tree>
   </state>
</option>
```

Figure 4: Sample XABSL striker definition

be inherited. For example, defenders may need to know the line of shot from the ball to our own net. From a design perspective, this allows us to use inheritance to simplify the work that needs to be done. However, proper *OO* principles cannot be the panacea to our problems. Some behaviours may need to be modified quickly and dealing with a complex inheritance tree may not be intuitive enough. For example, in changing a behaviour of a defender to a goalkeeper, the developer needs to be aware of certain restrictions placed on a goalkeeper. (e.g. the goalkeeper is not expected to come far away from its own goal.) Adding this behaviour is not as simple as copying the behaviour over to the new group of behaviours. Interaction between this behaviour and the existing behaviours can be severely degrade the overall performance. The trigger to execute this behaviour may be too rigid (e.g. a simple if-statement is not flexible enough). In a complex system, maintaining a complex if-structure is not ideal. Thus, our proposed architecture proposes to deal with this issue by allowing the behaviour to discover how applicable it is to a particular scenario. With our architecture, the developer can design the actions for the behaviour, and easily integrate the behaviour(see Fig. 5). Currently, we are in the process of developing a more complex architecture that uses XML to represent important information.

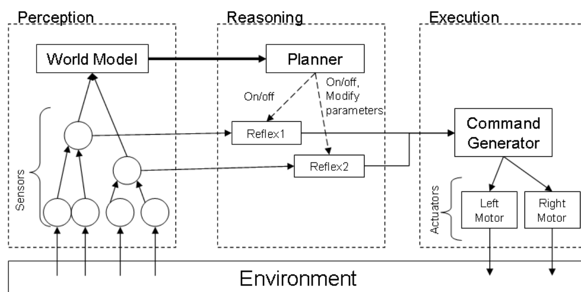

Figure 5: Proposed Architecture

Complex behaviours are represented as behaviour trees. The reasoning behind this is to create a system where it would be easy to add, remove, or modify complex behaviours. The leaf nodes of the tree communicate to the command generator to control the actuators (reflex behaviours). The expected transition from one behaviour to the next is mainly modelled via finite state machines.

From a high level appearance our architecture has two major levels for our behaviours. Level 1 are critical behaviours that will override level 2 behaviours. This override system is seen in other architectures such as the Brooks' subsumption architecture [2].

Level 1 behaviours are simple and critical behaviours that follow strict guidelines. Thus, they are mostly a collection of finite state machines loosely connected with each other by simple state conditions. These behaviours are intended for the designer to have full control over behaviour switching using simple transitional conditions. In our model domain, these behaviours are used for system diagnostics and obeying the referee commands such as free kicks and penalty kicks.

Level 2 behaviours compose approximately 95 percent of the robot's normal dynamic functions. Each behaviour is a node in the main behaviour tree. In our domain, we divided our behaviour tree further into 4 levels (see Fig. 6): Field strategy, role differentiation, role strategy, lower level behaviours. More complex behaviours are placed closer to the top of the tree *vis-a-vis* simpler, reflexive behaviours are closer to the bottom. Using a hierarchical approach not only allows us to control the complexity of the behaviours but it has proven to be flexible enough to allow deliberative and opportunistic actions [8].

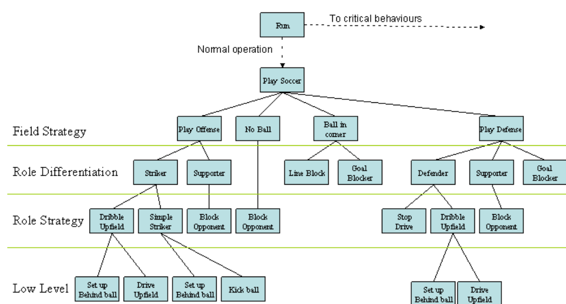

Figure 6: Level 2 dynamic behaviours [16]

However, the core of our new hybrid architecture is our task manager, a main component of the planner described in Fig. 5 and Fig. 7, which we use to control the behaviors of the robot. All behaviors in the system have two additional functions: *calc_applicability(state)* and *calc_reward(state)*. These functions are quick approximations of the assumptions and effects of a behaviour. For example, to determine whether a given behaviour's assumptions are met may require the be-

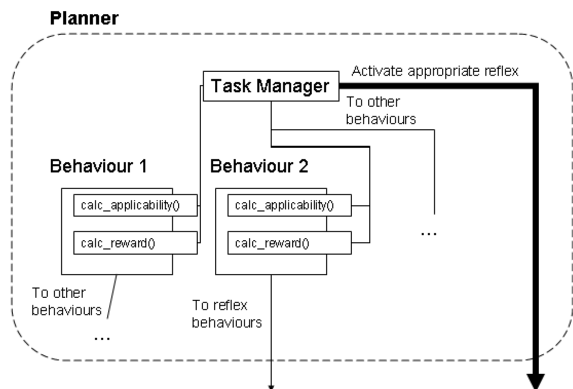

Figure 7: Planner: Task Manager

haviour to plan a complete path to the goal destination, which is computationally expensive. Instead, the *calc_applicability(state)* function estimates this applicability by the distance of the robot to the goal position.

The function *calc_applicability(state)* returns how close the current world state matches the assumptions of the behavior. These values are modelled as fuzzy sets [17]. For example, the behaviour *shoot-goal* returns a high applicability if the robot and the ball are lined up and close to the opponent's goal. The function *calc_reward(state)* returns the expected reward should the behavior succeed. For example, the *score- goal* returns a reward of 0.9 whereas *block opponent* returns 0.3. The *calc_applicability(state)* and *calc_reward(state)* functions are limited in the amount of processing that they are allowed to do. They are expected to return very quickly.

The task manager calculates the applicability and reward for all enabled behaviours and then based on the current strategy selects a behaviour. This behaviour is then activated and executed until the next time step. To avoid the common problem of oscillating between behaviours, the task manager enforces a minimum threshold for changing from one behaviour to another. For future research, learning can be incorporated into the behaviours and task planner. Behaviours can adjust the value of the *calc_applicability(state)* and *calc_reward(state)* depending on the success of the behaviour. Other researchers are looking into learning with their research as well [18].

Our goal to make information explicit must be carefully balanced with the need for an intuitive architecture. Obviously, the most extensible architecture is a description of a C++ program in XML. However, the resulting system would be even more difficult to understand than the original C++ code. Therefore, the type and amount of information that is expressed explicitly must be carefully controlled.

The sensor and perception processing part of the system is heavily data driven and a number of principle candidates for adaptation are easily apparent. For example, it should be trivial to change the colour of the ball or the number of opponents. Therefore, the following information is expressed in XML: number and types of objects, colours of the objects in our 15 parameter colour model, minimum and maximum sizes and aspect ratios for objects.

Similarly, the XML description of the reasoning component includes a description of the hierarchy of behaviours, a case-based description of the assumptions and the effects of a behaviour. Instead of describing maximum applicability and reward, the XML description includes a series of sample scenarios and a their associated applicabilities and rewards. The system automatically interpolates for states that do not match a given scenario.

The XML description of the execution module includes gains for the motor controllers (we use a CMAC with 64 parameters to determine motor outputs for desired control inputs) as well as the minimum distance between obstacles and the robot. For example, our *block-shot-on-goal* behaviour ignores other obstacles. We are currently still developing this XML aspect of our architecture.

5 Evaluation

Some aspects of an architecture can be easily evaluated (i.e., average runtime of the control cycle). However, measuring the extendability or flexibility of an architecture is difficult since there are no known quantitative measures for these features of an architecture. Furthermore, concepts such as flexibility are subjective in themselves.

Therefore, we use anecdotal evidence and a series of challenge tasks will be used to evaluate these features of the architecture. We measure the time it takes for a developer to implement the following tasks using this architecture: (a) path tracking, (b) treasure hunt, (c) obstacle run, (d) goal scoring, (e) ball passing, and (f) playing soccer.

Preliminary results from our current system are encouraging. In 2004, the RoboBisons used a configuration file to describe the tree structure of the behaviours and the maximum applicability and reward for each individual behaviour. Even making this simple change had a huge impact. We estimate that we saved several hours in the 2004 competition which allowed us to focus on other more important problems. Furthermore, it was easy to compare the influence of different behaviours since their maximum applicabilities were readily available to the developer.

6 Conclusions

This paper describes the motivation and design of a novel architecture for intelligent mobile robots. The agent based architecture is easier to extend, adapt, and more intuitive since the relevant information is explicitly available to the developer.

Using such an architecture allows developers to create behaviours quickly and easily. The flexibility of the architecture should allow developers to modify the behaviours to suit their needs. The immediate application of this novel architecture is in future mobile robotic competitions such as RoboCup and FIRA, but we believe many other applications for intelligent mobile robots will benefit from such an architecture.

References

[1] John Anderson, Jacky Baltes, Doug Cornelson, Terry Liu, Clint Stuart, and Adam Zilkie. The university of manitoba uleague team. In *Proceedings of the RoboCup Symposium*, Padova, Italy, July 2003.

[2] Rodney A. Brooks. A robust layered control system for a mobile robot. In *IEEE Journal of Robotics and Automation*, 1986.

[3] Tim Gorton and Bakhtiar Mikhak. A tangible architecture for creating modular, subsumption-based robot control systems. In *Extended abstracts of the 2004 conference on Human factors and computing systems*, pages 1469–1472. ACM Press, 2004.

[4] Raúl Rojas, Sven Behnke, Achim Liers, and Lars Knipping. FU-Fighters 2001 (Global Vision). In *RoboCup 2001: Robot Soccer World Cup V*, pages 571–574. Springer-Verlag Berlin Heidelberg, Inc., 2002.

[5] Mark M. Chang, Brett Browning, and Gordon F. Wyeth. ViperRoos 2000. In *RoboCup 2000: Robot Soccer World Cup IV*, pages 527–530. Springer-Verlag Berlin Heidelberg, Inc., 2001.

[6] Raffaello D'Andrea, Tams Kalmr-Nagy, Pritam Ganguly, and Michael Babish. The Cornell Robocup Team. In *RoboCup 2000: Robot Soccer World Cup IV*, pages 41–51. Springer-Verlag Berlin Heidelberg, Inc., 2001.

[7] Kian Hsiang Low, Wee Kheng Leow, and Jr. Marcelo H. Ang. A hybrid mobile robot architecture with integrated planning and control. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 219–226. ACM Press, 2002.

[8] Monica N. Nicolescu and Maja J. Matarić. A hierarchical architecture for behavior-based robots. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 227–233. ACM Press, 2002.

[9] Mattias Werner, Helmut Myritz, Uwe Düffert, Martin Lötzsch, and Hans-Dieter Burkhard. Humboldt Heroes. In *RoboCup 2000: Robot Soccer World Cup IV*, pages 651–654. Springer-Verlag Berlin Heidelberg, Inc., 2001.

[10] Vincent Decugis and Jacques Ferber. Action selection in an autonomous agent with a hierarchical distributed reactive planning architecture. In *Proceedings of the second international conference on Autonomous agents*, pages 354–361. ACM Press, 1998.

[11] Andrew Howard. MuCows. In *RoboCup 2000: Robot Soccer World Cup IV*, pages 535–538. Springer-Verlag Berlin Heidelberg, Inc., 2001.

[12] J. Ll de la Rosa, B. Innocenti, M. Montaner, A. Figueras, I. Munoz, and J. A. Ramon. RoGi Team Description. In *RoboCup 2001: Robot Soccer World Cup V*, pages 587–590. Springer-Verlag Berlin Heidelberg, Inc., 2002.

[13] AliReza Fadaie Tehrani, Peyman Amini, Hamid Reza Moballegh, Pezhman Foroughi, Omid Teheri, Behrouz Touri, Ahmad Movahedian, and Mohammd Ajoodanian. IUT Flash Team Description. In *RoboCup 2003: Robot Soccer World Cup VII*, 2003.

[14] Ingo Dahm, Uwe Düffert, Jan Hoffmann, Matthias Jüngel, Martin Kallnik, Martin Lötzsch, Max Risler, Thomas Röfer, Max Stelzer, and Jens Ziegler. Germanteam 2003. In *RoboCup 2003: Robot Soccer World Cup VII*, 2003.

[15] Martin Lötzsch. XABSL: The Extensible Agent Behavior Specification Language. http://www.ki.informatik.hu-berlin.de/XABSL/.

[16] John Anderson, Jacky Baltes, Brian McKinnon, Terry Liu, Paul Furgale, and Shawn Schaerer. Robocup 2004 Presentation, 2004.

[17] Lotfi Zadeh. Fuzzy Sets. In *Information and Control*, volume 8, pages 338–353, 1965.

[18] R. Peter Bonasso and David Kortenkamp. An intelligent agent architecture in which to pursue robot learning. In *Working Notes: MCL-COLT '94 Robot Learning Workshop*, July 1994.