MBA: A Modular Hierarchical Behavior-Based Architecture

Roberto Paredes, Francisco Martín, Vicente Matellán and Carlos E. Agüero

Grupo de Robótica (GSyC)
Universidad Rey Juan Carlos
{rober, fmartin, vmo, caguero}@gsyc.es

Abstract

This paper describes an autonomous modular hierarchical control system for robots. This architecture is designed to control robots that perform complex tasks in a highly dynamic environment. Some of the key features included in this architecture are: unlimited number of behaviors and levels in the architecture; easy addition, removal and edition of any element of this architecture while the robot is switched on. This is a very positive factor to debug the behaviors in both design and implementation phases. This system has been successfully integrated in our Sony aiBo four legged team software.

1 Introduction

In this paper we present a high level control architecture for robots. This control architecture generates actions depending on the sensory information about the environment, and the information shared among its team mates. This decision is organized in a hierarchical architecture. The higher levels take decisions concerning the robot role, that is strategy-based decisions. The lower levels set the actuation commands to perform simple tasks.

Behaviors are the basic modular blocks of the architecture. Each behavior has a goal that must satisfy, and may have one or many prerequisites. For example, if the goal of the current active behavior is *looking for the ball*, the behavior performs all the possible actions to look for the ball: turn, move to the last position where the ball was seen, etc. If any behavior's prerequisite are not met, it is set as a goal to the lower level. For example, if the goal of the active behavior is going to the ball, and it has as prerequisite *seeing the ball*, to find the ball may be a goal to the lower level. In the lower level the behavior that look for the ball will be active to achieve that goal.

This architecture is modular. The number of levels is configurable to let the behavior designer decide the degree of abstraction of each level. Each level may be composed by an undetermined number of behaviors. We can add or remove behaviors belonging to a level without make any other change in the architecture or other behavior.

The mechanism for behavior combination offered by this architecture makes a way to control the robot to perform complex tasks in real time.

This architecture has been successfully tested in a real robot, the Sony aiBo robot, integrated into the software developed by the TeamChaos[**?**] four legged league team for the RoboCup.

There are many works in the literature about behavior-based architectures. One of the best known is the Subsumption architecture [**?**] [**?**] developed by Rodney Brooks. This architecture was inspired in ethology and it has attributes such as close connection of sensors to actuators, pre-wired patterns of behavior, simple navigation techniques. The Subsumption architecture provides these capabilities through the use

of a combination of simple machines with no central control, no shared representation, slow switching rates and low bandwidth communication. Ronald C. Arkin reviewed behavior based systems in [?].

Another example is JDE [?], a hierarchical approach to a robotic control architecture, ethologically inspired, based on the selective activation of *schemas*, which generates the dynamic hierarchies that govern the robot behavior.

Alexandro Saffiotti also designed a control system that organized behaviors in an hierarchical way [?]. Later, he adapted this system to the RoboCup domain [?]. Also in the RoboCup domain we remark the work developed in the CMU team, in which an architecture similar to our approach is used [?].

We will start in the section ?? describing the environment,in particular the TeamChaos architecture and how this control system is integrated into it. In the section ?? we will fully describe this architecture and its elements. In the section ?? we will explain how to design, implement, and debug the complete architecture. An example of this architecture will be shown in section ??. Finally, in section ?? we will discuss the results.

2 Domain

RoboCup is a very challenging and motivating robotic competition where under a simple soccer match, are hidden lots of research in several areas: localization, perception, object tracking, recognition, locomotion, behavior design, cooperation, etc.

In this paper we will be focused on the 4-legged league where teams made by four Sony aiBo ERS7 compete. The robot processing is performed by an embedded MIPS processor running at 576MHz and with 64MB of memory. The main input comes from a 350K-pixel color camera, and a pair of infrared sensors. In this sense, aiBo can be considered legged active vision based mobile robot. The communication is based on a LAN wireless IEEE 8011b card.

The architecture that we present in this paper is integrated in the TeamChaos four legged team software[?]. TeamChaos development is organized in two projects: TeamChaos and ChaosManager. TeamChaos contains all code related to the robot, ChaosManager is a suite of tools for calibrating, preparing memory sticks, editing and monitoring different aspects of the robots and the games.

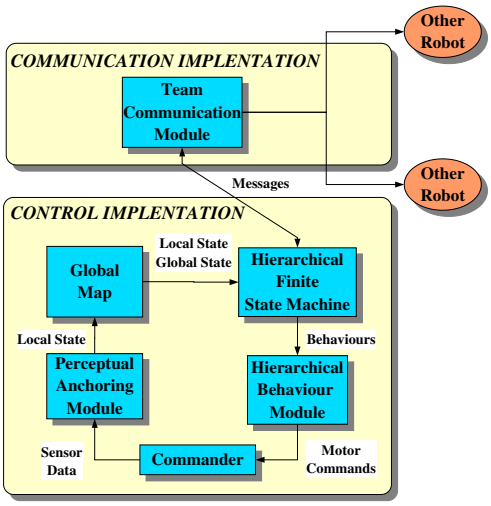

Figure 1: TeamChaos Architecture

TeamChaos software running in the robot is organized as a layered architecture shown in Figure ??. This architecture is an implementation of ThinkingCap architecture [?].

- The lower layer (CoMmanDer module, or CMD) provides an abstract interface to the sensorimotor functionalities of the robot.
- The middle layer maintains a consistent representation of the space around the robot named Perceptual Anchoring Module, or PAM, and implements a set of robust tactical behaviors (Hierarchical Behavior Module, or HBM). The HBM

realizes a set of navigation and ball control behaviors.

- The higher layer maintains a global map of the field (Global Model, or GM) and makes real-time strategic decisions based on the current situation. Situation assessment and role selection is performed in the HFSM, or Hierarchical Finite State Machine. The HFSM implements a behavior selection scheme based on finite state machines.
- Radio communication is used to exchange position and coordination information with other robots (via the TCM, or Team Communication Module).

Nowadays, the robot control is divided in two levels: in the high level, HFSM select the low level behavior that HBM should execute. Once the control system presented in this paper is implemented, it will replace the HFSM and the HBM modules, as is shown in figure **??**.

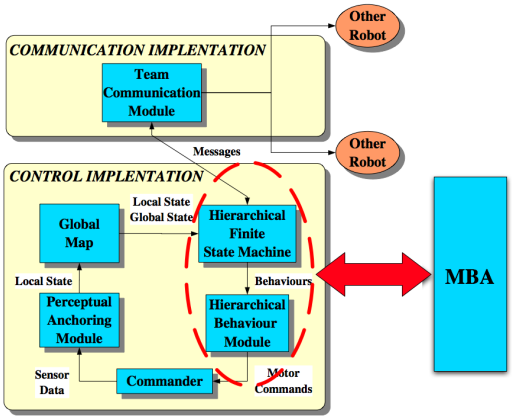

Figure 2: In the TeamChaos Architecture, the HBM and HFSM will be replaced by MBA.

3 Architecture description

In the architecture proposed in this work, *behaviors* are the basic block. A behavior is defined as a small piece of code with a *goal*. A behavior must perform the sequence of actions needed to achieve this goal. In many cases the actions that the behavior performs set new goals that another behaviors try to achieve.

As we can see in Figure **??**, the behaviors are organized in *levels of abstraction* $1..n$, where n is the total number of levels. It is usual to have several behaviors in the same level.

The levels are communicated by *goal buffers* $1..m$, where $m = n - 1$. The buffer 1 divides level 1 and 2, buffer 2 divides level 2 and 3, and so on. The execution of behaviors generates some necessities that are converted in goals for level $N - 1$ and stored in the $N - 1$ buffer.

All the behaviors in every level have access to local perceptual information, shared information and global model. The Local perceptual information stores the relative position with respect any element it can detect: ball, nets, landmarks. The shared information received by the team mates is stored too. The behaviors have access to the Global Model, that stores localization information.

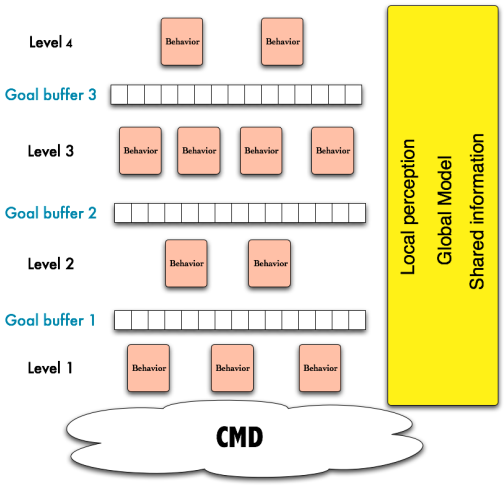

Figure 3: MBA Architecture

Only one behavior will be running in each level, that will be named "active". To select the active behavior, each behavior examines

the goals in the upper buffer and calculates how good it can achieve the goals, taking into account all the information available. Each behavior must calculate its own value.

There are situations in which two or more behaviors in the same level are able to achieve the same goal. The difference among them is how each behavior tries to achieve the goal. Not in every moment the environment conditions benefits identically to each behavior. For example, let's suppose than in a level we have 2 behaviors that achieve the goal of *going to a global position* in the field: *go2position* and *go2positionvff*. The difference among them is the navigation algorithm used. *go2position* supposes no obstacles in the path towards the destination and it is very fast. *go2positionvff* is able to avoid the obstacles, but it is slower. The active behavior in each moment depends on the obstacle detected in the path. The winner behavior in each level will be executed.

The behaviors in the level 1 (the lowest) set the actions that the architecture must send to the CMD module to be performed by the robot. These basic actions are velocities and kicks.

4 Design tool

The design tool has been integrated integrated in the ChaosManager (Figure **??**). The main goal of this tool is to design the complete architecture, implementing every behavior and its evaluation function. Once implemented, the complete architecture is sent to the robot. So, we can debug and change any element without robot rebooting.

5 Example: designing a robotic player

To illustrate how this architecture works, we will design a simple player. First of all, we design the architecture. As we see in figure **??**, there are three levels.

The robot stores a game state, and depending on it, this player will be in one of these three phases: A *home phase* when the robot goes to a predetermined initial position, a *kickoff phase* when the robot is aware of the ball to do a kickoff, and a *play phase*. In this last phase the robot is playing. In this simple example, the robot looks for the ball, goes to it and kick it.

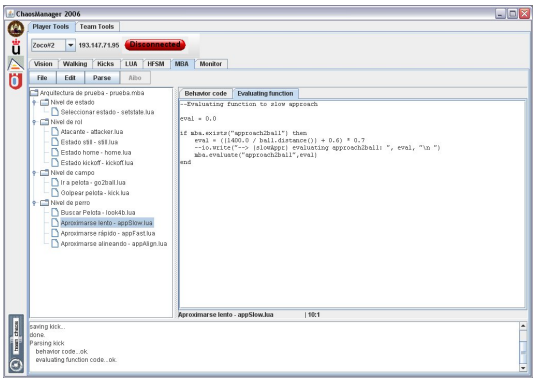

Figure 4: Design tool integrated in the ChaosManager

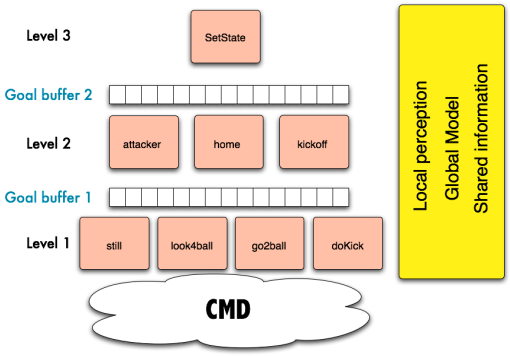

Figure 5: Simple player MBA Architecture example

In the level 3, there is only the *setState* behavior that, depending on the current phase (home, kickoff or playing), set the goals in the goal buffer 2:

– setState behavior
state = mba.getGameState()
if state == mba.state$_{p}laythenmba.addGoal(\text{"}beAttacker\text{"})$
elseif state == mba.state$_{k}ickoffthenmba.addGoal(\text{"}doKickoff\text{"})$
elseif state ==

mba.state$_h$omethenmba.addGoal("goHome")end

Only one goal (*beAttacker*, *doKickoff* or *goHome*) is set in the goal buffer 2. In this example we will only set one goal at the same time, but it is usual to set more than one. Each behavior in level 2 evaluates if to be active is appropriate depending on the goal and all the available information. For example, the evaluation function for the *attacker* behavior is:

– attaker behavior evaluation function

if mba.exists("beAttacker") then mba.evaluate("attacker", 1.0) end

The evaluation function of the *attacker* behavior gives the maximum value to be executed if the goal is *doAttacker*. In other case, it is evaluated to 0.

In this level we decide that in *home* and *kick* behaviors, we must be stopped, and in both cases the goal is *beStill*:

– home behavior (same for kick behavior)

mba.addGoal("beStill")

In the *attacker* behavior, if the ball is not detected (the ball's `anchored` property in the code), the goal is *seeBall*. If the ball is detected, but it is far away from robot, the goal is *approachBall*. If the robot is near from ball, the goal is *Kick*:

– attacker behavior

if ball.anchored() ¿ 0.9 then if ball.distance() ¡ 295 then mba.addGoal("kick") else mba.addGoal("approachBall") end else mba.addGoal("seeBall") end

Once explained the level 2 and 3, the only goals allowed in goal buffer 1 are *kick*, *approachBall* and *seeBall*, set by *attacker* behavior; and *beStill*, set by *home* and *kick*. In the level 1, the goal *beStill* is achieved by the *still* behavior, that send directly velocity commands to the CMD:

– still behavior

dog.setVlin(0) dog.setVlat(0) dog.setVrot(0)

The *go2ball* and *look2ball* also send directly velocity commands to the CMD depending on the ball position, or the implemented algorithm to search the ball. The *doKick behavior* sends the correct kick command to CMD depending on the localization information (`dog.angle stores the global robot orientation`):

– doKick behavior

if (dog.angle() ¡ pi$_4$ + 0.2)$and(dog.angle() > -pi_2 -$ 0.2)$then dog.setKick("LeftHeadKick")elseif(dog.angle() <$ $pi_{34} + 0.2)and(dog.angle() <= -pi_2 -$ 0.2)$then dog.setKick("RightHeadKick")elsedog.setKick("FrontKick")endend$

A video of this player example can be found in `http://gsyc.es/~fmartin/player.avi`.

6 Discussion and future work

In this paper we have presented a modular hierarchical control system for robots. The global robot behavior arises from the combination of the behaviors belonging to the architecture. The separation on levels let us to design behaviors with different level of abstraction. The debugging tool is essential for a fast implementation and debugging of the robot behavior, contributing the possibility of tuning any element *on the fly*. That feature supposes a powerful stone for the development stage..

The major contributions of the control system presented in this paper are:

- We provide an architecture in which only one behavior is active per level.
- We provide a distributed arbitration mechanism to decide the active behavior.
- We give the behaviors access to all the sensors and shared information, independently of which level is.
- We give only the low level behaviors access to the actuators to avoid conflicts in the actuation.
- We integrate this architecture in the TeamChaos software and we provide several tools to add, edit and remove behaviors, and to send to the robot while it is switch on, reducing the implementation time.

Future work focuses on the parallel activation of behaviors on the same level, and the tuning of the arbitration mechanism for behavior activation.

7 Acknowledgement

The authors would like to thank other members of TeamChaos, specially to Murcia University colleagues, for their valuable contributions to the tools used in this work.

This research has been partially sponsored by grants No. S-0505/DPI/0176 by Community of Madrid and No. DPI2004-07993-C03-01 by Spanish Ministry of Education corresponding to RoboCity and Acrace projects respectively.