INITIAL DEVELOPMENT OF HABLA (HARDWARE ABSTRACTION LAYER)
A Middleware Software Tool

Andrés Faíña, Francisco Bellas and Richard J. Duro
Integrated Group for Engineering Research, University of Coruña, Mendizabal S/N, Ferrol, Spain

Keywords: Middleware software tool, control system abstraction, autonomous robotics.

Abstract: In this work we present the initial implementation of a middleware software tool called the Hardware Abstraction Layer (HABLA). This tool isolates the control architecture of an autonomous computational system, like a robot, from its particular hardware implementation. It is provided with a set of general sensors and typical sensorial processing mechanisms of this kind of autonomous systems allowing for its application to different commercial platforms. This way, the HABLA permits the control designer to focus its work on higher-level tasks minimizing the time spent on the adaptation of the control architecture to different hardware configurations. Another important feature of the HABLA is that both hardware-HABLA and HABLA-control communications take place through standard TCP sockets, permitting the distribution of the computational cost over different computers. In addition, it has been developed in JAVA, so it is platform independent. After presenting the general HABLA diagram and operation structure, we consider a real application using the same deliberative control architecture on two different autonomous robots: an Aibo legged robot and a Pioneer 2Dx wheeled robot.

1 INTRODUCTION

The origin of this work can be found in the research on autonomous robotics carried out in our group. It usually implies the acquisition of one or more commercial robots or the construction of new ones. In any case, the robots must be programmed in their own programming language (C, C++, Lisp, etc) and their particular hardware architecture must be taken into account when developing its control software. Manufacturers usually provide an API (Application Program Interface) with a reduced set of high level commands to develop basic functionalities. Examples of these tools are AIBO SDE (AIBO, 2007) (a software development environment for AIBO robots) or Aria (Aria, 2006) (that supports different robot models from Activmedia Robotics). The main problem with these tools is that they are specific for a given family of robots and they require the designer to develop the control architecture in a preestablished programming language.

Nowadays, there is no standardization or even a general preference about the most appropriate programming language to be used in autonomous robotics, and the trend is to continue in the same way. As a consequence, even though two robots may have the same sensors and actuators, if we want to execute the same control architecture on both, we must modify the programming and adapt it to the particular language and implementation of each robot.

In order to deal with this problem, more general frameworks have been developed in recent years trying to achieve complete independence from the robot manufacturer. These tools can be classified as *middleware software* that abstracts the control architecture from the particular hardware. Examples of this kind of tools are Miro (Utz, 2002), Webots (Michel, 2004) or YARP (Metta, 2006) that permit development in a broad range of different robotic systems. Miro is a distributed object oriented framework for mobile robot control, based on CORBA (Common Object Request Broker Architecture) technology. It is focused on wheeled robots such as Pioneer or Sparrow and it does not provide support, for example, for legged robots like Sony's AIBO (very popular in autonomous robotics research) or humanoid prototypes. Webots "provides a rapid prototyping environment for modelling, programming and simulating mobile robots". It includes several libraries that allow the designer to transfer the control programs to many commercially available real mobile robots. Finally, YARP "is written by and for researchers in humanoid robotics, who find themselves with a complicated pile of

Faíña A., Bellas F. and J. Duro R. (2007).
INITIAL DEVELOPMENT OF HABLA (HARDWARE ABSTRACTION LAYER) - A Middleware Software Tool.
In *Proceedings of the Fourth International Conference on Informatics in Control, Automation and Robotics*, pages 53-58
DOI: 10.5220/0001627800530058
Copyright © SciTePress

hardware to control with an equally complicated pile of software”.

The abstraction level provided by these tools could be necessary in other autonomous computational systems (different from robotics) with sensors and actuators of very different nature and with a complex control system like, for example, domotic applications. Taking into account these generalization, in this work we propose the creation of a middleware tool to be applied in different autonomous computational systems characterized by different sensors and actuators controlled through a complex architecture that could be executed remotely. The only reference we have found that follows this philosophy and supports different hardware devices and application fields is Player (Gerkey, 2003). This tool provides an interface and a protocol to manage sensorial devices and robots over a network and accepts different programming languages for the control architecture. It runs on several robotic platforms and supports a wide range of sensors. At this time the developments outside the robotics field are limited. The general middleware tool we propose is called the Hardware Abstraction Layer (HABLA).

2 HARDWARE ABSTRACTION LAYER (HABLA)

The desired features for the Hardware Abstraction Layer can be summarized into a group of six:

Device independence: it must support the most common sensors and actuators present in autonomous computational systems such as cameras, microphones, infrared sensors, sonar sensors, motion sensors, etc. In addition, it should be provided with particular implementations for the most typical commercial platforms, for example, the robots used in research like Pioneer, Kephera, Aibo, etc.

Virtual sensing and actuation: it must provide typical sensorial processing such as color segmentation or sound analysis so that higher level information like distance to nearby objects or sounds can be considered by the control architecture.

Computational cost distribution: it must support communications through a computer network by TCP sockets in order to execute the control architecture, the low-level control program and the different elements of the HABLA itself over different computers. It seems obvious that, for example, sound or image processing should not be executed directly in the robot.

Control architecture independence: it must be independent of the programming language used in the control architecture, this is, we do not impose any particular programming language.

Scalability: the HABLA should present a modular design and an open architecture in order to increase the number of supported sensors and actuators corresponding to new commercial platforms.

Operating System independence: it must be implemented in JAVA to achieve operating system independence. In addition, JAVA is the most standard object oriented language, so the HABLA could easily include contributions from the research community.

Figure 1 shows a general diagram of the Hardware Abstraction Layer for a typical autonomous computational system. The left block represents the control architecture that requests sensorial information from the low level devices and provides the action or actions to be executed through the actuators. A basic idea behind the HABLA development is that we assume that the control architecture requires high level sensorial information, this is, the basic sensorial processing is not executed in the control architecture. In addition, the actions selected can be complex actions, and not only individual commands to the actuators.

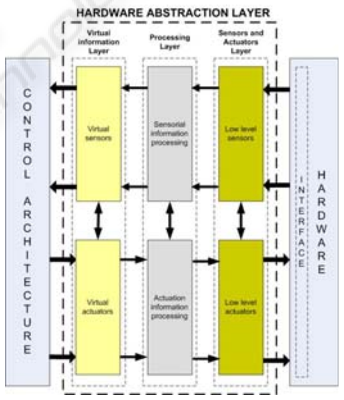
HARDWARE ABSTRACTION LAYER
Virtual information Layer
Processing Layer
Sensors and Actuators Layer
CONTROL ARCHITECTURE
Virtual sensors
Sensorial information processing
Low level sensors
Virtual actuators
Actuation information processing
Low level actuators
INTERFACE
HARDWARE

Figure 1: General diagram of the Hardware Abstraction Layer for a typical robotic system.

The right block in Figure 1 represents the hardware (sensors and actuators) that provides sensorial information to the control architecture and receives the action or actions that must be applied. In this case, the sensors provide low level information (with no processing) and the actuators require low level data too.

As shown in Figure 1, the middle block represents the Hardware Abstraction Layer, an element that isolates the high level information handled by the control architecture from the low level information handled by the sensors and actuators. The communications between control architecture and HABLA and between HABLA and hardware use TCP sockets, as commented before, in order to permit the distributed execution of these three basic elements.

Inside the HABLA we can see three sequential layers: the *sensors and actuators layer*, the *processing layer* and the *virtual information layer* in order of increasing processing of the information. The HABLA is implemented using JAVA and each layer contains methods that perform a particular function. The methods of a given layer can use and provide information from/to the neighboring layer as represented by the arrows in Figure 1. The information exchange between methods of the same layer is also possible, but not the exchange between non-neighboring layers (such as the case of the *sensors and actuators layer* and the *virtual information layer*). The *sensors and actuators layer* includes a general set of methods that store the sensorial information provided by the physical devices and that provide the commands to be applied to them. These methods may perform some kind of processing, as we will see later, and provide their outputs to the methods of the *processing layer*. In this layer, the sensorial information is processed in a general way, carrying out common signal processing tasks. In addition, the general processing of the commands is executed in this layer when required. The last layer is the *virtual information layer* where the information provided by the methods of the *processing layer* is treated and presented to the control architecture. As we can see, we are assuming a very general case where the low level sensorial information must be treated in two higher levels prior to the presentation to the control architecture. This scheme includes the simple case where the control architecture requires low level information, because "trivial" methods that simply transmit this information without processing could be present in the *processing layer* and *virtual information layer*.

Although the HABLA has been designed to be run in a single computer, the methods are independent and can execute a routine or program in a different computer by means of TCP socket communications. This way, a highly time consuming process can be run outside the HABLA computer to improve efficiency.

All of the methods present in the HABLA must be as general as possible in order to apply the HABLA to very different hardware devices or robotic platforms without changes. This is achieved by establishing a clear methodology in the creation of the methods for the *sensors and actuators layer* and the *virtual information layer*. In the case of the low level methods, we have created a protocol that must be followed by any software that controls the hardware at low level. As displayed in Figure 1, the right block that represents the hardware includes an internal part called *interface*. This element represents the methods or routines that must be programmed in the native language of the hardware device controller in order to provide sensorial information to the HABLA. For example, if the HABLA is working with a given robotic platform and we want to use a different one, we will have to program this interface layer in the new robot to communicate it with the HABLA according to our simple protocol.

In the case of the high level methods (*virtual information layer*), the HABLA is endowed with a configuration file that provides the list of active TCP sockets and the information that is provided on each. The control architecture that uses the HABLA must be reprogrammed in order to read the sensorial information or to write the commands in the appropriate socket. But, as commented before, a very important feature of the HABLA is that no limitation is imposed on the type of programming language for the control architecture.

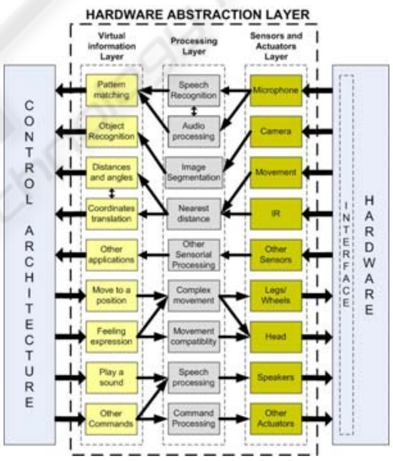

Figure 2: Diagram of the Hardware Abstraction Layer with sample methods for the case of an autonomous robot.

Figure 2 shows an example of a more detailed diagram of HABLA in the typical autonomous computational system we are dealing with, containing some of the methods that have been implemented in the HABLA at this time. In the *sensors and actuators layer* we have methods that store sensorial information from typical sensors such as microphones, cameras, infrared sensors, sonar sensors, bumpers, light sensors, GPS, motion sensors, etc. In addition, in this layer we have methods that send actuation commands, such as movements of the legs, wheels or head, or a sound to be played by the speakers, to the interface layer.

In the *processing layer* we have methods that carry out, for example, speech recognition or image segmentation and methods that can compose complex actuations or control and prevent impossible movements. These are typical processing routines, general to different robotic platforms and environmental intelligence systems. On one hand, these methods need sensorial information from the low level methods and provide information to be used by different methods in the *virtual information layer*. On the other, these methods receive data from the high level ones and execute low level methods to apply the commands. In general, the methods of this layer perform a processing function required by more that one high level method or that affect more than one actuator. For example, as represented in Figure 2, the information provided by the sonar sensors and by the infrared sensors could be combined in method that provides the distance to the *nearest object*.

The *virtual information layer* has methods that perform high level processing and present the information to the control architecture. An example of this kind of applications could be an *emotion recognition method* that provides the control architecture a string of data corresponding to a sentence or word that the user has said to the system with information about the intonation or the volume to detect the emotion in the user. This method needs information from the *speech recognition method* that provides the spoken sentence or word and information from the *audio processing method* that provides details of the physical signal in order to determine an emotion.

In Figure 2 we have represented two typical communications between methods of the same layer. For example, in the *virtual information layer* communications could take place between the method that calculates the *distance and angle* to all the objects in the vision field of the robot and the method that performs the *translation of coordinates*.

After presenting the general HABLA structure, in the next section we will try to make it clearer through robotic application examples.

3 PIONEER 2 WITH MDB

In order to show the basic operation of the HABLA in a real experiment with a real robotic platform, we have decided to reproduce the example presented in (Bellas, 2005). In this experiment we used a wheeled robot from Activmedia, the Pioneer 2 DX model, and a deliberative control architecture developed in our group called the Multilevel Darwinist Brain (MDB) and first presented in (Duro, 2000).

The MDB is a general cognitive architecture that has been designed to provide an autonomous robot with the capability of selecting the action (or sequence of actions) it must apply in its environment in order to achieve its goals. The details of the MDB are not relevant in this work and can be found in (Bellas, 2005). In the experiment presented in that paper we demonstrate the basic operation of the MDB in a real robot with a high level task. As commented before, the robot was a Pioneer 2 DX robot, a wheeled robot with a sonar array around its body and with a platform on the top in which we placed a laptop where the MDB was executed. Basically, the experiment consists on a teacher that provides commands to the robot in order to capture an object. The commands were translated into musical notes perceived by a microphone. Initially, the robot had no idea of what each command meant. After sensing the command, the robot acts and, depending on the degree of obedience, the teacher provides a reward or a punishment through a numerical value as a pain or pleasure signal introduced via keyboard.

The main objective was to show that the MDB allows the agent to create, at least, two kinds of models that come about when modeling different sets of sensors: one related to the sound sensor for the operation when the teacher is present and an induced model or models relating to the remaining sensors. The robot will have to resort to these models when the teacher is not present in order to fulfill its motivations. In this experiment, an induced behavior appears from the fact that each time the robot applies the correct action according to the teacher’s commands, the distance to the object decreases. This way, once the teacher disappears, the robot can continue with the task because it developed a satisfaction model related to the remaining sensors that tells it to perform actions that reduce the distance to the object.

The execution of this experiment as explained in (Bellas, 2005) involved the programming of all the sensorial processing in the MDB. The sonar values were processed in a function to calculate the distance and angle to the object, and the audio signal perceived through the microphone was analyzed and treated in another function. The action selected by the MDB was decoded into the Pioneer ranges in another function that was programmed in the MDB. As we can see, with this basic set up we were overloading the laptop’s CPU with the low level tasks and with the high level calculations (MDB).

At this point, we decided to introduce the HABLA with the set of sensors and actuators of this robot. Figure 3 shows the basic HABLA diagram particularized for this example, with the methods developed. In the *sensors and actuators layer* we have five methods according to the sensors and actuators used in the experiment. For example, the *Microphone* method reads from a socket the sound data received in the laptop's microphone and performs a basic filtering process to eliminate signal noise and provides the data to the methods of the next layer. In the *processing layer*, we have included six methods that provide typical processed information. For example, the *Nearest Distance* method receives an array of sonar values, detects the nearest one and provides this information to the *Distance and Angle* method. In the last layer (*virtual information layer*) we have programmed six high level methods. To continue with the same example, the *Distance and Angle* method calculates the angle from the information provided by the *Nearest Distance* method and sends the MDB the exact distance and angle in a fixed socket.

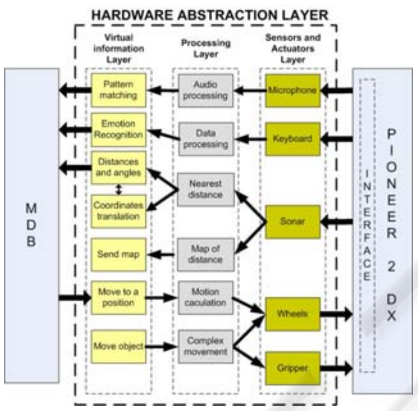

Figure 3: HABLA diagram with the particular methods of the Pioneer 2 robot.

With this basic implementation of the HABLA, we have re-run the example presented in (Bellas, 2005) obtaining the same high level result, this is, the Pioneer 2 robot autonomously obtained an induced behavior. What is important in this experiment is that, using the HABLA, the MDB doesn't have to compute low level processes. This allows us to work with a more general version of the architecture which is highly platform independent. In addition, we can execute the MDB in a much more powerful computer and use the laptop just for the HABLA and the communications.

4 AIBO WITH MDB

Once the successful operation of the MDB with a real robot has been shown, our objective is simply to repeat the experiment but using a different robot, in this case the robot is a Sony Aibo. The example is the same as in the previous case from the control architecture's point of view, but, as the robot is different, we have used a different group of sensors and actuators. In this case, the Aibo robot has to reach a pink ball it senses through a camera and the commands are spoken words provided by the teacher. In addition, the punishment or reward signal is provided by touching the back or the head of the robot, this is, using a contact sensor. The robot movements are different and it is able to speak some words through its speakers to show some emotion. In this case, the experiment was performed in a closed scenario with walls.

Figure 4 represents the HABLA with the new methods included for this robot. The philosophy behind the programming of the new methods is that they should be as general as possible in order to be useful for other robotic platforms similar, in this case, to the Aibo robot. Furthermore, we can see in Figure 4 that the previous methods developed for the Pioneer 2 robot are still present and, as we will explain later, some of them are used again.

The first thing we had to do in this experiment was to program the low level routines in the Interface layer of the Aibo robot using the Tekkotsu development framework (Touretzky, 2005). In this case, this tool follows the same idea as we use in the HAL, and all the sensorial information from the robot can be accessed by TCP sockets, so programming cost involved was very low. In the case of commands, Tekkotsu provides very simple functions to move the legs with a given gait that are accessed by sockets again.

In the sensors and actuators layer, we have included very general methods to deal with sensors such as a camera, infrared sensors, buttons or with actuators like a head or a speaker. In the processing layer we have included, as in the case of the Pioneer robot, very general processing related with the new sensors of the previous layer, such as image segmentation or speech recognition. In fact, the speech recognition method was the most important development in this experiment because this feature is not present in Tekkotsu software or in Sony's original framework. We think that owner-dog communication through spoken words is very important because it is a very intuitive way to teach this robot. In fact, the speech recognition was implemented using Sphinx-4 (Walker, 2004) which is a speech recognizer written entirely in the Java programming language. In our case, the Speech Recognition method basically executes Sphinx-4, which obtains the sound data from the microphone method, and outputs a string of data with the recognized word or phrase. In this case, we have used a reduced grammar but Sphinx includes a configuration file where more words or phrases can be added, so the method is very general. Aibo is always sending the data of the two microphones to a fixed port with a Tekkotsu behavior called "Microphone Server".

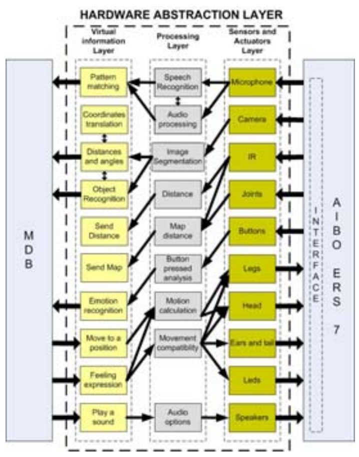

Figure 4: HABLA diagram with the particular methods of the Aibo robot

As shown in Figure 4, in the *processing layer* we find, for example, a method called *Audio Processing* that was created for the Pioneer robot, and is reused here. In the *virtual information layer* we have created more abstract methods than in the previous case, because the new sensors and actuators of this robot (like the buttons in the back or the head) permit us to create new methods such as *Emotion Recognition*, that provide information to the MDB related to the teacher's attitude.

Finally, we must point out that the execution result was successful, obtaining exactly the same behavior as in the Pioneer robot (Bellas, 2006). What is more relevant in this case is that there was no time spent in MDB reprogramming, because using the HABLA the low level processing was absolutely transparent to the control architecture. In addition, in this experiment we have executed the Tekkotsu software on the Aibo's processors, the HABLA in another computer and the MDB in a different one, optimizing this way the computational cost.

5 CONCLUSIONS

In this paper we have presented the initial implementation of the Hardware Abstraction Layer (HABLA) middleware tool. Its main features are: hardware devices independence, virtual sensing and actuation capabilities, computational cost distribution, control architecture independence, scalability and operating system independence. We have presented practical implementations of the methods in the HABLA that support two very different robotic platforms (Pioneer 2 and Aibo) in a real application example using the MDB control architecture. Currently, we are expanding the HABLA concept to different application fields, developing a practical example in an "intelligent" room.

ACKNOWLEDGEMENTS

This work was partially funded by the Ministerio de Educación y Ciencia through projects DEP2006-56158-C03-02/EQUI and DPI2006-15346-C03-01.

REFERENCES

AIBO SDE webpage, 2007: http://openr.aibo.com/

ARIA webpage, 2006: http://www.activrobots.com/SOFTWARE/aria.html

Bellas, F., Becerra, J.A., Duro, R.J., 2005. Induced behaviour in a Real Agent using the Multilevel Darwinist Brain, *LNCS*, Vol 3562, Springer, 425-434.

Bellas, F., Faiña, A., Prieto, A., Duro, R.J., 2006, Adaptive Learning Application of the MDB Evolutionary Cognitive Architecture in Physical Agents, LNAI 4095, 434-445

Duro, R. J., Santos, J., Bellas, F., Lamas, A., 2000. On Line Darwinist Cognitive Mechanism for an Artificial Organism, *Proc. supplement book SAB2000*, 215-224.

Genesereth, M.R., Nilsson, N., 1987. Logical Foundations of Artificial Intelligence, *Morgan Kauffman*.

Gerkey, B. P. , Vaughan, R. T., Howard, A., 2003. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems, *In Proc. of the International Conference on Advanced Robotics*, 317-323.

Metta, G., Fitzpatrick, P. Natale, L., 2006, YARP: Yet Another Robot Platform, *International Journal on Advanced Robotics Systems*, 3(1):43-48

Michel, O., 2004. Webots: Professional Mobile Robot Simulation, *International Journal of Advanced Robotic Systems*, Vol. 1, Num. 1, 39-42.

Touretzky, D. S., Tira-Thompson, E.J., 2005. Tekkotsu: A framework for AIBO cognitive robotics, *Proc. of the Twentieth National Conference on Artificial Intelligence.*

Utz, H., Sablatnög, S., Enderle, S., Kraetzschmar, G, 2002. Miro - Middleware for Mobile Robot Applications, *IEEE Transactions on Robotics and Automation, Special Issue,* Vol. 18, No. 4, 493-497.

Walker, W., Lamere, P., Kwok, P, Raj, B., Singh, R., Gouvea, E., Wolf, P., Woelfel, J., 2004. Sphinx-4: A flexible open source framework for speech recognition, *Technical Report SMLI TR2004-0811, Sun Microsystems, Inc.*