# Programming Model Based on Concurrent Objects for the AIBO Robot

Francisco Martín Rico, Rafaela González-Careaga, Jose María Cañas Plaza, Vicente Matellán Olivera

Grupo de Sistemas y Comunicaciones, ESCET, Universidad Rey Juan Carlos,
C. Tulipn S/N CP. 28933 Mstoles (Madrid), Espaa.
{fmartin,rafaela,jmplaza,vmo}@gsyc.escet.urjc.es

**Abstract.** *This paper presents the object-oriented programing environment for the AIBO robot, focusing in its concurrency model. Concurrency problems arise when programming a robot due to the various sensors and actuators that the programmers must manage. As this management has some aspects of real time and communication, involves some coplexity. In order to deal with this complexity Sony has developed a framework for programming the AIBO Robot. The description of this API called OPEN-R is presented. Also we will discuss how the underlying operating system, APERTOS, hides almost all the complexity of implement applications compounded of concurrent objects intercommunicated.*

## 1 Introduction

Robotics is a research area that has gained increasing attention in recent years. Its main goal is to build machines capable of carrying out task, with the flexibility, robustness and performance exhibited by humans. The robots are primarily intended to dangerous, dull, dirty, or difficult scenarios. Wielding and painting in car factories, space rovers, or nuclear plant cleaning are only some samples of their real application scope.

One of the main issues in robotics research is the autonomy, that is, the generation of autonomous behaviors in robots. Growing in robot autonomy opens the door to new applications like service or entertainment robotics. Remarkable advances in such direction are the robotic vacuum cleaner from iRobot and the AIBO dog from Sony, a best-seller robotic pet.

Mobile robots are no more than platforms where sensors, actuators and computers are combined. The software that runs in such computers determines the robot behavior, and provides its autonomy steadily deciding how to act. Generating autonomous behaviors lies in writing programs for the robot computers. From that programming standpoint the robots, like general purpose computers, are endowed with some operating system which provides an API for basic access to hardware resources like sensors and actuator devices.

Besides such basic access, the programming environment may provide some features to make easier the software development of robot applications. For instance, robot programs usually have to take care of many tasks at the same time: reading sensor measurements, sending motor commands, deciding what to do next, planning what to do in the future, displaying and receiving events from some graphical interface, etc.. To support such parallel nature the development environment has to provide a model for concurrent programming. Multi-threading also allows for distribution in the robot applications.

Another desirable feature of robot software is modularity in order to ease reusing of its components. This results specially useful in robotics as long as new behaviors can be achieved modulating or combining the existing ones. In such a way, new software environments have appeared in last years which offer Object Oriented Programming for several robot platforms: ARIA, for Pioneer robots (from Activmedia), Mobility for B21 robots (from iRobot) and even OPEN-R software for AIBO dog (from Sony).

In this paper we are going to explore in depth the concurrent object oriented programming model for the AIBO robot. This robot is very popular and has become an exciting benchmark for Artificial Intelligence and Robotics research (RoboCup [7]). Surprisingly we have detected a lack of detailed documentation about its programming. The already existing one is scarce and fragmented.

The API, build on top of C++, runs on APERTOS and it is provided by Sony as the OPEN-R SDK (figure 3). With OPEN-R SDK API, features as, make AIBO's joints move, get information from sensors, get image from camera, use wireless LAN(TCP/IP) can be achieved easily.

The Sony AIBO ERS7 (figure 1 and 2) robot is a completely autonomous robot which incorporates an embedded MIPS processor running at a clock frequency of 576MHz, and 64MB of main memory. It gets information from the environment through various hardware components including: a 350K-pixel color camera, 2 infrared sensors to detect distance and edges, an acceleration sensor, a vibration sensor, a stereo microphone, paw sensors in the feet to detect whether a leg is or not on the ground, and tactile sensors in the head, chin and back. AIBO moves and interacts with its environment through various actuators including: a speaker, and 20 motors to move each of the three joints of its four legs, each of the three joint of its head, its mouth, its ears, and both joints in the tail. AIBO also incorporates a IEEE 802.11b Wireless LAN card.

This paper is organized as follows: in section 2 we will discuss the main ideas of APERTOS, we will also show the advantages that OPEN-R, a framework for programming at the top of APERTOS, gives to the robot programmer. We will illustrate the OPEN-R SDK API with an example application in section 2, in which we will see how the concurrency problems are solved in a transparent way by the system. In section 3 an application of this will be shown. Finally, in section 4, some remarks are made.

## 2 OPEN-R

The OPEN-R SDK is a framework that offers a transparent way to program the Sony AIBO Robot. OPEN-R is the interface that Sony promotes to expand the capabilities of entertainment robots. This interface is layered and optimized

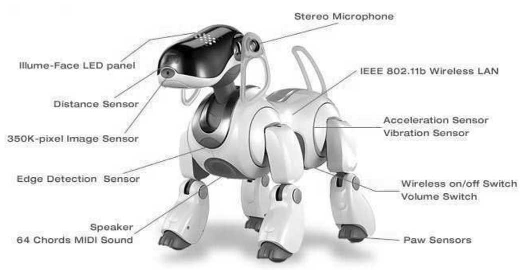

**Fig. 1.** Sony AIBO ERS7. Front

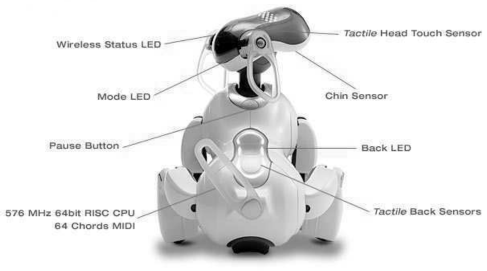

**Fig. 2.** Sony AIBO ERS7. Back

to let the programmers develop efficient software for AIBO. OPEN-R is built on top of APERTOS operating system. In next sections we will describe both elements making emphasis in OPEN-R.

### 2.1 APERTOS

APERTOS is an object-oriented embedded operating system based on meta-level architecture[1][2]. Many of the APERTOS' design concepts have a heavy weight in the way AIBO is programmed with OPEN-R.

Everything in APERTOS is an object. Each object encapsulates the state, methods which access such state, and a virtual processor which executes its methods.

The communication inter-objects is made by message passing and the object execution is guided by events. After the initialization, an object uses to be idle. When an object wants to communicate with others, it sends a message, writing the data in shared memory and sending an event to the destination object, which will eventually be activated, it will read the data and handle it depending on the message type has arrived. These events has some assigned priority so that

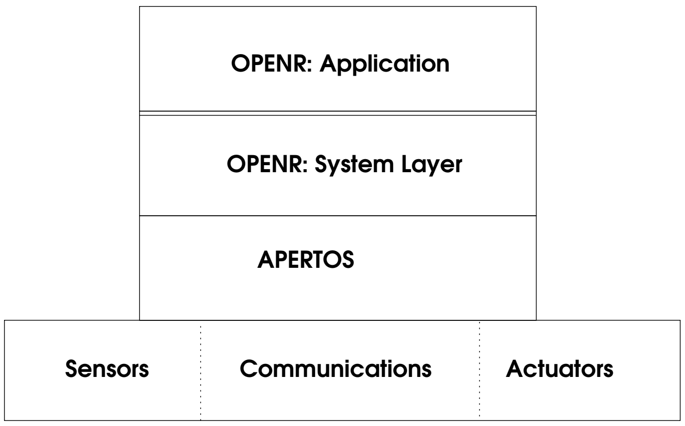

**Fig. 3.** Software/Hardware Architecture

it can be distinguished between an ordinary event from a hardware interruption or other events.

When an object is performing an operation (a method typically) it will not be interrupted by a event or interruption until the operation is finished to avoid race conditions. It will achieve this by interruption masking. The message that can't be immediately managed is stored in a message buffer allocated in shared memory. Although all the objects share the same memory space, no object can overwrite any data belonging to other object. The only exception is in message buffer case. The message delivered to other object is allocated in a shared region in memory. APERTOS does not provide a transparent way to protect this memory. It only provides a counter for references to a memory region.

The objects uses a meta-hierarchy of meta-objects to define its behavior. The set of meta-object that a object uses is called meta-space. If a object, for example, wants TCP/IP communication and its meta-space do not support this, the object can migrate to other meta-space that support this kind of communication.

APERTOS is used in the AIBO robot and in DST-MS9, the Set Top Box for CS satellite broadcasting (only in Japan). We have no further information about others devices works with APERTOS.

## 2.2 OPEN-R basis

The features of OPEN-R software are:

- **Modularized Software and inter-object communication.** OPEN-R inherits the properties of its underlying operating system and simplifies it. In the inter-object communication process OPEN-R does not make use of the message priorities, all messages have the same priorities.
  This communications between objects are set up in an external configuration file, which is loaded when the robot boots. Communication ports in objects

are uniquely identified by the service name, which let high modularity in object construction. Modularity allows parallel processing, clarity of design and an easy way of reusing pre-existing objects.

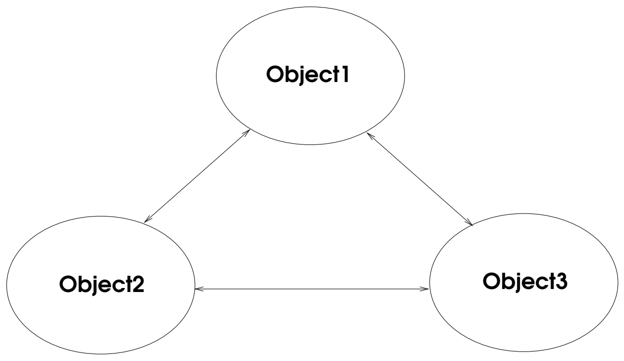

**Fig. 4.** OPEN-R Application Software

– **Layered structure of the software and services provided by the system layer.** It is important to note a difference between the so called "system layer" and the "application layer". The first one contains all the services necessary to access the robot hardware (through the special objects, and the application layer, is the one programmed by the user and the one that uses the system layer interface so that it can access to the robot hardware without knowing it in detail.
The three principal objects provided by the system layer are:
  1. **OVirtualRobotComm**: this object provides the following services to the application layer:
     - **OVirtualRobotComm.Effector.** This service receives joint and LED commands from the application layer's objects. The structure in which the data is stored is OCommandVectorData. In the OCommandVectorData structure, different kinds of commands can be stored at the same time, so with one single command, we can change several LEDs and joints values.
     - **OVirtualRobotComm.Sensor.** This service sends data referred to the sensor and joint values. The structure in which the data is stored is OSensorFrameVectorData. Also, each OSensorFrameVectorData might store information about several sensors at the same time.
     - **OVirtualRobotComm.OFbkImageSensor.** This service sends image data that is captured through the camera situated in the front side of the robots head. The structure in which the camera information is stored is OFbkImageSensor. By this structure we can access to the image in high, medium and low quality. Too we can access by this structure to a color based image segmentated. This is because

   OPEN-R implements a fast customizable method for color segmentation based on color.

2. **OVirtualRobotAudioComm**: this object provides the following services to the application layer:
   - **OVirtualRobotAudioComm.Mic.** This service sends data taken from the microphones situated in the robot's ears. The structure in which data is stored is OSoundVectorData. Sound data is sent every 32ms. The sound data is in the following format: PCM data, 16KHz and 16bit stereo.
   - **OVirtualRobotAudioComm.Speaker.** This service receives sound data that will be emitted from the robot's speaker. The structure in which data is stored is OSoundVectorData.
3. **ANT object**: The TCP/IP comunnication service is provided by the ANT (OPEN-R Networking Toolkit) object. We create an endpoint for the communication, and two buffers: one for input data and other for output data.

   First of all, we must create two buffers, one for incoming packets an another one for outgoing messages. This is performed by ANT primitives. Once this is done, we create an endpoint for each connection we can handle. For this two steps, this object send synchronous calls to the ANT object. The ANT object implements IPv4 stack. Four functions are commonly used for sending and receiving TCP/IP messages:
   - **receive.** We send an OPEN-R message to the ANT object telling it our intention to receive a TCP/IP packet. This is asynchronous, and because of this, in this message we tell which is the method (let's say *receiveCont*, for example) which has to handle the packet when ANT object receives a message.
   - **receiveCont.** When a packet arrives to the endpoint we had set up., ANT activates *sendCont* method to handle the information that the TCP/IP packet contains.
   - **send.** As we did in *receive*, we send an OPEN-R message to the ANT object telling it our intention of sending a TCP/IP packet and we setup data to be sent. This is synchronous too, so we specify which method (let's say *sendCont*, for example) has to be activated when the packet has been successfully sent, or has been an error in the communication.
   - **sendCont.** This method usually handles communications errors and calls *receive* to get next packet.

The mentioned System Layer's services that get data from the application layer (Effector and Speaker), send a event indicating the object that they are ready to receive more data once they have handled the last sent data. On the other hand, the services that send data to the application layer (Sensor, FbkImage and Mic), send a event indicating the object that they have actually sent the required data.

## 2.3 OPEN-R objects

The concept of an object in OPEN-R is similar to the concept of a process in UNIX, considering the following characteristics:

- Ojects are single-threaded.
- In order to solve the mutual exclusion problem in the shared memory region exposed previously, OPEN-R provides the `RCRegion` Class that can access the shared memory segment (provided by OPEN-R so that objects can place the data they want to send to each other) and gives a reference counter which holds the number of objects that have a pointer to a memory region allocated and behave as a mutual exclusion lock for this memory region.
- The objects in the system communicate by message passing, we say that an object that sends a message behaves as a subject, and an object that receives a message has an observer. The message contains data, (any C++ type) and an identifier (selector) that specifies which method to be executed when the message arrives. So, an object has several entry-points, unlike usual programs in personal computers which execution starts at `main()` function, in OPEN-R not only the `DoInit()` method, but any of the functions or methods that are invoked when a message is received, are also entry-points. As objects are single-threaded, they can only handle one message at a time, so if a message is received while another one is being handle, it is put in the message queue and processed later. There is one message queue per object. While the body of a message is being executed, if a new message is received, the execution does not stop, it goes on till the method is completely executed, and then the new entry point will be activated.
  This is the flow of execution of OPEN-R objects (figure 5):
  - The object is loaded when the robot boots, the textttDoInit method is invoked
  - The object waits for a message to arrive.
  - When a message arrives, the method corresponding to the selector included in the message is invoked. The object might send some message to other objects.
  - When the method invoked finishes its execution, it waits for another message to arrive and so on. In this case, the object we are referring to in this example is an observer as it is waiting for a message coming from a subject. An observer knows that a message containing new data has arrived when it receives a Notify Event from the Subject. A Subject knows that the Observer is ready to receive new more data when it gets a Ready Event from the Observer.

## 2.4 Using OPEN-R

In this section we will explain all the process in the generation of a new application, from the configuration of objects, comunication and services, to the compilation and loading in the robot. We are going to illustrate this with the Teleoperator example configuration files (Section 5)

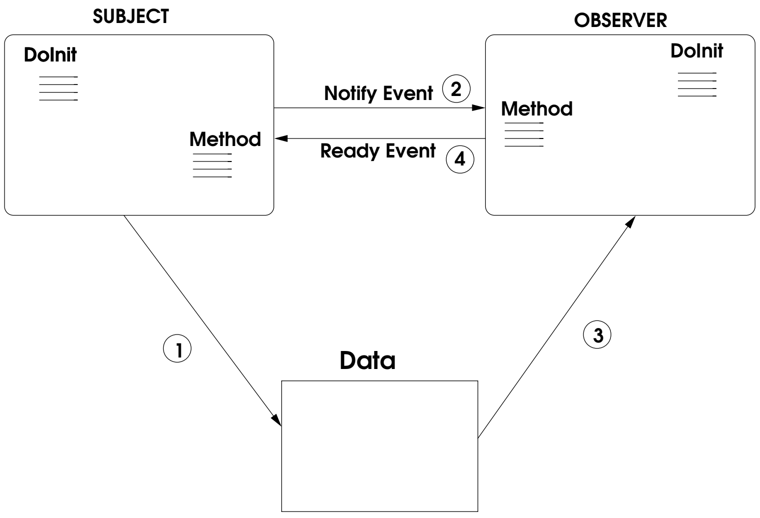

**Fig. 5.** Inter-Object Communication Flow

**Object definition** One object corresponds to one executable file (with .bin extension) that will be loaded when the system boots. The objects to be loaded are set up in the `OBJECT.CFG` file, which is the list of object to load.

**Object's services configuration** The number of subjects and observers for an object is defined in the configuration file `STUB.CFG`. In this file, the type of sent data and the methods to be invoked, are also specified. The notion of stub file is similar to *IDL file*s in CORBA and *stubs* in Java RMI, where these files defines the inter-object interfaces to achieve remote method invocation. OPEN-R also provides a way for execute objects in a distributed way. This is useful, for instance, for debugging the objects in a PC. This is a example of `STUB.CFG` file:

```
 ObjectName : teleoperatorServer
NumOfOSubject   : 2
NumOfOObserver  : 1
Service : "teleoperatorServer.SendString.char.S", null, null
Service : "teleoperatorServer.Motion.char.S", null, null
Service : "teleoperatorServer.ReceiveString.char.O", null,
Notify()
```

**Network configuration** through the `WLANCONF.TXT` file OPEN-R provides a method for configure a wireless connection with AIBO. Here we specify all the aspects related to the network.

```
HOSTNAME=AIBO
ETHER_IP=193.147.71.22
ETHER_NETMASK=255.255.255.128
IP_GATEWAY=193.147.71.1
ESSID=GSYC_WLAN_SS4
WEPENABLE=0
#WEPKEY=
APMODE=2
#CHANNEL=
DNS_SERVER_1=193.147.71.64
```

**Object's services communication configuration** In the `CONNECT.CFG` file, the way in which observers and subjects are connected, is specified:

```
#
# OVirtualRobot --> ImageProcesser
#
OVirtualRobotComm.FbkImageSensor.OFbkImageVectorData.S
        ImageProcesser.Image.OFbkImageVectorData.O

#
# ImageProcesser <--> TeleoperatorServer
#

TeleoperatorServer.SendString.char.S
        ImageProcesser.ReceiveString.char.O
ImageProcesser.SendString.char.S
        TeleoperatorServer.ReceiveString.char.O

#
# TeleoperatorServer --> Motion

TeleoperatorServer.Motion.char.S Motion.Orden.char.O

#Motion--> locomotion
Motion.ExecuteAA.AtomicAction.S
        actuatorControl.ExecuteAA.AtomicAction.O

actuatorControl.ECommander.OCommandVectorData.S
        OVirtualRobotComm.Effector.OCommandVectorData.O
actuatorControl.Head.OCommandVectorData.S
        OVirtualRobotComm.Effector.OCommandVectorData.O
actuatorControl.Indicators.OCommandVectorData.S
        OVirtualRobotComm.Effector.OCommandVectorData.O
actuatorControl.Effectors.OCommandVectorData.S
        OVirtualRobotComm.Effector.OCommandVectorData.O
```

**compiling objects** For the compilation, we use in a cross compiler for the MIPS architecture.

**Transferring binaries to the robot** Once the binaries are generated in the PC, we transfer them through a card writer to a memory stick that will be inserted in a slot at the robot

**Monitorization** through a telnet session at a given port the booting and the output messages are displayed. This can be used for debugging purposes and so on.

## 3 Teleoperator

Here we present an example application that uses the concept discussed before. The application is an AIBO Teleoperator which is divided in a client and a server.

The **Teleoperator client** runs in a PC. It creates a GUI ( figure 6) which is composed by an image box for display the images captured in the AIBO camera, box for controlling the direction and velocity of actuators and a set of buttons for selecting the image size and mode. Too a small box at the bottom shows the frame per second rate.

This application is designed for display images on demand from AIBO and sending commands to actuators.

Three image sizes are defined (hight, medium and low resolution) and two modes (color and black & white). Obviously, this has an impact in the number of frames per second the robot can transmit.

When we have active any mode of image capture, the client demands an image to the server. When it receives the image, it displays the image and it asks for another one.

When we have the Teleoperator active, clicking in the visual joystick we can setup the velocity and direction of movement. We can stop the movement in any moment too, clicking at the stop button.

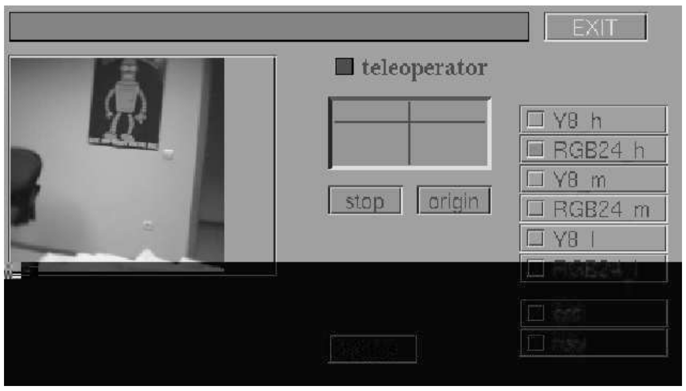

**Fig. 6.** Teleoperator GUI on PC side

The communication with the AIBO side uses a TCP/IP protocol for sending commands and receiving images.

The **Teleoperator server** runs on the AIBO robot and perform the motion commands selected in the client, sending the images from its camera.

## 3.1 Client/Server design

The figure 7 shows the high object level design. The blue circles represents OPEN-R services that provides communication, motion control and camera data. The left side separated by a vertical line correspond to the client allocated in the PC. The right side represent the server allocated in AIBO. The red arrow represent TCP/IP communication and the black arrow OPEN-R messages. Note that not all the arrows are bidirectional.

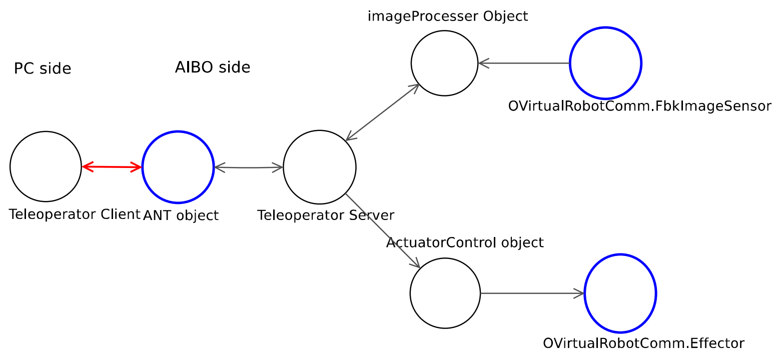

**Fig. 7.** Teleoperator high level design

The OPEN-R objects implemented in the server are:

**Teleoperator Server** This object receives commands from the client in the Pc side via TCP/IP. First, it checks whether the message is a motion command or a vision command. If the message is an image command it sends an OPEN-R message with the mode of the image asked to the imageProcesser object. If the message is a motion command it sends an OPEN-R message with the new kind of movement to the actuatorControl object.

**imageProcesser object** Each time this object receives and message asking for a image, the object communicates with the camera and get the image. It may include some image, and this is the reason for heaving a separate object.

**actuatorControl object** This object commands the movement of the robot according to the motion order it received last time. When it receives a new motion command, it changes its dynamic of movement.
This object has not been implemented by us. It is the motion module implemented in the University of New South Wales (UNSW) [8] and National ICT Australia (NICTA) for the Robocup Championship. This is so by the complexity of implement our own motion module.

## 4 Conclusions

As we have shown, the programming in AIBO is performed by the OPEN-R API. OPEN-R has several characteristics that APERTOS provides: single-threaded objects, TCP/IP communication, memory protection, message passing. The OPEN-R API goal is to provide an interface for the managing of these aspect in a convenient and easy way, providing an interface to the robot programmer, so that he only has to worry about the access to low-level components of AIBO as actuator and sensors.

In the Teleoperator example we shown how an application is implemented defining the objects and the communication that will be established between them when writing such example. We have not had any concurrency or mutual exclusion problems in mind, and we have concentrate our efforts in the access and management of sensors and actuators of AIBO Sony robot.

## References

[1] Yasuhito Yokote. “The Apertos Reflective Operating System: The Concept and its implementation”. *Sony Computer Science Laboratory Inc. Japan 1992*

[2] Jun-ichiro Itoh, Yasuhito Yokote. “Concurrent Object-Oriented Device Driver Programming in Apertos Operating System”. *Sony Computer Science Laboratory Inc. Japan 1994*

[3] Franois Serra, Jean-Christophe Baillie. “Aibo programming Using OPEN-R SDK. Tutorial”. *ENSTA. France 2003*

[4] OPEN-R Documentation. *http://openr.aibo.com/*

[5] Jean-Charles Fabre and Tanguy Prennou. ”A Metaobject Architecture for Fault-Tolerant Distributed Systems: The FRIENDS Approach”, *IEEE TRANSACTIONS ON COMPUTERS, Vol 47. January 1998.*

[6] H. Mashuhara, S.Mtsuoka, T.Watanabe, and A. Yonezawa. “Object-Oriented Concurrent Reflective Languages Can Be Implemented Efficiently”, *Proc. FTCS-22, pp.386-395, Boston, 1992*

[7] Robocup Official Site. *http://www.robocup.org/*

[8] rUNSWift Robocup Team. University New South Wales (Australia).*http://www.cse.unsw.edu.au/ robocup/index.phtml*