KRolog: A Prolog based interface for the Khepera robots

Edgardo Ferretti
Laboratorio de Investigación y Desarrollo en Inteligencia Computacional (LIDIC)
Universidad Nacional de San Luis
Ejército de los Andes 950 - Local 106, (5700) - San Luis - Argentina
ferretti@unsl.edu.ar

Marcelo Errecalde
Laboratorio de Investigación y Desarrollo en Inteligencia Computacional (LIDIC)
Universidad Nacional de San Luis
Ejército de los Andes 950 - Local 106, (5700) - San Luis - Argentina
merreca@unsl.edu.ar

and

Guillermo Simari
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Av. Alem 1253, (8000) Bahía Blanca, Argentina
grs@cs.uns.edu.ar

ABSTRACT

In this paper we present *KRolog* a Prolog based interface to work with simulated and real *Khepera* robots. The interface hides low-level robot-computer communication and provides a high-order set of predicates to develop programs in a declarative manner. This paper describes the software we have developed to support the hardware platform we use in cognitive robotics research at the LIDIC.

Keywords: Khepera, Webots, Prolog, Cognitive Robotics, Coordination models.

1. INTRODUCTION

One of the main objectives of the research line "Intelligent Agents" of the LIDIC, is the design, implementation, and application of high-level multi-agent coordination models. This study is carried out through a theoretic and practical approach. The confrontation with the real world is done using a group of *Khepera 2* [1] mobile robots, with capabilities to pick and transport objects and perform different kinds of environment sensing. Moreover, before the direct experimentation with the robots we also perform robots simulations with *Webots* [2], a 3D realistic professional simulator. Furthermore, the use of this simulator allows us to model situations with more than three robots, the number of robots that we have at the laboratory.

Our aim is to develop deliberative agents to control the robots coordination, and many of the aspects related to the robots' behavior require an expressive representation language that easily reflect the decision processes made by the agents. At this end, we decided to develop an interface in Prolog, a programming language that has already been used to develop applications in the field of cognitive robotics [3, 4].

In this way, the use of this interface allow us to ignore the low-level details related with the robots (e.g. dealing with the size of the robot, the steering angles needed, the slippage of the wheels, sensors noise, etc.), and helps us to concentrate on the high-level problem specification.

2. KHEPERA 2 ROBOT OVERVIEW

The *Khepera 2* robot, is a miniature mobile robot that allows confrontation to the real world of algorithms developed in simulation for trajectory execution, obstacle avoidance, pre-processing of sensory information, hypothesis on behaviors processing, among others. Its small size (60 mm diameter, 30 mm height), light weight (approx. 70 grams), and compact shape are ideal for micro-world experimentation. The *Khepera 2* has eight infrared sensors to sense both ambient light levels and proximity to nearby objects. It also has two DC motors that are capable of independent variable speed motion, allowing the robot to move forward, backward, and complete a variety of turns at different speeds.

As can be observed in Figure 1, the *Khepera 2* has several extension modules that can be plugged into the top of the robot. These include an arm with a gripper, a linear vision system, and a matrix vision cam-

era. The *Khepera 2* has an on-board Motorola 68331 (25MHz) processor, 512 KB RAM, 512 KB Flash memory programmable via serial port, and rechargeable NiMH batteries that allows it up to 60 minutes of autonomy. Thus, the *Khepera* has sufficient sensors and actuators to ensure that it can be programmed to complete a wide variety of tasks.

When connected to a host computer through the serial port, the `SemCor` control protocol is used to send control messages to the robot. As the robot may need to send an answer message to the host, ASCII messages are used to communicate between them. Each interaction consist of:

- A command, beginning with one or two ASCII capital letters and followed, if necessary, by numerical or literal parameters separated by a comma and terminated by a carriage return or a line feed, sent by the host computer to the *Khepera 2* robot.

- A response, beginning with the same one or two ASCII letters of the command but in lower case and followed, if necessary, by numerical or literal parameters separated by a comma and terminated by a carriage return and a line feed, sent by the *Khepera* to the host computer.

During the entire communication, the host computer acts as a master and the robot as a slave. All communications are initiated by the master.

Code can also be uploaded into the *Khepera's* memory for a standalone execution. Programs written in C language or in M68000 assembly language, can be compiled under many environments using a cross compiler and uploaded in RAM or flashed in non volatile memory. A complete API is available, either in C or assembly language, for programs to interface with the robot hardware.

3. THE KROLOG INTERFACE

The *KRolog* interface is currently developed as a *Ciao* Prolog [5] module running under the Linux operating system. To our view, *Ciao* is one of the most complete Prolog systems that allows the programmer to use sockets, multi-threads, Java and C embedded code in Prolog programs and vice versa, among others. In addition, it provides a fully integrated programming environment with the text editor *Emacs*, that allows the programmer to run, debug, compile, and syntax correction of Prolog programs.

This interface uses the *KRobot class* [6] to manage the serial port communication with the robot. The *KRobot class* developed by Harlan et al., hides low-level robot-computer communication and allows developers to focus on robot/environment interaction.

As our interface has been programmed in Prolog, it extends the functionalities provided by the *KRobot class* in that it allows representing the knowledge about the world in a declarative manner, and to derive new representations of the world, and use them to deduce what to do.

In Figure 2 the *KRolog* interface scheme is shown. As can be observed it has a three layer architecture and it is able to interact with real robots and simulated ones. This interface has been designed to communicate with *Webots* in the same way it does with the real *Khepera 2* robots.

Next, we describe in details this three layers that compose our interface.

The low level communication layer

This layer handles all the details related with serial communication among the robots and the high-level predicates of our interface. This layer is composed by two modules, one for the real robots and another for interfacing the simulator.

The KRobot class

The *KRobot class* is the base building block for the module that communicates with the robots. This C++ class maintains the information of the robot' state and provides a set of methods equivalent to the `SemCor` protocol commands. For instance, the command to read from the proximity sensors situated around the robot is:

```
N
```

where to this command the *Khepera* would respond with the following string, if it had hit an object by its front part:

```
n,0,259,1023,1023,278,0,0,0
```

The response is returned as a C-style string and must be parsed to determine the values of each of the proximity sensors.

In contrast, if we want to read the proximity sensors of a *Khepera 2* robot associated with an object `r` of the type `KRobot`, we just have to invoke the method `r.readProxSensors();` and it saves these values in an internal structure of the object. Then, each of these sensor values can be accessed by the method `r.getProxSensor(i);` with $0 \leq i \leq 7$.

Webots interface

In *Webots*, the `DifferentialWheels` node defines any differentially wheeled robot. Thus, the *Khepera 2* robot is an instance of the `DifferentialWheels` node with its fields completed to match its shape and functionalities.

In this way, the module that handles the communication between the Prolog interpreter and the simulator translates the predicates available in the *KRolog'* API, to their respective commands of the *Webots'* controllers API.

The interconnection layer

The development of this layer adheres the paradigm of a TCP/IP connection-oriented protocol, using Berkeley sockets.

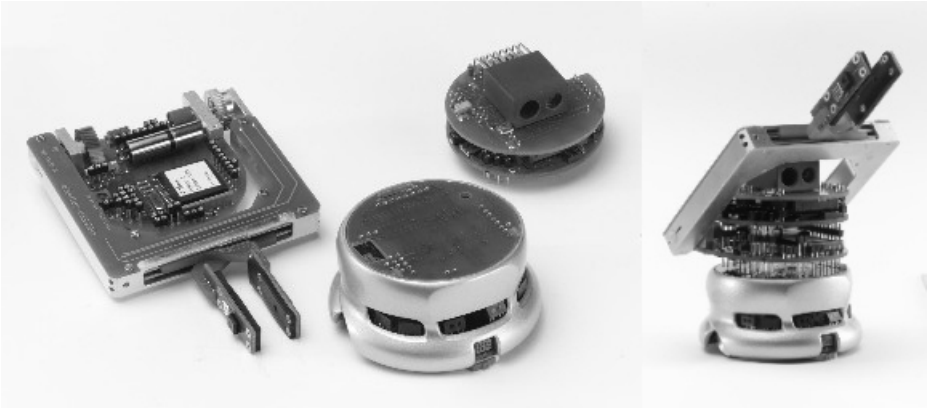

Figure 1: *Khepera 2* robot and its accessories

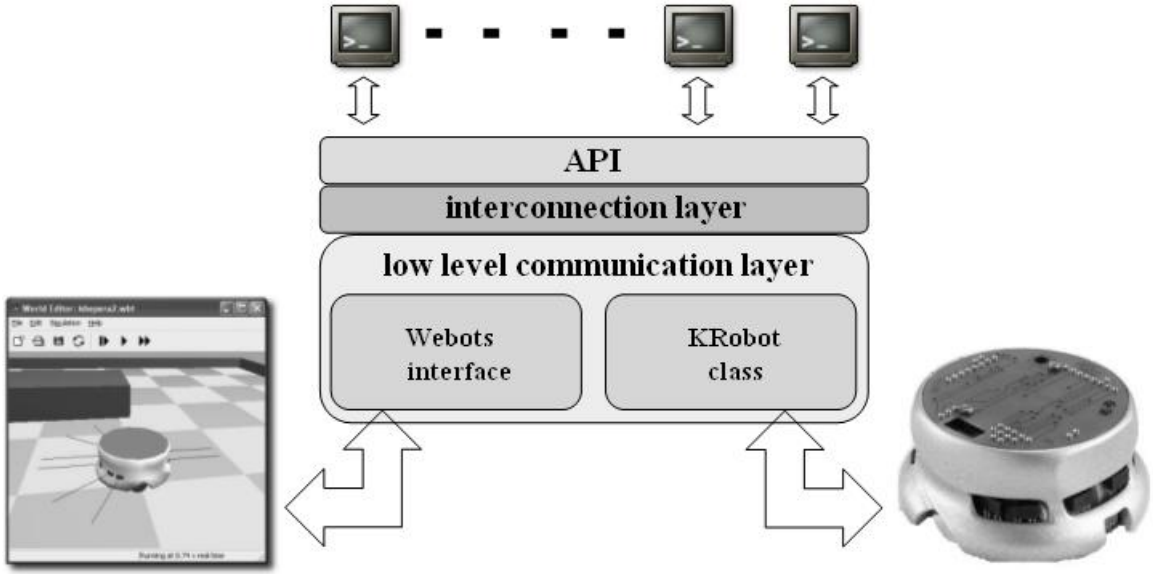

Figure 2: The *KRolog* interface scheme

In Section 2 was mentioned that during the entire communication, the host computer acts as a master and the robot as a slave, and that all the communications were initiated by the master. In consequence, the robots' control modules were programmed with the corresponding code of a server, while the predicates available in the API are seen as clients.

When the API's predicates should sent a command to a real or simulated robot, they launch a temporary client (programmed in C) that communicates to the server (the real or simulated robot) and waits for its answer. This operation is repeated as many times as predicates are used in the Prolog code that controls the robots behavior. In Figure 3, this communication process is depicted.

As a final remark, one advantage of using TCP/IP sockets to develop this layer, is that it makes it possible to interact with a global camera (that covers the robots' world) and its video and command communication servers that process the images it obtains, and generate information packets that are then made available to be used by the agents that control the robots. For instance, one alternative would be using the *Doraemon* video server [7], the one used in the E-League competition [8].

The API

The API is composed by 24 predicates, one for each *KRobot class'* methods. As all the `SemCor` commands after being issued receive a result or a confirmation from the *Khepera* robots, all the predicates have a variable as parameter that matches this answer. Those variables named `Outb` matches boolean values, `Outint` matches integer values, while `Outlist` matches list of atoms. These predicates are used without distinction to communicate with a real or simulated robot. Next, we can see the predicates and a brief explanation for each one:

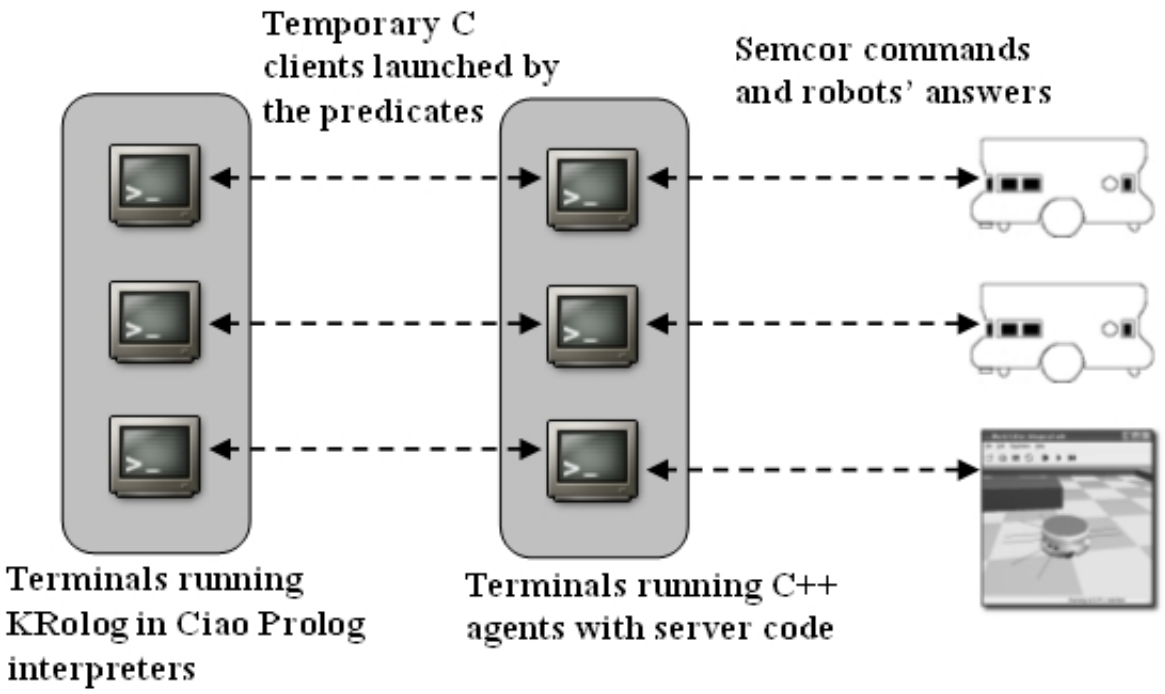

Figure 3: Functioning of the interconnection layer

- `reset(Outb)`: Resets the wheel counters to zero. Sets speed and acceleration to default settings.
- `moveForward(Lw,Rw,Outb)`: Makes the robot' left and right motors to move forward indefinitely at speeds `Lw` and `Rw`, respectively.
- `moveForwardDistance(D,Outb)`: Makes robot move forward `D` millimeters.
- `moveBackward(Lw,Rw,Outb)`: Makes the robot' left and right motors to move backward indefinitely at speeds `Lw` and `Rw`, respectively.
- `moveBackwardDistance(D,Outb)`: Makes robot move backward `D` millimeters.
- `stop(Outb)`: Stops the robot' movement.
- `turnLeft(Dg,Outb)`: Makes robot turn left `Dg` degrees passed as parameter.
- `turnRight(Dg,Outb)`: Makes robot turn right `Dg` degrees passed as parameter.
- `getLeftWheelCounter(Outint)`: Reads and returns the left wheel counter.
- `getRightWheelCounter(Outint)`: Reads and returns the right wheel counter.
- `getLeftWheelSpeed(Outint)`: Reads and returns the speed and direction (+/-) of the left wheel/motor in mm/sec.
- `getRightWheelSpeed(Outint)`: Reads and returns the speed and direction (+/-) of the right wheel/motor in mm/sec.
- `setWheelSpeed(Ls,Rs,Outb)`: Sets to `Ls` and `Rs` the left and right motors' speed and direction (+/-) in mm/sec.
- `getLeftWheelAcceleration(Outint)`: Reads the acceleration of left wheel/motor in mm/sec^2.
- `getRightWheelAcceleration(Outin)`: Reads the acceleration of right wheel/motor in mm/sec^2.
- `setWheelAcceleration(La,Ra,Outb)`: Sets to `La` and `Ra` the left and right motors' acceleration and direction (+/-) in mm/sec^2.
- `readLightSensors(Outb)`: Reads the values of each of the eight light sensors.
- `writeLightSensors(Outb)`: Displays the values of the eight light sensors.
- `getLightSensor(Ls,Outint)`: Gets the value of the requested light sensor `Ls`.
- `readProxSensors(Outb)`: Reads the values of each of the eight proximity sensors.
- `writeProxSensors(Outb)`: Displays the values of the proximity sensors.
- `getProxSensor(Ps,Outint)`: Gets the value of the requested proximity sensor `Ps`.

- `getAllLightSensors(Outlist)`: Return all the light sensors' values in a list.
- `getAllProxSensors(Outlist)`: Return all the proximity sensors' values in a list.

Even though this interface has not been tested under the Windows operating system, we think that its code should be easily ported because *Ciao* and *Webots* versions for Windows exist. Only the serial port definition should be changed from `/dev/ttyS0` to `COM1`.

4. FUTURE DEVELOPMENTS OF THE KROLOG INTERFACE

As further developments in the *KRolog* interface, we plan to extend the interface to be able to control real and simulated robots plugged with gripper-arms and linear and matrix vision systems. In consequence, we will have to add new classes to Harlan et al. C++ interface with one class per extension module, and methods for each `SemCor` command of the k213 linear vision extension turret, k6300 matrix vision extension turret, and the gripper-arm extension turret.

In second place, we are going to extend the low-level layer to allow the communication among the robots. This is a key feature to develop coordination models. However, as we are interested in developing coordination models where point to point and broadcast explicit communication exist, only this kind of facilities will be provided. In this way, this interface would also be useful to those researchers that have the *Khepera'* radio base module, because the robots could communicate among them in a wireless mode.

Finally, as our aim is to use *Defeasible Logic Programming* (DeLP) [9] to build applications that deal with incomplete and contradictory information in dynamic domains, we will try to interconnect an existent DeLP interpreter [10, 11] with our interface.

5. CONCLUSIONS

In this paper we have presented a flexible interface that helps researchers, teachers, and students in the development of Prolog based applications for the *Khepera* robots. The interface hides low-level robot-computer communication and provides a high-order set of predicates to help us to concentrate on the high-level problem specification.

Besides, it has the advantage of communicating with *Webots* in the same way it does with the real *Khepera 2* robots.

6. ACKNOWLEDGMENTS

We thank the National University of San Luis and the ANPCYT for their unstinting support.

7. REFERENCES

[1] K-Team, "Khepera 2." http://www.k-team.com. A miniature mobile robot designed as a research and teaching tool.

[2] O. Michel, "Webots: Professional mobile robot simulation," *Journal of Advanced Robotics Systems*, vol. 1, no. 1, pp. 39–42, 2004.

[3] A. J. García, G. I. Simari, and T. Delladio, "Designing an agent system for controlling a robotic soccer team," in *X Congreso Argentino de Ciencias de la Computación*, 2004.

[4] H. J. Levesque and M. Pagnucco, "Legolog: Inexpensive experiments in cognitive robotics," in *The Second International Cognitive Robotics Workshop*, (Berlin, Germany), pp. 104–109, August 2000.

[5] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López-García, and G. Puebla, "The ciao prolog system. reference manual," Tech. Rep. CLIP3/97.1, School of Computer Science, Technical University of Madrid (UPM), August 1997. Available from http://www.clip.dia.fi.upm.es/.

[6] R. M. Harlan, D. B. Levine, and S. McClarigan, "The khepera robot and the krobot class: a platform for introducing robotics in the undergraduate curriculum," *SIGCSE Bulletin*, vol. 33, no. 1, pp. 105–109, 2001.

[7] B. Vosseteig, J. Baltes, and J. Anderson, "Robocup e-league video server." http://sourceforge.net/projects/robocup-video.

[8] "Oficial e-league webpage." http://agents.cs.columbia.edu/eleague/.

[9] A. J. García and G. R. Simari, "Defeasible logic programming: an argumentative approach," *Theory and Practice of Logic Programming*, vol. 4, no. 2, pp. 95–138, 2004.

[10] A. J. García and G. R. Simari, "Un compilador para la programación en lógica rebatible," in *III Congreso Argentino de Ciencias de la Computación*, 1997.

[11] A. J. García, "La programación en lógica rebatible su definición teórica y computacional," Master's thesis, Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur, Bahía Blanca, Argentina, 1997.