A new programming interface for Educational Robotics

Javier Caccavelli, Sol Pedre, Pablo de Cristóforis, Andrea Katz, and Diego Bendersky

Departamento de Computación,
Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires
Buenos Aires, Argentina
{jcaccav,spedre,pdecris,akatz,dbenders}@dc.uba.ar

Abstract. Educational Robotics uses robots as a tool for teaching a variety of subjects other than specifically robotics in undergraduate curricula. To achieve this goal is vital to have an adequate interface that allows inexperienced students to interact with robots in an easy manner. In this paper we present the current development of ERBPI (*Easy Robot Behaviour Programming Interface*), a new application that doesn't require any previous programming experience to control robots. To accomplish this, we propose to abandon the imperative programming paradigm and take a behaviour-based approach. Thus, the new application is based on the connectionist paradigm, accomplishing behaviours by establishing configurable connections between sensors and actuators. Moreover, different defined behaviours can be connected using a subsumption architecture. The new application is designed to work with different robots and simulators, and it is simple for adding new ones. Learning experiences with high school students allowed us to test its effectiveness.

Key words: educational robotics, behaviour-based programming interface.

1 Introduction

The use of robots in undergraduate curricula has grown over the last years. The availability of low cost, easy-to-use platforms has even led to the use of robots in middle and high schools. Many teachers have an interest in introducing robots into their classrooms for teaching a variety of subjects other than specifically robotics. Thus, Educational Robotics arises, proposing the use of robotics as a teaching resource that allows inexperienced students to approach different scientific fields such as mathematics, experimental sciences, technology, information science and computing, among others. One of the aims of Educational Robotics is to aid students in building their own representations and concepts of science and technology, through the use, handling and control of robotic environments.

This approach uses the constructionist process of designing, building, programming and debugging robot's behaviours, as well as collaboration and teamwork, as powerful means of enlivening education.

Several educational robotics programming interfaces have been presented. Most are designed for university-level or late high school students and are implemented as extensions to existing programming languages. That is the case of Pyro [1], a Python-based programming framework which provides a set of abstractions that allow students to write platform-independent robot programs. Other interfaces include Not-Quite C (NQC) [2] based on C, BrickOS [3] based on C++ and leJOS [4] that is based on Java. These three interfaces are particular for Lego Mindstrom. All these interfaces require programming experience or interest in learning a particular programming language. This makes them unsuitable for middle or high school students that do not handle any imperative or procedural programming concepts, such as the idea of loops, conditions, forks or variables in a program. Microsoft also offers the commercial tool Microsoft Robotics Developer Studio [5]. This includes a visual programming interface based on a data flow approach, but again it requires knowledge of programming concepts, making it quite complex for inexperienced users.

There are also several graphical environments for simulated robots aimed at middle schools. That is the case of StartLogo [6], Squeak Etoys [7] and Scratch [8]. These are easy-to-use programming interfaces, allowing inexperienced students to make a quick start, although they maintain an imperative programming influence and are designed only for particular simulated environments. Another programming interface used in instructional settings at the K-12 level is RoboLab [9] for the LEGO Mindstorms robot. This is a graphical environment in which students are given palettes of icons that they can drag and drop on a canvas. The icons represent robot components like motors and sensors, as well as abstract programming structures such as loops and counter variables. This interface is particular for the Mindstrom robot, and once again uses imperative programming structures that add complexity to the robotic environment. Finally, in [10] authors present an extension to RoboLab in order to work with other robotic platforms.

Our experiences working with classroom teachers and young students have raised several issues that have motivated us to pursue the development of a behaviour-based interface, which abstracts away the imperative programming constructs and the low-level motor and sensor commands that often confuse inexperienced programmers or deter techno-phobic students. To accomplish this, we propose to abandon the imperative programming paradigm and take a behaviour-based approach. Thus, the proposed interface is based on a connectionist paradigm. The idea is that stimuli captured by the robot sensors are processed in a network of connections and result in a response for the robot actuators. To program the robot behaviours we have to establish connections between its sensors and actuators. This connections may include different mathematical functions [11]. Moreover, different defined behaviours can be connected using a subsumption architecture [12], making it possible to achieve more complex behaviours. In this

way, we get a state automaton (or machine), where each state represents a behaviour and each transition a change in the environment. This state machine can be easily translated into an imperative program that the robot executes to perform the behaviour.

This paper describes the current development of ERBPI(*Easy Robot Behaviour Programming Interface*), a behaviour-based, easy to use robotic programming interface for Educational Robotics that allows to program different robotic platforms and simulators. The design of ERBPI follows the next criteria:

- *Ease of use:* The user is not supposed to have any previous programming knowledge. The interface must be intuitive and easy to learn, providing all the tools to program robot behaviours graphically, making it possible to perform drag-and-drop with robot's sensors and actuators, build sensor-actuator connections, and easily configure them.
- *Platform independence:* The application must work with a variety of robots and simulators, and be easily expandable to control new ones. The different bodies, sensors, actuators, low-level commands and protocols to communicate with high-level systems of different robots must be abstracted. Moreover, users must be able to test and run the same behaviours on multiple robots.
- *Portability:* The application must work in different operating systems and platforms to accommodate the different hardware and software available in schools or other educational institutions.
- *Flexibility:* Students from a wide range of backgrounds and teachers with a broad range of goals must be able to use the programming interface effectively, accommodating different levels, curricular needs, academic subjects and physical environments for instruction.

This paper is organized as follows. In the next section we describe ERPBI's architecture and features, and give examples of use. Afterward, we comment some experiences using ERPBI in the classroom with high school students. Finally, we draw some conclusions and discuss directions for future work.

2 The ERBPI: Easy Robot Behaviour Programming Interface

ERBPI uses a behaviour-based approach following the connectionist paradigm. It enables users to create basic behaviours, and then connect them using a subsumption architecture to create more complex ones.

To create basic behaviours, the user establishes connections between sensors and actuators, and configures them with a chain of mathematical functions. Each function can have several inputs, including sensed data and outputs from previous ones. It uses all the inputs to compute a single output, that in turn may be used to set actuators or input following functions. The user chooses functions from predefined families and configure their parameters. This schema defines an

execution graph that allows to infer a computation order and thus translate it to an imperative program for the robot.

All the function outputs, sensors and actuators values are normalized to the same scale $(-100, 100)$. This scale is a "percentage of activation" for each element in the schema. Normalizing all the data permits the previous schema to work properly and also that one behaviour can be used for several robots that may have different sensors or actuators.

The application has a modular design, clearly decoupling the different responsibilities in the system. The GUI (Graphical User Interface) module allows the user to graphically design the robot behaviour, and then exports this behaviour in a file. The CORE module reads this file and executes the defined behaviour. To communicate with the robotic platform (simulator or real robot), the CORE uses a particular RAL (Robot Abstraction Layer) module. There is one RAL for each robotic platform ERBPI manages. This module is in charge of normalizing all the sensor and actuator values, and also of communicating with the actual robot using its particular communication protocol. The basic architecture, including the three modules and their communication, is shown in Fig. 1. Each module is explained in the following sections.

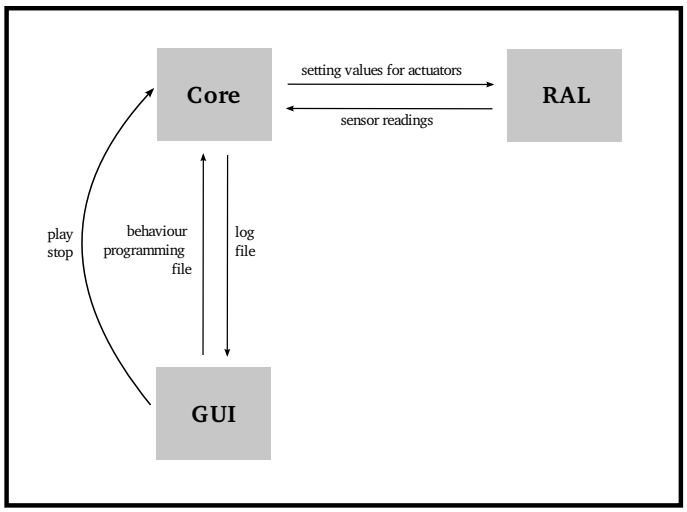

Fig. 1. ERBPI's modular architecture.

2.1 GUI (Graphical User Interface) Module

The GUI module is in charge of interacting with the user. To start, the user selects a robot or simulator to work with. The GUI enables the user to drag and drop elements (sensors, actuators, functions) to a work canvas, and then connect

them using the mouse. Different functions may be selected from a menu, dragged to the canvas, and then configured with a pop-up configuration window. Fig. 2 shows a screenshot of the GUI and Fig. 3 an example of the pop-up configuration window.

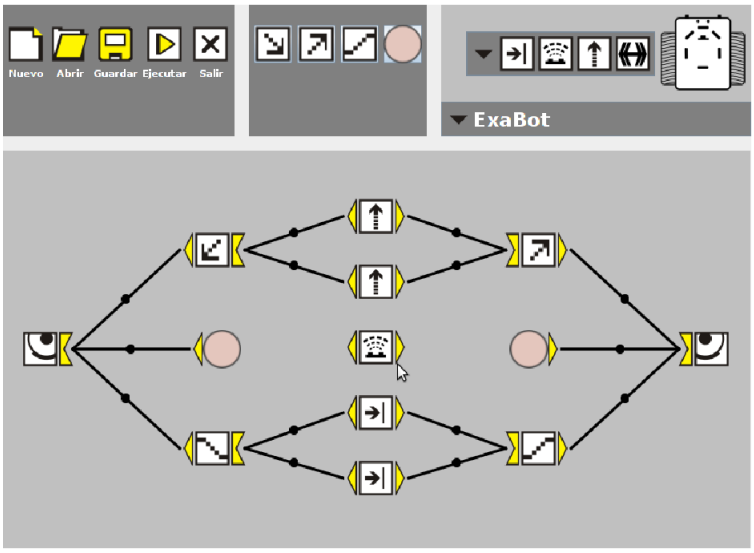

Fig. 2. A screenshot of the application. The upper panel shows the configurations and elements. The general operations (new, open, save, play, quit) are in the upper left corner. The upper-center panel has different families of functions the user can choose from (decreasing, growing, broken and constant functions in this screenshot). The upper right panel shows the selected robot with its sensors. In the canvas we can see a line following behaviour that stops when the bumpers are activated.

Once the behaviour is finished, the user can select a robot to execute it on. The created behaviour and the minimal sensor and actuator configuration needed for its execution are stored in a file (the behaviour-file), that will be read by the CORE. The execution of the behaviour may be started and paused at any moment from the GUI. The GUI also provides general operations to open and save files.

The GUI is implemented in Java, a good language for graphical interfaces and portable to several operating systems, only requiring the installation of the JVM (Java Virtul Machine). The behaviour-file is in XML(Extensible Markup Language), making it very simple to add new robots, sensor types, functions and other features we might add to ERBPI. A simple example of a behaviour-file is shown in table 1.

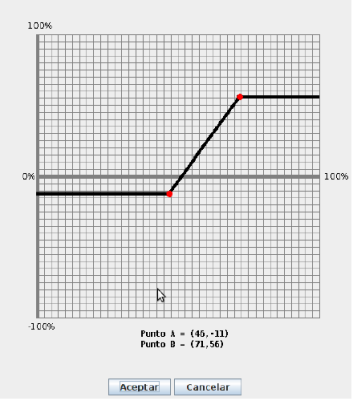

Fig. 3. The configuration pop-up window for a broken function. The change points in the function and the low and high levels can be easily configured moving them with the mouse. The scale for the normalized values $(-100, 100)$ is also observable.

2.2 CORE module

The CORE module is in charge of executing the behaviour. It reads the XML behaviour-file and establishes a connection with the appropriate RAL. At regular intervals, the core receives from the RAL the normalized values of the sensors, executes the behaviour, and gives to the RAL the normalized values to set the actuators. The CORE stops when the GUI signals the user has stopped the execution.

To be able to execute the behaviour, the CORE has to transform the execution graph defined by the GUI in the behaviour-file to a corresponding ordered execution list, to guarantee that all the inputs for a function are ready when its turn to execute is up. For this, we used a topological sorting [13] of the execution graph. The CORE also asserts that the behaviour can be executed in the selected robot, for example that the graph is not cyclic (i.e, cannot be ordered) or that the robot has enough sensors and actuators to execute the behaviour. It also defines the communication frequency with the RAL depending on the robots working frequency. Finally, the CORE makes a log-file where all the values at a certain time are registered, including each sensor value, the output value of each function and the value of each actuator. This log file is communicated to the GUI. We plan to use it to implement a debug function in the future.

2.3 RAL (Robot Abstraction Layer) module

The RAL modules encapsulates all the knowledge of the particular robot or simulator, providing a standard interface to the CORE module, dealing with everything necessary to communicate with the actual robot. The RAL abstracts the particular robot, its communication protocol, and normalizes the values of the particular sensors and actuators. In this way, all the specific characteristics

```
<behaviour>
  <sensors>
    <sensor id='sonar.0'/>
    <sensor id='encoder.motor.left'/>
  </sensors>
  <functions>
    <function id='function1'>
      <inputs>
        <input id='sonar.0'/>
        <input id='encoder.motor.left'/>
      </inputs>
      <points>
        <point x='100' y='0'/>
        <point x='150' y='255'/>
      </points>
    </function>
  </functions>
  <actuators>
    <actuator id='motor.left'>
      <inputs>
        <input id='function1'/>
        <input id='function2'/>
      </inputs>
    </actuator>
  </actuators>
</behaviour>
```

Table 1. Example of an XML behaviour-file. In this case, two sensors are needed to perform this behaviour (a sonar and an encoder). A broken function (`function1`), defined by two points, takes both sensors as inputs. The value for the motor.left actuator is set by the output of two functions

of the robot are transparent to the CORE. A RAL must implement a standard interface that include the following methods: get the list of sensors and actuators in the robot, get the frequency the robot can work in, get the sensor values, and set the values for the actuators. To add a new robotic platform for ERBPI to work with, a developer must only program a particular RAL for the platform implementing the general RAL interface.

All RALs are implemented as dynamic libraries. In this manner, we can add new RALs without having to recompile the CORE or the GUI. Moreover, this allows the CORE to load a different RAL on runtime, without having to restart the application. This makes ERBPI easily extendable to control different robots.

To the date, we have implemented RALs for the Khepera [14] and Exabot [15] robots, and for the YAKS (Yet another Khepera Simulator) [16] and the Player/Stage [17] simulator.

3 ERBPI in the classroom

To explore the adequacy of the tool, a preliminary version of ERBPI was used in a special course designed for high school students during July 2010.

The class was composed of 20 students from seven high schools of Buenos Aires, Argentina. The student's programming experience varied from none to some experience with procedural languages like C, C++ or Pascal. There were also few students with some electronics background, and all mastered basic mathematical concepts such as constant, monotonous and broken functions. None of the students had experience with robots.

The course covered basic concepts of robotics and of behaviour based robotics. The preliminary version of ERBPI used for the course only had the basic conectionnist approach, lacking the subsumption feature to compose more complex behaviours. We followed a hands-on approach, programming easy behaviours from day one in groups of two or three students. Using the Braitenberg model [11], the students developed behaviours in simulators that were also tested in real robots, experiencing with the Yaks simulator [16], Khepera [14] and ExaBot [15] robots. Different behaviours were proposed for the students to solve with the robots, ranging from follow a line in the floor, go to a light spot, avoid obstacles or solve mazes. Those behaviours were developed by teaching students the scientific method, encouraging them to propose hypothesis, contrast the expected results with the ones obtained in the testing phase, and then propose explanations and changes to the original robot control. Each group also shared their findings with the rest of the class, showing different approaches and solutions to the given problems during a discussion phase. Overall, the students picked up the use of ERBPI easily and could program all the proposed behaviours quickly.

After the course, a questionnaire was given to the students in order to explore their reviews of the course and the programming interface. All students answered that they had found the interface easy to learn and use, despite they had no idea about programming robots and had not had the opportunity to control a robot before the course. All students felt that they met the objectives of the course successfully and most of them showed interest in taking more courses of robotics, computer science and engineering after the course.

In this experience we found some improvements to be made in ERBPI. For example, we plan to execute the resulting program directly in the robot when possible, instead of executing the program in the external PC with the consequent transmission of sensor data and commands back and forth. Another improvement is to increase the amount of possible functions, to broaden the tools capabilities for math teachers. We also realized how important the subsumption architecture feature is, to allow the construction of more complex behaviours from simpler ones. Students also suggested some improvements, like "cooler" or friendlier names for different objects of ERBPI.

We also tested ERBPI in shorter courses, lectures, exhibitions and others activities of popularization of science for high school students and broader public. The most important was part of a three day exhibition of the University of Buenos Aires that took place in October 2010 [18]. Many of the ideas in

ERBPI were born in previous experiences with high school students, mainly two eight-week workshop on robotics we organized as a part of a program from the Vocational Orientation Department of our Faculty during 2006 [19] and 2009 [20].

We are planning on using a full featured ERBPI in Educational Robotics courses for more high schools of Buenos Aires during the present year, as a part of a popularization of science project of our Faculty.

4 Conclusions and ongoing work

In this paper we presented the design and current development of ERBPI, an application to provide a simple, graphical, behaviour-based interface for Educational Robotics courses aimed at inexperienced students. To reach this goal, we propose to abandon the imperative paradigm and take a behaviour based approach. The tool is capable of controlling different robotics platforms, and it is designed to be easily expandable for new ones. It is also portable to different operating systems to accommodate the available hardware and software in schools. Experiences with high school students show that the current version of ERBPI allows inexperienced public to quickly start working with robotic environments.

We are currently working on the subsumption feature of the tool, important to allow the development of more complex behaviours. We also plan to expand the application in order to execute the behaviour directly on the robot when possible. Another improvement we are working on is to include a play-back facility to debug behaviours, using the execution log-file and a web-cam that films the robot behaviour. The idea is to use the execution log-file to show at the same time which sensors, connections and behaviours were activated when the robot took a certain action, and the play-back of the video captured with the web-cam.

During the present year we will prepare and teach educational robotics courses for more high schools in Buenos Aires, using the full featured ERBPI tool. We hope this experiences will provide extra feedback to continue and improve this project.

References

1. D.S. Blank, D. Kumar, L. Meeden, and H. Yanco. Pyro: A python-based versatile programming environment for teaching robotics. Journal on Educational Resources in Computing(JERIC), Special issue on robotics in undergraduate education. Part 2, 4(3):115, 2004.
2. Baum. NQC. http://bricxcc.sourceforge.net/nqc/, accessed March 17, 2011.
3. Markus. brickOS. http://brickos.sourceforge.net/, accessed March 17, 2011.
4. J. Solorzano. leJOS. http://lejos.sourceforge.net/, accessed March 17, 2011.
5. Microsoft Robotics Developer Studio, http://www.microsoft.com/robotics/, accessed March 17, 2011.
6. MIT's Scheller Teacher Education Program (STEP), http://education.mit.edu/drupal/starlogo-tng, accessed March 17, 2011.

7. http://www.squeakland.org/, accessed March 17, 2011.
8. http://scratch.mit.edu/, accessed March 17, 2011.
9. Tufts University. RoboLab. http://www.ceeo.tufts.edu/robolabatceeo/, accessed March 17, 2011.
10. M. Q. Azhar, An agent-oriented behavior-based interface framework for educationa robotics, In Agent-Based Systems for Human Learning (ABSHL) Workshop at Autonomous Agents and MultiAgent Systems (AAMAS-2006),2006.
11. V. Braitenberg, Vehicles: Experiments in Synthetic Psychology, MIT Press, Cambridge, 1986.
12. R. C. Arkin, Behavior-Based Robotics, MIT Press, Cambridge, 1998.
13. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Topological Sort, Introduction to Algorithms, MIT Press, Cambridge, 2009.
14. K-Team, Khepera I, http://www.k-team.com/, accessed March 17, 2011.
15. Sol Pedre, Pablo de Cristforis, Javier Caccavelli and Andrs Stoliar, A mobile mini robot architecture for research, education and popularization of science, Journal of Applied Computer Science Methods, Guest Editors: Jacek Zurada and Pablo Estevez, vol 2, no 1 pp 41-59, ISSN 1689-9636.
16. Yet Another Khepera Simulator, http://freshmeat.net/projects/yaks/, accessed March 17, 2011.
17. Player/Stage Simulator, http://playerstage.sourceforge.net/, accessed March 17, 2011.
18. ExpoUBA 2010, Plaza de las Ciencias, Universidad de Buenos Aires, Argentina, http://www.uba.ar/expouba, accessed March 17, 2010.
19. Robot programming workshop for high school students, www.fcen.uba.ar /dov/talleres_de_ciencia/2006/computacion.htm, accessed March 17, 2011.
20. Robot programming workshop for high school students, www.fcen.uba.ar /dov/talleres_de_ciencia/2009/computacion.htm, accessed March 17, 2011.