An Agent-oriented Behavior-based Interface Framework for Educational Robotics

M.Q. Azhar
Dept of Computer Science
Graduate Center
City University of New York
365 Fifth Avenue
New York, NY 10016, USA
mazhar@gc.cuny.edu

Rachel Goldman
Google, Inc.
1440 Broadway
New York, NY 10018 USA
rjg@google.com

Elizabeth Sklar
Dept of Computer and Information Science
Brooklyn College
City University of New York
2900 Bedford Ave
Brooklyn, NY, USA 11210
sklar@sci.brooklyn.cuny.edu

ABSTRACT

This paper describes the development of an agent-oriented behavior-based interface framework for educational robotics. The framework is designed to interact with multiple agent platforms through an XML-based agent behavior language. Our longterm goal is to create a standard middle ground that can act as a sort of "magic black box" for current and future robotic platforms, structured for use in educational settings where agent platforms and operational environments vary greatly. The benefits of such a system include:- *ease of use:* programmers only have to deal with high-level abstractions; *disappearing boundaries:* programmers are able to test and run the same behaviors on multiple platforms; and *interoperability:* a standard behavior language is used for multiple platforms.

1. INTRODUCTION

We use the term *educational robotics* to refer to the use of robotics as a hands-on learning environment [17]. Many educators are using robotics as a teaching tool for university-level computer science, science and engineering courses as well as schoolteachers at primary through pre-college levels for technology and physical science subjects [17, 11, 20, 6, 3, 10, 4, 2]. Instructors have used robotics-inspired projects to teach AI and robotics courses [10, 17]. Klassner and Anderson [11] demonstrate the suitability of LEGO Mindstorms [12] robots to support the ACM computing curriculum through lab exercises and projects from beginning courses in programming to advanced courses in operating systems, compilers and networks. Fagin has used Ada-based robotics [6] successfully for teaching college-level introduction to programming courses.

One of the earliest efforts in this direction was undertaken by Lynn Stein [20] who argues that computer science (CS) students need to be better prepared for writing code that will operate in a dynamic world. She promotes teaching students more multi-threaded, dynamic, re-active programming techniques. The introduction of simple, inexpensive, easy-to-use robotics kits (like the LEGO Mindstorms, which became publicly available in 1999) is the ideal platform for implementing Stein's revolutionary approach to teaching introductory computer science.

More recently, Blank et al. [3] introduced Pyro, a Python-based programming framework which provides a set of abstractions that allow students to write platform-independent robot programs. This versatile programming environment has been successfully integrated into a wide variety of existing computer science courses, from introductory programming to advanced mobile robotics courses. One of the unique features of Pyro is the write once/run anywhere (on any robot!) approach; whereas most robot programming interfaces tend to be specific to particular robotic platforms.

Traditional computer programming environments are not meant for responsive and autonomous objects, such as robots. It is partly due to the fact that the models and metaphors underlying traditional programming languages are not particularly suited to the task. The idea of programming with "agents" help people create worlds involving responsive, interacting agents [21]. In addition, we are not looking to teach programming syntax but rather programming concepts. Our agent-oriented behavior-based framework effectively seperates four educational topics - agent-based concepts, programming basics, mechanical engineering and physical world constraints [5].

There are several important practical factors that any educator needs to take into consideration when integrating robotics materials into their curriculum.

- *Hardware:* What kinds of robotics platforms are suitable and affordable for the particular curriculum? Currently, there are a wide variety of low-cost robotics platforms available.
- *Software:* What kinds of programming environments are available for the particular hardware platform chosen and how student-friendly are they?
- *Lack of "Practice" Environment:* Most schools cannot afford to let the students take the robotics kits home. So, students can only program the robot during the lab time [7]. What limitations does that put on the types of projects that an instructor can assign?

- *Multi-step Debugging Process:* A continuing source of irritation for anyone working in robotics lies in the process of debugging whereby changes in code must be tested by downloading modified code and then running it on the robot. This is a time-consuming, though necessary, procedure; however, students tend to lose patience when they have to test their changes repeatedly this way [5].

- *Real-world Interaction:* The unpredictability that accompanies work on robots operating in the "real world" tends to lead to an even longer debugging process. Again, this can take up valuable lab time and jeopardize students' ability to finish lab assignments and projects on time.

In general, while the use of low-cost robotics platforms in the classroom has many attractive features, such as motivating students and engaging non-traditional learners in technology-based subjects, there are still several shortcomings that must be overcome in order to realize the full potential of educational robotics as a practical learning environment. Particularly since time for "practice" on real robots is limited, there is a need to reduce debugging time when using physical robots in instructional settings.

Most robotics programming interfaces are designed for university-level or late high school students and are implemented as extensions to existing languages. For example, Pyro is based on Python, Not-Quite C (NQC) [1] is based on C, BrickOS [14] is based on C++ and leJOS [18] is based on Java. There are fewer interfaces for students who lack programming experience or interest in learning a programming language. Probably the most well-known programming interface used in instructional settings at the K-12 level for the LEGO Mindstorms robot is RoboLab [22]. 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. Figure 1 contains a sample RoboLab program.

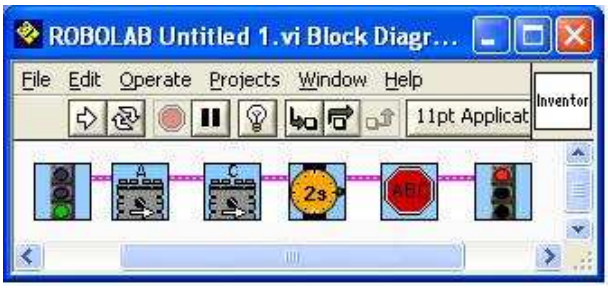

Figure 1: Sample RoboLab program.
This program assumes that the robot has motors attached to the ports labeled A and C. When executed, the program will make the robot go forward for 2 seconds and then stop.

This paper describes the early development of an agent-oriented, behavior-based interface framework designed to address some of the shortcomings associated with the current state of educational robotics. Our framework has the capability to interact with multiple agent platforms through an XML-based agent behavior language. Our longterm goal is to create a standard middle ground that can act as a sort of "magic black box", for current and future robotic platforms, following several design criteria:

- *ease of use:* programmers only have to deal with high-level abstractions;

- *disappearing boundaries:* programmers are able to test and run the same behaviors on multiple agent platforms;

- *interoperability:* a standard behavior language is used for multiple platforms; and

- *flexibility:* students from a wide range of backgrounds and teachers with a broad range of goals can use the system effectively, accommodating different levels, curricular needs, academic subjects and physical environments for instruction.

This paper is organized as follows. The next section provides an overview of our development approach, highlighting key design criteria that have guided our process. Section 3 describes the implementation of the system, giving details for one of three platforms for which we have written drivers thus far. We conclude with a summary and discussion of longterm goals and near future work.

2. OUR APPROACH

In essence, our approach encompasses the needs of both users who want to utilize a pre-defined robot/agent behavior hierarchy and users who want more control to define their own behaviors for the robots/agents, using new combinations of low-level control structures and robot functions. This ability to re-use and/or re-define at multiple levels blurs the boundaries between the different robotics modules and makes the entire system more adaptable to the user's needs. One of our long term aims is to be able to provide programmers with the option to enter the framework at any point in the interface continuum and to move either towards integrating pre-defined behaviors or towards re-defining existing behaviors.

Our system provides control through a single integrated interface. If, for a moment, we move away from the bottom-up approach and focus on the top-down perspective, we can begin to see what will be necessary for our behavior interface. We have created a minimal set of agent *behaviors* that can be programmed by selecting them from a specially designed "behavior palette" (see Figure 2). Whereas standard RoboLab groups icon palettes by functionality, our interface groups icon palettes by behavior classes. Each class of behaviors can be expanded into sub-palettes that display the associated lower-level behaviors.

Within the scope of the big picture, we not only look at being able to run the same programs on multiple platforms, but also hope to generate platform-specific code that can be used as a learning tool and guide. Programmers would have the ability to take generated code, modify and add to it directly before it is run on the target platform. With the big picture, we aim to create a consistent set of capabilities across various software environments and platforms. Although there are differences between the platforms that

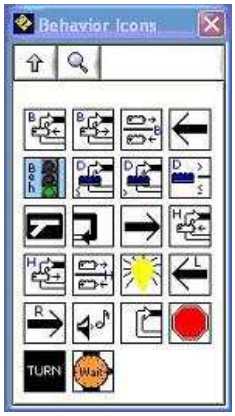

Figure 2: Our Behavior Palette for RoboLab

need to be addressed (primarily related to sensors and actuators), we provide a standardized method for specifying and generating behavior-oriented agents.

Because our particular application domain is a classroom environment, we have identified a small subset of tasks that are commonly explored in the classroom [17]. For example, students are typically asked to develop the following high-level behaviors when studying mobile robots:

- *line following*
- *wall following*
- *object pushing*
- *obstacle avoidance*

Each high-level behavior can be broken down into multiple middle-level actions which, in turn, may be broken down further into multiple low-level, platform-specific functions, thus forming a functional hierarchy. Our initial development step is to identify and implement the platform-specific low-level functions that can be executed on a variety of platforms, both in the virtual and physical worlds, with certain basic commonalities. For development purposes, we have chosen three target platforms and written drivers for each of these:

1. the LEGO Mindstorm Robotics Invention Kit [12],
2. the Sony AIBO legged robot [19], and
3. a virtual robotic agent inhabiting a simulated world, implemented in Flash [13].

Our development strategy has been to combine a bottom-up approach with a top-down approach. Using a top-down approach, we identified (as above) a standard short list of behaviors; we refer to these as the *high-level behaviors*. Then, we shift to a bottom-up approach and build the *low-level functions* necessary to execute these behaviors on each of our sample platforms. The final development step, which is the primary contribution of the work presented here, has been to connect the low-level functions to the high-level behaviors in such a way as to avoid any platform-specific features or semantics, to allow for future expansion of our repertoire at all levels, as we look toward adding new behaviors and robot/agent platforms. For each of the four high-level behaviors listed above, we deconstruct the behavior into its subsidiary parts. We select a set of common low-level functions that are applicable to all the high-level behaviors we have defined.

The following is an example of our decompostion of the abstract behavior for *line following*. We assume that the robot operates in an environment in which there is a colored line, i.e., a line that is a sufficiently different color from the background of the field on which it lies. We also assume that the robot possesses some type of visual sensor that can detect the line (i.e., a camera or a light sensor) Given these constraints, the pseudo code for *line following* is:

```
follow_line {
 loop( forever ) {
   locate_line()
   re_orient()
   deactivate( motors )
   deactivate( sensors )
  }
}
```

This pseudo code is comprised of one low-level function (`deactivate()`), one low-level programming construct (`loop`), and two high-level behaviors (`locate_line()` and `re_orient()`). These latter two are further decomposed as:

```
locate_line {
  activate( motors )
  activate( sensors )
  values = read( sensors )
  process( values )
}

re_orient {
  position( forward, backward, turn )
}
```

These behaviors consist only of low-level functions: `activate()`, `read()`, `process()` and `position()`.

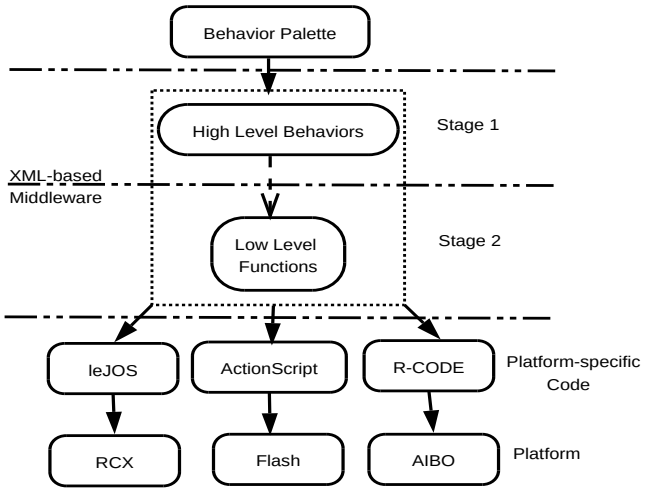

Figure 3: System architecture

3. IMPLEMENTATION

Figure 3 illustrates the architecture of our framework, which revolves around middleware executed in two stages.

stage 1 translates the RoboLab output, which is in the form of LEGO Assembly Language (LASM) commands, into behavior-based XML using a module called `lasm2xml`. *stage 2* consists of a platform-specific driver that converts our behavior-based XML to the control code required by the agent platform being used. Currently, we have developed three individual second-stage drivers, one for each of the three target platforms described in section 2.

The steps involved in using the system are as follows. The user creates the program in RoboLab with our behavior palette (see Figure 2), which is saved as a LASM file. We will use the following RoboLab behavior-based program (see Figure 4) through out this section as an example:

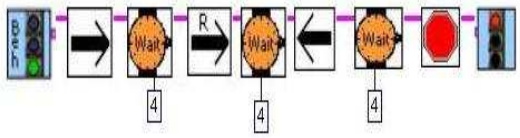

Figure 4: Behavior-based RoboLab Program

creates the following LASM (see Figure 5):

```
delt 0
task 0
pwr 1,2,7
dir 2,1
out 2,1
pwr 4,2,7
dir 2,4
out 2,4
wait 2,400
out 1,7
pwr 1,2,7
dir 2,1
out 2,1
pwr 4,2,7
dir 0,4
out 2,4
wait 2,400
out 1,7
pwr 1,2,7
dir 0,1
out 2,1
pwr 4,2,7
dir 0,4
out 2,4
wait 2,400
out 1,7
endt
```

Figure 5: LASM Program

In *stage 1*, the LASM file is used as input to the `lasm2xml` module, which generates the robot's behaviors in our XML format. The `lasm2xml` module takes the LASM input and creates XML equivalents (see Table 1) for the commands that originated with the behavior palette in RoboLab.

Table 1: XML equivalence for behavior palette

RoboLab Icons	XML Equivalence	Description
	`<forward>value</forward>`	Move forward for a specified distance (mm)
	`<backward>value</backward>`	Move backward for a specified distance (mm)
	`<turnright>value </turnright>`	Turn right for a specified degree
	`<wait>value</wait>`	Wait for a specified time (sec)

We use the UNIX utilities *lex* and *yacc* [15] to implement `lasm2xml`. In order to build an interpreter with lex and yacc, there are two source files that need to be created: the lex specification (a `*.l` file) and the yacc specification (a `*.y` file). Figure 6 depicts how the two source files produce a working compiler. Our `lasm2xml.l` holds the grammar for translation and `lasm2xml.y` holds corresponding functions for the grammar of our XML-based behavior language.

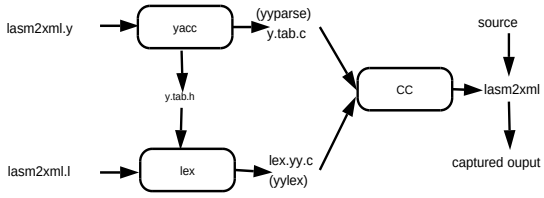

Figure 6: lasm2xml driver using lex and yacc

Our `lasm2xml` produces the following XML (see Figure 7) for the behavior-based RoboLab program (see Figure 4) where the target platform is Flash:

In *stage 2*, the user selects a destination platform and executes the corresponding module to translate from the XML-based middleware to the low-level, platform-specific functions that can run directly on the platform of choice:

- `xml2lejos` → LEGO Mindstorms platform
- `xml2aibo` → Sony AIBO
- `xml2flash` → Flash simulator

```
<?xml version="1.0"?>
<behavior source=lasm destination=flash>
<command>
<forward>4</forward>
<turnright>4</turnright>
<backward>4</backward>
</command>
</behavior>
```

Figure 7: Behavior-based XML

In earlier work [5], we developed an agent-oriented simulator in Flash [13] for the LEGO Mindstorms robot, and built a prototype system for programming the agent in RoboLab and feeding the LASM output directly into Flash. The current implementation takes this notion one step further, by abstracting the connection between the programming interface (RoboLab) and the destination platform (Flash agent), using XML, as output by `lasm2xml`. The first task is to convert the XML input into the corresponding Flash "action script" (the programming langauge used inside Flash) commands. In this manner, low-level operations are masked and grouped into abstracted behaviors. For instance, there is no direct reference to the LEGO motors in the simulator description. It should be noted that the `xml2flash` module only handles simple sequential behaviors. Basic control structures are under implementation. Because the `xml2flash` simulation module only describes behaviors, the simulator world is dynamic and can be created to model any scenario. Lights, colors, and physical obstacles can be added and removed. This structure follows the CML architecture [8] where the agent behavior is separated from their virtual world.

We have used the same lex-yacc environment for creating code to run on the other two target platforms: the Sony AIBO (i.e., using a module called `xml2aibo`) and the LEGO Mindstorms via the leJOS Java-based programming environment (i.e., using a module called `xml2lejos`). The `xml2aibo` module creates an "R-CODE" [16] file, which will be executed in the R-CODE environment on the AIBO robot. An example is shown in Figure 8.

The `xml2lejos` module creates a "leJOS" [18] specific Java file, which will be executed in the leJOS (LEGO Java Operating System) on a LEGO robot.

4. CONCLUSION

We have described the development of our agent-oriented behavior-based interface framework for educational robotics. Our current system is based not only on previous research, but also on the foundation of relevant pedagogical and technical theory [9]. However, there are several questions that still need to be addressed.

The main outstanding questions are:

1. where should the borderlines for the behavior decomposition be drawn?
2. should the different behavior levels overlap?
3. how should the different behavior levels interact?
4. how can we create coherent sets of XML tags for each behavior that will work on every robot platform?

```
:Start
PLAY:ACTION:STAND
WAIT
PUSH:4
CALL:Forward:1
WAIT
PUSH:4
CALL:Turn_Right:1
WAIT
PUSH:4
CALL:Backward:1
WAIT
*********************************************
*                 BEHAVIOR FUNCTIONS          *
*********************************************

/*********************************************
 * walk forward for a specified distance in mm
 *********************************************/
 :Forward //pass a distance
  ARG:distance
  PLAY:ACTION:WALK:0:distance
  RETURN

/*********************************************
 * walk backward for a specified distance in mm
 *********************************************/
 :Backward //pass a distance
  ARG:distance
  PLAY:ACTION:WALK:180:distance
  RETURN

/*********************************************
 * turn a specified number of degrees 2 the right
 *********************************************/
 :Turn_Right //pass a degree
  ARG:degrees
  PLAY:ACTION:TURN:-degrees
  RETURN
```

Figure 8: R-CODE example

For instance, R-CODE which is specific to AIBO does not have a `time` parameter for the *forward* command; instead it takes a `distance` parameter. On the contrary, most platforms (i.e., leJOS, brickOS, RoboLab) specific to LEGO Mindstorms platform take a `time` parameter for the *forward* command. Thus, the parameter differs across various target robot platforms for the same low-level functions.

The answers to these questions depend on the overall focus: simplicity versus control. For simplicity (and currently for the purposes of testing), all the behaviors including the higher-level behaviors should be implemented on the target platform. This will make the middleware specification simpler but will remove control from the user. The user will be locked into the high-level behavior implementations that have been pre-defined. For control, the user should have the ability to define mid-level actions and high-level behaviors based on the system hierarchy. By allowing the user to have more control, the degree of specification will increase. In this case, the middleware will have to provide a means for decision-making, looping and structuring be-

haviors. The immediate conclusion after implementing the low-level behaviors is that our system will not be a set of discrete components. The boundaries between the different modules are without precise definition. This will allow for more flexibility in the long term.

We hope to address these issues and more in the future. We believe that the successful implementation of XML-based middleware demonstrates the feasibility and viability of our proposed architecture.

5. REFERENCES

[1] D. Baum. NQC. `http://bricxcc.sourceforge.net/nqc/`, accessed January 16, 2006.

[2] R. D. Beer, H. J. Chiel, and R. F. Drushel. Using autonomous robotics to teach science and engineering. *Communications of the ACM*, 42(6), June 1999.

[3] 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):1–15, 2004.

[4] M. Carbonaro, M. Rex, and J. Chambers. Using LEGO Robotics in a Project-Based Environment. *The Interactive Multimedia Electronic Journal of Computer-Enhanced Learning (IMEJ)*, 6(1), June 2004.

[5] K.-H. Chu, R. Goldman, and E. Sklar. Roboxap: an agent-based educational robotics simulator. In *Agent-based Systems for Human Learning Workshop at AAMAS-2005*, 2005.

[6] B. Fagin. Using ada-based robotics to teach computer science. In *ITiCSE '00: Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and Technology in Computer Science Education*, 2000.

[7] B. Fagin. Ada/Mindstorms 3.0: A computational environment for introductory robotics and programming. *IEEE Robotics and Automation Magazine*, 10(2):19–24, June 2003.

[8] J. Funge, X. Tu, and D. Terzopoulos. Knowledge, reasoning and planning for intelligent characters. In *In Siggraph 1999, Computer Graphics Proceedings, Alyn Rockwood (Editor)*, pages 29–38. Addison Wesley Longman, Los Angeles, 1999, 1999.

[9] R. Goldman. From robolab to aibo: Capturing agent behavior. Master's thesis, Columbia University Department of Computer Science, NY, 2005.

[10] F. Klassner. A case study of lego mindstorms suitability for artificial intelligence and robotics courses at the college level. In *Proceeding of the 33rd SIGCSE Technical Symposium on Computer Science Education*, 2002.

[11] F. Klassner and S. Anderson. LEGO MindStorms: Not just for K-12 anymore. *IEEE Robotics and Automation*, 10(2):12–18, June 2003.

[12] LEGO. Mindstorms robotics invention kit. `http://www.legomindstorms.com/`.

[13] Macromedia. Flash. `http://www.macromedia.com/software/flash/`, accessed January 16, 2006.

[14] N. Markus. brickOS. `http://brickos.sourceforge.net/`, accessed January 16, 2006.

[15] T. Niemann. *A Compact Guide to Lex and Yacc*. epaperpress.com, 2006.

[16] R-CODE. SDK. `http://openr.aibo.com/openr/eng/no_perm/faq_rcode.php4`, accessed January 16, 2006.

[17] E. Sklar, S. Parsons, and P. Stone. Using RoboCup in university-level computer science education. *Journal on Educational Resources in Computing (JERIC), Special Issue on robotics in undergraduate education, part I*, 4(2), September 2004.

[18] J. Solorzano. leJOS. `http://lejos.sourceforge.net`, accessed January 16, 2006.

[19] Sony. AIBO. `http://www.us.aibo.com/`, accessed January 16, 2006.

[20] L. A. Stein. Rethinking cs101: Or, how robots revolutionize introductory computer programming. *Computer Science Education*, 1996.

[21] M. D. Travers. *Programming with Agents: New metaphors for thinking about computation*. PhD thesis, MIT, 1996. Supervisor- Marvin Minsky and Mitchel Resnick.

[22] Tufts University. RoboLab. `http://www.ceeo.tufts.edu/robolabatceeo/`, accessed January 16, 2006.