RoboXAP: an agent-based educational robotics simulator

Kar-Hai Chu[1], Rachel Goldman[1] and Elizabeth Sklar[2]
[1] Department of Computer Science
Columbia University
New York, NY 10027 USA
kmc2103,rg2020@columbia.edu
[2] Department of Computer and Information Science
Brooklyn College, City University of New York
Brooklyn, NY 11210 USA
sklar@sci.brooklyn.cuny.edu

Abstract

We describe an agent-based educational robotics simulator for children that we have built based on the popular RoboLab graphical programming environment and the LEGO Mindstorms robot. Motivated by practical as well as pedagogical issues, we developed the RoboXAP environment to try and overcome some of the key difficulties encountered when implementing educational robotics curriculum for kids.

1. Introduction

We have developed an agent-based simulation environment for children, designed to be used in conjunction with the popular RoboLab [25] graphical programming interface and LEGO Mindstorms [21] robotic platform. The idea is to give students an opportunity to learn about agent-based programming by using RoboLab in a "safe and friendly" place – they can "try out" programs in our simulator before loading them onto the robot platform and before being faced with real-world, physical constraints and issues such as noise. Once students have debugged their programs in the simulator, they can then download them onto the LEGO robot and see, e.g., how noise within the physical world may affect their robot and control code. This effectively separates four educational topics — agent-based concepts, programming basics, mechanical engineering and physical world constraints.

From a practical standpoint, students can work in readily-accessible computer labs and make progress with programming tasks on days when they do not have access to the physical robots, either because of lack of time or shortage of equipment. From a pedagogical standpoint, students have a simulation environment in which they can check if a source of error is in their program (rather than in the mechanics of their robot or from noise in the environment).

This paper presents the design of our RoboXAP (pronounced "robo-zap") agent-based simulation environment. We begin with a brief overview of the system architecture and background. This is followed by technical details of how the system works. Then we highlight an informal study we conducted with a small number of children at an after-school program in order to get feedback on the design of the simulator. The closing section outlines plans for a pilot study to be conducted in Spring 2005.

We acknowledge that many, many simulation and robotic environments exist for kids, starting with Papert's Turtle LOGO [23]. From MicroWorlds [20] to NetLogo [22] to googol-Choo-Choo [13] to AddictingGames.com [1], there is a vast variety in application technologies, goals and content. Our work capitalizes on the popularity of RoboLab and aims to solve particular implementaion problems, as described above. As such, we do not in this paper provide any comparison with other simulation or robotic environments for kids; there simply is not another simulator built to be used in conjunction with RoboLab that addresses the particular issues we have highlighted and emphasizes an agent behavior-based model for constructing robot control code. Here we focus on the technical description and background that contributed to our design decisions. Future work will constrast RoboXAP with other kids' environments and will also strive to compare on-line versus hands-on learning experiences.

2. System Overview

The RoboLab [6, 25] programming environment, developed as a joint project between the Tufts School of Education, the Tufts School of Engineering and the National Instruments Corporation, combines the National Instruments' LabVIEW graphical programming approach with RCX code (developed by LEGO) to create a flexible and understandable way to write control software for the LEGO Mindstorms robots. A sample is shown in figure 1. Hardware entities such as motors and sensors are represented by rectangular icons on the screen, and users drag and drop them, using a mouse, to create "code". The icons must be strung together using "wires", and all programs must be *downloaded* from RoboLab onto the LEGO brick in order to be executed. The environment is highly visual and gives a good first experience with procedural programming concepts.

The purpose of RoboLab is to provide an educational tool for both children and adults. The graphical programming environment scaffolds its levels of programming, allowing users to produce results and acquire skills early on without having to read large and complicated manuals. The ability for students of all ages to employ the same tool-set enables all users to focus more on exploring and learning about programming and robotics, and less on learning how to use the software development environment.

Figure 1. A RoboLab program that tells the robot to go forward until the light sensor reads a value less than 40; then stop. "Modifiers" hang down from the main control loop, which moves horizontally across the diagram, and indicate speeds, thresholds and port connections.

Our goal with RoboXAP is to provide an intermediate simulation layer, between the RoboLab environment and the LEGO robot, for the practical and pedagogical reasons outlined earlier. A further, longterm goal is to develop an interface by which RoboLab could be used to program multiple types of robot platforms, in addition to the LEGO Mindstorms and our RoboXAP simulator. For example, we have been using Sony AIBO robots in our lab (see figure 3) for conducting research on robot coordination tasks using an agent, behavior-based control architecture [11]. The

(a) The RCX microcontroller; (b) A light sensor.

Figure 2. Components of the LEGO Mindstorms Robotics Kit. The RCX has 3 output ports and 3 input ports. Motors plug into the output ports. Sensors, such as the light sensor shown here, plug into input ports.

AIBO's are programmed in C++ using an open development environment called *OPEN_R*. This work has informed the development of RoboXAP as detailed in the following sections.

Figure 3. Sony AIBO robot

The overall system architecture is illustrated in figure 4. The basic components are (1) a graphical programming environment (RoboLab), (2) a middleware layer that translates between the programming environment and code that can be downloaded into a simulator or onto robot hardware, and the (3a) simulation environment or (3b) the robot itself. The solid lines and arrows in the diagram indicate the flow of code, from RoboLab through the translation step and into the simulator. The dashed lines and arrows indicate a user's development cycle — students write code in the programming environment, which gets sent either to the simulator or the robot for testing; and then they can return to the programming environment to fix bugs and expand their programs.

When dealing with varying robotic platforms, even with a relatively simple one like the LEGO Mindstorms, it is effective for students to be able to classify and specify behavior patterns for their robots. Behavior patterns can range from basic behaviors, such as "move forward for 4 sec-

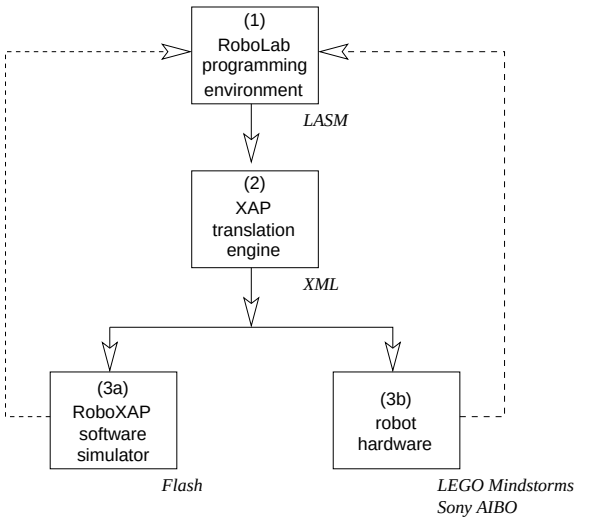


Figure 4. Overview of system architecture

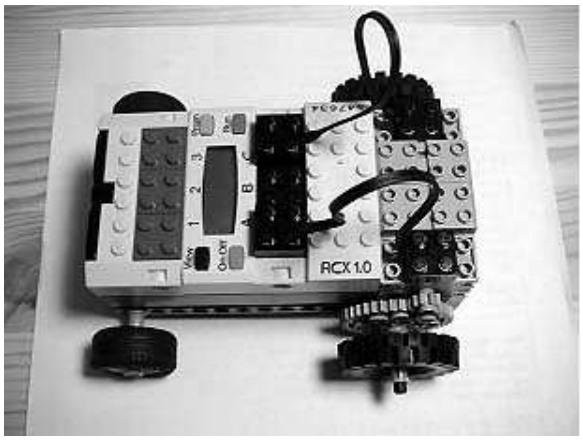


Figure 5. LEGO Mindstorms robot configured for RoboXAP

onds", to complex behaviors such as "open a gate" or "move food to where the pets are kept"[1]. Due to the complexity of defining and implementing robot behaviors, many control architectures are designed to build complex behaviors out of simple, low-level commands [18]. We take advantage of this trend for constructing our middleware layer, which uses the popular Extensible Markup Language XML. The idea is to define low-level behaviors, such as "move forward for 4 seconds", that are constructed out of multiple RoboLab icons but translate into a parameterized XML "tag" so that specification of behaviors is platform independent. For example, for a LEGO robot to move forward, it must turn on two motors that make its wheels spin; but for an AIBO robot to move forward, it must coordinate multiple motors that control the joints in its four legs.

It is important to note here that RoboXAP currently assumes the use of a standard hardware configuration for the LEGO robot, namely a wheeled design that has motors controlling each of two wheels as shown in figure 5. When RoboXAP is deployed in classrooms, students will be given a set of building constraints corresponding to the limitations of the simulator. As the simulator becomes more sophisticated, we will be able to support a more diverse set of robot body constructions.

The remainder of the paper is focused primarily on the technical background and details of the system.

1 This is one of the challenges for the FIRST Lego League 2005 competition (http://www.firstlegoleague.org/).

3. Technical Background

Since the initial release of the Extensible Markup Language XML [3] in 1998, a myriad of XML-based behavior languages have been developed capitalizing on many of the key features [26] of XML. Specifically, XML is popular for structuring and representing complex data; however, one does not need to be an experienced programmer to use or learn it. Like HTML, XML uses tags and attributes and is both easy to read and write. Tags in XML are used to delimit information and are interpreted in context by different applications. By defining a set of rules, one can ensure that the generated or read data is unambiguously structured. Additionally, XML is well supported and many free tools exist to both validate and display XML data. The main attraction of XML is that it is extensible, modular, platform-independent and free. Here we examine a few XML-based behavior specification languages which have been developed for robotic architectures.

The Extensible Robot Control Language (XRCL) [2] is an XML-based language intended for programming and simulating mobile robots. XRCL includes both a proposed language specification and an environment for interpreting that language. XML was chosen as the basis for the XRCL for three reasons. First, it provides a standard for representing the structure of data. Second, a large variety of editing, processing and validating tools exist. Third, XML has become popular with both programmers and non-programmers and is used in varying contexts. Based on the model of embedding JavaScript within HTML documents, the XRCL implementation intersperses C++ code and XML tags. Although C++ is used to code low-level behaviors, Java or Lisp could easily be used as well. The system was designed to be easy enough for a novice, but pow-

erful enough for advanced researchers.

The Extensible Agent Behavior Specification Language (XABSL) [16] presents a flexible, hierarchical behavior control architecture, which allows for scalable agent behavior solutions. XABSL was developed in order to simplify the behavior specification process for autonomous agents with the intention of supporting both reactive and deliberative behaviors. It was designed for use with AIBO robots operating in a robotic soccer environment (RoboCup [24, 14]). Building on the XML model, XABSL is both modular and extensible. Behavior patterns can be used in different contexts, old behaviors can be extended and new behaviors can be added without adversely affecting others. The XABSL framework takes advantage of existing XML technologies and contains numerous visualization and debugging tools. Although XABSL is designed to run on any robotics platform, it has currently been implemented and tested on the Sony AIBO platform and in a simulation robot soccer environment.

XABSL is better suited for specifying and mapping behaviors than native programming languages as the number and complexity of behaviors increases. Designing and implementing complex behaviors in native languages, like C++, is an involved task and can result in an unclear and convoluted structure. Due to intricacies of implementation, extending and maintaining large complex behavior control systems can become difficult and error prone. Higher-level specifications, like XABSL, separate the behavior design from the implementation of the target platform thereby simplifying the entire process. For this reason XABSL's high-level abstraction of behaviors, using an XML-based representation, makes it well suited for mapping RoboLab icons (or groups of icons) to particular options and states. Additionally the modularity of XABSL allows the behaviors to be reused in different contexts removing the need to recode an entire behavior when a slight modification is needed or when new RoboLab icons need to be mapped to a specific behavior. Based on the extensibility and scalability of XML-based behavior specification languages, extensions of RoboLab can be easily accommodated.

Using XML technologies is more attractive than defining a new grammar for numerous reasons. Many of these reasons are why the XRCL and XABSL projects were motivated to choose XML as their backbone. XML support tools have the ability to validate an XABSL document before runtime. Furthermore, XML provides flexible data representation, and easy transformation from and to other programming languages. Last, complex behaviors, specified using XABSL, are easily visualized using packages like XSLT [27] or DotML [5].

Another approach to designing control architectures for agent-based systems uses *cognitive modeling*. Unlike XML-based behavior languages, which are primarily designed to simplify the specification and structure of behaviors thereby achieving abstraction, many non-XML based behavior languages like the Cognitive Modeling Language (CML) achieve behavior abstraction through logic based reasoning systems. CML was designed to aid in behavior specification in simulation environments including computer games and animation. Cognitive models go one step further than behavior models. Cognitive models govern what an agent knows, how that knowledge is acquired and how it is used to plan. The Cognitive Modeling Language (CML) [12] was designed to help build these cognitive models.

Cognitive modeling can be broken into two parts: domain knowledge specification and agent behavior specification. This helps create modularity by separating knowledge from control. CML forms a middle ground between logic programming languages like Prolog and traditional imperative programming languages like C. The main features of CML are the intuitive nature with which domain knowledge can be specified and the presence of control structures that can be used to focus the reasoning engine. This is extremely important for animation and computer games where fast development is more important than fast execution. The theoretical background for CML is highly based in traditional AI planning.

Another feature of CML is that from the user's perspective, the underlying theory is hidden. Users are not required to write any first-order logical axioms; rather, they can use descriptive keywords that have a clear-cut mapping to the underlying formalism. Figure 6 depicts the relationship between the user, the cognitive model and a reactive system. The goal of CML is to equip characters with a model that enables them to interpret high-level instruction from the animator.

Our XAP solution combines the language specification advantages of working with XML and the cognitive modeling advantages of CML, wherein agent behaviors are separated from the environment where the robots act. An XML-based middleware solution is best suited for the task of bridging RoboLab and a target platform/environment because of its modular, scalable and extensible properties. XML-based behavior languages simplify the specification of robot behavior and have the ability to handle real-time sensor data. The design of XAP is informed by XRCL and XABSL, but contains its own set of primitives, with the intension of encapsulating the types of simple behaviors that kids tend to code in RoboLab.

4. System Details

RoboXAP has three main components:

- the translater that takes the output of RoboLab (in the form of LASM commands – LEGO assembly lan-

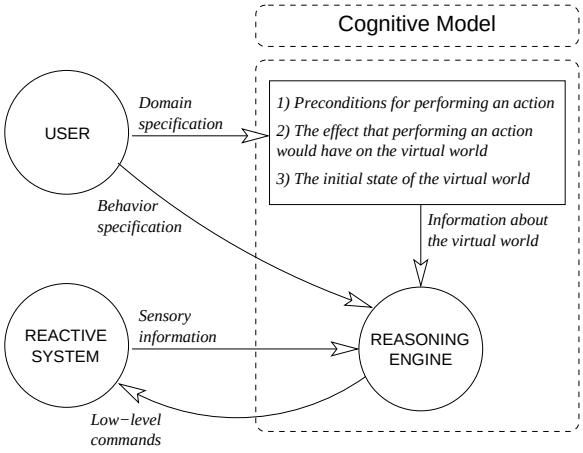


Figure 6. Interaction between the user, cognitive model and reactive system (from [12])

guage) and converts them into behaviors specified using our XML-based language;

- our XML-based Agent Protocol language (XAP); and
- the RoboXAP simulation environment, implemented in Macromedia's Flash [8].

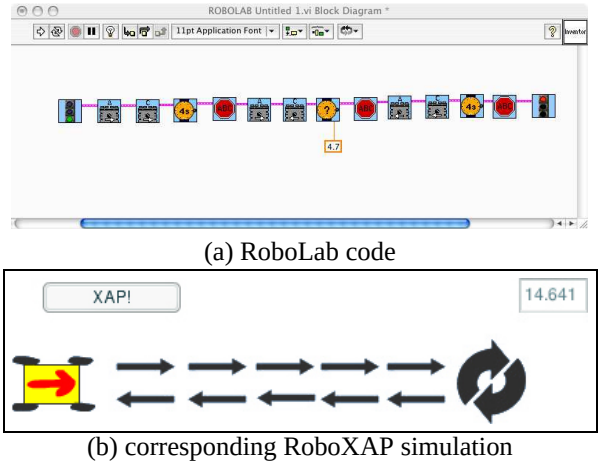
Even though the simulator is the only component that is visible to the user and contains any visual output, all three parts reside together and are implemented in Flash. The main purpose for this is to avoid requiring the user to install any additional tools in order to run RoboXAP. Macromedia Flash Player can be downloaded and run on many different web browsers and over a dozen operating systems [10].

By keeping all of RoboXAP's components within the Flash environment, we can maintain compatibility with almost any system that has a web browser capable of running the Flash plug-in. The three-tier design serves to keep the functionality of the front-end simulator and the back-end translator independent of each other. This not only allows updates to be made easily, but also for other components to be substituted in without too many changes in the software. We note that when interfacing with other front-end destinations, such as the AIBO, the translation engine will be required to convert XAP behaviors into C++.

RoboXAP is built entirely with Macromedia's ActionScript 2.0 [9]. ActionScript is Macromedia's proprietary language used solely within Flash. It is object-oriented, allows users to create more dynamic content, and gives serious programmers strong tools to refine their Flash presentations [9]. In designing the simulator, an environment with both strong multimedia capabilities and flexible programming was needed. It is because of ActionScript's powerful features, combined with Flash's already established multimedia functionality, that makes it the ideal medium for development.

The RoboXAP process cycle begins with a user creating a program in RoboLab. The system is designed so that at the press of a button, RoboXAP will be launched from within the RoboLab environment[2]. It will generate LEGO assembly code (LASM) from the user's block-diagram program, convert it to XAP, send the XAP code into the robot world simulator and display visually what the robot is doing, i.e., how the robot operates in the simulated world using the control code that was written in RoboLab.

Figure 7 illustrates the RoboLab code and resulting movement in a RoboXAP world. The string of RoboLab commands tells the robot to go forward for 4 seconds, turn for some number of seconds (N) and then go forward again for another 4 seconds. This example is a challenge we gave to a group of 3rd and 4th grade students. They were told to figure out what value N would need to be in order to get the robot to return to its starting place at the end of the program. The RoboXAP world contains a clock in the upper right portion of the window, so that students can make a (calculated) guess for what N should be, then run the simulation and actually see what N should be by watching the clock. In the figure, N = 4.7 seconds.



(a) RoboLab code

(b) corresponding RoboXAP simulation (arrows were added for clarity, to indicate the motion of the animation)

Figure 7. RoboXAP example

RoboLab generates assembly code based on the block-diagram program that the user has constructed. RoboXAP's

2 The automatic launching of RoboXAP from within RoboLab is currently under development and at the moment, RoboXAP must be started up manually, outside of RoboLab.

first process is to convert the LASM into an XAP representation. Our purpose is not to have a one-to-one translation of every LASM line that RoboLab produces, but rather to convert groups of lines to behaviors and generate an XML description of each behavior. Table 1 contains sample LASM code. The first three lines instruct motor one (A) to turn on at full power in the forward direction[3]. Similarly, the last three lines instruct motor three (C) to do the same thing. However, we do not want to refer to motors in XAP; that does not tell us what behavior the robot is following and it is too specific for the Mindstorms hardware. Instead, we want something more generally descriptive about the robot's behavior. The corresponding XAP code in table 2 only contains two commands: `forward`, which encompasses most of the 6 lines of LASM code, except for the power level setting; and `power`, which handles the power level setting, indicating the robot's speed (relative to its maximum capabilities). In this case since the Mindstorms power level ranges from 0 to 7, the power level of 7 represents 100% of its capacity; hence the translation in XAP is 100.

```
pwr 1,2,7
dir 2,1
out 2,1
pwr 4,2,7
dir 2,4
out 2,4
```

Table 1. LASM code example

```
<instruction>
<command>forward</command>
<power>100</power>
</instruction>
```

Table 2. XAP code example

Difficulties arise when different commands are mixed to produce an effect that is not quickly decipherable. For example, to make a robot go in an arc, both motors are turned on at different speeds. It is not easy to capture this behavior by looking at the code itself, as it can be compounded by other commands. Control structures are also problematic, as the LASM frequently uses JUMP commands. Creating a multi-pass translator would require considerable processing power, something we wish to avoid as Flash player is a client-side application. We do not want to force users to have an extremely powerful machine in order to run RoboXAP.

Although the translator uses only a single pass, it looks ahead several lines in order to determine the purpose of each "chunk" of LASM code. The assembly is very basic, and only by clumping several lines together can we establish some type of understandable functionality. The need to frequently look ahead several lines is a process-intensive procedure. However, the tradeoff is that by spending time to produce more descriptive XML, the simulator can later animate the robot with less lag. This also gives RoboXAP more portability. It serves to allow the XML to be easily inserted into other simulators or external devices, or even to be converted into a higher level XML that can possibly describe sophisticated behaviors, such as those outlined in section 3.

Flash ActionScript is the driving force behind RoboXAP's front-end simulator. With many animation tools natively built-in, it is the ideal environment for creating a dynamic simulator. ActionScript's XML parsing tool is too simple to perform complex operations, but it can still traverse an XML tree and manipulate child and parent nodes. After receiving XAP from the translation engine, each instruction, along with its arguments, is performed in a step-wise manner. Most of the robot's functionality can be reproduced in the simulator: all different movements (e.g., forward and backward), timers, and external sensors can be programmed to perform as they would in the real world. Because XAP only describes the robot behavior, the world in the simulator is dynamic, independent of the user's program, and can be created to suit any scenario. Lights, colors, and physical obstacles are all components in the simulator that can be added and removed. This structure follows the CML architecture in which the agent behavior is kept separate from their (virtual) world.

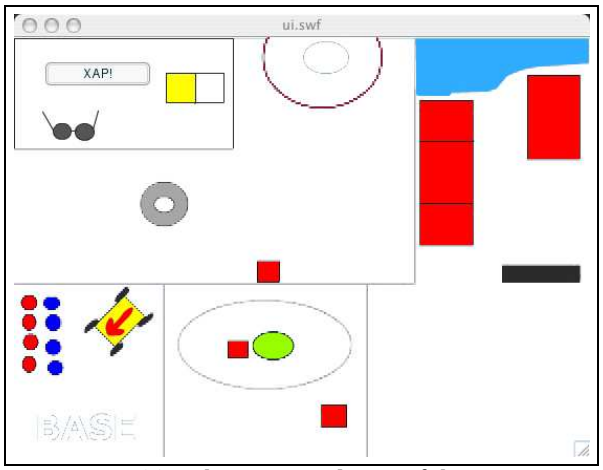
Each XAP instruction corresponds to a specific animation. Flash displays the robot movement in real-time and each step is displayed as it is parsed. Because most of the difficulties of defining what each motion should do is handled by the translator, this significantly decreases the amount of processing needed at this point. The result is smoother viewing and no frame skipping. It is here, with Flash's strong animation and presentation powers, that we can fully demonstrate what the robot is capable of doing. For example, with native controls for collisions, we can produce extremely customizable worlds and interactions between objects. Simple features such as timers are built-in and allow for an authentic reproduction of a real robot.

5. Preliminary Study

An informal demonstration of the RoboXAP prototype was given to a class of thirteen third and fourth grade stu-

3 LASM numbers the three output ports 1, 2 and 4, corresponding to the labels A, B and C on the LEGO robot.

dents in order to provide us with preliminary feedback on the project. The class was part of an after-school robotics program at a local elementary school. All of the students had some experience programming in RoboLab and using the LEGO robots. A brief explanation of how RoboXAP would work in parallel with their robots was given. Because the students had been working on a project which involved their robots traversing a field and performing certain tasks[4], the simulation mimicked the real-life arena that was already familiar to them. The students watched in astonishment as the virtual robot completed several of the real life objectives they had been working on.



(a) RoboXAP emulation of the FIRST Lego League world 2005

(b) participants provide feedback about RoboXAP

Figure 8. Informal focus group

4 The FIRST Lego League challenge 2005

The reaction was unanimously positive. Every student immediately recognized what the simulation was doing and compared it with their own tasks. We asked the class several questions, the most important being: "do you think the simulation would be useful in helping with your project, and if so, how?" Some of the responses included: "it would shorten the amount of time needed to fix programs", "you won't need to take turns on the playing field" and "how realistic is it?" Not surprisingly, the first two responses were the immediate practical issues that we had hoped RoboXAP would solve. Regarding the first comment – using RoboXAP to help debug code – one of the quickest ways students, especially younger ones, lose patience is when they have to repeatedly test out small changes on their robots; it would be interesting to see if having a simulation to assist with debugging could alleviate their anxiety.

6. Discussion

Just as every student is a unique individual, the methods used to teach the students must be unique as well. Conventional teaching methods generally pass information in a single direction: from the teacher to the student. Simulations allow for a different instructional technique, making teachers promoters of information. The student's role also changes from a passive receiver of information to an active contributor to a lesson. By providing students with content that is both real and dynamic, an ability to change what is happening in front of them, and a two-way flow of information, there is a greater motivation for them to learn [4].

Simulators are unbreakable, unlike the real-world counterparts that they are mimicking. This has benefits for every age group. It gives younger kids experiences that might be unsafe for themselves or any equipment they might be using. For older students, it offers them an opportunity to push experiments beyond their limits [17]. Letting students have free reign over their experiments or projects can make for a more enjoyable educational experience.

In addition to a simulator's indestructible nature, it provides the ability to engage in activities that they otherwise would not experience, either because there is no practical equivalent that can be implemented in a classroom setting or because some of the subjects covered in a simulator may be outside standard curricula. Linser's 1999 [15] study gave students the opportunity to try their hand in world politics. The results showed that an interactive, collaborative learning environment creates more student interest and motivation than a traditional classroom lecture. Matson's [19] implementation of a robotic simulator for K-6 grade children does not teach a regular school subject, but attempts to develop logical and critical thinking at an early age.

Few studies have been completed that compare the use of a simulator to traditional teaching methods. Finkelstein's

2004 [7] quantitative study is a rare instance of using simulations in a university undergraduate circuits lab while having control groups learning the same material. Both pre- and post-tests were conducted, and post-test results showed slightly higher learning rates when comparing the simulation group with the control group (who approached the circuits material in a traditional hands-on lab setting).

Our development of RoboXAP is nearing the end of its technical development phase and will move into testing and educational research phases over the next 6 to 12 months. In Spring 2005, a pilot study will be conducted with students in a primary school to determine how effective the environment is in addressing practical (time and equipment), pedagogical (divide-and-conquer and frustration) and academic (agent-based concepts and programming basics) issues. In 2005-06, we will begin to consider more broadly the role of simulators in classroom settings, and we plan to conduct comparative studies to evaluate use of simulated versus hands-on (physical) learning environments.

Acknowledgments

We are grateful to Professor Chris Rogers at Tufts University for his help and advice with developing for the RoboLab programming environment. We are also grateful to Shawn Mishler for enabling the preliminary study. This work was made possible in part by funding from NSF #GK12-03-38329.

References

[1] http://www.addictinggames.com/.

[2] D. S. Blank, J. H. Hudson, B. C. Mashburn, and E. A. Roberts. The XRCL project: The university of arkansas entry into AAAI 1999 mobile robot competition. Technical Report Technical Report CSCE-1999-01, University of Arkansas, 1999.

[3] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. W3C recommendation: Extensible markup language (XML) 1.0 (third edition). http://www.w3.org/TR/REC-xml/, 2004.

[4] L. Chwif and M. R. P. Barretto. Simulation models as an aid for the teaching and learning process in operations management. In *Proceedings of the 2003 Winter Simulation Conference*, volume 2, pages 1994–2000, 2003.

[5] DotML 1.2: The dot markup language. http://www.martin-loetzsch.de/DOTML/, 2003.

[6] B. Erwin, M. Cyr, and C. B. Rogers. Lego engineer and robolab: Teaching engineering with labview from kindergarten to graduate school. *International Journal of Engineering Education*, 16(3), 2000.

[7] N. D. Finkelstein, K. K. Perkins, W. Adams, P. Kohl, and N. Podolefsky. Can computer simulations replace real equipment in undergraduate laboratories? In *PERC Proceedings 2004*, 2004.

[8] Macromedia flash. http://www.macromedia.com/software/flash/.

[9] Flash action script. http://www.macromedia.com/devnet/mx/flash/actionscript.html.

[10] Flash system requirements. http://www.macromedia.com/software/flashplayer/productinfo/systemreqs/.

[11] V. Frias-Martinez, E. Sklar, and S. Parsons. Exploring auction mechanisms for role assignment in teams of autonomous robots. In *Proceedings of the RoboCup-2004: Robot Soccer World Cup VIII*, 2004.

[12] J. Funge, X. Tu, and D. Terzopoulos. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. In A. Rockwood, editor, *Siggraph 1999, Computer Graphics Proceedings*, pages 29–38, Los Angeles, 1999. Addison Wesley Longman.

[13] http://www.googolplex.co.jp/index_us.html.

[14] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proceedings of the First International Conference on Autonomous Agents (Agents-97)*, 1997.

[15] R. Linser and S. Naidu. Web-based simulations as teaching and learning media in political science. In *AusWeb99 Conference Proceedings*, 1999.

[16] M. Lotzsch. XABSL: Extensible agent behavior specification language. http://www.ki.informatik.hu-berlin.de/XABSL/, 2003.

[17] G. B. Lu, M. Oveissi, D. Eckard, and G. W. Rublioff. Education in semiconductor manufacturing processes through physically-based dynamic simulation. In *Proceedings of the 1996 Frontiers in Education Conference*, volume 1, pages 250–253, 1996.

[18] M. J. Mataric. Behavior-based robotics as a tool for synthesis of artificial behavior and analysis of natural behavior. *Trends in Cognitive Science*, 2(3):82–87, March 1998.

[19] E. T. Matson, R. Pauly, and S. DeLoach. Robotic simulators to develop logic and critical thinking skills in under served K-6 school children. In *Proceedings of the 2003 ASEE Midwest Section Meeting*, 2003.

[20] http://www.microworlds.com/.

[21] http://www.legomindstorms.com/.

[22] http://ccl.northwestern.edu/netlogo/.

[23] S. Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. BasicBooks, 1980.

[24] http://www.robocup.org.

[25] Tufts RoboLab. http://www.ceeo.tufts.edu/robolabatceeo/.

[26] W3c xml in 10 points. http://www.w3.org/XML/1999/XML-in-10-points, 2003.

[27] XSL transformations (XSLT) version 1.0. http://www.w3.org/TR/xslt, 1999.