iCricket: A Programmable Brick for Kids' Pervasive Computing Applications

Fred Martin, Kallol Par, Kareem Abu-Zahra, Vasiliy Dulsky and Andrew Chanler

University of Massachusetts Lowell,
Computer Science Department, Engaging Computing Group,
1 University Avenue, Lowell MA 01854 USA

Abstract. The iCricket is a new internet-enabled embedded control board with built-in motor and sensor interface circuits. It is designed for use by pre-college students and other programming novices. It includes a Logo virtual machine with extensions that allow networked iCrickets communicate with one another, retrieving sensor values and remotely running each other's Logo procedures. The underlying implementation uses standard HTTP protocols. The iCricket's key contribution is that it will allow programming novices (children, artists, and other non-engineers) to implement pervasive computing applications with an easy-to-use, interactive language (Logo). This paper focuses the iCricket hardware and software design. Later work will evaluate results of using the design with various users.

1 Introduction

There is a long history of construction toys for use by children, to encourage their creativity and inventiveness [1]. In the computer age, researchers developing programmable materials for children generally use one of two approaches. Early work, led by Seymour Papert at the MIT AI Laboratory, was explicitly based on programming. The Logo language was developed to give children a way to express their ideas in code; the leading application was making drawings using commands to a virtual or physical robot "turtle" [2].

Recent work provides children with a form of implicit programming, also known as "programming by example." Phil Frei's *curlybot* was inspired by the Logo turtle, but children programmed it by moving it directly on a table with their hand. The robot recorded their gesture and then played it back with various permutations [3]. The Topobo project takes inspiration from *curlybot* but allows children to build their own mechanical bodies, with any joint having a similar form of record-and-playback programmability [4].

Because of its pedagogical value, we base our work on the former of these two approaches. Children's programs become a symbol-based representation of their intentions. These representations then act as mediators in their learning experience [5].

Our earlier work includes development of the MIT Programmable Brick, the Handy Board, and the Cricket. These technologies have been used for everything from ele-

Martin F., Par K., Abu-Zahra K., Dulsky V. and Chanler A. (2005).
iCricket: A Programmable Brick for Kids' Pervasive Computing Applications.
In *Proceedings of the 2nd International Workshop on Ubiquitous Computing*, pages 75-81
DOI: 10.5220/0002574100750081
Copyright © SciTePress

mentary children's construction of science experiments to graduate level robotic design courses [6–9].

In this paper, we introduce the iCricket, the latest in a series of microcontroller boards designed for use by children, teachers, hobbyists, and researchers.

1.1 Motivations

Over the last 10 years, light-weight TCP/IP stacks have started to appear in embedded devices, including commercial products [10, 11]. These are based on an inexpensive 8- or 16-bit CPUs with custom TCP stack code. Our goal in designing the iCricket is to create a platform that allows children and other programming novices to create applications that use pervasive computing technology.

1.2 Why the iCricket?

Designing a new board is not the only way to make pervasive computing accessible to novices. Alternately, we could have developed programming environments for a more powerful embedded device (e.g., a PDA with a wireless card) or used existing TCP-enabled devices (e.g., Dallas-Maxim's TINI product [11]).

Partly, we chose to build on our previous work since the Logo language is accessible to children [12], and allows them to interactively design and program their own sensor and motor control systems [13]. Also, we had specific goals for the iCricket device:

- Integrating motor/sensor circuitry. No existing commercial device includes jacks for connecting DC motors and simple analog sensors. The design of the iCricket means that kids and other hardware novices can build functioning systems without needing knowledge of electronics.
- Extensions to the Logo language that support meaningful and easy-to-use communication between peers. Much of our work on the iCricket is contained in the new Logo communications primitives. With just a few lines of code, an iCricket programmer can link sensors on one iCricket to motors on another.
- Minimal hardware design cost. When developing code on the iCricket, the user takes advantage of the keyboard, screen, and overall computational power of a conventional desktop or laptop PC. The user's code is then compiled into Logo bytecodes for the iCricket. This arrangement lets the iCricket itself be a very simple and inexpensive device.
- Integrated TCP/IP communications. Standard network protocols, such as HTTP, let iCrickets communicate with one other and conventional web services.

In an important sense, the iCricket's contribution is a "whole that is greater than the sum of its parts." Individually, the pieces of the iCricket (a programmable brick, the Logo language, an embedded stack) are not new. But taken as a whole system, our intention with the iCricket is to make the ideas of the research community readily accessible to a wide range of users, who otherwise would not have an opportunity to participate in this work.

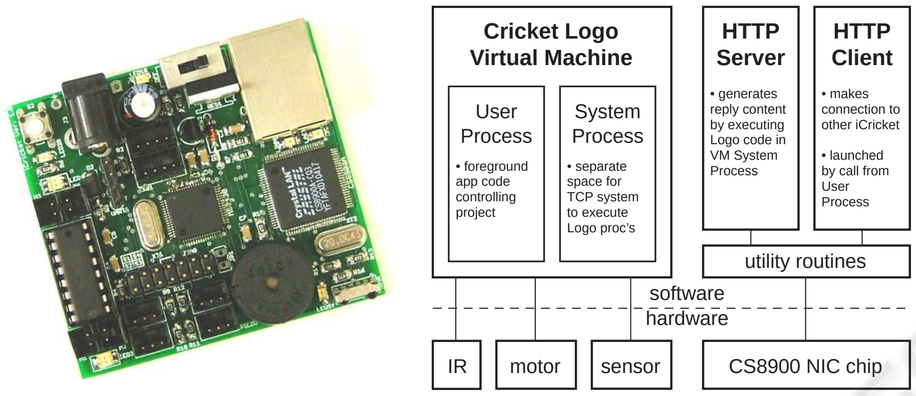
Cricket Logo Virtual Machine
User Process
• foreground app code controlling project
System Process
• separate space for TCP system to execute Logo proc's
HTTP Server
• generates reply content by executing Logo code in VM System Process
HTTP Client
• makes connection to other iCricket
• launched by call from User Process
utility routines
software
hardware
IR
motor
sensor
CS8900 NIC chip

Fig. 1. The iCricket Microcontroller and Block Diagram

From a technical standpoint, the iCricket's central innovation is its interface between the Logo language and the TCP stack. As is described in the following, we have created a remote procedure call-like mechanism that allows iCrickets to execute each other's procedures. Also, we developed a call-back mechanism that allows the built-in web server to transparently execute Logo procedures in order to generate server reply content.

2 Hardware

The iCricket is closely modeled after the Handy Cricket [14], a commercial version of the MIT Media Lab Cricket [15]. (Please note that the work described in this paper has no connection to the MIT CSAIL "Cricket Indoor Location System.")

Figure 1 (left) shows the iCricket device. It is based on Texas Instruments' MSP430 microprocessor, a low-cost 16-bit CPU with a 64k address space and good on-chip peripherals. We use the 'F149 variant, which has 60k of internal flash and 2k of internal RAM. The CPU is coupled with the Crystal Semiconductor CS8900 ISA ethernet adapter chip for 10BT networking. We chose this pair of devices for their ease-of-use, including the availability of prototyping hardware and TCP/IP sample code [16].

The iCricket includes outputs for two DC motors, inputs for four analog sensors (with powered sensor ports), IrDA communications, a piezo beeper, and a JTAG connector (for development purposes). It also has a connector for the "Cricket Bus," a custom 1-wire bus that allows a single iCricket to control a number of locally-attached slave devices [15].

3 Software

Figure 1 (right) illustrates the software running on the iCricket device. The primary subsystems are the Logo Virtual Machine, HTTP Server, and HTTP Client. The HTTP client and server are based on Adam Dunkels' `uip` code [17].

3.1 Logo Virtual Machine

The Logo Virtual Machine (VM) is a stack-based, byte-coded interpreter with 16-bit integer numerals and support for procedures (including arguments, return values, and recursion). The VM includes primitives for interacting with the iCricket's sensor ports, motor outputs, and local IR communications [15].

The iCricket extends previous Cricket VMs by introducing two parallel Logo threads. The User Process runs the application code that would control an iCricket project—e.g., taking sensor readings and controlling local motor outputs based on them. The System Process is a separate execution area that is used by the HTTP Server and Client to execute Logo procedures as part of their operation.

The iCricket VM adds public global variables (called xglobals) and public procedures (called xdo's) which can be accessed by the Client/Server system.

3.2 HTTP Server

When responding to incoming connection requests, the iCricket's HTTP Server has two primary functions. First, it processes the message supplied by a client (another iCricket or any web browser). This may contain requests to execute procedures and/or set variable values. Then, it constructs a reply packet, which announces the values of all of its xglobal variables and contains custom content generated by Logo code.

For background, consider the embeddable HTTP server code presented by Jones in a 2001 article [18]. This server supported "dynamic content in HTML files with an API to provide the content." In this design, a new tag, <`DATA x`>, was used to insert dynamic content into the HTML stream. The parser searched for the `DATA` keyword, then used the embedded variable name to retrieve the actual content.

The iCricket takes this further. We generate reply content by having the HTTP Server make a callback to a specially-named Logo procedure (`answer`) that is provided by the Logo application programmer. Thus, the iCricket not only reports variable values (the xglobals), but also allows arbitrary Logo procedures to execute and supply response data.

The `answer` procedure itself contains a series of calls to a `reply` primitive. These build up a table of name-string/integer-value pairs in a RAM-based table. When the `answer` procedure concludes, control returns to the HTTP Server and the table is exported, producing the reply content.

This design allows the user to create custom reply content by writing just a few lines of Logo code. If an iCricket has a temperature sensor plugged into its sensor A port, it can publish the sensor's value with:

```
to answer
reply "temperature sensora
end
```

The server reply itself is an XML file which can be viewed in a web browser and parsed by another iCricket's HTTP Client. For example, the `answer` procedure above would produce XML like:

```
<?xml version="1.0"?>
<icricket>
<r n="temperature" v="68"/>
<g n="xglobal1" v="0"/>
</icricket>
```

The “r” tag indicates a value reported by the `reply` primitive. Xglobal variables declared in the user’s code are automatically published in the XML reply; these are indicated by the “g” tag.

3.3 HTTP Client

The HTTP Client allows one iCricket to connect to another iCricket. The Client sends an HTTP request with the GET syntax (e.g. “`GET index.xml?xglob1=0&fan=1`”). Here, the Server iCricket would set its `xglob1` to 0 and then would execute its procedure `fan` with an argument of 1. The Server then sends back an XML reply that the Client will parse.

The Client’s actions are scripted with two Logo primitives. The `tell` primitive accepts a name-string and integer-value; these are built up in a command table. The `talk` primitive initiates communication with the remote iCricket. It uses the command table to generate the HTTP GET request and then sends it to the given IP address.

The Client then receives the Server’s XML reply and parses it into another table. Two Logo primitives, `reply?` and `getreply`, examine this table. Respectively, they test for the existence of a name-string and return its value.

4 The iCricket IDE and Application Examples

To build applications with the iCricket, the programmer runs the iCricket IDE on a normal PC or Mac. The IDE includes a compiler (which translates the programmer’s Logo into bytecodes), a downloader (for installing the bytecodes into the iCricket), and a command console (to run code interactively on the iCricket).

The command console is the primary way for interacting with an iCricket. Here, the user can type commands, and they are immediately compiled, downloaded, and executed. This gives the iCricket the flavor of an interactive system. Also, the user can print debug information, which is displayed in the IDE.

Suppose a user wishes to create a thermostat project with two iCrickets: a “Temp iCricket” (has a temperature sensor) and a “Fan iCricket” (controls a fan). This can be done at least two ways: a polling method, in which the Fan iCricket asks the Temp iCricket for temperature readings, and an interrupt method, where the Temp iCricket issues commands to the Fan when the temperature changes.

Let’s look at the polling method first. The Temp iCricket needs to report its temperature reading. The way to do this is to have it report the temperature in its answer procedure:

```
to answer
reply "temp sensora
end
```

To test if this is working, the user can connect to the iCricket from a standard web browser. This would yield a XML file that looked like the example shown earlier, revealing the iCricket's local temperature measurement.

Next, the code for the Fan iCricket is constructed. It has a procedure named `startup` (which automatically runs when the iCricket is powered on) that repeatedly polls the Temp iCricket, and based on the temperature, decides to turn the fan on or off:

```
to startup
talk "temp-IP-addr
ifelse getreply "temp > 70
[a, on][a, off]
startup
end
```

Alternately, an interrupt method can be employed.

Based on the local temperature reading, the Temp iCricket tells the fan to turn on or off. To allow itself to be commanded, the Fan iCricket provides an `xdo` procedure:

```
xdo fan :n
ifelse :n = 1
[a, on][a, off]
end
```

The `xdo` procedure can be interactively tested from the iCricket IDE. After it is working, the Temp iCricket is programmed. It uses the `waituntil` primitive to wait until a temperature threshold has been crossed, and then sends the appropriate on/off command to the Fan:

```
to startup
waituntil [sensora > 70]
tell "fan 1
talk "fan-IP-addr
waituntil [sensora < 70]
tell "fan 0
talk "fan-IP-addr
startup
end
```

5 Discussion and Future Work

The iCricket system provides simple and effective way to implement pervasive computing applications. Sensors and actuators can easily be connected to the internet, and multiple iCrickets can coordinate their actions.

As of this writing, the iCricket system as described is functional. From a technical standpoint, our future work plan includes augmenting basic internet services (DHCP; DNS), manufacturing a larger set of prototypes, and building more demos. Also, we plan a middleware layer that would run on conventional computers, interfacing iCrickets with internet services in general.

More broadly, our research focus is the pedagogical value of the iCricket when it is used by children and other programming novices. By giving children the opportunity to work with this new technology, we will study its effectiveness as a design tool for children, its impact on their attitudes toward technology, and its value in encouraging imaginative applications.

We are planning a variety of venues for bringing the technology to children, including programs in school, after-school, and with community partners.

We also plan to make iCrickets available to the research community. Please connnect to our live demo at `icricket.cs.uml.edu` and give us feedback.

References

1. Petroski, H.: Back to the future. Prism **9** (2000)
2. Papert, S.: Mindstorms: Children, Computers, and Powerful Ideas. Basic Books (1980)
3. Frei, P., Su, V., Mikhak, B., Ishii, H.: curlybot: designing a new class of computational toys. In: CHI ’00: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM Press (2000) 129–136
4. Raffle, H.S., Parkes, A.J., Ishii, H.: Topobo: a constructive assembly system with kinetic memory. In: CHI ’04: Proceedings of the 2004 conference on Human factors in computing systems, ACM Press (2004) 647–654
5. Ackermann, E.: Direct and mediated experience: Their role in learning. In Lewis, R., Mendelsohn, P., eds.: Lessons from Learning. Proceedings of the IFIP TC3/WG3.3 Working Conference (1993)
6. Martin, F.: Children, Cybernetics, and Programmable Turtles. Master’s thesis, Massachusetts Institute of Technology, Cambridge, MA (1988)
7. Martin, F.: Circuits to Control: Learning Engineering by Designing LEGO Robots. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA (1994)
8. Martin, F.: Robotic Explorations: A Hands-On Introduction to Engineering. Prentice-Hall (2000)
9. Resnick, M., Berg, R., Eisenberg, M.: Beyond black boxes: Bringing transparency and aesthetics back to scientific investigation. Journal of the Learning Sciences **9** (2000) 7–30
10. Inc., N.: SitePlayer Embedded Internet Server, NetMedia Inc: `www.siteplayer.com` (2005)
11. Products, M.I.: TINI (Tiny InterNet Interface), Dallas Semiconductor/Maxim Inc: `www.maxim-ic.com/TINIplatform.cfm` (2005)
12. Kafai, Y.: Learning through making games: Children’s development of design strategies in the creation of a computational artifact. In Kafai, Y., Resnick, M., eds.: Constructionism in Practice. Lawrence Erlbaum Associates (1996) 71–96
13. Martin, F., Mikhak, B., Resnick, M., Silverman, B., Berg, R.: To mindstorms and beyond: Evolution of a construction kit for magical machines. In Druin, A., Hendler, J., eds.: Robots for Kids: Exploring New Technologies for Learning. Morgan Kaufmann (2000) 9–33
14. Martin, F.: The Handy Cricket: `handyboard.com/cricket/` (2005)
15. Martin, F., Mikhak, B., Silverman, B.: Metacricket: A designer’s kit for making computational devices. IBM Systems Journal **39** (2000)
16. Dannenberg, A.: MSP430 Internet Connectivity. Technical Report SLAA137A, Texas Instruments (2004)
17. Dunkels, A.: Full TCP/IP for 8-Bit Architectures. In: Proceedings of MOBISYS 2003, San Francisco, CA (2003)
18. Jones, M.T.: An Embeddable HTTP Server: Web-enabling embedded devices. Dr. Dobbs Journal (2001)