eBlocks – An Enabling Technology for Basic Sensor Based Systems

Susan Cotterell, Ryan Mannion, Frank Vahid[¥], Harry Hsieh
Department of Computer Science and Engineering
University of California, Riverside
Riverside, California, USA
{susanc, rmannion, vahid, harry}@cs.ucr.edu; http://www.cs.ucr.edu/eblocks
[¥] Also with the Center for Embedded Computer Systems at UC Irvine

***Abstract*—We describe the development of a set of embedded system building blocks, known as eBlocks. An eBlock network can be viewed as a basic form of sensor network that can be developed by non-programming engineers, scientists, and others. Each eBlock has a defined function, either one of a few pre-defined combinational or sequential functions, a custom-programmed function defined by an automated tool, or by user with programming skills. A user creates an application simply by connecting blocks, and possibly performing simple configuration via dials and switches. We have built over 100 physical eBlock prototypes, and tested their usability with over 100 non-programming users to date. We will describe the architecture of the blocks, including design tradeoffs we considered and the benefit of an exploration tool that we developed to help optimize the power and performance of the design. We have also built a graphical eBlock simulator that users can utilize to quickly build and test systems before deployment, and that we have used in experiments with over 300 non-programming users to help us define intuitive block functions and interfaces. We will describe the simulator architecture, as well as a tool that automatically converts a user's eBlock network into a much smaller network of programmable blocks with accompanying automatically generated programs.**

Keywords-eBlocks; Sensor networks; Embedded Computing Systems; User Interfacing

I. INTRODUCTION

We are developing a set of electronic building blocks (eBlocks) to enable non-computing-specialists to build basic sensor/actuator systems. Presently, people without computer programming skills cannot themselves setup sensor/actuator systems (with the exception of some limited home automation applications), inhibiting numerous useful applications. There exists a large class of applications consisting of basic sensor-based systems that simply transform sensor data using basic logic and state functions and feed the processed data to output or actuator devices, or to PCs or sensor-network compute nodes for further processing. For example, one such application is a sleepwalking detector, which uses motion sensors and light sensors distributed through rooms and hallways to detect motion in the dark, causing a buzzer to sound at a nursing station or bedside. Another application is detecting the presence of endangered species by photographing or recording nocturnal animals when they feed at a particular location. A third application monitors a section of farmland for the presence of insects and dispenses insecticide accordingly. We have encountered several dozen similar "basic" sensor-based applications and can easily think of hundreds of potential applications.

Users of these systems are not computing specialists, but rather biologists, hospital workers, secretaries, teachers, and so on. Our goal is to enable those users to setup systems to implement the users' particular customized applications, without requiring such users to learn a temporal programming language, which requires users to create a time-ordered sequence of operations in a textual or graphical language, and which others and we have found to be too difficult for most non-computing-specialists. Our blocks all have predefined functions and thus do not require programming. Users build systems simply by connecting blocks together, and configuring dials and switches of certain blocks. While not as general a paradigm as temporal-language programming, the eBlock approach addresses a useful and large class of applications. Existing as well as proposed technologies do not meet the needs of non-computing-specialists.

"Sensor network" technology is an emerging class of systems [9][18][27] consisting of tens to thousands of small wired or wireless sense and compute nodes with stringent battery, size, and cost constraints. Applications include military surveillance, medical monitoring, inventory control, monitoring machines for wear, monitoring the structural integrity of buildings and bridges, home automation, and environmental monitoring. Sensor-network technologies presently require expert programming to tailor to a particular application [11][27] – even described as requiring "2.5 Ph.D.'s" [10]. Recent sensor-network programming methods still assume temporal-language programming skills, introducing application programmer interfaces specific to common sensing tasks [1][28]. Numerous electronic building block and robotic device technologies emphasize ease of use, but still require temporal-language programming [16][19][24][26]. Others technologies are targeted towards children and are too simplistic to build useful monitor/control sensor-based systems [6][13]. Pre-manufactured products are available but are designed for a specialized task [22]. These products solve only one problem and not easily customizable. X10 based products [30], which communicate over home power lines, provide some configuration features on devices or using a PC, but are oriented to specific home automation tasks. Some sensor-based systems are designed for education of basic logic theory and consist of low-level gate components and microprocessors that are unfamiliar to users without a computer science background [12][14].

In this paper, we highlight eBlocks themselves, and then describe the physical eBlock prototypes and the graphical eBlocks simulator/synthesis tool that we have developed.

II. PHYSICAL EBLOCK PROTOTYPES

The primary goal of the eBlocks project is to build a set of electronic blocks that would enable people with little or no programming or electronics expertise to create basic sensor-based

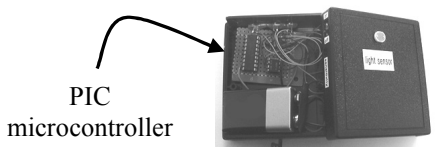

Figure 1: Internals of a light sensor eBlock.

systems simply by connecting blocks. We achieved this by adding inexpensive low-power microcontrollers to each block, as shown in Figure 1. Previously "dumb" components like a button or an LED would then become a tiny compute node with a specified functionality, and connecting these components together would create a small computer network. Furthermore, we created blocks whose sole purpose is to compute a pre-defined combinational or sequential function. The manner in which a user chooses and connects blocks defines the system's overall functionality. For example, Figure 2(a) illustrates a sleepwalking detector. A user connects a light sensor and motion sensor to a logic block. The user configures the logic block, via a DIP switch, to perform A'B. When the system detects no light and motion, the system transmits a signal to the LED, causing the LED to illuminate. Notice that no traditional form of programming is involved. By selecting different nodes, we can build another application, a daytime doorbell that sounds a buzzer when a visitor presses the doorbell only during the day. This type of doorbell may be useful to a person who has a small child and doesn't want the doorbell disturbing the child while sleeping at the night. Again, the user simply connects the various sensors as shown Figure 2(b), configuring the logic block to perform AB. Figure 2(c) presents another application designed to record nocturnal animals. Many similar simple monitor/control systems exist which deal with human-scale events. Figure 2 presents several basic systems, however, increasingly complex systems are achievable simply by connecting additional eBlocks. In [4][5], we provide a more detailed discussion of the motivation for eBlocks as well as describe the types of applications that we can construct using eBlocks.

A. Block Definition

In defining the set of available eBlocks, a robust library of blocks allowing users to create any system conceivable must be balanced with the number of blocks available such that the library itself does not become overwhelming. We defined the library of blocks by considering dozens of applications. We developed 30 sensor-based systems, decomposed those systems to determine the types of blocks required to build each system. As we created new systems with the library of blocks, we refined the blocks or added new blocks.

In general, there are two types of eBlock systems: integer-based systems that operate on number values (temperature, seconds of time, decibels, etc.) and Boolean-based systems that operate on yes/no values (motion/no motion, light/no light, sound/no sound, etc.).

We currently focus only on Boolean blocks and in future extensions plan to consider integer-based blocks. Boolean eBlocks can be broken down into four basic categories:

- Sensor blocks - monitors the environment and include motion sensors, buttons, contact switches, and so on.
- Output blocks - provides stimuli and include light-emitting diodes (LEDs), beepers, and so on.
- Intermediate blocks - performs basic logic transformations (e.g. AND, OR, NOT) or basic state functions (e.g. prolong, toggle, trip).
- Communication blocks - provides wireless point-to-point communication.

Each Boolean eBlock maintains an internal state of yes, no, or error. Blocks communicate with one another by sending packets consisting of the block's current state. Blocks transmit packets when a sensor's state changes or upon a timeout (currently defined as 3 seconds). [5] provides more detail on eBlock compute and communicate protocols.

B. Block Design

The development of the underlying architecture, computation protocols, and communication protocols require balancing competing multi-level considerations. The high-level design goals include lifetime, latency, responsiveness, and reliability. For example, it is not feasible for every block to have a wall power supply, thus blocks must have the option of being battery-powered. Blocks must have a sufficient lifetime so that users are not constantly changing batteries. Latency is an important consideration because high block latency compounds into a slow network response to environmental input. For example, the sleepwalking detector illustrated in Figure 2 must quickly alert users of a sleepwalker, a delayed alert of 5-10 minutes defeats the purpose of this system. Responsiveness plays an important role in situations when users disconnected blocks from or connected blocks into a working network (hot-swapped). If a new sensor is added into the network, but the network does not recognize the newly added block for ten seconds or so, the user might assume the newly added sensor is not working. Conversely, if a block fails or is disconnected by a user, the network should detect the failure/disconnect promptly and respond accordingly. Lastly, the data transmitted within the network must ensure a certain level of reliability.

Our design goal is to select the right microprocessor to embed within each block. We selected the PIC16F628 due to the microcontroller's low power operation, consuming only 20 μA when active and 0.20 μA in power-down mode [17]. By powering down the

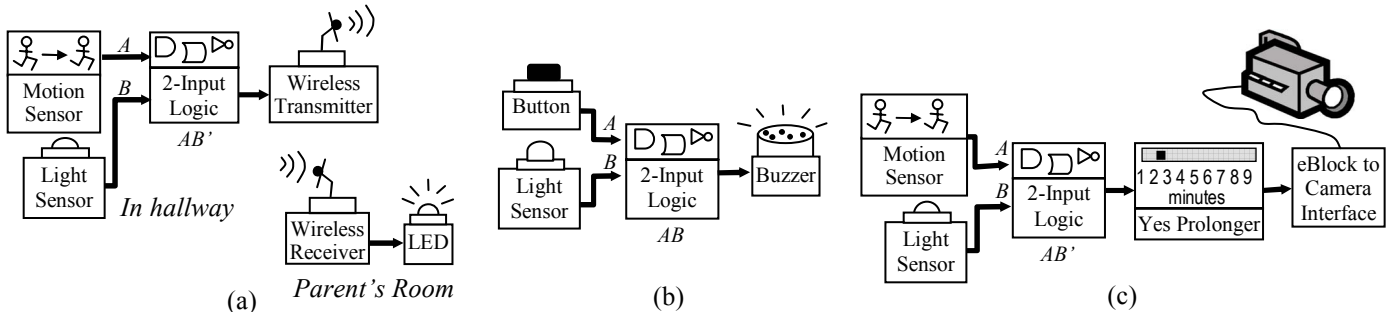

Figure 2: Connecting eBlocks forms an end-applications without any programming but perhaps slight configuration., (a) Sleepwalk Detector, (b) Daytime Doorbell, (c) Endangered Species Monitor.

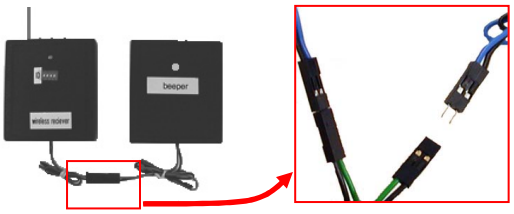

Figure 3: eBlocks connectors.

microcontroller between the computation and communication, a block is able to last several years on a battery.

The block communication medium, wired verses wireless, is another important design issue. Most sensor networks communicate wirelessly and are ad hoc networks in which nodes dynamically self-configure the network. Nodes may be added to, removed from, or fail within the network without changing the functionality mapped to the network. In an eBlock network, the manner in which blocks are connected defines the functionality of the network. Wired connections provide an explicit connection between blocks, such that users can easily decipher the network's corresponding functionality, making wired connections actually preferable for system configuration by novices. Furthermore, power consumption of a wired connection is orders of magnitude lower than a wireless connection, and is lower in cost since the need for specialized circuitry or antennas is eliminated. Many applications are localized and have no need for every block to communicate wirelessly. We realize, however, there are benefits to wireless connections in simplifying the physical design of the network, and we therefore provide the option for wireless point-to-point links within an eBlock system by including wireless transmit and receive blocks configurable to particular channels.

The underlying configuration of hardware along with the computation and communication protocol heavily impacts each of the high-level design goals. We consider parameters such as supply voltage, clock frequency, baud rate, packet size, packet transmission frequency, and error checking/correcting strategy. Because of the sheer number of possible configurations and interdependencies, we developed an exploration tool to evaluate design space. The tool includes a set of equations, derived primarily from data sheets and straightforward electrical equations, relating the design parameters to the design metrics of lifetime, latency, responsiveness, and reliability. The tool also includes a user-weighted cost function of those metrics. The tool exhaustively searches all configurations of design parameter values and returns the configuration yielding the lowest cost.

By carefully defining the architecture, computation protocols, and communication protocols, our physical eBlock prototypes can last several years on a 9-volt battery. We determined the parts cost in moderate volume to be between $1.80 and $3.67, depending on the block. A Harvard Business School project involving eBlocks estimated off-the-shelf costs of the blocks in moderate volume to be between $4 and $12, depending on the block, with costs decreasing each year due to technology trends, and lower costs for higher volumes [25].

C. *Physical Prototype Usage*

Building physical prototypes enabled us to observe and test block usage and to refine our block designs. To test usability, we conducted experiments with dozens of people [4] building a variety of pre-defined sensor-based applications of varying difficulty. Such applications included a doorbell which only rings during the day, a system which sounds an alert when motion is detected at night in any of multiple locations around a home, a paging system to alert a receptionist of a person waiting in the lobby, and so on.

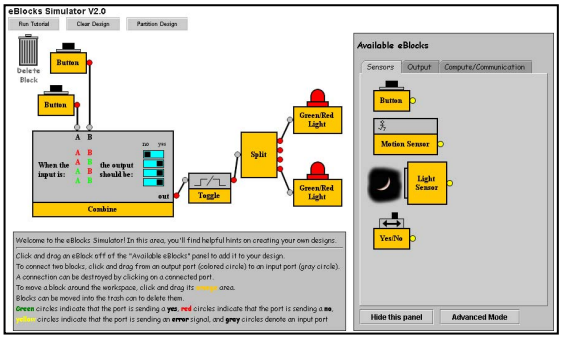

Figure 4: eBlock Simulator.

A key design criterion we found is that all eBlocks should be self-explanatory. No special training, including reading a separate set of instructions or taking a tutorial, should be required to use the blocks. We initially provided users with handouts describing the functionality of each block, however these handouts were either ignored or not understood by users. We further tried including descriptions on the back of each block, but users continued to ignore the descriptions. We found that non-engineers often do not read or understand such instructions, which is consistent with other studies showing users instead preferred exploratory learning [7][21]. Thus, we found the need to limit instructions to a 10 word or less description on the front of the block, perhaps even captured in the block's name itself. For example, we initially defined a "tripper" block with two inputs, data and reset. The tripper block mimics a classic SR latch, a *yes* on the data line sets the block's state to *yes*. The tripper block remains in the *yes* state until a *yes* is received on the reset line. We found that tripper is a technical term not readily understood by non-engineers, and by renaming the block to "once yes, stays yes," a non-technical term describing the block's function, usability greatly increased.

We placed status/debugging lights consisting of small, colored LEDs on the side of each block to indicate the status of the block (green means yes, red no, and yellow error). Originally, we intended the LEDs as an aid to ourselves in the debugging of blocks – we did not expect users to utilize the LEDs extensively. However, users of our physical prototypes placed extensive focus on the LEDs while developing their systems. Users were able to follow chain of blocks and check to see if each block was is in the desired state. If not, users were easily able to detect at which point the system no longer followed the desired specification and the user could make the changes accordingly. The LEDs contributed to the exploratory building methodology users preferred. We therefore integrated the status/debugging lights in the final design to aid users in deciphering how the system was responding to various stimuli.

Smaller details also played an important role in ensuring usability. In order for users to easily connect blocks together, eBlocks utilize an inexpensive 2-pin connector, shown in Figure 3. The black wire corresponds to ground, and the blue and green wires correspond to data. The initial connector did not limit the connections between wires. In many instances, users would connect the ground wire to the data wire, and vice versa, and could not figure out why a system did not function correctly. To alleviate the aforementioned problem, we updated the eBlock connectors such that the connectors could only snap together in a single configuration, aligning the ground and data wires.

Furthermore, each eBlock contains an on/off switch. Users commonly forget to turn on each block within the network and are unable to figure out why the network does not respond to stimuli in the desired effect. Future implementations will situate the on-off switch in a more obvious position.

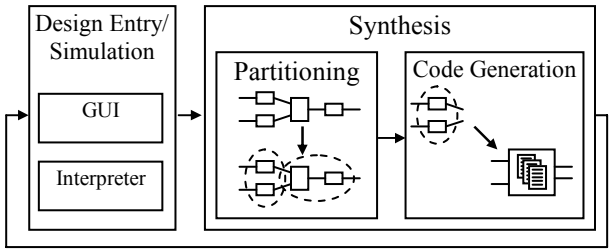

Figure 5: Partitioner Design Framework.

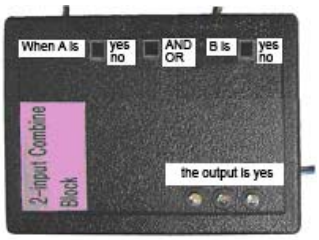

Figure 6: Physical prototype based on a logic sentence.

III. EBLOCK SIMULATOR

Although the physical prototypes were imperative in the initial testing of eBlocks, physical prototypes are expensive and time consuming to produce. To reach larger numbers of users, ease data collection of usage experiments, and facilitate our evaluation of different eBlock interfaces, we developed a graphical eBlock simulator, shown in Figure 4.

A. Simulator Implementation

The eBlock simulator is a Java-based graphical user interface (GUI) for the design entry and simulation of an eBlock system. The user is able to drag a block from a catalog of blocks, situated on the right edge of the simulator, to the workspace and connect the blocks together by drawing lines between circular representations of blocks' input/outputs. Users are able to choose between a "simple mode" and "advanced mode." In order not to bombard novice users, the simple mode displays a core subset of blocks from the library. The advanced mode provides a greater variety of blocks to build systems that are more complex. eBlocks that sense or interact with their environment include accompanying visual representation of their environmental stimuli/interaction to simulate the corresponding environment. For example, the light sensor shown in Figure 4, is accompanied by a "day/night" icon. Users click on the icon to alternate between day and night, causing the light sensor's output to change accordingly.

The gray text box situated in the bottom-left quadrant of the simulator, displays context-sensitive help. As users hover their mouse cursor over blocks in the simulator, the block's description and interface automatically appears in the text box. Additionally, we included an online tutorial in the simulator that demonstrates two simple eBlocks systems and shows how to connect blocks together. Because users favored the status/debugging lights integrated in the physical eBlocks, the simulator also colors blocks' output ports indicating the internal state of each block.

We implemented the simulator in an object-oriented approach such that developers could incorporate new types of blocks with minimal effort. We developed a base eBlock class that implements the underlying communication protocol and methods by which the GUI and partitioner interface with each block. On top of the base eBlock class, we define specific subclasses, one for each type of block. The subclasses contain a behavioral representation of that block's functionality. In addition, the subclasses describe how the simulator draws each block in the GUI. In the base eBlock class, we provide methods to draw a standard block and its ports as well as user-configurable components such as switches (as illustrated in the Combine block in Figure 4) and multi-position switches used in other blocks. eBlock subclasses can further define extra components to draw, such as the "day/night" icon in the Light Sensor.

The eBlock simulator is behaviorally correct, however the simulator does not capture all low-level timing detail. Communication between blocks is done serially using packets and hence globally asynchronous. Given the asynchronous nature of our communication protocol and the fact that input blocks react to human-scale events, our simulation models are able provide the equivalent physical network functionality despite the lack of precise timing.

B. Program Synthesis

For more advanced users, we provide a tool in the simulator for automatically programming programmable eBlocks. The tool automatically converts internal (non sensor or output nodes) blocks in an eBlock network into a much smaller network of programmable blocks, where possible, that preserve the design's functionality, and the tool automatically generates code for each programmable block. The design framework, illustrated in Figure 5, is broken down into three steps: specification, partitioning, and code generation.

1) Specification

Utilizing the eBlock simulator discussed in the previous section, a user can define and test the functionality of a given eBlock system.

2) Partitioning

After the specification phase, the design specification enters the partitioner. The goal of the partitioning phase is to minimize the total number of blocks by replacing pre-defined compute blocks with a fewer number of programmable blocks. A programmable block contains i input ports and o output ports. Furthermore, a programmable compute block is assumed to have slightly higher cost than a pre-defined compute block due to the hardware required to permit programmability, but less cost than two pre-define compute blocks.

The partitioning phase works in the following manner. First, the eBlock system is characterized as a directed acyclic graph $G = (V, E)$ where V is the set of nodes (blocks) in the graph and E is the set of edges (connections) between the nodes. Sensor and output nodes are primary blocks and invalid selections for partitioning. The partitioner's objective is to find a set of subgraphs of G such that:

- Each subgraph has at most i inputs and o outputs, corresponding to the number of inputs and outputs available in a programmable block
- Each subgraph must be replaceable by a programmable block that can provide equivalent functionality
- The number of internal blocks after replacement is minimized.

Utilizing a decomposition heuristic referred to as PareDown, all internal blocks of a design are selected as a candidate partition. The partitioner removes individual blocks from the partition until the input and output constraints are met. This process continues until the partitioner finds no new partitions. The choice of which block to remove from an invalid candidate partition is based on selection strategy we developed that considers the effect block has on the candidate partition's ability to meet the criteria we defined for a valid subgraph. For a more detailed description of the decomposition heuristic, refer to [15]. The decomposition method achieved optimal, or within 15% of an optimal solution, compared to an exhaustive search methodology.

TABLE I. PHYSICAL PROTOTYPE AND SIMULATOR USAGE SUMMARY

	Date	Num. of Participants	Prototype	Simulator	Description
Non-computing users	02/09/04	13	●		Usability of physical block interfaces
Non-computing users	04/06/04	4	●		Usability of physical block interfaces
Non-computing users	05/10/04	41		●	Usability of various logic block interfaces
Non-computing users	05/20/04	40		●	Usability of various logic block interfaces
Non-computing users	05/21/04	67		●	Usability of various logic block interfaces
Non-computing users	06/29/04	10		●	Usability of various logic block interfaces
Non-computing users	06/30/04	16		●	Usability of various logic block interfaces
Non-computing users	08/03/04	18		●	Usability of various state based block interfaces
Non-computing users	08/06/04	8	●	●	Usability of physical, logic, and state block interfaces
Non-computing users	10/12/04	9		●	Usability of logic and state block interfaces
Non-computing users	10/19/04	9	●		Usability of physical block interfaces
Beginner computing users	05/18/04	21		●	Usability of various logic block interfaces
Beginner computing users	06/01/04	84		●	Usability of various state based block interfaces
Beginner computing users	07/08/04	22		●	Usability of various logic block interfaces
Beginner computing users	08/04/04	14		●	Usability of various state based block interfaces
Intermediate computing users	06/03/04	50		●	Usability of various logic block interfaces
Advanced computing users	02/09/04	6	●		Usability of physical block interfaces
Advanced computing users	05/27/04	39		●	Usability of various state based block interfaces
Advanced computing users	10/27/03	21	●		Usability of physical block interfaces
Miscellaneous	varied	5	●		Usability of physical, logic, and state block interfaces
Total Participants:		**497**			

3) *Code Generation*

After a set of partitions is determined, the framework passes each partition to a code generation tool we developed. The code generation tool translates the functionality of a given partition into a Java-like language, and automatically transforms the new behavior into a syntax tree. The newly merged syntax tree specifies the behavior of the programmable block that will replace the partition. The simulator's interpreter evaluates the syntax tree in the same manner as a non-programmable block. Furthermore, since physical eBlocks exists, we were able to translate the syntax tree into C code, compile and download the generated code, and use the output of the synthesis tool chain in real-world systems. The programmable eBlock prototype utilizes a Microchip PIC16F628 microcontroller with 2 Kbytes of program memory.

C. *Simulator Usage*

The eBlock simulator proved to be an invaluable tool for testing eBlock usability and providing for rapid iterations of eBlock designs. For example, most systems do not directly connect a sensor block to an output block, but rather require a logical transformation of two or more sensor values. The eBlock library provides 2-input and 3-input "combine" blocks. In engineering, the combine block's functionality is equivalent to simple Boolean equations. However, the expression of Boolean equations by non-computing-specialists is not a simple problem. A survey of front-end approaches for searching online public access catalogs [8] found users had great difficulty in composing Boolean expressions and required the aid of an expert. Web search engines also posed problems for users, 7 out of 10 users are dissatisfied with Internet search engines and less than 6% of users use Boolean search terms "and", "or", "+", and "-" [23]. Users are familiar with AND and OR because they are found in natural language (i.e. English) but these words take on a different meaning when used in a Boolean search [20]. Many times users incorrectly substitute the AND logical operator for the OR logical operator. Furthermore, users do not understand that the scope of the NOT operator varies, and users often ignore parenthesis. While many alternatives for specifying Boolean equations exist [2][20][29], they do not translate well to the combine block interface. Additionally, eBlocks are limited by power, cost, and physical dimensions. Many of these methods require a computer to configure the various blocks.

Utilizing the eBlock simulator, we have implemented six different combine block interfaces, ranging from truth tables to sentences. We had approximately 200 students build a variety of systems requiring a Boolean transformation of sensor data. Such systems include a doorbell ringing only during the day (AB), an alarm indicating a garage door left open at night (A'B'), and an alert indicating motion on the property (A+B). We have found that using color in truth tables improves success as well as a combine block having a sentence-like structure with some configurable switches, as shown if Figure 6. Furthermore, renaming the block from its original name, "logic block," to "combine block," enables users to better recognize the block's functionality, coinciding with the intuitive notion of needing to combine two or more blocks' outputs together in a certain manner. A more detailed description of the logic block interfaces considered, the testing methodology, and results is presented in [3].

Furthermore, the simulator is able to save and accumulate various user-defined systems into a centralized database. We can then later evaluate these user-defined systems of various eBlocks and determine common user pitfalls. Future work will include further automation.

IV. EBLOCK USAGE

Table I summarizes the various usages over the past year of our physical eBlock prototypes and our graphical tool. Participants who were non-science, non-engineering majors are classified as non-computing users. Typical majors of these participants include business, history, and psychology. Participants classified as beginner computing users consisted of students in their first course of introductory programming (<10 weeks). Participants classified as intermediate computing users are students with 2-3 courses of programming. Further, advanced computing users are participants with both programming and electronics experience. Lastly, miscellaneous users consisted of middle-school or high-school kids, or non-engineering adults. The hands-on testing has provided invaluable

TABLE II. USER SUCCESS RATES BUILDING VARIOUS CATEGORIES OF EBLOCK SYSTEM IN 10 MINUTES ACROSS DIFFERING SKILL LEVELS

Categories of eBlock Systems	Percentage Success	Number of Students
Sensor-to-output	56%	91
Sensors-with-logic	54%	281
Sensors-with-state	66%	168
Sensors-with-logic-and-state	12%	39
Overall	**55%**	**579**[a]

a. Total number of students is higher than TABLE I because individual students may have participated in multiple tests

information regarding user interfacing, usability, and comprehension of eBlocks. Through testing, we were able to observe ambiguities and common mistakes, we then re-designed problematic areas and tested the refined designs as discussed in previous sections.

TABLE II summarizes students of varying expertise levels building a variety of systems involving just sensors and outputs, sensors with logic and outputs, sensor with state and outputs, and sensors with logic and state and outputs. We saw that more than half of all users were able to build the first three types of systems in less than 10 minutes. In comparison, advanced students required days to weeks to build similar systems without the benefit of eBlocks. In [4], we provide a more in-depth description of the experimental methodology and results.

V. CONCLUSIONS AND FUTURE WORK

The creation of physical prototypes and a graphical tool has been essential in developing an effective set of eBlocks that can be used by non-computing-specialists to develop basic sensor/actuator applications. We highlighted various details of the physical prototypes and the graphical tool, and summarized their use in a large variety of user experiments. Future work includes developing a set of integer-based blocks, involving new physical prototypes and extensions to the graphical tool. Additional related work includes projects underway with companies to apply and refine eBlocks for use by healthcare workers and adult caregivers in remotely monitoring at-home elderly people with cognitive impairment, and for use in an agricultural project enabling automated targeted local application of pesticide in response to insect counts in farm fields.

ACKNOWLEDGMENT

This work is being supported by the National Science Foundation (CCR-0311026).

REFERENCES

[1] Abdelzaher, T., et. al. EnviroTrack: an Environmental Computing Paradigm for Distributed Sensor Networks. Int. Conf. on Distributed Computing Systems, 2004.

[2] Anick, P. G., et. Al. A Direct Manipulation Interface for Boolean Information Retrieval via Natural Language Query. Proc. ACM SIGIR Conf. On Research and Development in Information Retrieval, User Interfaces (1990), 135-150.Beyond Black Boxes http://llk.media.mit.edu/projects/bbb/, 2004.

[3] Cotterell, S., F. Vahid. A Logic Block Enabling Logic Configuration by Non-Experts in Sensor Networks. CHI, 2005.

[4] Cotterell, S., K. Downey, F. Vahid. Applications and Experiments with eBlocks - Electronic Blocks for Basic Sensor-Based Systems. IEEE SECON, October 2004.

[5] Cotterell, S., F. Vahid, W. Najjar, H. Hsieh. First Results with eBlocks: Embedded Systems Building Blocks. CODES+ISSS, 2003.

[6] Electronic Blocks, http://www.itee.uq.edu.au/~peta/Electronic%20Blocks.htm.

[7] Gammon, B. Everything we currently know about making visitor-friendly mechanical interactives. British Interactive Group, http://www.big.uk.com, 1999.

[8] Hidreth, C. R. Intelligent Interfaces and Retrieval methods for Subject Search in Bibliographic Retrieval Systems. Advances in Library Information Technology, 2 (1989).

[9] Hill, J., D. Culler. MICA: A Wireless Platform For Deeply Embedded Networks. IEEE Micro 22, 6 (2002).

[10] Horton, M. Commercial Wireless Sensor Networks: Status, Issues and Challenges. Keynote Presentation, IEEE SECON, 2004.

[11] Horton, Mike. et. al. MICA: The Commercialization of Microsensor Notes. Sensors Online, April 2002.

[12] Kharma, N. and L. Caro. Magic Blocks: A Game Kit for Exploring Digital Logic. American Society for Engineering Education Annual Conference, 2002.

[13] Logiblocks, http://www.logiblocs.com/.

[14] Logidules, http://diwww.epfl.ch/lami/teach/logidules.html, 2004.

[15] Mannion, R., H. Hsieh, S. Cotterell, F. Vahid. System Synthesis for Networks of Programmable Blocks. Design, Automation and Test in Europe (DATE), March 2005.

[16] Martin, F., et. al. Crickets: Tiny Computers for Big Ideas. http://lcs.www.media.mit.edu/people/fredm/projects/cricket/.

[17] Microchip, http://www.microchip.com.

[18] National Research Council. Embedded, Everywhere: A Research Agenda for Networked Systems of Embedded Computers. National Academies Press, 2001.

[19] Phidgets, http://www.phidgets.com/.

[20] Pane, J. and Myers, B. Tabular and Textual Methods for Selecting Objects form a Group. Proc. Visual Languages, pp. 157-164, 2000.

[21] Sikorski, M. Teaching Computers the Young and the Adults: Observations on Learning Style Differences. CHI, pp 42-43, 1998.

[22] Smarthome Inc., http://www.smarthome.com.

[23] Tanaka, J. The Perfect Search. Newsweek 134, 13, pp 71-72, 1999.

[24] Teleo, http://www.makingthings.com/.

[25] Vahid, F., S. Cotterell, S. Bakshi. eBlocks: Embedded Systems Building Blocks. Harvard Business School Business Plan Contest, 2004.

[26] Wallich, P. Mindstorms Not Just a Kid's toy. IEEE Spectrum, September 2001.

[27] Warneke, B., M. Last, B. Liebowitz, K. Pister. Smart Dust: Communicationg with a Cubic-Millimeter Computer. Computer Vol. 34, Issue 1, pg. 44-51, 2001.

[28] Welsh, M., and G. Mainland. Programming Sensor Networks Using Abstract Regions. First Symposium on Networked Systems Design and Implementation, 2004.

[29] Young, D. and Shneiderman, B. A Graphical Filter/Flow Representation of Boolean Queries: A Prototype Implementation and Evaluation. Journal of American Society for Information Science 44 (1993), 327-339.

[30] X10 protocol, http://www.x10.org