TOWARD AMBIENT PROGRAMMING FOR CHILDREN

Nwanua Elumeze and Michael Eisenberg
Department of Computer Science
University of Colorado, Boulder USA 80309-0430
nwanua@colorado.edu, duck@cs.colorado.edu

ABSTRACT
Children’s programming is typically envisioned as an activity that takes place solely in front of a desktop screen. This paper argues that, given an environment in which increasing numbers of objects are “computationally enriched”, we can move children’s programming toward an activity that is itself undertaken away from the desk, in all sorts of day-to-day situations. We use three programmable objects–tiles, a shirt, and a pendant–as a springboard for discussing ways in which children’s programming may be re-thought as an ambient activity, in which programming is done wirelessly via PDAs and using novel means of gathering input for running programs.

KEYWORDS
Children's programming; ambient programming; SmartTiles.

1. INTRODUCTION: AMBIENT PROGRAMMING FOR CHILDREN

One of the most enduring and passionate debates in educational computing is whether (and if so, how) programming should be introduced to children. On the one side, proponents argue that children can be intellectually empowered by programming, and that important ideas–e.g., of mathematics, software design, or engineering–may be introduced to children through this medium. (Probably the best-known and most eloquent argument along these lines was Papert's [1980] book Mindstorms, which discussed the then-recent Logo language as a medium in which children could represent and explore "powerful ideas" such as abstraction and recursion.) Opponents of children’s programming, on the other side, argue that the medium is too difficult–or perhaps boring–for children to learn. In practice, this theme is often coupled with a more expansive general critique of the role of computational media in education (cf. [Oppenheimer 2003]); for such opponents, the argument is that programming is merely one of the ways in which computer technology has largely proven harmful (or perhaps wasteful) in education.

These larger debates on educational computing aside, it is also possible to take a middle ground–respecting the potential of programming as a creative medium for children, while admitting that relatively few children seem to successfully appropriate this medium. Indeed, many prominent researchers have shared Papert's enthusiasm for programming, but have acknowledged the cognitive or motivational barriers that make the subject difficult for children. Some have responded by creating powerful programming environments intended (in a variety of ways–graphical programming, programming by example, and so forth) to alleviate the difficulty of writing long or complex programs. These innovations are themselves part of a larger tradition of research in "end-user programming"–a subject focusing on the ways in which nonprofessionals can be introduced to programming (e.g., through spreadsheets). On this view, children represent a certain type of "end-user"–a type whose interests and cognitive abilities are mediated by age and experience. Good summaries of research in this area can be found in [Cypher 1993] and [Nardi 1993]; both sources deal with the subject of children's programming as a specific instance of end-user programming. (A recent very brief account of the research issues surrounding end-user programming, also mentioning children's languages, can also be found in [Myers et al. 2006].) Two excellent recent discussions of programming language design specifically in an educational context can be found in Resnick [2005] and diSessa [2000].
Clearly the subject of children's programming is complex and controversial—and we will return to several of the above-mentioned themes a bit later in this paper. For our present purposes, however, it is worth focusing on one (usually unstated) assumption that has held sway in all previous traditional discussions of the subject: namely, that the act of programming itself is done at the desktop. That is, the notion of "children's programming" has virtually always been associated with the visual image of youngsters sitting before a desktop computer. There have been creative exceptions: Wyeth and Wyeth [2001], for example, describe a (very simple) “building block” programming medium for young children in which the programming elements are communicating blocks (e.g., representing sensors and logic gates) that can be composed into larger playful constructions. Still, most research has proceeded on the assumption that “programming” means working at a desk, in front of a screen. In most of these examples, the program itself is likewise something that runs on a desktop screen—e.g., a graphics program or interactive story; but even in those cases (like the beautiful “programmable Lego brick” and its successors [Resnick et al. 1996]) where the newly-created program runs in a setting away from the desktop, the act of writing the program—the act of programming itself—takes place at the desktop.

The advent of “ambient computing”, with its plethora of computationally-enriched objects scattered throughout the day-to-day environment, suggests new and potentially quite powerful directions for children’s programming. First, programming the behavior of a physical object—as in the programmable Lego brick—typically entails writing a program that is much shorter and simpler than the huge files encouraged by earlier media for children’s programming. In part, this is because simple one-page programs can generate delightful effects when they control the behavior of physical objects. Moreover, programming physical objects meshes well with a large range of children’s activities—activities that involve physical movement and play in a wide variety of settings.

In this paper, we argue for a novel re-examination of children’s programming. In particular, we argue that ambient computing should be accompanied by ambient programming for children—a style of program-writing geared toward creating (as well as reading, discussing, and debugging) small programs embedded within interactive physical objects in children’s environments. It is our belief that those very technological developments that have woven computers into a variety of physical objects and interactions likewise hold the potential for revolutionizing traditional assumptions about children’s programming.

We begin this argument, then, with several "objects-to-think-with". The following section of this paper presents three programmable physical objects designed for use and exploration by children. Our focus, however, is not so much the particular objects themselves (two of them are discussed at greater length elsewhere), but the means by which they may be programmed and customized. We proceed, then, by describing our current techniques for programming these objects (on and off the desktop), and we use the objects as motivation for the beginning of a much wider-ranging discussion on new directions in ambient programming for children. The final section describes our (very early) pilot tests of the various objects with groups of children, and outlines directions for a busy agenda of future work.

2. THREE PROGRAMMABLE AMBIENT OBJECTS

SmartTiles

SmartTiles are small, touch-sensitive, independently programmable tile objects that can be combined to cover various sorts of planar surfaces. Each SmartTile contains a microcontroller that dictates the rules by which it communicates with neighboring tiles; red and green LEDs to indicate (via color) an internal state value; and a piezoelectric disk to make the tile touch-sensitive. When assembled on a grid that provides power and communication facilities, the tiles collectively enact user-customizable cellular automaton programs.

A brief digression is in order here on the subject of cellular automata. In cellular automaton systems, a large array of simple, synchronized computers is arranged on a grid; each of these computers communicates only with its immediate neighbors, and each computer changes its own internal state over time in accordance with some (usually simple) rule that depends on its own current state and that of its neighbors. To take a particularly powerful and well-known example: in the "Game of Life" automaton system [Gardner 1983], each computer in the array has one of two states, labeled "alive" (corresponding to a red color) and "dead"
Each computer then changes its state at time \( t \) according to the following rule: if this computer (cell) was alive at time \( t-1 \) and had two or three live neighbors, it remains alive at time \( t \); if this cell was dead at time \( t-1 \) and had exactly three live neighbors, it becomes alive at time \( t \); otherwise, this cell is dead at time \( t \). A simple rule of this sort, as it happens, can give rise to remarkably fascinating dynamical patterns over time. Although there is no space here for a more thorough discussion of the subject, cellular automata are extremely powerful media in which to represent and simulate an astonishing variety of systems in mathematics, the natural sciences, and even the social sciences. (Toffoli and Margolus [1987] and Poundstone [1985] are good introductions to the subject.)

**Figure 1:** A ten-by-ten array of 1-inch cubical SmartTiles running a cellular automaton program.

To bring the discussion back to SmartTiles: by programming each tile with its own simple rule, the overall array may be used as a cellular automaton system. Figure 1 shows a ten-by-ten array of tiles; in the figure, each of the tiles holds an identical program corresponding to the above-mentioned "Game of Life" system. (A more detailed description of the SmartTiles implementation may be found in [Elumeze and Eisenberg 2005].)

For our immediate purposes, the crucial aspect of SmartTiles is contained in the phrase "independently programmable" mentioned in our initial description of the device. Because each tile includes its own microprocessor, it may be customized to hold its own particular cellular-automaton procedure. Thus, although it is often the case that each tile is programmed to run according to the same identical procedure (this is the case in Figure 1), it is not hard to imagine alternative situations in which individual tiles can run their own specialized programs; for example one could stipulate that one of the tiles in Figure 1 is always "alive" (corresponding to a bright red color). Thus, the tiles collectively comprise a "surface covering" of computational elements, each of which is associated with its own program, and each of which communicates with its immediate neighbors.

**Figure 2:** The programmable TankTop shirt. In the figure, the user is reprogramming her own shirt using a wireless device, as discussed later on in the text.
TankTop

The programmable TankTop (see [Buechley et al. 2006] for more detail) is a shirt into which 140 individual LEDs—arranged in a 7x20 grid pattern and controlled by an AVR microcontroller—have been sewn. This array of LEDs can be used to display a range of low-resolution animations, as well as different cellular automata programs of the sort described above. Figure 2 shows the shirt being wirelessly reprogrammed with a starting pattern for the “Game of Life” system. One important difference between the SmartTiles of the previous subsection and the TankTop is that the latter is only associated with one microcontroller rather than many: this controller can be programmed to run a simulation of a cellular automaton system, but it contains a single program (and processor), rather than a large set of distinguished programs (and processors) as in the SmartTiles example. We will return to this distinction later, in the discussion section of this paper.

Pendantif

The Pendantif is a programmable pendant that incorporates a PIC microcontroller, switch, and multi-color LED; the LED softly illuminates a plastex "stone", and also doubles as a color sensor. While the pendant itself may be used in a variety of ways, our first instantiation of the object simply changes, gradually, between two colors that were "picked up" from everyday objects. The color change incorporates intermediate combinations of colors between the two points in the color spectrum retrieved by the user. Figures 3 and 4 show the object itself, and illustrate the means by which endpoint colors may be selected.

3. PROGRAMMING LANGUAGES FOR AMBIENT OBJECTS

The previous section introduced programmable objects of significantly different types: tiles, a shirt, and a pendant. In this section, we describe our current work in designing programming media for these objects.
Subsequently, we use these current efforts as a springboard for a wider-ranging discussion of ambient children’s programming.

**Cellular Automaton Programming: SmartTiles and TankTop**

We are currently developing a cellular automaton programming interface that can be used for both the TankTop and SmartTiles. In effect, the interface consists of several distinct modules for programming cellular automaton rules, and for directly initiating the values of automaton elements in a running simulation. Importantly, this interface runs on a handheld PDA, and is able to download automaton programs “in situ”, wirelessly, to the objects. That is, it is not necessary to write the desired cellular automaton rules on a desktop computer, nor is it necessary to bring the tiles or shirt to a desktop computer for programming: instead, the rules may be downloaded via an IR link to the objects. One could thus imagine a child using a PDA to directly program or interact with a running simulation on the shirt of a friend sitting nearby.

The automaton programming interface consists of several modules: a textual programming language that permits the user to write sophisticated cellular automaton rules (this is currently only applicable to the tiles); a menu-driven “beginner’s interface” that is much simpler than the textual language, but allows the creation of only a limited set of rules; and an interface that allows the user to initialize the automaton’s state (by setting the values of individual cells to “live” or “dead”).

Space does not permit a complete description of the entire interface here, but Figure 5 shows a view of the textual programming interface used to program the tiles on a PDA screen. The rule being written here is that for the classic “Game of Life”: if a red cell is surrounded by two or three red neighbors, it remains red (“live”) at the next time step; if a dark (“dead”) cell has three red neighbors, it turns red at the next time step. (In all other cases the cell is dead at the next step.)

![Figure 5. A textual rule that can be downloaded to the programmable tiles. As of this writing, textual rules cannot be sent to the programmable shirt, nor to the pendantif.](image)

Another portion of this same programming interface permits the user to directly set cell values in the automaton (using the PDA pointer to toggle cell values on a grid shown on the PDA screen). Once the user has created the desired pattern of “live” and “dead” cells on the PDA screen, she can communicate that pattern wirelessly to the shirt or tiles. (Figure 2 above in fact depicts the user/wearer initializing the state of her own shirt’s program in just this way.) This portion also allows the user to interact with a running simulation by (e.g.) pausing, changing running speed, or restarting a simulation.

The textual language shown in Figure 5 is not especially syntactically “friendly” (and will undoubtedly undergo revision as we go along), but again it should be noted that the programs under design are invariably brief. It would not be too hard, for instance, to edit this program so that (e.g.) an “off” cell changes to red on the next time step if and only if it has four (as opposed to three) red neighbors. Moreover, as noted above, we are in fact designing distinct “levels” of programming interface, so that at least some (relatively constrained) types of rules may be programmed via menu choices, without using the textual language. For the purposes of this argument, the key point here is that a very small, spare (but still powerful) mobile programming interface may be designed for natural usage with ambient objects like the tiles.
Bringing Ambient Objects “Back to the Desk”

One interesting and perhaps subtle difference between the SmartTiles and TankTop is that in the latter case, the entire automaton is inseparably located on one surface (the garment). SmartTiles, however, are individual objects that may be removed and replaced within an array. Each tile is in fact a stand-alone lightweight “computational nugget” that can (if the user wishes) be plucked out of a simulation and reprogrammed individually.

In this case, the style of “ambient programming” is much closer to that of the classic programmable brick: an individual object is brought to a desktop computer, reprogrammed, and then placed back into its “off-the-desktop” environment. Figure 6 shows an individual tile being removed from a small array for just this purpose.

Our own desktop “tile-programming” environment is in fact similar to that implemented on the PDAs: there is a menu-driven “simple” version and a more powerful textual language. Again, the crucial point here is not to dwell on the specifics of our implementation, but rather to note some interesting features of ambient programming: namely, that depending on the type of object involved, there may be constraints (or opportunities) for the corresponding programming environment. An object like the SmartTile, which can be individually removed and easily brought to a desktop machine, may be reprogrammed with a programming environment designed for a large desktop screen. Conversely, an artifact in which the programmed elements are unmovable (say, tiles cemented to a wall) or permanently grouped (as in the TankTop) may best be programmed exclusively via the portable handheld interface.

Figure 6: Removing a tile from a five-by-five array in preparation for reprogramming it at a desktop machine.

The Pendantif as Programmable Object and Ambient Input Device

Thus far we have discussed programming techniques for the TankTop and SmartTiles—and in both cases, the natural “programming domain” is that of cellular automata. The Pendantif represents an interesting departure from this domain. In one sense, it too may be viewed as a programmable object in its own right: certainly, its capabilities as a “color recorder and playback” are dictated by its embedded program. (At present, we do not have an interface–desktop or PDA—for programming the Pendantif.) More loosely, one might call the user’s choice of endpoint colors a very coarse-grained kind of “programming” for the object; we would more likely call this “customization” or “parameter selection” rather than programming per se.

Our own feeling is that the most interesting role of the Pendantif is as a new type of ambient object that can be used, eventually, as an element of a larger ambient programming system. One scenario that we have implemented employs the Pendantif as an "ambient input device" which can pick up a color from the physical environment and transfer that color to another object. Figure 7 shows this scenario in action: in the first photograph, the user employs the Pendantif to retrieve a selected (reddish) color from a plastic box. In the second photograph, the Pendantif's color is transferred via infrared signal to a single SmartTile in a large array of tiles. (The tile has been augmented to retrieve colors via an IR signal and to adopt the color that it
retrieves.) Once the chosen tile has changed color, the entire array of tiles acts as a cellular automaton to propagate that color in "waves" over the surface of tiles. Thus, Figure 7 overall may be read as a scenario in which a portable pendant-like object may be used to pick up environmental cues (such as a color) and transmit those cues to running programs in other ambient media.

*Figure 7.* Using the Pendantif as an "ambient input device". Upper left: a reddish color is retrieved from a plastic box in the environment. Upper right: that color is transmitted to a single SmartTile. Bottom: the array of tiles, acting as a cellular automaton, propagates the transmitted color over the entire surface.

4. CENTRAL THEMES IN AMBIENT PROGRAMMING; ONGOING AND FUTURE WORK

The previous discussion suggests a variety of themes that should, we believe, form a foundation for research in ambient children’s programming. While we only have space for a summary of these themes here, they prominently include the following:

*Writing programs "in situ".* As the SmartTiles and TankTop examples suggest, it is often the case that a user might wish to program (or reprogram) an object without recourse to a desktop computer. In this case, it is important to develop programming interfaces (e.g., for handheld devices) that are less elaborate than full-fledged language systems, but that are sufficient for writing very simple or short programs.

*Multi-level Programming Environments.* The previous paragraph suggests, in turn, that new sorts of programming environments can be designed at various "levels" of complexity and power. For example, a handheld environment to program a SmartTile may be very simple; but should the user wish to bring a tile back to a desktop machine, she might then have recourse to a more featureful programming system that takes advantage of greater speed, memory, and screen size, among other advantages. Thus, rather than viewing programming environments as single-level entities, we might partition them into "desktop" and "portable" versions.

*Motivational and Affective Issues.* One of the traditional barriers to children's programming has been the relative sparseness of programming projects geared toward children's interests. Ambient programming
potentially offers children the ability to create customized clothing (like the TankTop), as well as a myriad other programmable objects (trains, marionettes, mobiles, dioramas, just to name a few possibilities). We believe that there is a strong possibility of enhancing children's involvement in programming through the use of computationally-enhanced physical objects.

As to ongoing and future work: initial pilot tests of the various objects described in this paper have already begun. For instance, we recently held an informational session with two classes at a local high school, where the students got to play with these and other computationally-enhanced craft items. Though the students’ exposure to the objects was brief, the enthusiastic response is encouraging. We have also begun to engage students in longer-term, closely-observed “case studies” that assess the objects and their associated programming systems. We are also continuing active development of the programming systems (both PDA and desktop) for all these objects.

In the longer term, the objects themselves are likely to evolve (and to suggest still newer opportunities for ambient programmability). For instance, a wider range of SmartTiles that respond with (e.g.) musical tones or mechanical behavior rather than only with LEDs, should be possible; likewise, we imagine versions of the TankTop geared toward “communities of garments”, in which (e.g.) a program can be communicated from shirt to shirt. In the case of the Pendantif, our immediate term plan is to develop a rudimentary programming interface, and then to implement means for “sending” a color value from the pendant to other objects.

Clearly, there is much still to do; but we believe that these initial steps reflect a possibility for a much broader change in children’s programming. By devising programmable objects for children’s environments—tiles, garments, wearables—we can move programming away from its traditional association with the desktop. Rather, we can re-imagine the nature of programming as an activity woven, in small playful ways, into children’s day-to-day lives.

ACKNOWLEDGEMENT

Leah Buechley created and designed the TankTop. Thanks to Mark Gross and members of Craft Technology Lab. This research was funded in part by the National Science Foundation (awards no. EIA-0326054, and REC0125363).

REFERENCES