Rapid Construction of Functioning Physical Interfaces from Cardboard, Thumbtacks, Tin Foil and Masking Tape

Scott E. Hudson and Jennifer Mankoff
HCI Institute
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213
{hudson, jmankoff}@cs.cmu.edu

ABSTRACT
Rapid, early, but rough system prototypes are becoming a standard and valued part of the user interface design process. Pen, paper, and tools like Flash™ and Director™ are well suited to creating such prototypes. However, in the case of physical forms with embedded technology, there is a lack of tools for developing rapid, early prototypes. Instead, the process tends to be fragmented into prototypes exploring forms that *look like* the intended product or explorations of functioning interactions that *work like* the intended product – bringing these aspects together into full design concepts only later in the design process. To help alleviate this problem, we present a simple tool for very rapidly creating functioning, rough physical prototypes early in the design process – supporting what amounts to interactive physical sketching. Our tool allows a designer to combine exploration of form and interactive function, using objects constructed from materials such as thumbtacks, foil, cardboard and masking tape, enhanced with a small electronic sensor board. By means of a simple and fluid tool for delivering events to "screen clippings," these physical sketches can then be easily connected to any existing (or new) program running on a PC to provide real or Wizard of Oz supported functionality.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Design, Human Factors

Keywords: Tools, rapid prototyping of physical interfaces, sketching of interactive physical forms.

INTRODUCTION
Paper prototypes, sketches, and other early design prototypes are an important first step for many GUI designers. One reason they are valuable is the speed with which they can be constructed, tested, and thrown away or modified [18]. This allows early communication and discussion about potential designs both with users and with other product team members. Early, rough forms can be particularly valuable because of the way end-users react to them: they are often seen as unfinished, inducing end users to provide richer design suggestions [11].

UIST'06, October 15–18, 2006, Montreux, Switzerland.

However, as computing moves off the desktop and into other physical forms, paper prototypes have become less complete [15], functioning interactive prototypes tend to be limited to fixed forms, such as existing mobile devices [14], and rapid prototyping tools are likewise limited. As a result, designers tend to be constrained to either create prototypes that *look like* the final product (either physically, *e.g.* using media such as foam mockups, or visually, *e.g.* using tools such as Flash™) or *work like* the interactions envisioned for the final product (*e.g.* provide equivalent interaction via an on-screen simulation). As a result, as illustrated in Figure 1a, the design process often involves a splitting of form and interactivity early, with only a gradual combining of these into a full product design. This results in a less fluid process with potentially slower and weaker communication and iteration.

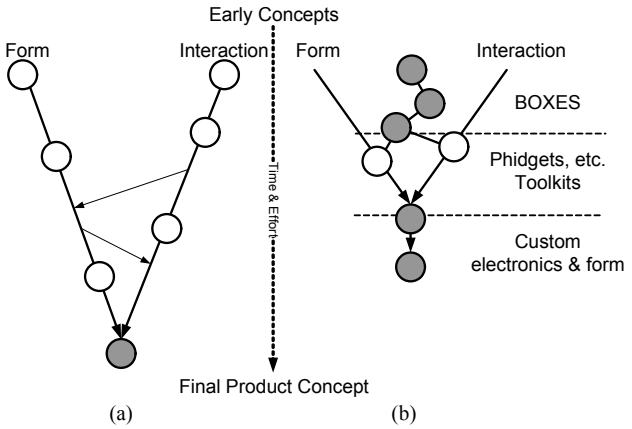


Figure 1: BOXES helps to reduce the need for parallel but separate development of form and interface.

The work described this paper seeks to alleviate this problem at the earliest stages of prototyping with BOXES (Building Objects for eXploring Executable Sketches), a tool that enables rapid construction of early interactive physical prototypes that both look like *and* work like the intended product. As illustrated in Figure 1b, this allows for an integrated design process that combines form, interface, and function in an earlier and more fluid fashion.

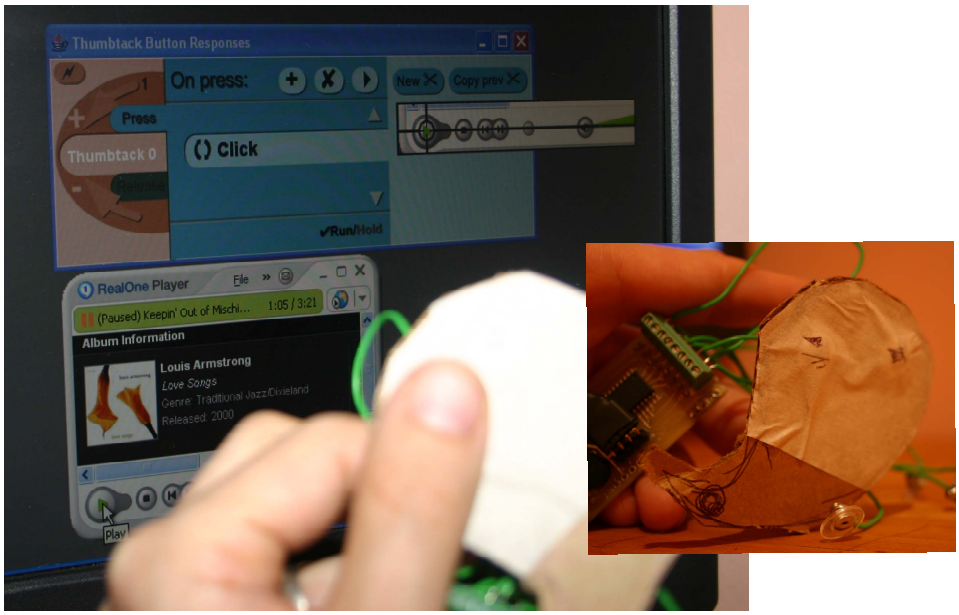


Figure 2: A rough cut at a portable MP3 player that can play/pause (button under thumb) or stop a song. In the background, the on-screen implementation is visible. The "play" button (Thumbtack 0) causes a mouse click over the play button on the RealOne media player. Inset image is a close up of the physical sketch prototype with the BOXES circuit board cupped in the user's hand.

Designers using our tool can construct prototype forms from everyday objects like cardboard, thumbtacks, aluminum foil and masking tape, and can implement the functionality for prototypes using any rapid prototyping language or existing interface on a PC.

Thumbtacks and foil (optionally placed underneath tape and labeled) are touch sensitive when connected to BOXES and can be easily and rapidly (re)positioned on a piece of cardboard. A simple wire connects each thumbtack (optionally pressed through a foil patch of arbitrary shape and size) to a small sensor board, which in turn communicates with a PC through a USB connection. The designer can then assign on-screen functionality to each touch sensitive component. Functionality is defined in terms of translations into mouse and keyboard actions delivered to specific portions of the desktop. Thus, as illustrated in Figure 2, the designer of a portable MP3 player can cause a simulated mouse button click to be delivered to the "play" button of an on-screen media player when the user presses the thumbtack labeled "play."

The remainder of this paper is organized as follow: First, we describe how a designer might construct a simple application using our tool. We then describe the hardware and software implementation that makes BOXES possible. We used an iterative, designer-centered process to develop BOXES, and we end by describing the experiences of designers using our tool.

BACKGROUND

The underlying requirements driving the design of the BOXES system came from our past ethnographic work with designers [2]. In those investigations, we learned the following high level critical points that motivated the design of BOXES:

First, in order to facilitate design exploration; communication; and design testing; we should support a highly interactive system with a short design/implement/test cycle – making changes in seconds rather than minutes or hours.

Second, the physical side of the tool needs to be close in size to the final interactive components that will be used, and needs to be able to integrate into existing design platforms such as cardboard and foam. This suggests the need for a tiny, portable piece of technology that can be "stuck into things" … such as thumbtacks.

Third, the digital side of the tool needs to be sensitive to design practice. We particularly wanted something that did not require coding, that was interactive, that could support rapid iteration, and that could work with existing tools such as Director™ and Flash™.

After completing our formative work, we conducted two pieces of exploratory research – we created the *Switcheroos* system which supported a Director™-based set of pinned components based on RFID technology [2] and we created a programmable set of more complex interactive components that integrated with Java (the Calder toolkit [12]). The BOXES system has been constructed based on lessons learned from these systems including the need for small size, the efficacy of button-based approaches and the need for rapid experimentation with component positioning.

Our approach is distinguished by its ability to be used at the very earliest stages of design. In contrast to past work such as the Calder hardware toolkit [12] and the physically much larger components of the Phidgets [7] and iStuff toolkits [4] it allows designers to construct forms using everyday, non-interactive objects, and add functionality to them simply by inserting thumbtacks and/or placing foil patches.

Closely related to the work presented are previous systems which have used devices configured in a tack or push pin including [13,21,22]. These devices offer a similar fluidity of placement. However, each includes a processor in the externally attached portion of the device, and hence has a notably larger form factor.

Our approach is not end-user programming: our system does not include rules, and is not intended to encode complex actions. Rather, it is intended to use the delivery of simulated mouse and keyboard events to *trigger* complex actions that are available in existing or designer-created applications or prototypes. In comparison to past systems [2, 4, 7, 10, 12], no coding is needed, nor is a compatible piece of software required: any action or series of actions that can be done with the keyboard and mouse can be attached to a touch sensitive physical button in BOXES.

Our approach is most similar to the end-user accessible "widget picker" provided with Phidgets [6], but uses an interactive *window clipping* approach to action specification inspired by the Snip [9] and WinCuts [20] systems. The Phidgets widget picker enabled end users of applications built with standard Windows™ components to tie activation of physical components to the activation of a GUI component. Our approach to action is similar, but is targeted at designers and will also work with mock-ups done in Flash™ or Director™ (as well as Java) that do not use standard components.

In summary, our approach is smaller, simpler, more fluid, more application independent, and requires less coding than similar previous tools. While it also has less functionality (smaller range of sensors, *etc.*), this tradeoff enables it to target a much earlier stage of design than past tools.

SCENARIO

Suppose a designer is creating a portable MP3 player. She can create an appropriate physical form using cardboard. She begins to iteratively decide where to place buttons on the form and what functionality they should control. For example, the physical interface in Figure 2 has a "play/pause" and a "stop" thumbtack/button. The thumbtacks are connected by wire to a circuit board taped to the back of the cardboard. On screen, the designer must now specify what should happen when the user touches a thumbtack.

First, she needs to create or run the interface that will provide behaviors – in this case a free media player suffices. She then "cuts" out the relevant portion of her desktop (essentially grabbing the portion of the screen where that interface is running). She selects the first thumbtack (Thumbtack 7, in our scenario), and indicates, using our GUI, that when the user touches that thumbtack (and/or foil patch), it should cause the mouse to "click" in the spot indicated by the cross hairs (over the play/pause button of a media player). A similar process allows her to associate "stop" with the second thumbtack. For visual effect, she may place masking tape over the thumbtacks or patches and draw appropriate icons to indicate how they should be used.

In this scenario, the entire specification process involves selection of the screen cut area, followed by a pair of action specifications each consisting of selecting the thumbtack involved, selecting an action type (in this case a click) and setting the position of the click using a crosshair indicator – a total of seven selection or positioning actions. Our observations show that after minimal training designers can create such a specification in well under a minute.

Some important things to note here are that:

(1) BOXES is implementation independent. The same process could be used with any media player, or with a custom application developed by the designer using her tool of choice

(2) BOXES is form independent. Assuming it is possible to attach a wired thumbtack or foil pad, the designer can approach physical design using her tools and process of choice

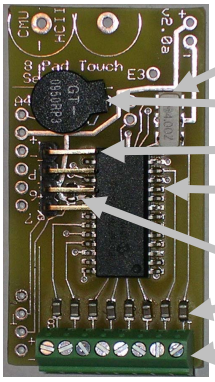
(3) BOXES is intended to support a flexible, integrated *design process*. It enables rapid back-and-forth exploration of issues of form (such as button position) and issues of interactivity (such as whether navigating to a new song should pause or stop the song that is currently playing).

(4) BOXES is "interpreted" and immediately executable. At any point, the developer can "run" the actions associated with any button.

(5) BOXES does not include a display. Although this limits the fidelity of the prototype being created, it helps to maintain the flexibility that designers in our interviews value. It also allows designers to explore rich visual affordances cheaply and quickly on the more capable PC platform. This is appropriate for the early stage of the design process on which we are focusing.

IMPLEMENTATION

BOXES consists of two core pieces: one or more hardware boards to which thumbtacks are attached by wires, and a software interface in which thumbtacks are associated with on screen actions.



Parts List

1	4 MHz resonator
1	Piezo-speaker
1	Serial connector
1	PIC 16F767 µcontroller
1	0.1 µF (bypass) capacitor (hidden under serial connector)
8	0.1µF (accum.) capacitors
1	8 Connector terminal block

Figure 3. BOXES touch sensor board (approx. actual size) and parts list

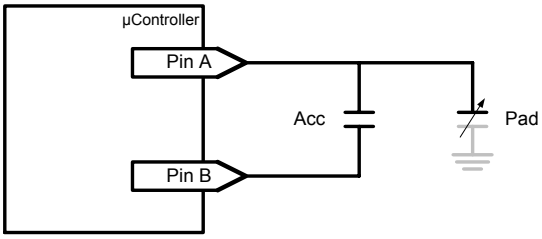


Figure 4a. Capacitive touch sensing circuit

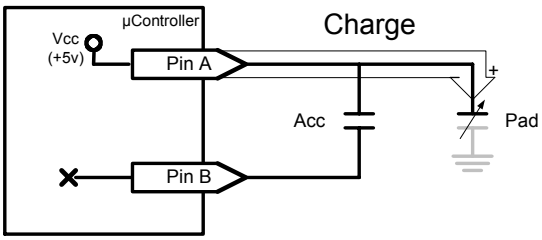


Figure 4b. Charge phase of sensing.

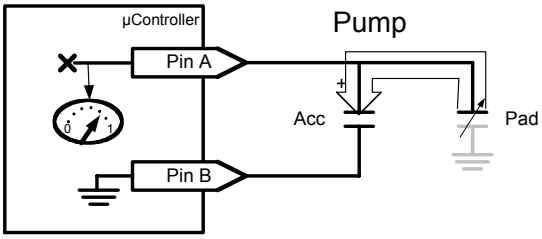


Figure 4c Pump phase of sensing

Hardware

The BOXES system makes use of small custom circuit boards (50mm x 27mm). As shown at approximately actual size in Figure 3, each board contains a single inexpensive microcontroller (a PIC 16F767) nine small (0.1uF) capacitors, a 4 MHz ceramic resonator, a small piezo speaker, and two connectors. Each board provides eight capacitive touch sensors that can be connected to a thumbtack or foil pad by a single wire. Currently we solder this wire to a small earring clasp that easily fits over and grasps the thumbtack pin, creating an electronic connection to the thumbtack. Foil pads are typically held in place by a wired thumbtack and tape. Inputs are reported from the board using a TTL level (5v) serial signal. Currently this signal is delivered to the PC on the USB serial bus by means of a small, off the shelf, adaptor board. It may also be possible to make use of compact, off the shelf, BlueTooth wireless modules which have recently become available.

The board provides eight copies of the capacitive touch sensing circuit illustrated in Figure 4a. (This circuit has also been employed for example in [5] and see [8] for additional approaches) With proper firmware (described below), this circuit can measure the very small capacitance existing along the capacitively coupled path between the attached thumbtack and ground (possibly passing through the user). This path is represented in Figure 4 by the "Pad"

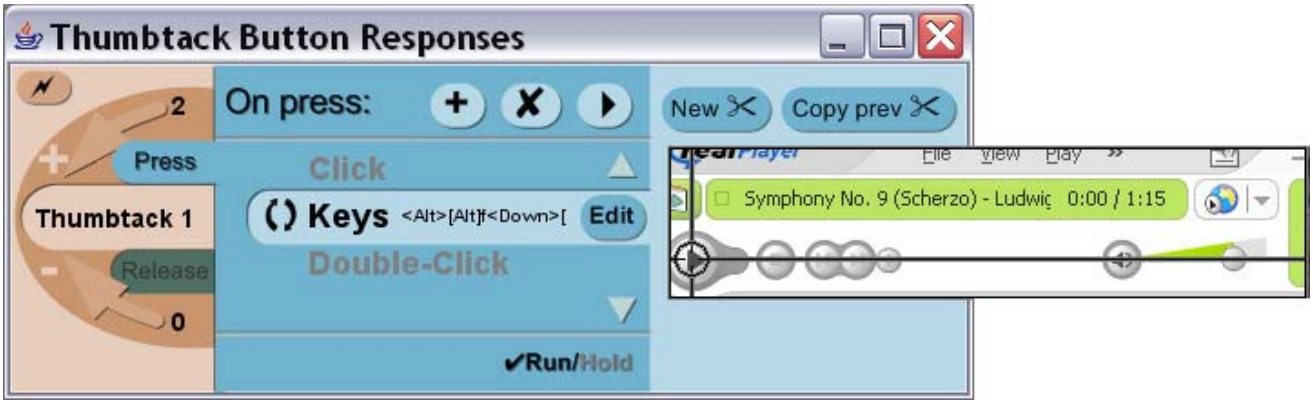
```
// Reset charge Acc capacitor by grounding
// both sides
Output Low(Pin_A);
Output Low(Pin_B);

// Loop, counting pump cycles needed to
// raise Acc capacitor above logic 1 min
count = 0;
while (Input(Pin_A) != 0 && count < max_cnt)
  {
    // Charge cycle (Figure 4b)
    Input(Pin_B);
    Output_High(Pin_A);

    // Pump cycle (Figure 4c)
    Input(Pin_A);
    Output_Low(Pin_B);

    // Count this round
    count++;
  }

// the more cycles required the lower
// capacitance of the pad...
return count;
```

Figure 5. Capacitive sensing code

capacitor at the right (which we will refer to as a *virtual* capacitor since it is not an actual electronic component). Although this capacitance is always quite low, it is much higher when a person is very near to or touching the thumbtack or foil patch than when the path goes through free air. The touch sensor operates by detecting significant changes in this measured capacitance. Note that no direct electrical connection is required, and the sensor is sensitive enough to operate reliably when the thumbtack is placed under a layer of masking tape. This allows the designer to optionally cover buttons and draw meaningful labels over them (a capability that we found important in our preliminary testing).

The circuit in Figure 4a works by repeatedly pushing a charge out to the (virtual) capacitor associated with a thumbtack or foil, then transferring (or *pumping*) that charge back into the larger *accumulation* capacitor (marked "Acc" in the center of Figure 4). When a human is near the thumbtack, the capacitance in the virtual capacitor increases, causing more charge to be held in the first step, and causing the accumulation capacitor to receive more charge in the second step. Measurement is performed using only digital components, by counting the number of pump cycles necessary to raise the accumulation capacitor from an initial ground state to above the minimum voltage that registers as a logic 1. This is done in a series of steps illustrated in Figures 4b and c and detailed in the code shown in Figure 5.

As indicated in Figure 5, the main algorithm is a loop that repeats a charge phase and a pump phase until input Pin A reaches logic 1. In the *charge phase*, a small charge is pushed to the thumbtack and beyond using the circuit



Thumbtack selection (indicating "press" of thumbtack #1")

Action sequence specification (indicating 3 sample event delivery steps)

Screen cutting and focus area (focused on the "play" button of the RealPlayer™)

Figure 6: The Action Cutter interface

shown in Figure 4b: Pin A is configured to `Output_High` (the logical equivalent of connecting it to the positive supply voltage, in this case +5V) and Pin B to high impedance input (effectively disconnecting it from the circuit). The amount of charge that accumulates at the thumbtack (labeled Pad in Figure 4) depends on whether or not a human is present. Next, in the *pump phase*, this charge is pulled back to the accumulating capacitor using the configuration in Figure 4c: Pin A is configured as a high impedance input and Pin B as `Output_Low` (effectively grounding it). This causes charge to flow from the thumbtack towards Pin B, and into Acc, the accumulating capacitor. During each repetition of this cycle, charge accumulates in Acc and its voltage rises. The number of charge/pump cycles necessary to raise the voltage across this capacitor above logic 1 minimum is inversely proportional to the capacitance of Pad and is thus directly related to whether or not a human is touching the thumbtack.

The firmware in the BOXES sensor board carries out this sensing algorithm in parallel for our eight sensing circuits. Using a 4 MHz clock rate (corresponding to one million instructions per second on the PIC 16F767 processor) the current firmware completes a charge/pump cycle for all eight sensors every 132 μsec and a complete capacitance sensing approximately once every 52 msec (or just short of 20 times per second). The firmware uses these raw sample values to maintain a running average of the current sensed value for each circuit and reports an event to its serial interface when a sufficiently large change from the running baseline is observed. When an event is reported a very short audio feedback chirp is also produced to help compensate for the lack of tactile feedback in these touch buttons. By working relative to a running average the sensor is able to adapt to different baseline capacitance values automatically. This is an important property both because of the "floating" ground inherent in this circuit, and because the use of tape over sense pads in some places but not others, as well as the vagaries of wire routing and other environmental conditions, can result in rather different absolute values at different times or different places.

The firmware for the sensor board is currently comprised of about 1300 lines of C code and consumes about 1.75K (14 bit) words of program memory (of 8K available) and 120 bytes of RAM (of 368 bytes available).

Connecting Inputs to Actions

Whenever a thumbtack or foil patch is touched by a person, the hardware sensor board sends an event notification to the *Action Cutter* subsystem, the PC-side software associated with the BOXES system. This subsystem is responsible for translating these events into actions that the designer would like the prototype to carry out in response.

To achieve the fluidity and flexibility needed for rapid prototyping, it is critical that the Cutter subsystem allow very rapid specification of actions. At the same time, it is important both to provide access to a very rich set of capabilities and to allow designers to create custom actions using familiar tools, when desired. In short, the system needs both a very low threshold for first use and a relatively high ceiling [17] in terms of range and scope (but not polish, robustness, or ability to support "production quality" results)

Rather than trying to accomplish this through a sophisticated, custom made programming infrastructure, we take the approach of reusing the functionality of other programs. This allows us to provide nearly instant access to the rich capabilities of commercial programs such as media players, web browsers, personal productivity tools, and many others. This approach can also support custom actions programmed in tools already familiar to designers such as Flash, Director or Visual Basic.

To avoid limiting the set of programs and programming environments our designers can make use of we have taken the approach of simulating interactive actions – each sensed physical action can be translated into a series of mouse and keyboard events which are injected into the

system as if a user had carried them out (this capability is implemented in a platform independent way using the Java Robot class [3]). This allows physical button presses to execute actions by means of on-screen button presses, menu selections or more complex sequences of actions.

The interface shown in Figure 6 is used by the designer to specify the translation between sensed physical actions and on-screen actions. This interface has three main sections: thumbtack selection, action specification, and screen cutting selection and focus.

Thumbtack selection

On the left, the designer can select a thumbtack number, and indicate whether an action or actions should be taken in response to *press* or *release* (or both) of the corresponding thumbtack or foil button. To promote fluidity, all buttons are active by default – touching them causes the corresponding specification label to flash (if visible) and any associated actions to be carried out. This helps to encourage interactive testing as actions are specified. The user also has the option of temporarily disabling action execution to avoid accidental firings (for example, while thumbtacks are being repositioned).

Screen cutting selection and focus

The heart of the action specification system is the *screen cutting* facility appearing on the right side of the interface. The designer creates a cutting (similar to those supported by the Snip [9] and WinCuts [20] systems) by selecting an area of the desktop. This cutting serves as a visual proxy for the corresponding screen area, and is updated regularly to track the appearance of that area. Within each cutting, a crosshair can be positioned to indicate a focal position for event delivery. Simulated events are delivered as if the mouse were pointing at the focal position.

Each step in an action sequence can make use of a different cutting. This allows actions to be carried out across several different underling programs if desired. For example, an audio prompt could be played with a sound player in one step and a custom Flash interface used for the next. For many prototypes, however, we have observed designers reusing (copies of) a single cutting over the interesting part of a program, but with the crosshair repositioned as needed. To facilitate this common use pattern, the system by default supplies a copy of the most recently used cutting each time a new action step is introduced.

Action specification

In the middle portion of the interface the designer specifies the particular series of mouse or keyboard events that are associated with the thumbtack press or release specified on the left. In our experience, most functionality can be specified with one or more mouse button clicks over the proper button(s) on a screen cutting. However, keystrokes are also used, and are particularly valuable for controlling menus since in most programs all standard menus are accessible through the keyboard (e.g., using the Alt and arrow keys). *Action Cutter* provides designers with a small keystroke recorder/editor for entering sequences of keystrokes.

Note that actions carried out by the Action Cutter are *stateless*, in that they always perform the same action regardless of the past history of interaction. This means for example, that toggle switches which invoke different actions on every other press are not directly supported within the cutter itself. However, this kind of interaction can easily be supported using custom interactive programs provided by the designer (in the prototyping platform they are most comfortable with). Based on our needs finding work, we believe this tradeoff of power for simplicity is justified. However, one of our preliminary test subjects did raise an issue with this, so additional testing will be required to determine whether this supposition holds, or instead a more complex action expression mechanism is justified.

DEVELOPMENT, VALIDATION AND USE

Range of support

Our system is intended to support early-stage iterative design. As such, it is important to consider both the range of applications that can be prototyped, and the range of processes for iterative design that can be supported or augmented by our tool.

In many regards, the range of applications supported by our system is in the designer's hands. Interactively speaking, our system can be used to explore a range of interactive systems. While a significant range of interactions can be simulated with the BOXES system, our hardware is best suited to a range of physical forms typical of today's mobile technologies, such as mobile phones, remote controls, MP3 players, mobile tour guides (*e.g.* [1]) and assistive technology for elders (*e.g.* [19]), which do not make use of large displays as a central component.

In terms of process, our approach also enables a range of user feedback. We can learn about everything from what weight and form factor is effective to the ergonomics of the physical layout of buttons to the usability of interaction choices. Many applications can be completely implemented (in rough form) using our system. More blue-sky systems, such as those that depend on hundreds of users, context, or highly accurate machine interpretations of human speech or actions can be supported via a Wizard-of-Oz protocol.

User feedback

We have conducted two rounds of tests supporting the iterative design of BOXES. The first, very early round, involved a single designer who wished to explore the design space for a mobile tool enabling people who are deaf to ask "what just happened" (see [16] for more details). At this point in the project, the ability to program was required to attach functionality to the buttons.

The designer who used this version of BOXES was able to program. However, she was more comfortable with design environments such as Flash, and in this case she decided to implement all actions *via* Wizard-of-Oz. She encountered several problems because of limitations (and bugs) in the early sensor implementation. The designer was creating a device that was meant to be carried around for most of each day. For this reason, she was interested in exploring button

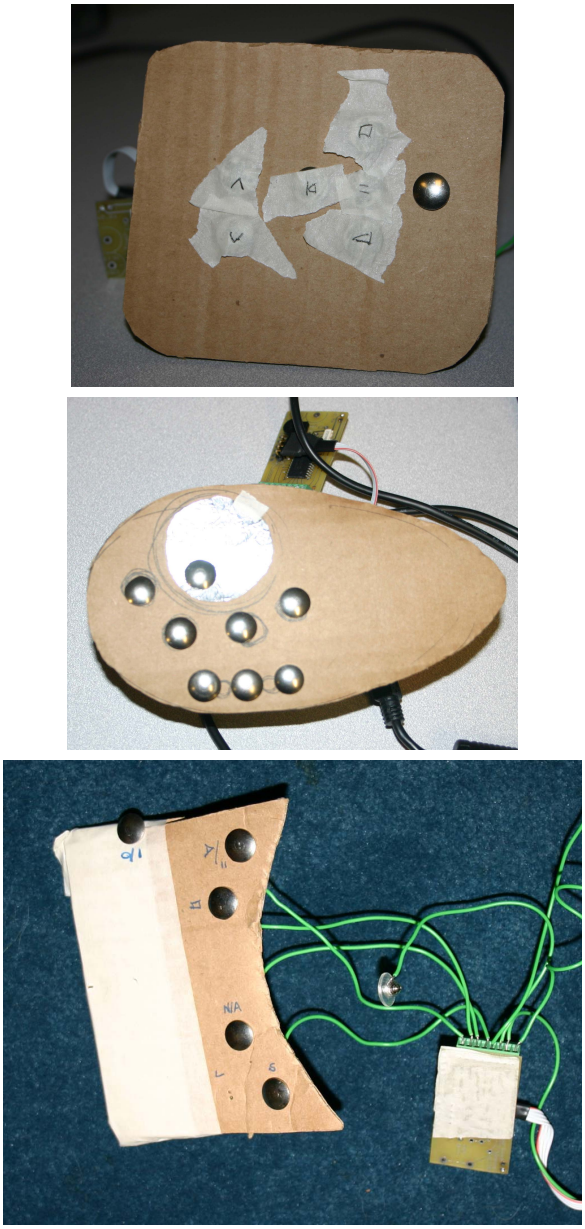


Figure 7: Three designs created during our second round of tests.

placement on the human body. Her designs included in the user's pocket and a "bracelet-type band." She tended to use the buttons without any cardboard to simulate a very small device. She did attach labels using paper and tape. However, the lack of cardboard made her designs a little too fragile (wires could break off easily, for example). Also, the presence of long wires between board and button may have contributed to implementation issues.

After the initial test, we re-implemented parts of the BOXES hardware. We also added the *Action Cutter* tool to better support the implementation styles of designers. We then asked three more designers to create interfaces using BOXES. Unlike the previous test, which was conducted in the field, these tests were scripted and were conducted in the lab. Two of the designers were comfortable programming in Flash, while the third worked almost exclusively in PhotoShop and on paper and was uncomfortable with any sort of programming. This test helped us to evaluate the comfort level of designers with BOXES, and to see what sort of process our tool supported.

Each designer was shown a demo in which the experimenter created a simple calculator that could add or subtract two numbers. The experimenter then opened a media player (RealOne Player) and asked the designer to create a portable MP3 player using it as the back end. In addition to the audio player, designers were given cardboard, scissors, tinfoil, thumbtacks, and our hardware and software. Figure 7 shows final designs that were created.

All of the designers were very comfortable with physical prototyping. All designers clearly iterated on both physical layout and interaction choices as they went through the design process. However, interestingly, two chose to test their prototypes interactively only after all of the thumbtack buttons had been placed and assigned actions, while the other tested his prototype interactively as he specified the action associated with each button.

The designer who tested along the way began by sketching on the cardboard, and then chose a highly iterative process in which placement, interaction, and testing were all modified with the addition of each thumbtack. He ended by cutting the cardboard to a physical shape, and testing the physical layout of the buttons one last time. The other designers planned ahead more.

One began by attaching all of the thumbtacks his functionality called for to the cardboard. He then created all of the interactive mappings in sequence, iterating on his choice of mapping and button location as he went. After all of this he conducted interactive tests. He too ended by cutting the cardboard to a physical shape. He commented "I think I'll cut things at the end … because then I can lay out the buttons and cut it to shape later."

The other started with the physical shape. She explored and designed the shape in multiple dimensions, taping pieces of cardboard together to create a form. Unconstrained by a programmatic understanding of the particular limitations of Action Cutter, she made use of a mode button so that she could overload multiple functions on other buttons. She then placed her buttons and kept careful track of the order in which she connected them on a separate piece of paper. She then started drawing labels indicating button functionality on the cardboard neighboring each button. When asked if she wanted to use masking tape to put labels directly on the buttons she responded "yes definitely but since it seems like such a low fidelity prototype ... these are more annotations than instructions"

In all cases we observed that the designers quickly became fluent with action specification using the Action Cutter interface. Given a program providing the right functionality, they were typically able to specify action responses in a few seconds. The designer with no programming experience required occasional help: we had to explain that

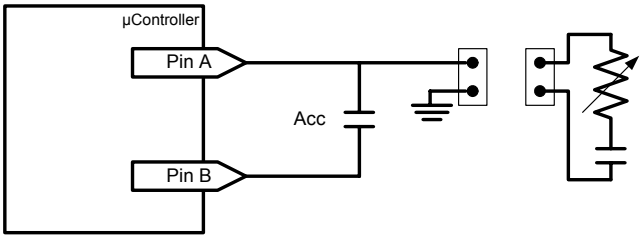


Figure 8: Circuit for measuring a variable resistor by measuring RC decay time.

modes were not possible, that the numbering scheme started at 0 and not 1, how to assign two actions to the same button, and when to move the crosshair. Also, we had to suggest that she simulate a volume slider by associating discrete volumes with a series of buttons placed on her device. With all three designers, we found that after the first few specifications, designers were able place their focus on the object being designed, rather than on our tool. Based on what we observed, we feel confident that the fluidity of our system is high, and is appropriate for early rapid explorations.

All three designers felt that the system could be useful. One commented "industrial designers want to reduce the number of buttons (*e.g.* remote control)… just with a mock-up you cannot imagine how it would work, so this kind of tool would be helpful…"

However, the designers pointed out some limitations. The lack of a physical screen, and the inability to handle modes without writing custom software (due to stateless actions as discussed above) were issues. Another issue that arose was in fidelity of form achievable with the early stage materials we provided. In the words of one designer: "I would like to have some sort of form I could [create] actual 3D shapes [with], like blue foam." Based on this, part of our future work will explore additional mechanisms better suited to materials such as blue foam (see below), and the tradeoffs associated with stateless actions will need to be tested more thoroughly.

CONCLUSIONS AND FUTURE WORK

In conclusion, we have presented the design of BOXES, a system for prototyping interactive physical forms at the earliest stages of design. We used a user-centered process to create BOXES, including early ethnography and iterative design based on tests with designers.

Using BOXES, interactive interfaces can be created from everyday objects such as cardboard, masking tape, thumbtacks, and foil. Button behavior can easily be linked to existing applications or custom-designed interfaces using the *Action Cutter* desktop tool. By leveraging existing applications to implement interactivity, we have created a system with an extremely low threshold (because it can interoperate with familiar, existing tools) yet, along the dimensions of interest for prototyping, presents a comparatively high ceiling when needed (because it can also work with potentially complex custom programs created by designers in the tools of their choice).

In the future, we will be exploring additional physical forms for touch sensitive areas, looking in particular at new attachment mechanisms which work particularly well with additional form prototyping materials. For example, to better support blue foam prototyping, we are exploring the use of foil patches attached with specially prepared stiff wire that can be pushed through the foam.

In addition, we are currently examining additional capabilities for handling other forms of inputs with the same sensing hardware (but with firmware additions, and possible connector changes). For example, the circuit in Figure 8 can be used to measure a variable resistance (as might be provided by a potentiometer, light sensor, force sensitive resistor, or thermistor). This measurement is performed by charging the (known, fixed) capacitor to the positive supply voltage, then measuring the time necessary for it to decay below the minimum voltage for logic 1. Note that replacing the variable resistor with a physical switch can also be handled in a uniform fashion with this circuit – as it can simply be treated as a special case where resistance is either very low or very high. The firmware to support this sensing is fairly simple. An open challenge, however, will be to find a way to mix and match sensing modalities while maintaining the zero configuration ("*it just works*") property of the original system, hence maintaining its full fluidity.

ACKNOWLEDGMENTS

This work was funded in part Intel and IBM, as well as the National Science Foundation under grants IIS-0121560, IIS-032535, IIS-0205644 and IIS-0511895.

REFERENCES

1. G. D. Abowd, C. G. Atkison, J. Hong, S. Long, R. Kooper, and M. Pinkerton, Cyberguide: A mobile context-aware tour guide. *Wireless Networks,* 1997. 3(5):421-433.

2. D. Avrahami and S. E. Hudson. Forming interactivity: A tool for rapid prototyping of physical interactive products. in *Proceedings of DIS'02: Designing Interactive Systems: Processes, Practices, Methods, & Techniques*. 2002, pp. 141-146.

3. R. G. Baldwin, Introduction to the Robot class in java, in *www.developer.com*. 2003.

4. R. Ballagas, M. Ringel, M. Stone, and J. Borchers, iStuff: A physical user interface toolkit for ubiquitous computing environments, in *Proceedings of ACM CHI 2003 conference on human factors in computing systems*. 2003. pp. 537-544.

5. P. H. Dietz and W. S. Yerazunis, Real-time audio buffering for telephone applications, in *Proceedings of the ACM symposium on user interface software and technology*. 2001. pp. 193-194.

6. S. Greenberg and M. Boyle, Customizable physical interfaces for interacting with conventional applications, in *Proceedings of the ACM symposium on user interface software and technology*. 2002. pp. 31-40.

7. S. Greenberg and C. Fitchett, Phidgets: Easy development of physical interfaces through physical widgets, in *Proceedings of the ACM symposium on user interface software and technology*. 2001. pp. 209-218.

8. Hinckley, K. and Sinclair, M. 1999. Touch-sensing input devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1999. pp. 223-230.

9. D. R. Hutchings and J. Stasko. Shrinking window operations for expanding display space. in *Proceedings of the working conference on Advanced Visual Interfaces*. 2004. Gallipoli, Italy: ACM Press, pp. 350-353.

10. S. R. Klemmer, J. Li, J. Lin, and J. A. Landay, Papiermache: Toolkit support for tangible input, in *Proceedings of ACM CHI 2004 conference on human factors in computing systems*. 2004. pp. 399-406.

11. J. A. Landay, Interactive sketching for the early stages of user interface design. PhD Thesis, 1996, School of Computer Science, Carnegie Mellon University.

12. J. Lee, D. Avrahami, S. Hudson, J. Forlizzi, P. Dietz, and D. Leigh. The Calder toolkit: Wired and wireless components for rapidly prototyping interactive devices. in *Proceedings of the ACM Symposium on Designing Interactive Systems (DIS'04)*. 2004, pp. 141-146.

13. J. Lifton., Seetharam, D., Broxton, M., and Paradiso, J. Pushpin Computing System Overview: A Platform for Distributed, Embedded, Ubiquitous Sensor Networks. in *Proceedings of the First international Conference on Pervasive Computin*g. 2002, pp. 139-151.

14. Y. Li, J. I. Hong, and J. A. Landay. Topiary: A tool for prototyping location-enhanced applications. in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'04)*. 2004. Santa Fe, NM: ACM Press, pp. 217-226.

15. L. Liu and P. Khooshabeh, Paper or interactive? A study of prototyping techniques for ubiquitous computing environments, in *Proceedings of ACM CHI 2003 conference on human factors in computing systems.* 2003. pp. 1030-1031.

16. T. Matthews, S. Carter, C. Pai, J. Fong, and J. Mankoff. Scribe4me: Evaluating a mobile sound transcription tool for the deaf. to appear in *Proceedings of Ubicomp 2006*.

17. B. Myers, S. E. Hudson and R. Pausch, Past, present, and future of user interface software tools. *TOCHI,* 2000. 7(1):3-28.

18. M. Rettig, Prototyping for tiny fingers. *Communications of the ACM*, April, 1994. 37(4):21-27.

19. T. Starner, J. Auxier, D. Ashbrook, and M. Gandy. The gesture pendant: A self-illuminating, wearable, infrared computer vision system for home automation control and medical monitoring. in *ISWC 2000.* 2000, pp. 87-94.

20. D. S. Tan, B. Meyers, and M. Czerwinski, Wincuts: Manipulating arbitrary window regions for more effective use of screen space, in *Proceedings of ACM CHI 2004 conference on human factors in computing systems*. 2004. pp. 1525-1528.

21. K. Van Laerhoven, A. Schmidt and H.-W. Gellersen. Pin&Play: Networking Objects through Pins. in *Proceedings of Ubicomp 2002*, Lecture Notes in Computer Science; Vol. 2498, Göteborg, Sweden. Springer Verlag, September 2002, pp.219 - 229.

22. Wrensch, T., Blauvelt, G., and Eisenberg, M. 2000. The rototack: designing a computationally-enhanced craft item. In *Proceedings of DARE 2000 on Designing Augmented Reality Environments* (Elsinore, Denmark). 2000. pp. 93-101.