ABSTRACT

Prototyping tomorrow’s computing technologies—which transcend the desktop PC—poses a unique challenge for designers. Prototyping tools should offer a high-level approach that generates working results quickly. d.tools is a design tool for prototyping the bits and the atoms of physical user interfaces in concert. It enables designers without specialized engineering or programming knowledge to quickly build functional interactive prototypes. d.tools offers a visual authoring environment that allows for drag-and-drop specification of interaction models for tangible user interfaces in a matter of minutes.

Keywords: Interaction design, prototyping, tangible UIs

INTRODUCTION

We have entered the age of ubiquitous computing: processor enabled products, each with its own user interface, have pervaded everyday life. While interfaces on the desktop have ossified around the WIMP paradigm, interfaces for mobile, ambient, and tangible devices are still evolving through experimentation. Designers of desktop computing applications can rely on a vast library of development aids, but tools that support the design of novel ubiquitous computing devices are just emerging. Current designers of information appliances have to possess expert knowledge in a number of specialized areas, such as programming, embedded microcontrollers, and circuit design in order to build usable prototypes (see FIGURE 1). These requirements make the current development process solution-information driven—much time and resources are spent on determining the right implementation—rather than need-information driven, where innovation is directed by the desired user functionality.

Von Hippel [6] highlights the importance of appropriate high-level toolkits to separate need-intensive design tasks from solution-intensive implementation tasks. d.tools empowers designers to perform this need-based innovation. We envision a world where end users can design, program and fabricate their own devices. As a first step in this direction, we are developing a toolkit for product and interaction designers. This user group likely has some knowledge about fabrication, content creation, and interaction design, but does not necessarily possess the engineering expertise to build and program all involved electronic components.

TOOLS FOR PROTOTYPEING

Prototyping is the iterative process of building approximations to reflect on design decisions. It is a central activity of the product design process. Our fieldwork with design professionals and students taught us that in order to be useful, prototyping tools must provide a fast, low-overhead path to generate working results. d.tools achieves this low-threshold requirement by initially insulating users from circuit design and programming APIs through plug-and-play hardware and a visual authoring interface.

In d.tools, designers create interaction prototypes using our PC-based visual authoring environment (see FIGURE 2). The PC can simulate an embedded controller and execute these interaction models, communicating with the prototype device through a tether (see FIGURE 3). States in the editor define device outputs; state transitions are triggered by physical inputs. At any time, designers can switch between authoring and run modes. When authoring, designers visually lay out the device’s interaction model. In run mode, the current interaction model is executed as a finite state automaton on the PC, processing sensor inputs and generating output signals on the physical prototype. The integration of design and execution into the same tool allows for fast, fluid iterations of the think–design–test cycle [4].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

UIST’05, October 23–27, 2005, Seattle, Washington, USA.
Copyright ACM 1-59593-023-X/05/0010...$5.00.
CURRENT IMPLEMENTATION
Our current architecture uses a Phidgets InterfaceKit [2] to provide a set of continuous (analog voltage-varying) and discrete (digital) inputs, along with a compatible set of input devices such as sliders, knobs and buttons. For graphical output to small screens, d.tools provides a serial color LCD controller (EarthLcd ezLcd01). Both Phidgets and LCD controller boards are housed in an I/O box that is tethered by ribbon cable to the prototype on one side and connected to a host PC through USB and RS232 on the other side. On the PC, we have implemented the visual state editor using Macromedia Flash as our rendering engine. A TCP socket server written in C# connects the Flash GUI editor to external hardware. It marshals physical input events as XML messages and unmarshals Flash XML commands into API calls for Phidgets and our own LCD library.

SCENARIO
The following sample scenario envisions a product designer assigned the task of developing a handheld media player. She first builds a shape prototype and embeds interactions elements such as buttons, sliders and an LCD screen. She then connects the devices to the PC through our I/O box. On the PC, she arranges icons depicting the recognized physical I/O components into a virtual representation of the physical device. This iconic representation affords rapid recognition of device elements. The designer then builds an interaction statechart by dragging graphics, sounds or other output specifications onto states. New states are automatically created by clicking and dragging input symbols in the state display. After authoring a few key interaction paths, the designer can hit the run button and hand the now-functional prototype to her office mate for on-the-spot user feedback.

RELATED WORK
Recent toolkits [1-3, 5] have opened up physical interface development to software programmers. However, utilizing these tools still requires programming experience. Commercial applications such as Max/MSP, a music synthesis and control platform, have shown that domain experts (musicians in this case) can successfully author complex systems visually without having to be proficient coders. d.tools seeks to combine the advantages of these two application classes.

WORK IN PROGRESS: EXTENSIBILITY
Having lowered the entry threshold for physical interaction prototyping, we turn our attention to raising the complexity ceiling of devices that can be prototyped. Toward this end, we are developing a new d.tools interface in Java as an Eclipse IDE plug-in. This approach exposes textual programming to users that wish to transcend the possibilities of a purely visual editor.

A hallmark of desktop GUI toolkits is their extensibility: expert users can create their own widgets. Inspired by this, we are simultaneously researching a high-flexibility hardware architecture. We are moving to Atmel AVR microcontrollers and adopting the industry standard I2C bus for component-microcontroller communication. This will allow users to add any device adhering to the I2C protocol into the d.tools architecture. Using a programmable microcontroller also allows us to run the interaction model on the embedded processor itself and remove the PC tether. The prototype can then be used for field testing outside the confines of a usability lab.

ACKNOWLEDGMENTS
We thank Michael Bernstein for his Flash assistance.

REFERENCES