d.tools: Integrated Prototyping for Physical Interaction Design

Björn Hartmann, Scott R. Klemmer, and Michael Bernstein
Stanford University HCI Group
Computer Science Department, Stanford, CA 94305-9035, USA
{bjoern, srk, mbernst}@cs.stanford.edu

ABSTRACT
Designers tasked with imagining future information appliances currently employ separate tools for rapidly prototyping the form (the atoms) and the interaction model (the bits) because integrated prototyping of bits and atoms requires resources and knowledge outside the reach of design generalists. Based on interviews with product designers, we created d.tools, a system enabling non-programmers to prototype the bits and the atoms of physical user interfaces in concert. d.tools lowers the threshold to prototyping functional physical interfaces through plug-and-play hardware that is closely coupled with a visual authoring environment. We evaluated the d.tools use threshold through a first-use study with thirteen participants; the study showed that the tool is accessible and encourages reflective design practice. We tested the d.tools range of design support by recreating existing research and commercial devices; this demonstrated that the visual language was sufficiently expressive for existing and emerging real-world designs.

Author Keywords
Toolkits, information appliances, design tools, prototyping, hardware-software integration, physical user interfaces

ACM Classification Keywords
H.5.2. [Information Interfaces]: User Interfaces-input devices and strategies; interaction styles; prototyping; user-centered design. D.2.2 [Software Engineering]: Design Tools and Techniques—State diagrams; user interfaces.

INTRODUCTION
Information appliances—small portable devices such as mobile phones, digital cameras, and music players—are growing quickly in number and diversity. To arrive at usable interface designs, product designers commonly build a series of prototypes—approximations of a product along some dimensions of interest. Prototypes are the pivotal medium that structures innovation, collaboration, and creativity in the most successful design studios [19]. These prototypes play important roles for four distinct constituencies. First, designers create prototypes for their own benefit; visually and physically representing ideas externalizes cognition and provides the designer with *backtalk* [37]—surprising, unexpected discoveries that uncover problems or generate suggestions for new designs. Second, prototypes provide a locus of communication for the entire design team; through prototypes, the tacit knowledge of individuals is rendered visible to the team. Third, prototypes are integral to user-centric development by providing artifacts that can be used for user feedback and usability testing. Finally, prototypes are also important sales tools in client relationships—many product designers live by the principle "never enter a client meeting without a prototype in hand."

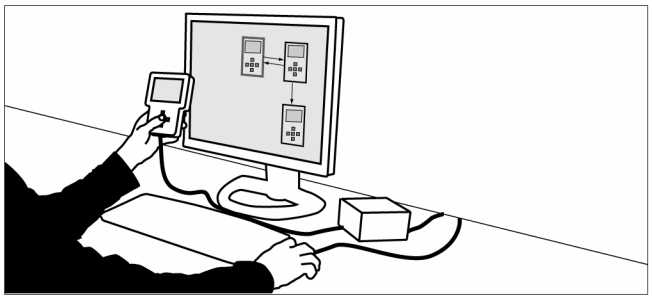

Figure 1. d.tools enables designers to rapidly prototype the circuits and code of information appliances using pictures and parts.

Through much of the design process, designers today create two separate sets of prototypes: "looks-like" prototypes that show only the form of a device (the atoms), and "works-like" prototypes that use a computer display to demonstrate its interaction (the bits). The time and expertise requirements for creating comprehensive prototypes that tie form and function together prohibit their use until late in development. At that time, monetary constraints and resource commitments prohibit fundamental design changes [42]. However, only integrated functional prototypes can uncover the interdependence between bits and atoms that characterizes the final user experience. To enable designers to prototype bits and atoms in concert, we have developed d.tools, a system that introduces integrated interactions [24] to enable rapid prototyping of information appliances (see Figure 1).

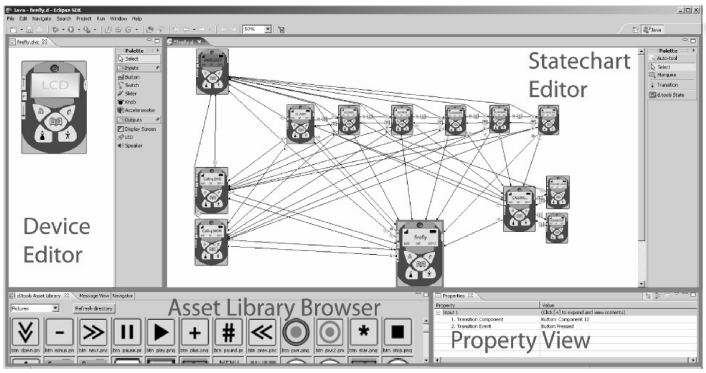

Figure 2. The d.tools visual authoring environment integrates software and hardware prototyping. This screen shows a statechart for a mobile phone prototype.

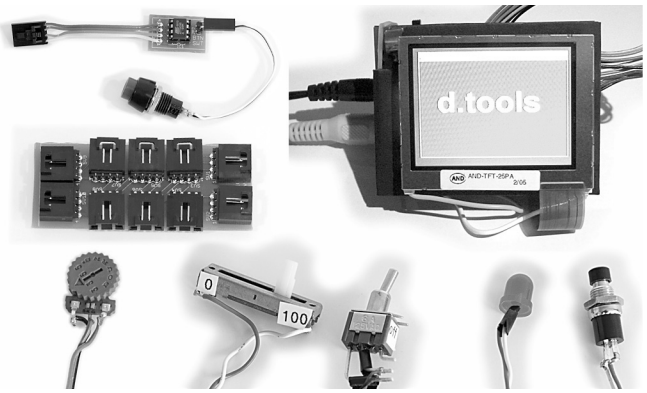

Figure 3. *Top left:* Smart components have their own RISC microcontroller; they are connected to plug boards to communicate over an I2C bus. *Top right:* d.tools supports output to small-form-factor LCDs. *Bottom:* Examples of supported plug-and-play hardware.

Design Tools

Myers *et al.* introduced the terms *threshold* and *ceiling* to describe use properties of a tool: the threshold is the difficulty of learning and using a system, while the ceiling captures the complexity of what can be built using the system [30]. Today, programming in general purpose languages and electronic circuit design are still the prevalent means of creating functional prototypes of physical user interfaces; the high threshold for these tools has been a gating factor to designers, and the time commitment of these tools makes them infeasible for rapid iterative exploration. These difficulties of high-threshold tools echo the experiences of developing graphical user interfaces (GUIs) twenty years ago; today nearly all GUIs are built with the assistance of user interface design tools [30].

Recent research and commercial systems have demonstrated the power of providing software abstractions to physical devices (*e.g.*, [9, 15, 21, 38]). However, the expertise threshold and time investment required make them inappropriate for designers, particularly at the early stages. The contribution of the d.tools research described in this paper is a system that lowers the threshold for functional prototyping and provides a sufficiently high ceiling to design useful systems.

Prototyping with d.tools

d.tools supports design thinking rather than implementation tinkering. With d.tools, designers place physical controllers (*e.g.*, buttons, sliders), sensors (*e.g.*, accelerometers, compasses), and output devices (*e.g.*, LEDs, LCD screens, and speakers) directly onto form prototypes, and author behavior visually in our software workbench (see Figure 2 and 3). d.tools employs a PC as a proxy for embedded processors so designers can focus on user experience-related tasks rather than implementation-related details. The d.tools library includes an extensible set of smart components that cover a wide range of input and output technologies.

Designers create interaction prototypes in d.tools using a PC-based visual authoring environment, inspired by the statecharts visual language [16]. States in the editor specify device outputs (see Figure 2); state transitions are triggered by physical inputs. Users graphically arrange icons of recognized physical I/O components into a virtual representation of the physical device. This iconic representation affords rapid matching of software widgets with physical I/O components. d.tools dynamically detects the presence and capabilities of attached hardware components, enabling the software editor to be couple to the hardware configuration.

UNDERSTANDING CURRENT DESIGN PRACTICES

To create a design tool offering these benefits in a manner felicitous with current design practices, we conducted structured interviews with designers and surveyed the interaction requirements of existing devices.

Interviews

We conducted individual and group interviews with eleven designers and managers at three product design consultancies in the San Francisco Bay Area. To understand how design students could benefit from prototyping tools that help them focus on the design aspects of their education, we interviewed three product design master's students.

Professional design companies have access to resources and expertise to create integrated functional prototypes that demonstrate interaction in a high-fidelity form factor (identified by one interviewee as "Comdex models" to convey their importance for trade shows). Pering has described the use of such comprehensive prototypes for user experience testing of PDAs [35]. However, these solutions are generally expensive one-offs that that can not be reused or easily modified. Their cost limits deployment to later stages of the design process and to large projects.

For earlier stages, designers reported using lower-fidelity prototyping solutions such as using PowerPoint and Excel spreadsheets or Photoshop layers to express UI control flow. One design manager noted that user interface evolution was harder and slower than iterating hardware designs. One reason is that many design consultancies have more mechanical engineers and design generalists than programmers

or electrical engineers on staff: technical work has to be queued or outsourced. d.tools addresses these difficulties with early-stage design by giving design generalists a ready-to-hand tool for building functional prototypes at their desk or workstation.

Design students' access to resources is even more constrained; two of the three interviewees reported giving up on trying to prototype the electronics of their projects. Students expressed a need for narrative or storyboard-based design that allows them to capture only key interaction paths without having to develop a comprehensive interface model up front. As with our professional interviewees, students fell back on lower-fidelity mock ups such as sliding transparencies in and out of acrylic blocks to simulate screen output. They expressed dissatisfaction with those techniques because they failed to suspend disbelief of their test users. In response, we designed d.tools focus on concrete interaction sequences and to support high-fidelity, low-latency output comparable to that of finished products.

Survey of existing information appliances

To inform the architecture and library requirements for d.tools, we tallied the use of input and output components by 24 devices in three categories: mobile consumer electronics, stationary control interfaces, and novel systems that introduce interactions not yet common in the mainstream.

Mobile appliances such as portable media players, cell phones and digital cameras predominantly use a large number of buttons (~5 to 71), and a small number of switches (~1 to 3) as inputs. With these digital controls, UI state is not apparent in the physical state of the input components and is mostly communicated through status LEDs and color LCD screens. Many devices feature microphone input and stereo sound output; capacitive and other sensors are gaining popularity in the commercial avant-garde (such as the iPod wheel) but are not yet commonplace.

Stationary interfaces include appliances from musical control surfaces to home automation/room control panels. They often rely on continuous dedicated controls such as rotary knobs (potentiometers or encoders) and sliders as inputs which offer a physical indication of their state. LED and text LCD output is common; graphical LCDs are less common, but gaining popularity.

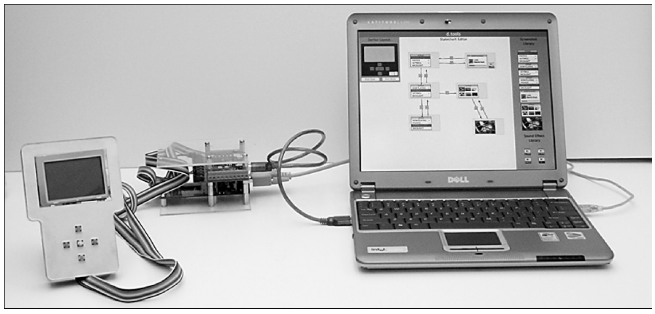

Figure 4. We brought this Flash + Phidgets prototype to design companies to elicit feedback during our fieldwork.

Entertainment and research devices form the vanguard of physical interactions, and helped us understand emerging trends. Devices such as the Nintendo Revolution game controller, experimental musical controllers, and research systems such as the Sensing PDA [18] and Tablet Whacking [17] make use of a broader range of sensors: orientation, acceleration data, pressure. Non-graphical output via solenoids or vibrating motors (*e.g.,* [40]) is currently rare, even in research, but is an important area for future growth.

These survey results offer important design guidelines for d.tools. The d.tools architecture should support both discrete and continuous input, and graphical and sound output. The architecture should not be a closed system; it should enable fluid integration of emerging sensor technologies as designers adopt them.

D.TOOLS ARCHITECTURE

d.tools was developed using the same iterative design process that our target audience employs. We summarize our exploratory work and then describe the full implementation.

Prototyping a Prototyping Tool

An integrated prototype requires interaction, physical input, and output. Our formative prototype employed Macromedia Flash [1] for the interaction, a Phidgets interface kit [15] for the input, and a RS232-controlled color LCD screen (earthLCD ezLCD001) for display. We designed visual interaction techniques for authoring the UI of one specific information appliance: a media player, for which we also built a physical device out of layered sheets of laser-cut ¼" acrylic (see Figure 4). Flash affords rapid interaction development and Phidgets provided a complete C# API for sensing physical input. A TCP socket server connected the Flash GUI editor to the LCD screen and Phidgets hardware by marshaling physical input events as XML messages and unmarshaling Flash XML commands into API calls for Phidgets and our LCD library.

This prototype anchored our discussions with professional designers. Designers found the visual authoring environment, in which states iconically represent the physical device, immediately compelling. We learned that the fluidity of our design tool should also extend to hardware components. Interviewees pointed out the disconnect between fluid drag-and-drop interactions in software and the soldering, screwing, and software development required to integrate hardware into an application. Armed with this information, we implemented the complete d.tools system.

Software

The d.tools visual authoring environment is implemented in Java J2SE 5.0 as an Eclipse IDE plug-in using its Graphical Editing Framework (GEF). Eclipse furnishes a standard application framework with flexible handling of multiple editors, views, and wizards. The d.tools Eclipse plug-in comprises 9,850 lines of logical code (18,600 lines of physical code, which includes comments and white space). The d.tools interface comprises a device designer, a state-

chart designer, and associated views for specifying properties.

Device designer

In the *device editor*, designers author an iconic representation of the appliance they are prototyping: they create, arrange and resize input and output components, then control their look by dragging and dropping images from the *asset library* image browser onto the device outline or onto individual components. (While d.tools recognizes the device type, designers may wish to add a particular look to their visual prototype.) d.tools supports image transparency and can read graphics in JPEG, GIF, and PNG formats. The component library of the device editor currently comprises the following inputs: buttons, switches, sliders, knobs, accelerometers, and general voltage varying sensors; and the following outputs: LCD screens, speakers and LEDs. Input and output components are presented according to their affordances. For example, while buttons and switches are both one-bit controls, d.tools retains their distinct affordances in software (switches physically maintain their state, while buttons return to a default position after each press).

Statechart designer

Designers define their prototype's behavior by creating interaction graphs with the statechart designer (see Figure 2). *States* are graphical instances of the device design; they describe the content assigned to the *outputs* of the prototype at a particular point in the UI: screen image, sound, LED behavior. As in the device editor, content can be assigned to output components of a state by dragging and dropping items from the asset library onto a component.

Transitions represent the control flow of an application; they define rules for switching the currently active state in response to user input (hardware events). They are represented graphically as solid-line arrows connecting two states. Transitions have one or more input components and an input event associated with each such component. For example, a designer can author that a button press transitions from one state to a second. Designers can specify multiple options for a transition's input (Boolean OR); they can also specify synchronous operation of multiple inputs (Boolean AND). When a user operates a physical control, d.tools checks whether that input matches the condition(s) for any transitions outbound from the current state. If there is a match, d.tools updates the visual interface's current state and sends this new state's content (images, sounds, LED behaviors, *etc*.) to the device. These state transitions express the *control flow* portion of the interaction design.

d.tools also supports *data flow*: the continuous attributes of output elements. Designers specify this data flow logic through intra-state *bindings*, which create a link between an input and an output component within a single state. For example, while a speaker receives discrete events to play and pause a sound (specified through transitions), continuous control specifies the volume (specified through bindings). Graphically, bindings are shown as arrows with dashed lines. Input components act as binding sources and expose different signals, such as the continuous position of a slider, or the state of a switch. These signals can be bound to different targets exposed by output components.

To facilitate a seamless map between the statechart authoring interface and the physical prototype, both are always live. Selecting a state in the visual authoring environment sets the physical device's current state, and vice versa. Integrating these representations and providing fast, global control of application state has two important benefits. First, it provides a clear and consistent mental model to designers. Second, it facilitates the designer's ability to extemporaneously control interactions during walkthroughs with clients and prototype testing with end users. As a first step toward enabling designers to analyze user interaction data, time-stamped state transitions are recorded in a log file.

Tight and loose coupling of hardware and software

d.tools introduces plug-and-play functionality for input and output components through a *tight coupling* between hardware components and their software duals (see Figure 5). The software listens for incoming messages and sends out hardware state change events via serial (RS232) and UDP ports. When a physical component is plugged in, a corresponding virtual component is created in the device editor and propagated to the statechart. When a physical component is unplugged, the software dual is decoupled from the physical component and deactivated. d.tools indicates deactivation by drawing a red × over the visual component.

d.tools uses a two-phase delete mechanism because components are bound to designer-authored information (state content and transitions). Separating deactivation from removal enables designers to review the action and remap the interaction logic to an alternate control if desired. To encourage exploration, all user interface actions are implemented on a command stack that supports multiple levels of undo. After initiating a delete in software or removing a hardware component, designers can reassign that component's transitions and content to a different component by dragging and dropping a new component on top of an equivalent inactive component.

Designers can switch to a *loose coupling* mode if they prefer software-centric development, if they want to work with only a subset of the hardware in their design, or if they do not have access to the hardware toolkit. When loosely

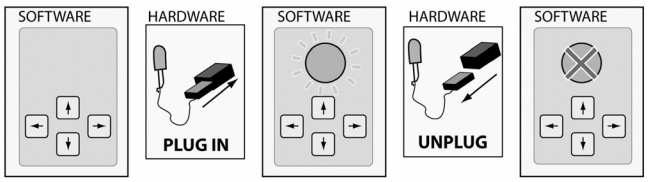

Figure 5. d.tools introduces an integrated approach: hardware components and their software duals are tightly coupled.

coupled, d.tools is agnostic to the presence of hardware components: it can receive data events from hardware and will follow matching transitions, but it will not add or deactivate software components based on connect/disconnect actions of physical components.

When a designer switches from loose to tight coupling, d.tools scans the hardware to ascertain which components specified in the hardware diagram currently have hardware duals attached. Visual components are *active* when their hardware dual is present. When the hardware is detached, the visual component is *inactive*.

Hardware

This section describes design rationale for and implementation of the d.tools hardware platform. d.tools provides plug-and-play integration of individual IO components by making each component smart (adding a dedicated small microcontroller) and networking the components on an I2C bus (see Figure 6). I2C offers a large base of existing compatible hardware, sufficient bandwidth characteristics for most interaction components and easy connection-point expansion through daisy-chaining. Output devices that require higher bandwidth are handled separately by PC peripherals.

A master controller board coordinates hardware-to-PC communication. This controller transforms hardware events into OpenSoundControl (OSC) messages. OSC is an open protocol developed specifically for real-time control of human performance systems.

Atmel microcontrollers are used to implement this architecture because of their low cost, high performance, and programmability in C. We used the open source WinAVR tool chain [6] and the commercial IAR AVR workbench [41] compilers. The d.tools microcontroller code comprises 1260 logical lines and 2100 physical lines.

I2C is a serial multi-drop bus architecture where one master and many slaves exchange messages through two data and clock lines. Up to 127 bus devices can be connected at a temporally-multiplexed data rate of 100 Kbps. We programmed an Atmel ATmega128-based Procyon Engineering AVRmini v3.1 board to serve as the communication liaison between the host PC and individual hardware components. It serves as I2C master and also sends OSC messages over an RS232 connection at 115 Kbps (or USB via VirtualCom-Port drivers). Individual interface components each have their own 8-pin microcontroller (Atmel ATtiny45) that runs an I2C slave program sending sensor data from an attached input to the master or setting the state of an attached output according to received commands (see Figures 3, 6). Components can be hot-plugged into the I2C bus via plug boards with polarized 4-pin plugs carrying I2C data and clock, +5 volts and ground. A polling loop over a database of known components allows the master controller to track presence and identity of hardware components. Sensors are polled at 250Hz to 1kHz, depending on the number of attached components. Most human input devices require less than 100Hz. The master generates additional OSC messages to notify the d.tools software of the configuration change. Occasionally, electrostatic discharge during plugging can interrupt operation of the I2C bus, which requires a manual 3 second hardware reboot; this problem can be avoided by introducing I2C bus buffer chips in the plug boards.

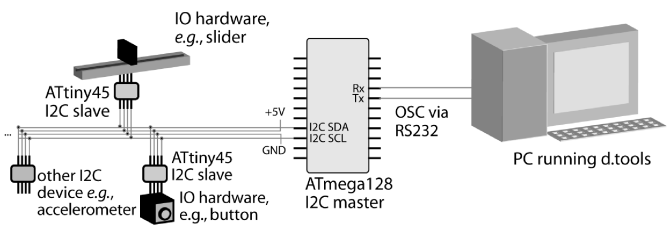

Figure 6. The d.tools plug-and-play hardware architecture.

d.tools distinguishes audio and video from lower-bandwidth components (buttons, sliders, LEDs, *etc.*). The modern PC A/V subsystem provides plug-and-play support for audio and video; for these components d.tools simply uses the existing infrastructure. For graphics display on the small screens commonly found in information appliances, d.tools includes an LCD display which can be connected to a PC graphics card with video output (Purdy AND-TFT-25PAKIT). This screen is controlled by a secondary video card connected to a video signal converter.

Modularity allows for substitution and extension

Tool support for physical computing operates at three levels: the wire protocol, the hardware-to-PC interface, and the software level. To facilitate extensibility by advanced users and the software development community at large, d.tools builds on existing open source APIs for all three layers; no existing systems provide a comparable level of extensibility. With closed architectures, designers are unable to create their desired prototype if the tool does not contain all required library elements. Our interviews and use of Phidgets in our university's interaction design course demonstrated that limited libraries are indeed a problem in current practice. d.tools employs I2C as its wire protocol; OSC for the hardware-to-PC interface, and Eclipse and Java as the software interface.

Each of these three APIs has a rich development community. For example, there are OSC implementations for the data flow languages Pd and Max/MSP [4]. The d.tools hardware connects these languages to the physical world. To show this modularity, we have used a d.tools slider to select pitch and a d.tools button to trigger sound in Pd. Similarly, other hardware toolkits can be used with d.tools by writing an OSC wrapper that communicates with the d.tools software. We have written a reference wrapper to connect a Phidgets InterfaceKit to d.tools. The Phidgets API affords detection of connection status for components such as servo motor controllers and component aggregators such as InterfaceKits, and LED banks. Phidgets does not provide this information for small, individual components such as

buttons and LEDs; thus, tight coupling behavior is not available.

The design choice of whether all components should be smart trades off larger size for plug-and-play functionality: d.tools components afford faster prototyping and a lower development threshold through tight coupling; however, plain electronics are smaller as they do not require a dedicated microcontroller per component. It is certainly also possible to mix smart and plain components in a particular device, or to initially prototype with smart components for their richer tool integration and later in the design process, as fidelity and form constraints increase, replace some of the smart components with the smaller plain components.

The d.tools hardware component library can be extended with other devices that conform to the I2C standard. In addition to our own ATtiny controlled components, we have successfully added I2C chips from other manufacturers into our system: a Phillips PCF8591 I2C A2D converter for voltage-varying inputs such as potentiometer knobs, and a Procyon Engineering ADXL accelerometer board, which uses an ADS7828 I2C A2D converter. Currently, adding such new components requires programming expertise; source code for both microcontroller and Eclipse environments is freely available to users.

Performance

Performance matters in three distinct areas of the d.tools system: designer interaction with the software workbench; plugging and unplugging of hardware in tightly coupled mode; and user interaction with a physical prototype.

Interaction with the d.tools software workbench is responsive on the 3.0 GHz PCs used in our evaluation. Noticeable lags only occur when operations such as device layout updates and state dragging involve repainting many graphic elements in complex statecharts. These delays are incidental rather than intrinsic to our architecture and are a result of using the GEF's suboptimal graphics engine. Plugging and un-plugging hardware is reflected in software representations after one to two seconds. This latency is sufficient to convey causality. Round-trip latency from the time a user generates a hardware event in an input component (*e.g.*, by pressing a button) until a signal is generated in an output component (*e.g.*, LED turns on) is on the order of 100ms, within the range of perceptual causality [12]. Refactoring code to separate hardware output from graphics updates may improve this performance.

EVALUATION

We present two evaluations of our system — a first-use study with thirteen participants, and the use of our toolkit to rebuild prototypes of three existing devices.

First-Use Study

We conducted a controlled study of d.tools in our laboratory to assess the ease of use and felicity of our tool for design prototyping. The study group comprised 13 participants whose skills matched those of the d.tools target audience: general design experience, but no required background in building physical user interfaces, electronics or programming. Most participants were students or alumni of design-related graduate programs at our university. Two undergraduates with design experience also participated. While all had some prior exposure to programming, only one participant reported to be fluent; none self-rated as experts. Participants' ages ranged from 20 years to 37 years; six were male, seven were female.

Pilot Study

Three participants served as pilot testers, which allowed us to iteratively refine our testing protocol. The pilots uncovered stumbling blocks not related to the technical part of our system: they alerted us to the importance of labeling otherwise identical hardware components to give users a way to differentiate them, of making available third party software tools for image creation during the study so participants could add their own graphics to their prototypes, and of providing online documentation and design patterns for participants to refer to. To support opportunistic design strategies, we also added a set of images of common navigation elements and symbols to the asset library.

Laboratory Evaluation

We began each two-hour session for the ten participants in the full study by demonstrating the d.tools software editor and the hardware components. We then gave the participants two narrowly-defined tasks and one open-ended design project. For the first task, participants were asked to complete a cell phone navigation menu that the experimenter had started during the demonstration. For the second task, participants were asked to build a functional physical prototype of a device with one button and one switch as inputs, and one LED and a speaker as outputs. Pressing the button should play a sound clip and toggling the switch should turn the LED on or off. The two components were to function independently of each other.

The third assignment was to begin prototyping a digital music player for children. Participants were given written guidelines such as "children prefer dedicated controls and like elements that move better than buttons," and were told to focus on the out-of-box experience. As the study allotted

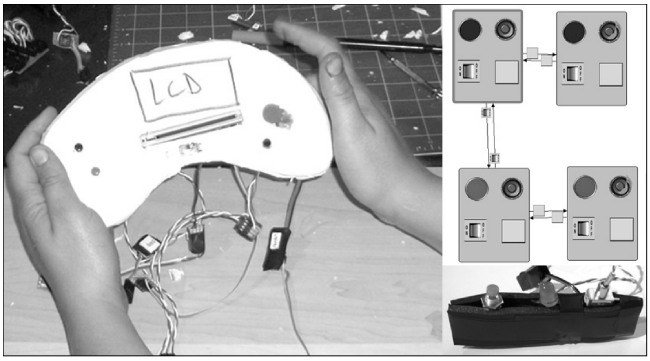

Figure 7. Prototypes created by our study participants. *Left*: a music player; *Right*: statechart and device for task 2

only 30 to 45 minutes for this part, participants were informed that they were not expected to produce a finished product. To sketch and build physical prototypes, we provided an 18" × 24" paper pad, sheets of foam core, pens, a selection of tools, glue and tape, and a label printer. As the final step of the study, participants were asked to complete a 26 question survey.

Study Results

All participants successfully completed both closed tasks, regardless of prior experience in user interface design or physical computing. Task one took a mean of 9 minutes while task two took a mean of 24 minutes to complete (see Figure 8).

Participants followed heterogeneous approaches: some started by exploring the ergonomics of different shapes to determine input component placement; others focused on requirements analysis on paper; yet others worked exclusively in software. d.tools was most frequently used for determining layout of interaction components in the device designer, and reasoning about the interaction model in the statechart designer. Two participants with prior physical computing experience built functional physical prototypes with navigation and sound playback in less than 30 minutes.

The success of a low threshold and tight coupling

Almost all users commented positively on the tight coupling of hardware components and their software counterparts, especially the automatic recognition of hardware connections. Authoring statecharts through link-and-create actions was immediately intuitive. Refining default behaviors through text properties and expressing functional independence in a statechart took longer; nevertheless, participants mastered these strategies by the end of the session.

After an initial period of learning the d.tools interface, participants spent much of their time with *design thinking*—reasoning about how *their* interface should behave from the user's point of view instead of wondering about how to implement a particular behavior. This was especially true for authoring UI navigation flows.

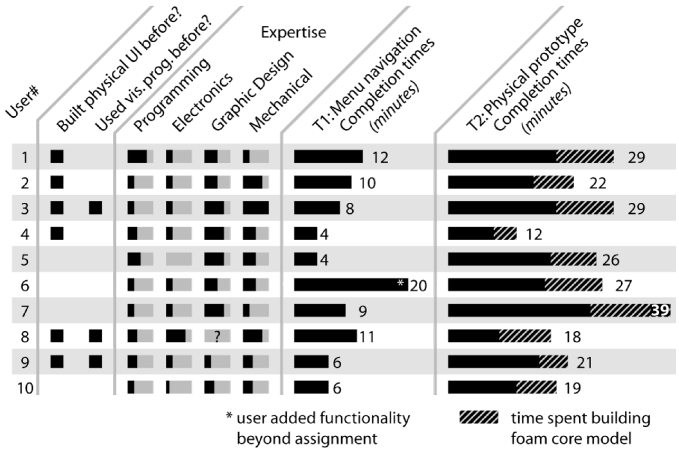

Figure 8: Task completion times, and prior experience and expertise of the study participants. Participants completed task 1 in an average of 9 minutes, and task 2 in an average of 24 minutes. These times demonstrate that prototyping with d.tools is fast enough to be appropriate for early-stage design.

The experimenter asked participants to hand over the devices built for task two—while observing this on-the-spot user test, many subjects expressed the wish to iterate on their designs and produced another version two to ten minutes later. This suggests the advantage of the rapid iteration cycles that d.tools enables.

Participants consistently gave d.tools high marks for enabling usability testing (μ=4.6 on a 1 to 5 Likert scale with neutral value 3; σ=0.70), shortening the time required to build a prototype (μ=4.3; σ=0.67) and helping to understand the user experience at design time (μ=4.25; σ=1.03).

Needs: software simulation, larger library, richer feedback

One significant shortcoming discovered through the study was the lack of software simulation of an interaction model: the evaluated version did not provide a mechanism for stepping though an interaction without attached hardware. A software simulation mode would complement the software-centric *loosely coupled* work flow.

Participants also found the d.tools hardware library too limited and noted that this constrained their designs. Some participants explicitly asked how difficult it would be to extend the hardware library. Some participants desired more robust hardware connectors: they were concerned about damaging or unhooking wires during plugging and unplugging and wished for a more "Lego-like" fit. We have yet to find a commercially available connector set that combines small size, polarized plugs, positive lock, and robustness to a high number of plug cycles. Users also wished for aggregate inputs that have become standard navigation elements for information appliances such as combined up-down buttons and five-way joysticks.

In the visual authoring interface, the study uncovered an inconsistency in our handling of click-and-drag actions between the device editor and the statechart editor. Furthermore, d.tools could benefit from better feedback on transitions and visualization of input component state in the statechart – some participants looked for transition information in the components of a state, instead of in the properties of transitions themselves.

Editing textual properties worked well for subjects who had some comfort level with programming, but was disliked by one product designer who described himself as a "visual person." Interaction techniques for graphically specifying properties would likely address this.

Rebuilding Existing Devices

To evaluate the current toolset's expressiveness, we recreated prototypes for three existing devices—an Apple iPod Shuffle music player, the back panel of a Casio EX-Z40 digital camera, and the Sensing PDA published in [18]. Figure 9 shows statechart diagrams of these prototypes. We

distilled the key functionality of each device and prototyped these key interaction paths.

Interactive physical prototypes have two scaling concerns: the complexity of the software model, and the physical size of the prototype. This rebuilding exercise demonstrated that d.tools diagrams of up to 50 states are visually understandable on a desktop display (1600 × 1200); this scale is sufficient for the primary interaction flows of current devices. Positioning and resizing affords effective visual clustering of subsections according to gestalt principles of proximity and similarity. However, increasing transition density makes maintaining and troubleshooting statecharts taxing. This limitation is shared by other visual authoring environments. The design of more complex interfaces would benefit from the ability to create reusable aggregations of functionality and implementing more of Harel's visual abstractions [16], especially hierarchical grouping.

Models were created at a scale of 1.5:1 since cabling and the small microcontrollers attached to each d.tools component require additional space inside the device enclosure. While miniaturization of our components can mitigate this issue to a certain extent, there still exists a principal mismatch between the physical scale appropriate for manipulation during prototyping and the scale of components used in computer-aided manufacturing.

Figure 9 shows the breakdown of prototyping times into graphic design time, d.tools design time, and physical construction time. d.tools successfully cut the time that is implementation-details-related to a small fraction, enabling the prototyping to be driven by design concerns.

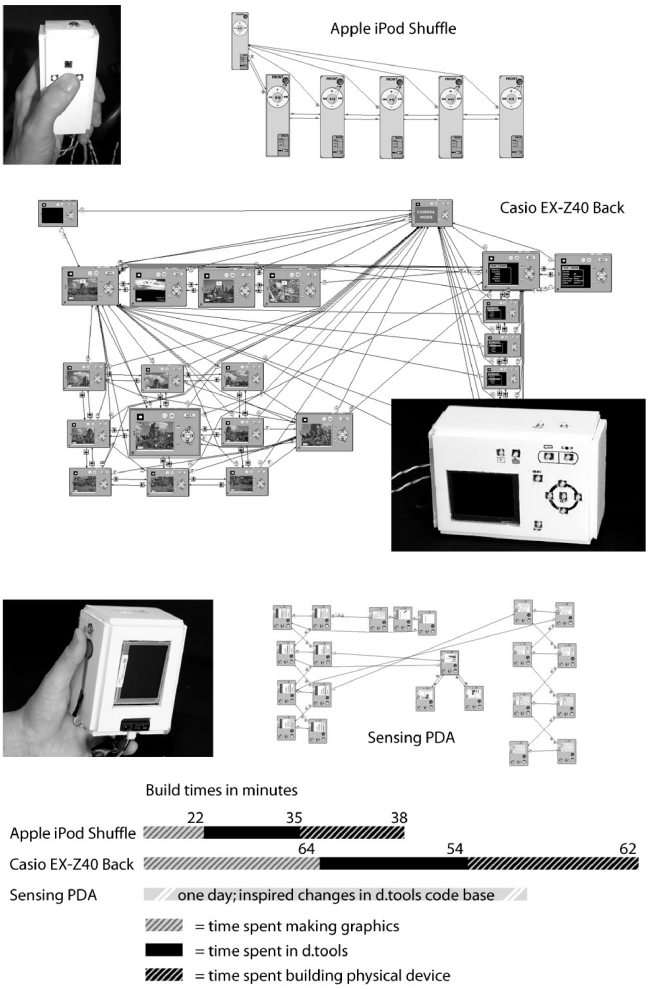

Figure 9. Statechart diagrams and build times of prototypes for three existing devices: iPod Shuffle, Casio digital camera back, Sensing PDA.

RELATED WORK

The d.tools system supports early-stage design activities. Prior work has created early-stage design tools for other domains, including graphical [26], web [22, 33], multimedia [8], speech [23], multimodal [39], and cross-device [28] user interfaces. The d.tools system also draws on related work in end-user programming and tools for physical computing. This section summarizes each area and how d.tools relates to each body of work.

End-user and visual programming

Nardi [32] notes that while general-purpose programming languages are too far removed from the tasks faced by domain experts to be adopted into their work processes, these users are not averse to using formal languages *per se*. Task-specific languages with abstractions that match the professional's work domain (*e.g.*, spreadsheets, CAD) have been tremendously successful.

Domain-tailored visual authoring environments have been successful for domains such as real-time music synthesis and control [4] and engineering simulation [5]. The HANDS system [34] shows that the same usability and human-centered design strategies used to construct task-specific languages can also be employed to develop more accessible models of general programming. d.tools seeks to transfer the benefits of end-user programmability to the domain of physical interaction design.

Tools for Physical Computing

d.tools builds on prior work on physical computing tools, including programming toolkits, multimedia authoring extensions, and visual authoring systems.

Toolkits for Programmers

The Phidgets [15] system introduced physical widgets: programmable ActiveX controls that encapsulate communication with USB-attached physical devices, such as a switch, pressure sensor, or servo motor. The graphical ActiveX controls, like the d.tools visual authoring environment, provide an electronic representation of physical state.

iStuff [9] extended this idea to support wireless devices, a loose coupling between input and application logic, and the ability to develop physical interactions that function across an entire ubiquitous computing environment. iStuff, in

conjunction with the Patch Panel [10], enables standard UIs to be controlled by novel inputs.

Papier-Mâché [20, 21] introduced a software architecture for acquiring and abstracting physical input, most notably computer vision. d.tools employs the user-centered tool design methods that Papier-Mâché introduced, but focuses on mechatronic input.

Wiring [11] is an IO board that extends the Processing [36] environment in which behavior is authored through a Java-extensible scripting language. Wiring and Processing differ from d.tools in that they seek to teach textual programming to design students.

Pin&Play [25] offers smart components with conductive pins that can be pushed into a multi-layer surface which acts as the network medium and power source. The act of attaching a component to the surface itself establishes a communications connection. Ergodex [13] follows a similar "put any element anywhere" approach, but uses RF technology to allow freeform placement of buttons on a tablet.

Application logic for these toolkits is created through a textual programming language such as Java or C. The d.tools visual authoring environment contributes a lower threshold tool and provides stronger support for rapidly developing the "insides of applications" [30] than these systems. However, textual programming offers a higher ceiling and allows for novel control of existing applications, which d.tools does not.

Extending Multimedia Authoring

Teleo [38] is a commercial system offering similar benefits to Phidgets. Teleo's primary distinction is that it is programmed with Max/MSP [4] or Macromedia Flash [1], rather than through Microsoft Visual Studio.

Calder [7, 27] integrates RFID buttons and other wired and wireless devices with C and Macromedia Director [1]. Fluid integration with physical mock-ups is aided by the small form factor of the devices.

DART [29] provides augmented reality authoring in the Macromedia Director [1] environment. It abstracts technology issues such as sensor input from designers.

This class of toolkits enables those in the interaction design community already familiar with scripting languages of multimedia applications to prototype physical devices such as remote controls and game controls. The goal of tools in this area should be similar to the goal of web authoring tools such as Macromedia Dreamweaver [1], where (for the most part) users can move fluidly between textual and visual authoring modes. d.tools shares this goal but offers a authoring environment focused on designing artifacts rather than creating a media experience.

Visual Authoring

The Lego Mindstorms Robotic Invention System [2] offers a visual environment based on control flow puzzle pieces to control sensors and actuators. While a benchmark for low-threshold authoring, Lego Mindstorms targets autonomous robotics projects; the abstractions are inappropriate for designing physical user interfaces.

STCtools [31] employs a statechart editor coupled with pen input and geo-referenced projection to prototype information appliances. It is the only other tool besides d.tools that provides explicit support for integrated raster graphics displays. There are several advantages to a small display over projection: higher resolution; freedom from occlusion; better mobility; and lower system complexity. At this point, the STCtools library comprises solely buttons and is not designed to support richer interactions.

Maestro [3] is a commercial design tool for prototyping mobile phone interactions. It provides a complex visual state language with code generators, software simulation of prototypes, and compatibility with Nokia's Jappla hardware platform. Maestro and Jappla together offer high ceiling, high fidelity mock-up development; however, the complexity of the tools make them too heavyweight for the informal UI sketching that d.tools targets.

CONCLUSIONS AND FUTURE WORK

This paper introduced d.tools, an integrated prototyping environment that lowers the threshold for creating functional physical prototypes. To better understand how d.tools is used in longitudinal practice, we have released d.tools to the design community as open source (see http://hci.stanford.edu/d-tools). This winter, we will deploy d.tools in our university's interaction design studio course to ascertain the strengths and weaknesses of this interaction model for design students. d.tools introduces an architecture that provides a low threshold for design generalists, and modular extensibility for developers. Currently, we are researching techniques for developers to more fluidly increase the software ceiling of the d.tools visual authoring language. We are also exploring opportunities for integrating digitally-controlled fabrication technologies such as 3D printing [14] into d.tools.

ACKNOWLEDGMENTS

We thank Nirav Mehta for help with fieldwork and the d.tools Flash prototype. We also thank the designers who participated in our fieldwork, the pilot, and the evaluation study; and Terry Winograd, Bill Verplank and Wendy Ju for stimulating discussions.

REFERENCES

1 Macromedia. http://www.macromedia.com
2 LEGO Mindstorms Robotic Invention System. http://www.mindstorms.lego.com/
3 Maestro, 2005. Cybelius. http://www.cybelius.com/products
4 Max/MSP. Cycling '74. http://www.cycling74.com/products/maxmsp.html
5 Simulink. http://www.mathworks.com
6 WinAVR. http://winavr.sourceforge.net

7 Avrahami, D. and S. E. Hudson. Forming interactivity: a tool for rapid prototyping of physical interactive products. In Proceedings of DIS '02: Conference on Designing interactive systems: ACM Press. pp. 141-46, 2002.

8 Bailey, B. P., J. A. Konstan, and J. V. Carlis. DEMAIS: designing multimedia applications with interactive storyboards. In Proceedings of Proceedings of the ninth ACM international conference on Multimedia: ACM Press. pp. 241–50, 2001.

9 Ballagas, R., M. Ringel, M. Stone, and J. Borchers. iStuff: a physical user interface toolkit for ubiquitous computing environments. CHI: ACM Conference on Human Factors in Computing Systems, CHI Letters 5(1). pp. 537–44, 2003.

10 Ballagas, R., A. Szybalski, and A. Fox. Patch Panel: Enabling Control-Flow Interoperability in Ubicomp Environments. In Proceedings of PerCom 2004 Second IEEE International Conference on Pervasive Computing and Communications: IEEE Press. pp. 241–52, 2004.

11 Barragan, H., Wiring: Prototyping Physical Interaction Design, Interaction Design Institute, Ivrea, Italy, 2004.

12 Card, S. K., T. P. Moran, and A. Newell, Chapter 2: The Human Information Processor, in The Psychology of Human-Computer Interaction. Lawrence Erlbaum: Hillsdale. pp. 23-97, 1983.

13 Ergodex, DX1 Input System. http://ergodex.com

14 Gershenfeld, N., Fab: Personal Fabrication, Fab Labs, and the Factory in Your Computer: Basic Books, Inc. 278 pp. 2005.

15 Greenberg, S. and C. Fitchett. Phidgets: easy development of physical interfaces through physical widgets. UIST: ACM Symposium on User Interface Software and Technology, CHI Letters 3(2). pp. 209–18, 2001.

16 Harel, D. Statecharts: A visual formalism for complex systems. Sci. Comput. Program. 8(3). pp. 231-74, 1987.

17 Hinckley, K. Bumping Objects Together as a Semantically Rich Way of Forming Connections between Ubiquitous Devices. In Proceedings of Ubicomp, 2003.

18 Hinckley, K., J. Pierce, M. Sinclair, and E. Horvitz. Sensing Techniques for Mobile Interaction. UIST: ACM Symposium on User Interface Software and Technology, CHI Letters 2(2). pp. 91–100, 2000.

19 Kelley, T., The Art of Innovation: Currency. 320 pp. 2001.

20 Klemmer, S. R., Tangible User Interface Input: Tools and Techniques, Unpublished PhD, University of California, Computer Science, Berkeley, CA, 2004. http://hci.stanford.edu/srk/KlemmerDissertation.pdf

21 Klemmer, S. R., J. Li, J. Lin, and J. A. Landay. Papier-Mâché: Toolkit Support for Tangible Input. CHI: ACM Conference on Human Factors in Computing Systems, CHI Letters 6(1). pp. 399–406, 2004.

22 Klemmer, S. R., M. W. Newman, R. Farrell, M. Bilezikjian, and J. A. Landay. The Designers' Outpost: A Tangible Interface for Collaborative Web Site Design. UIST: ACM Symposium on User Interface Software and Technology, CHI Letters 3(2). pp. 1–10, 2001.

23 Klemmer, S. R., A. K. Sinha, J. Chen, J. A. Landay, N. Aboobaker, and A. Wang. SUEDE: A Wizard of Oz Prototyping Tool for Speech User Interfaces. UIST: ACM Symposium on User Interface Software and Technology, CHI Letters 2(2). pp. 1–10, 2000.

24 Klemmer, S. R., B. Verplank, and W. Ju. Teaching Embodied Interaction Design Practice. In Proceedings of Conference on Designing for User eXperience, 2005.

25 Laerhoven, K. V., N. Villar, A. Schmidt, H.-W. Gellersen, M. Håkansson, and L. E. Holmquist. Pin&Play: The Surface as Network Medium. IEEE Communications Magazine 41(4): IEEE. pp. 90-96, 2003.

26 Landay, J. A. and B. A. Myers. Sketching Interfaces: Toward More Human Interface Design. IEEE Computer 34(3). pp. 56–64, 2001.

27 Lee, J., 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 ACM Symposium on Designing Interactive Systems: ACM Press. pp. 167–75, August, 2004.

28 Lin, J. Damask: A Tool for Early-Stage Design and Prototyping of Cross-Device User Interfaces. In Proceedings of Conference Supplement to ACM UIST 2003 Doctoral Symposium: ACM Press. pp. 13–16, 2003.

29 MacIntyre, B., M. Gandy, S. Dow, and J. D. Bolter. DART: a toolkit for rapid design exploration of augmented reality experiences. UIST: ACM Symposium on User Interface Software and Technology, CHI Letters: ACM Press. pp. 197-206, 2004.

30 Myers, B., S. E. Hudson, and R. Pausch. Past, Present, and Future of User Interface Software Tools. ACM Transactions on Computer-Human Interaction 7(1). pp. 3–28, 2000.

31 Nam, T.-J. Sketch-Based Rapid Prototyping Platform for Hardware-Software Integrated Interactive Products. In Proceedings of CHI '05 extended abstracts on Human factors in computing systems: ACM Press. pp. 1689-92, 2005.

32 Nardi, B. A., A Small Matter of Programming: Perspectives on End User Computing. Cambridge, MA: MIT Press. 162 pp. 1993.

33 Newman, M. W., J. Lin, J. I. Hong, and J. A. Landay. DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice. Human-Computer Interaction 18(3). pp. 259–324, 2003.

34 Pane, J., A Programming System for Children that is Designed for Usability, Unpublished PhD, Carnegie Mellon University, Computer Science, Pittsburgh, 2002. http://www.cs.cmu.edu/~pane/thesis

35 Pering, C. Interaction design prototyping of communicator devices: towards meeting the hardware-software challenge. interactions 9(6). pp. 36-46, 2002.

36 Reas, C. and B. Fry. Processing: a learning environment for creating interactive Web graphics. In Proceedings of SIGGRAPH 2003 conference on Web graphics: ACM Press. p. 1, 2003.

37 Schön, D. A. and J. Bennett, Reflective Conversation with Materials, in Bringing Design to Software, T. Winograd, Editor. ACM Press: New York. pp. 171–84, 1996.

38 Shiloh, M., Teleo: Rapid Prototyping Toolkit. San Francisco, CA. http://www.makingthings.com/teleo.htm

39 Sinha, A. K. and J. A. Landay. Capturing User Tests in a Multimodal, Multidevice Informal Prototyping Tool. In Proceedings of ICMI-PUI: ACM International Conference on Multimodal Interfaces: ACM Press. pp. 117–24, 2003.

40 Snibbe, S. S., K. E. MacLean, R. Shaw, J. Roderick, W. L. Verplank, and M. Scheeff. Haptic Techniques for Media Control. UIST: ACM Symposium on User Interface Software and Technology, CHI Letters 3(2). pp. 199–208, 2001.

41 Systems, I., Embedded Workbench for Atmel AVR, version 4.11. Foster City, CA. http://www.iar.com/

42 Ulrich, K. T. and S. D. Eppinger, Product Design and Development. 2nd ed: Irwin McGraw-Hill. 384 pp. 2000.