Papier-Mâché: Toolkit Support for Tangible Input

Scott R. Klemmer, Jack Li, James Lin
Group for User Interface Research
Computer Science Division
University of California
Berkeley, CA 94720-1776, USA
srk@cs.berkeley.edu

James A. Landay
DUB Group
Computer Science & Engineering
University of Washington
Seattle, WA 98195-2350, USA
landay@cs.washington.edu

Abstract
Tangible user interfaces (TUIs) augment the physical world by integrating digital information with everyday physical objects. Currently, building these UIs requires "getting down and dirty" with input technologies such as computer vision. Consequently, only a small cadre of technology experts can currently build these UIs. Based on a literature review and structured interviews with nine TUI researchers, we created Papier-Mâché, a toolkit for building tangible interfaces using computer vision, electronic tags, and barcodes. Papier-Mâché introduces a high-level event model for working with these technologies that facilitates technology portability. For example, an application can be prototyped with computer vision and deployed with RFID. We present an evaluation of our toolkit with six class projects and a user study with seven programmers, finding the input abstractions, technology portability, and monitoring window to be highly effective.

Categories & Subject Descriptors: D.2.2 [**Software Engineering**]: Design Tools and Techniques—*software libraries; user interfaces*. H.5.1 [**Information Interfaces**]: Multimedia Information Systems—*artificial, augmented, and virtual realities*. H.5.2 [**Information Interfaces**]: User Interfaces—*input devices and strategies; interaction styles; prototyping; user-centered design*. I.4.9 [**Image Processing and Computer Vision**]: Applications.

Keywords: tangible interfaces, computer vision, barcode, RFID, augmented reality, toolkits, API design

INTRODUCTION

Tangible user interfaces (TUIs) augment the physical world by integrating digital information with everyday physical objects [14]. Generally, TUIs provide physical input that controls graphical or audio output. Developing tangible interfaces is problematic because programmers are responsible for acquiring and abstracting physical input. This is difficult, time-consuming, and requires a high level of technical expertise in a field very different from user interface development — especially with computer vision. These difficulties echo the experiences of developing GUIs 20 years ago. An early GUI toolkit, MacApp, reduced application development time by a factor of five [23]. Similar reductions in development time, with corresponding increases in software reliability [10] and technology portability, can be achieved by a toolkit supporting tangible interaction.

This paper presents Papier-Mâché, a toolkit that lowers the threshold for developing tangible user interfaces. It enables programmers who are not input hardware experts to develop TUIs, as GUI toolkits have enabled programmers who are not graphics hardware experts to build GUIs. Papier-Mâché's library supports several types of physical input: computer vision (web and video cameras, the file system, and TWAIN), RFID, and barcodes (1D EAN, 2D PDF417, and 2D CyberCode [28]). Through technology-independent input abstractions, Papier-Mâché also improves application flexibility, allowing developers to retarget an application to a different input technology with minimal code changes.

A significant difficulty in debugging is the limited visibility of application behavior [4] (§7.2). The novel hardware used in tangible interfaces and the algorithmic complexity of computer vision exacerbate this problem. To facilitate debugging, Papier-Mâché provides application developers a monitoring window displaying the current input objects, image input and processing, and behaviors being created or invoked. The monitoring window also provides Wizard of Oz (WOz) generation and removal of input; it is the first post-WIMP toolkit to offer this facility. WOz control is useful for simulating hardware when it is not available, and for reproducing scenarios during development and debugging.

The design of Papier-Mâché has been deeply influenced by our experience in building physical interfaces over the past several years. This experiential knowledge is very powerful — toolkit designers with prior experience building relevant applications are in a much better position to design truly useful abstractions [22] (§2.1). As part of our user-centered design process, we also leveraged the experiential knowledge of others, conducting structured interviews with nine researchers who have built tangible interfaces.

In addition to its toolkit contributions, this paper introduces two methodological contributions. This is the first paper to

CHI 2004, April 24–29, 2004, Vienna, Austria.

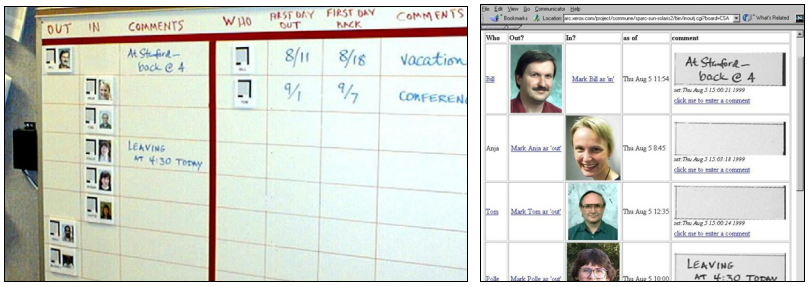

Figure 1. Collaborage [21], a *spatial* TUI where physical walls such as an in/out board (*left*) can be captured for online display (*right*).

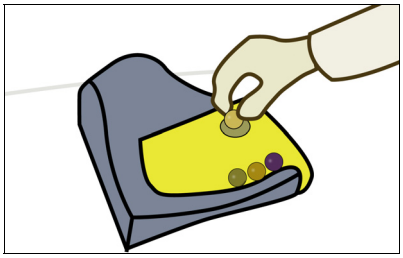

Figure 2. The marble answering machine [14], an *associative* TUI, uses marbles as a physical index to recorded answering machine messages.

employ fieldwork as a methodological basis for toolkit design. A toolkit is software where the "user interface" is an API, and the users are programmers. To our knowledge, this paper is also the first to employ a laboratory study as a method of evaluating an API *as a user interface*. (There have been studies of programming languages and environments, *e.g.*, [27]).

In this paper, we first summarize our literature review and interviews that informed Papier-Mâché. Next we discuss Papier-Mâché's architecture. We then present the results of our two evaluations. We close with related work on TUI taxonomies and ubiquitous computing toolkits.

INSPIRING TANGIBLE INTERFACES

To better understand the domain of tangible interfaces, we conducted a literature survey of existing systems employing paper and other everyday objects as input. The twenty-four representative applications fall into four broad categories: *spatial*, *topological*, *associative*, and *forms*.

In *spatial* applications, users collaboratively create and interact with information in a Cartesian plane. These applications include augmented walls, whiteboards, and tables. A majority of these applications use computer vision, often in conjunction with image capture. Collaborage, a spatial application, connects information on physical walls "with electronic information, such as a physical In/Out board connected to a people-locator database" [21] (see Figure 1).

Topological applications use the relationships between physical objects to control application objects such as media files or PowerPoint slides [25]. Paper Flight Strips [18] augments flight controllers' current work practice of using paper strips by capturing and displaying information to the controllers as the strips are passed around.

With *associative* applications, physical objects serve as an index or "physical hyperlink" to digital media. Durrell Bishop's marble answering machine [14] (see Figure 2) deposits a physical marble with an embedded electronic tag each time a message is left. To play a message, one picks up the marble and drops it into an indentation in the machine. Most associative applications employ either barcodes or electronic tags.

Forms applications provide batch processing of paper interactions. The Paper PDA [12] is a set of paper templates for a day planner. Users work with the planner in a traditional manner, then scan or fax the pages to electronically synchronize handwritten changes with the electronic data. Synchronization also executes actions such as sending handwritten email.

These twenty-four applications share much functionality with each other, including:

- Physical input for arranging electronic content
- Physical input for invoking actions (*e.g.*, media access)
- Electronic capture of physical structures
- Coordinating physical input and graphical output
- An add, update, remove event structure — these events should contain information about the input (such as size and color), and should be easily extensible

In all of these applications, feedback is either graphical or auditory. Graphical feedback is sometimes geo-referenced (overlaying the physical input, *e.g.*, [17, 20]), sometimes collocated but on a separate display [16, 25], and sometimes non-collocated (*e.g.*, Collaborage's In/Out web page [21]). For this reason, we have concentrated our current research efforts on input support. This taxonomy omits haptic and mechatronic user interfaces (which do provide physical output), as these UIs are not the focus of our research.

STRUCTURED INTERVIEWS WITH TUI DESIGNERS

As part of our user-centered design process, we conducted structured interviews with nine researchers who have built tangible interfaces. We conducted these interviews in person at the workplaces of researchers who were near our university, and over the phone or via an email survey otherwise. These researchers employed a variety of sensing techniques including vision, RF and capacitance sensors, and barcodes. Here, we summarize the findings that most directly influenced the toolkit architecture. We concentrate on the difficulties they encountered, where tools could have smoothed the process.

No Small Matter of Programming

By definition, tangible interfaces employ novel hardware. A general theme among interviewees was that acquiring and abstracting input was the most time consuming and challenging piece of application development. This is not only, as the cliché goes, a "small matter of programming." Acquisition and abstraction of physical input, especially with computer vision, requires a high level of technical expertise in a field very different from user interface development. In each of the three projects that employed

computer vision, the team included a vision expert. Even with an expert, vision proved challenging. In the words of one vision researcher, "getting down and dirty with the pixels" was difficult and time consuming.

Writing code without the help of a toolkit yielded applications that were unreliable, brittle, or both. This discouraged experimentation, change, and improvement, limiting researchers' ability to conduct user evaluation, especially longitudinal studies. One interviewee avoided these studies because his team lacked the resources to "add all the bells and whistles" that make a system usable.

The Appropriate Abstraction is Events, not Widgets

Model-View-Controller (MVC) [26] is a software design pattern for developing GUIs. In MVC-style user interfaces, a *controller* (input abstraction) sends input events to a *model* (application logic), and the model sends application events to a *view*. The view-controller combination is called a *widget*. While some post-WIMP toolkits have hoped to provide an analogue to widgets (*e.g.*, the Context Toolkit [5]), in practice *toolkit support for the view* (output) is distinct from *toolkit support for the controller* (input), and with good reason: a particular piece of input can be used for many different types of output. Interactors [24] extends MVC with higher-level input events. This higher-level API shields application developers from implementation details such as windowing systems. Papier-Mâché's event structure and associations provide a similarly high level of abstraction, allowing developers to talk about objects, events, and behavior at a semantic level, *e.g.*, "for each Post-it note the camera sees, the application should create a web page."

Authoring Behavior: Associations and Classifications

Tangible interfaces couple physical input with electronic behavior; for example, a marble represents an answering machine message [14]. This coupling implies both a *classification* describing the general case (marbles = messages), and an *association* describing each specific case (RFID tag 73 = "Hi, this is Aaron, please call me back"). While our interviewees provided these metaphors very clearly in English, not everyone felt they were implemented as clearly in software. Several interviewees wished they had a more flexible method of defining associations, making it easier to change the input technology and to explore alternative interactions for a given input technology.

Importance of Feedback for Users and Developers

Good feedback is a central tenet of user interface design. Feedback is particularly important to developers, because the complexity of their task is so high. One researcher found that, "One key issue was that sensing errors were pretty mysterious from the users' perspective." Providing visual feedback about the system's perception of tracked objects helped users compensate for tracking errors.

Debugging is one of the most difficult parts of application development, largely because of the limited visibility of dynamic application behavior [4]. The novel hardware used in tangible UIs, and the algorithmic complexity of computer vision, only exacerbate this problem. One interviewee had "the lingering impression that the system must be broken, when in fact the system was just being slow because we were pushing the limits of computation speed."

THE PAPIER-MÂCHÉ ARCHITECTURE

Our interviews and literature survey showed us that toolkit support for tangible input should support:

- Many simultaneous input objects
- Input at the object level, not the pixel level
- Application portability across multiple input technologies
- Uniform events across the multiple input technologies, supporting easy application retargeting
- Classifying input and associating it with application behavior
- Feedback for end users
- Visualizations helping programmers understand what objects were created and why, and the effect of events

Papier-Mâché is an open-source Java toolkit written using the Java Media Framework (JMF) and Advanced Imaging (JAI) APIs. JMF supports any camera with a standard driver, from inexpensive webcams to high-quality 1394 cameras. We explain the Papier-Mâché architecture using two examples: 1) an RFID implementation of Bishop's marble answering machine [14], and 2) a simplified version of PARC's Collaborage [21] using computer vision and barcodes. For each of these applications, a developer has two primary tasks: declaring the input that she is interested in and mapping input to application behavior via associations.

Input Abstraction and Event Generation

Papier-Mâché represents physical objects as `Phobs`. The input layer acquires sensor input, interprets it, and generates the `Phobs`. A developer is responsible for selecting input types, such as RFID or vision. She is not responsible for discovering the input devices attached to the computer, establishing a connection to them, or generating events from the input. These "accidental steps" are not only time-consuming, but require substantial hardware and computer vision expertise, a field very different from user interface development. For example, the marble answering machine developer adds her application logic as a listener to an RFID reader but does not need to manage a connection to the hardware. Similarly, the Collaborage developer tells Papier-Mâché that he is interested in receiving computer vision events with a video camera as the source.

Event generation

Once the developer has selected an input source, Papier-Mâché generates events representing the addition, updating, and removal of objects from a sensor's view. Event types are consistent across all technologies. Providing high-level events substantially lowers the application development threshold and facilitates technology portability.

While all technologies fire the same *events*, different technologies provide different *types of information* about the

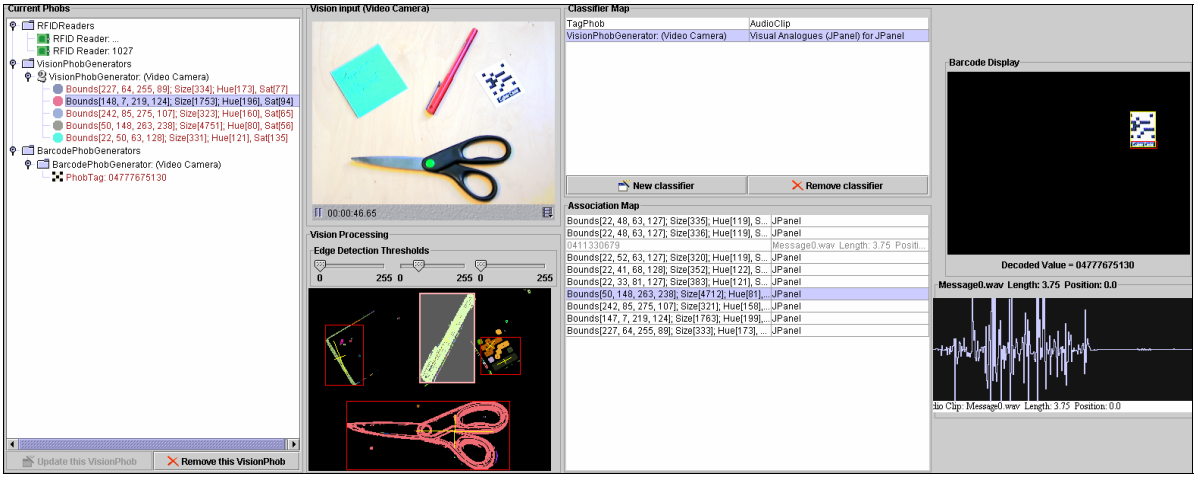

Figure 3. The monitoring window. In the 1st column, each current object appears in the hierarchy beneath the generator that sensed it. The 2nd column displays the vision input and output. The 3rd column displays classifiers (in this figure, RFID tags are associated with audio clips, and vision objects with graphical analogues). The red pen is selected in all three columns. The barcode recognizer is displayed in the top-right, and audio output is displayed on the bottom-right.

physical objects they sense. RFID provides only the tag and reader IDs. Vision provides much more information: the size, location, orientation, bounding box, and mean color of objects. (Size, location, and orientation are computed using image moments [9].) Because this set is commonly useful, but not exhaustive, `VisionPhobs` support extensibility: each stores a reference to the image the object was found in. Application developers can use this for additional processing. Barcodes contain their ID, their type (EAN, PDF417, or CyberCode [28]), and a reference to the barcode image, providing vision information such as location and orientation.

Generating RFID events requires minimal inference. Each reader provides events about tags currently placed on it. When a tag is placed on a reader, Papier-Mâché generates a `phobAdded` event. Each subsequent sensing of the tag generates a `phobUpdated` event. If the reader does not report a tag's presence within a certain amount of time, Papier-Mâché infers that the tag has been removed, generating a `phobRemoved` event. This technique was introduced by [31]. RFID events contain both the tag ID and the reader ID. Applications can use either or both of these pieces of information to determine application behavior.

Image analysis

Generating vision events requires much more interpretation of the input. Image analysis in Papier-Mâché has three phases: 1) camera calibration, 2) image segmentation, and 3) event creation and dispatching. The contribution of our research is not in the domain of recognition algorithms; the vision techniques we use are drawn from the literature. Additionally, each of these processing steps can be overridden by application developers if they are so inclined.

We have implemented camera calibration using perspective correction — an efficient method that most contemporary graphics hardware, and the JAI library, provide as a primitive. (More computationally expensive and precise methods exist, see [8], Chapters 1–3 for an excellent overview of the theory and methods.)

The segmentation step partitions an image into objects and background. (See [8], Chapters 14–16 for an overview of image segmentation.) We employ edge detection to generate a bi-level image where white pixels represent object boundaries and all other pixels are black. Labeled foreground pixels are grouped into objects (segments) using the connected components algorithm [13]. We create a `VisionPhob` for each object. At each time step, the vision system fires a `phobAdded` event for new objects, a `phobUpdated` event for previously seen objects, and a `phobRemoved` event for objects no longer visible.

Associations and Classifications

Tangible interfaces couple physical input with electronic behavior. In the In/Out board, a barcode ID represents a person, and its location represents whether they are in or out. Developers author these representation mappings by implementing an `AssociationFactory`, which listens to events from the input sources. The factory receives a callback to create a new representation instance (*e.g.*, audio message) for each new `Phob`. Association elements can be either *nouns* or *actions* [7]. *Nouns* (such as audio clips and web pages) represent content; they can be the selection focus of an application. *Actions* (such as fast-forward and rewind) control the current selection focus.

Program Monitoring: Application State Display

Papier-Mâché provides application developers a monitoring window (see Figure 3). It displays the current input objects, image input and processing, and behaviors being created or invoked with the association map.

Current objects and vision I/O

At the left-hand side of the monitoring window, Papier-Mâché displays a tree of all current input technologies, `PhobProducers`, and `Phobs`. This allows developers to see the current state of the system. Each `Phob` appears in the hierarchy beneath the generator that sensed it. The `Phob` displays a summary of its properties; `VisionPhobs` also have a circular icon showing their color.

Raw camera input is displayed at the top of the second pane. At the bottom of the second pane is the processed image; it displays each object's outline, bounding box, and orientation axis. Clicking on an object in either the "Current Phobs" view or the vision view highlights it in both views.

Wizard of Oz control

Papier-Mâché is the first post-WIMP toolkit to offer Wizard of Oz (WOz) generation and removal of input. This control is provided by the *add* and *remove* buttons at the bottom of the monitoring window; pressing these buttons causes the appropriate `PhobProducer` to fire an add or remove event, exactly as if it had come from the sensor. For computer vision, pressing *add* generates a `Phob` with a reference to the camera's current image. This WOz control is useful when hardware is not available, and for reproducing scenarios during development and debugging.

Performance

On contemporary hardware, Papier-Mâché runs at interactive rates. On a dual Pentium III computer running Windows XP, the vision system runs at 5 frames per second without monitoring, and 4.5 FPS with monitoring, at a CPU load of 80%. With the vision system and two RFID readers, the performance is 3 FPS. The performance is more than sufficient for *forms* and *associative* applications, and sufficient for *topological* and *spatial* applications with discrete events. Where tangible input provides a continuous, interactive control, current performance may be acceptable, but a minimum of 10 FPS is required for these controls to feel truly interactive [2]. Of the 24 applications we surveyed, five required this continuous direct manipulation. These performance numbers should be considered lower bounds on performance, as our code is entirely unoptimized.

Lowering the Threshold: A Simple Application

The following Java code comprises the complete source for a simple application that graphically displays the objects found by the vision system. It is only four lines of code, three of which are constructor calls.

Have the vision system generate objects from camera input.

```
1 PhobProducer prod = new VisionPhobProducer
  (new CameraImageSource());
```

Set up a map that associates each object seen by the camera with a `JPanel`.

```
2 AssociationFactory factory = new
  VisualAnalogueFactory(new PMacheWindow(
  gen, CALIBRATE), JPanel.class);
3 AssociationMap assocMap = new
  AssociationMap(factory);
```

Attach the map to the camera which will create, update, and remove `JPanel`s according to what the camera sees.

```
4 gen.addPhobListener(assocMap);
```

EVALUATION

In this section, we first discuss existing evaluation methods for toolkits. We then describe two evaluations of Papier-Mâché: use of the toolkit to build a group of class projects, and an informal laboratory evaluation.

Discussion of Evaluation Methods

Very little, if any, research has been published on evaluating a toolkit's API *as a user interface*. However, in designing API evaluation methods, we can draw inspiration from both the software engineering and the empirical studies of programmers communities.

Common evaluation metrics in the software engineering community include *performance*, *reliability*, and *lines of code* needed to produce an application. (For an excellent review of metric-based evaluation, see [3].) While these metrics are important, they do not address the end-user experience of software development.

The empirical studies of programmers community has identified several desirable properties of programming languages that we believe are also relevant for evaluating a toolkit such as Papier-Mâché:

- *Ease of use.* Programming languages and toolkits should be evaluated on how readable programs using the toolkit are by other programmers, how learnable the toolkit is, how convenient it is for expressing certain algorithms, and how comprehensible it is to novice users [29] (p. 1).
- *Facilitating reuse.* A development tool should provide solutions to common sub-problems, and frameworks that are reusable in "similar big problems" [4] (Ch. 4), minimizing the amount of application code.
- *Schemas yield similar code.* In our user study, we looked for similarity of code structure — both between programmers and for the same programmer across tasks. This code similarity implies that programmers employ a common schema (design pattern) to generate the solutions. This is desirable because it minimizes design errors, facilitates collaboration, and makes maintaining the code of others easier [4], (§ 5.2.1). From this perspective, the success of a toolkit is judged by the extent to which it is leveraged to generate the solution.

Applications Using Papier-Mâché in Coursework

Spring 2003, graduate human-computer interaction

Two groups in the Spring 2003 offering of the graduate HCI class at our university built projects using Papier-Mâché.

Physical Macros is a *topological* TUI for programming macros, such as "actions" in Adobe Photoshop. In this system, users compose physical function blocks that represent image editing functions. When examining their code, we found that presenting geo-referenced visual feedback was a substantial portion of the code. We then realized that many of our inspiring applications, including The Designers' Outpost [17], also require this feature. For this reason, we introduced the concept of associations.

SiteView (see Figure 4) is a *spatial* TUI for controlling home automation systems. On a floor plan of a room, users create rules by manipulating physical icons representing conditions and actions. The system provides feedback about how rules will affect the environment by projecting photographs onto a vertical display. SiteView employs a

ceiling-mounted camera to find the location and orientation of the thermostat and the light bulbs, and three RFID sensors for parameter input (weather, day of week, and time).

The thermostat is distinguished by size; the bulbs are distinguished by size and color. In general, the system worked well, but human hands were occasionally picked up. This inspired our addition of an event filter that removes objects in motion. With this in place, human hands do not interfere with recognition. SiteView is roughly 3000 lines of code; of this only about 30 lines access Papier-Mâché. As a point of comparison, the Designers' Outpost [17], using OpenCV, required several thousand lines of vision code to provide comparable functionality. We consider this substantial reduction in code to be a success of the API.

Fall 2003, ubiquitous computing

Four students in the Fall 2003 offering of a graduate course on ubiquitous computing at our university used Papier-Mâché for a one week mini-project. The goals of the mini-projects were tracking laser pointers, capturing Post-it notes on a whiteboard, invoking behaviors such as launching a web browser or email reader, and reading product barcodes.

These programmers were impressed with the ease of writing an application using Papier-Mâché. One student was amazed that, "It took only a single line of code to set up a working vision system!" Another student remarked, "Papier-Mâché had a clear, useful, and easy-to-understand API. The ease with which you could get a camera and basic object tracking set up was extremely nice."

The students also extended the toolkit in compelling ways. One student's extension to the monitoring system played a tone whenever an object was recognized, mapping the size of the recognized object to the tone's pitch. This provided lightweight monitoring feedback to the recognition process.

These projects also unearthed some shortcomings of the current vision algorithms. For example, the system tended to lose track of an object and then immediately find it again, causing the undesired firing of `phobRemoved` and `phobAdded` events. One student observed that vision algorithms are inherently ambiguous and requested better ways of dealing with the ambiguity.

In-lab Evaluation

We conducted an informal, controlled evaluation of Papier-Mâché to learn about the usefulness of our input abstractions, event layer, and monitoring window. Seven graduate students in our university's computer science department participated in the study: 1 in graphics, 3 in programming languages, 2 in systems, and 1 in AI. (We excluded HCI students due to potential conflicts of interest, and theory students because their background is less appropriate.) All participants had experience programming in Java.

Figure 4. SiteView, a *spatial* UI for end-user control of home automation systems. *Left:* A physical light-bulb icon on the floor-plan, with projected feedback above. *Right:* The six physical icons.

We began each evaluation session by demonstrating an application associating RFID tags with audio clips, including an explanation of the monitoring window. We then asked the participant to read a user manual of the system. Next, we gave participants a warm-up task and two full tasks. The evaluation was conducted in our lab on a dual Pentium II running Windows XP with the Eclipse IDE. We verbally answered questions about Java and Eclipse; for toolkit questions we referred participants to the user manual and online Javadoc. We asked participants to "think aloud" about what they were doing, and we videotaped the sessions and saved participants' Java code for further review.

The warm-up task was to change an application that finds red objects so that it finds blue objects. The first full task was to change an In/Out board written using computer vision to use RFID tags instead. The second full task was to write an application that used RFID tags to control a slide-show. One tag represented a directory of images; the two other tags represented *next* and *previous* operations.

Results

Every participant completed every task, though not without moments of difficulty. We take this to be a success of the API. In our first task, participants converted an In/Out board from vision to RFID in a mean time of 31 minutes using a mean of 19 lines of code. This shows that technology portability is quite possible.

Participants appreciated the ease with which input could be handled. In addition to their verbal enthusiasm, we noted that no one spent time looking up how to connect to hardware, how input was recognized, or how events were generated. In our second task, participants authored an RFID-based image browser in a mean time of 33 minutes using a mean of 38 lines of code. Note that participants on average wrote code twice as fast in the second task as in the first, indicating that they became familiar with the toolkit. Two of the participants directly copied code; one said, "So this is like the marble answering machine [in the user's manual]."

Ironically, the warm-up task—changing a colored-object finder from red to blue—proved to be the most challenging. The problem was that the classifier took a color parameter represented in a luminance-based color space (IHS), highly effective for image analysis but not intuitive to most computer scientists, who are used to the RGB color space. Participants had difficulty even though we explained that the color space was IHS, not RGB. Once a color in the proper color space was found, it took less than a minute to make the change. Ideally, these parameters should not be

specified textually at all. We are currently researching techniques for visual authoring of associations and classifications.

Overall, participants found the monitoring window to be very useful. For the warm-up task, they used it to understand the (confusing) color classifier. For the In/Out board task, they used the monitoring window to get information about the attached RFID readers. Participants also used the monitoring window to verify that the input was not the source of errors in their code.

We also uncovered several usability issues. The most glaring was an inconsistency in naming related elements: the superclass was named `PhobGenerator`, a subclass `RFIDReader`, and the accessor method `getSource`. Other points of confusion highlighted places where our documentation was insufficient. We have since addressed these usability issues by improving the API, documentation, and method names based on the feedback from this study.

RELATED WORK

We present related work in TUI taxonomies and ubiquitous computing toolkits. Papier-Mâché is heavily inspired by the projects described in this section. A general distinction between our work and prior work is that this is the first paper to employ fieldwork as a methodological basis for toolkit design and use a laboratory study as a method of evaluating an API *as a user interface.*

Emerging Frameworks for Tangible User Interfaces

Ullmer and Ishii [30] provide an excellent taxonomy of existing tangible interfaces. We have drawn heavily on both this taxonomy and the innovative ideas of their Tangible Media Group in creating our list of inspirational applications. They also propose MCRpd as analogue to MVC for physical UIs. The difference is that the view is split into two components: *Rp*, the physical representation, and *Rd*, the digital representation. However, from an implementation standpoint, it is unclear whether explicitly separating physical and digital outputs is beneficial. In fact, for reasons of application portability, it is important that the event layer be agnostic to whether the implementation is physical or digital (*e.g.*, for studies, it would be useful to create and compare physical and electronic versions of an application). Also, the approach is untested: no tools or applications have been built explicitly using the MCRpd approach.

Ubiquitous Computing Toolkits

The work most related to Papier-Mâché is Phidgets [11]. Phidgets are physical widgets: programmable ActiveX controls that encapsulate communication with USB-attached physical devices, such as a switch or motor. Phidgets are a great step towards toolkits for tangible interfaces. The graphical ActiveX controls, like our monitoring window, provide an electronic representation of physical state. However, Phidgets and Papier-Mâché address different classes of tangible interfaces. Phidgets primarily support tethered, mechatronic TUIs that can be composed of powered, wired sensors (*e.g.*, a pressure sensor) and actuators (*e.g.*, a motor). Papier-Mâché supports TUI input from untethered, passive objects, often requiring computer vision.

Papier-Mâché provides stronger support for the "insides of the application" than Phidgets. Phidgets facilitates the development of widget-like physical controls (such as buttons and sliders), but provides no support for the creation, editing, capture, and analysis of physical input, which Papier-Mâché supports.

IStuff [1] introduces compelling extensions to the Phidgets concept, primarily support for wireless devices. IStuff provides fast remapping of input devices into the iRoom framework, enabling standard GUIs to be controlled by novel input technologies. There are two main differences in our research agenda: First, like Phidgets, iStuff targets mechatronic tangible interfaces, rather than augmented paper tangible interfaces. For example, it is not possible to build computer vision applications using iStuff or Phidgets. Second, iStuff offers novel control of existing applications, while Papier-Mâché does not. Unlike iStuff applications, the tangible interfaces Papier-Mâché supports do not use a GUI input model.

Fails and Olsen have implemented a highly successful system for end-user training of vision recognizers, Image Processing with Crayons [6]. It enables users to draw on training images, selecting image areas (*e.g.*, hands or notecards) that they would like the vision system to recognize. They employ decision trees as their classification algorithm, using pixel-level features. The resulting recognizers can be serialized for incorporation into standard Java software. Crayons complements our work well, offering a compelling interaction technique for designating objects of interest. Papier-Mâché's recognition methods (*e.g.*, edge detection and perspective correction) are higher-level than the pixel-level processing employed by Crayons. We also offer higher-level object information (*e.g.*, orientation and aspect ratio), and most importantly, an event mechanism for fluidly integrating vision events into applications. Papier-Mâché's classifiers also supports ambiguity [19], an important feature unavailable in Crayons.

The Context Toolkit (CTK) [5] makes context-aware applications easier to build. We find this work inspiring for two reasons. First, it is one of the most rigorous and widely used post-WIMP toolkits to date. Second, it does not just provide a software interface to physical sensors (a la Phidgets), it "separates the acquisition and representation of context from the delivery and reaction to context by a context-aware application." Papier-Mâché provides more monitoring and WOz facilities than CTK, and it supports interactive tangible interfaces, which CTK does not.

The ARToolKit [15] provides support for building applications that present geo-referenced 3D graphics overlaid on cards marked with a thick black square. The ARToolKit provides support for 3D graphics; Papier-Mâché does not. The ARToolKit does not provide general information about objects in the camera's view, only the 3D location and ori-

entation of marker cards. It is not a general input toolkit; it is tailored for recognizing marker cards and presenting geo-referenced 3D graphics through a head-mounted display.

CONCLUSIONS AND FUTURE WORK

We have presented Papier-Mâché, a toolkit for building tangible interfaces using computer vision, electronic tags, and barcodes. Our event-based model for working with these types of input facilitates technology portability. From our literature review and interviews, we learned what functionality Papier-Mâché should provide. Class projects showed us how easy it was to apply Papier-Mâché to a variety of systems. A user study validated that even first-time users could build tangible interfaces and easily adapt applications to another technology.

Currently, we are extending WOz support and researching techniques for visually authoring associations and classifications. We plan to optimize the vision system, extending support to applications that demand lower latency. We are actively seeking more users, and we are researching improved methods for evaluating the usefulness of toolkits like Papier-Mâché. Papier-Mâché is open-source software available at http://guir.berkeley.edu/papier-mache.

REFERENCES

1 Ballagas, R., M. Ringel, *et al.*, iStuff: a physical user interface toolkit for ubiquitous computing environments. Human Factors in Computing Systems, *CHI Letters*, 2003. **5**(1): pp. 537–44.

2 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.

3 Clements, P., R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*. Boston: Addison-Wesley. 323 pp, 2002.

4 Détienne, F., *Software Design—Cognitive Aspects*. London: Springer Verlag. 200 pp, 2001.

5 Dey, A.K., D. Salber, and G.D. Abowd, A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction*, 2001. **16**(2-4): pp. 97–166.

6 Fails, J.A. and D.R. Olsen, A Design Tool for Camera-based Interaction. Human Factors in Computing Systems, *CHI Letters*, 2003. **5**(1): pp. 449–56.

7 Fishkin, K.P., T.P. Moran, and B.L. Harrison. Embodied User Interfaces: Towards Invisible User Interfaces. Proc. *Conf. on Engineering for Human-Computer Interaction*. pp. 1–18, 1998.

8 Forsyth, D.A. and J. Ponce, *Computer Vision: A Modern Approach*. Upper Saddle River: Prentice Hall. 693 pp, 2003.

9 Freeman, W.T., D. Anderson, P. Beardsley, C. Dodge, *et al.*, Computer vision for interactive computer graphics. *IEEE Computer Graphics and Applications*, 1998. **18**(3): pp. 42–53.

10 Grady, R.B., Practical Software Metrics for Project Management and Process Improvement, Prentice Hall: Englewood Cliffs, NJ. pp. 17, 1992.

11 Greenberg, S. and C. Fitchett, Phidgets: easy development of physical interfaces through physical widgets. User Interface Software & Technology, *CHI Letters*, 2001. **3**(2): pp. 209–18.

12 Heiner, J.M., S.E. Hudson, and K. Tanaka, Linking and messaging from real paper in the paper PDA. User Interface Software & Technology, *CHI Letters*, 1999. **1**(1): pp. 179–86.

13 Horn, B., *Robot vision*. Cambridge: MIT Press. 509 pp, 1986.

14 Ishii, H. and B. Ullmer. Tangible Bits: Human Factors in Computing Systems. Proc. *CHI: Human factors in computing systems*. pp. 234–41, 1997.

15 Kato, H., M. Billinghurst, and I. Poupyrev. ARToolKit. *University of Washington HIT Lab*, 2000. http://www.hitl.washington.edu/artoolkit/

16 Klemmer, S.R., J. Graham, G.J. Wolff, and J.A. Landay, Books with Voices: Paper Transcripts as a Tangible Interface to Oral Histories. Human Factors in Computing Systems, *CHI Letters*, 2003. **5**(1): pp. 89–96.

17 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. User Interface Software and Technology, *CHI Letters*, 2001. **3**(2): pp. 1–10.

18 Mackay, W.E., A.-L. Fayard, L. Frobert, and L. Médini. Reinventing the Familiar: Exploring an Augmented Reality Design Space for Air Traffic Control. Proc. *CHI: Human Factors in Computing Systems*. ACM Press. pp. 558–65, 1998.

19 Mankoff, J., S.E. Hudson, and G.D. Abowd, Providing Integrated Toolkit-Level Support for Ambiguity in Recognition-Based Interfaces. Human Factors in Computing Systems, *CHI Letters*, 2000. **2**(1): pp. 368–375.

20 McGee, D.R., P.R. Cohen, R.M. Wesson, and S. Horman, Comparing paper and tangible, multimodal tools. Human Factors in Computing Systems, *CHI Letters*, 2002. **4**(1): pp. 407–414.

21 Moran, T.P., E. Saund, W. van Melle, A. Gujar, *et al.*, Design and Technology for Collaborage: Collaborative Collages of Information on Physical Walls. UIST: User Interface Software and Technology, *CHI Letters*, 1999. **1**(1): pp. 197–206.

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

23 Myers, B. and M.B. Rosson. Survey on User Interface Programming. Proc. *CHI: Human Factors in Computing Systems*. ACM Press. pp. 195–202, 1992.

24 Myers, B.A., A new model for handling input. *ACM Trans. on Information Systems*, 1990. **8**(3): pp. 289–320.

25 Nelson, L., S. Ichimura, E.R. Pederson, and L. Adams. Palette: a paper interface for giving presentations. Proc. *CHI: Human Factors in Computing Systems*. ACM Press. pp. 354–61, 1999.

26 Olsen, D.R., Chapter 5: Basic Interaction, in *Developing User Interfaces*, Morgan Kaufmann. pp. 132–62, 1998.

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

28 Rekimoto, J. and Y. Ayatsuka. CyberCode: Designing Augmented Reality Environments with Visual Tags. Proc. *Designing Augmented Reality Environments (DARE 2000)*. ACM Press. pp. 1–10, 2000.

29 Shneiderman, B. Empirical Studies of Programmers: The Territory, Paths, and Destinations. Proc. *First Workshop on Empirical Studies of Programmers*. Ablex Pub. pp. 1–12, 1986.

30 Ullmer, B. and H. Ishii, Emerging Frameworks for Tangible User Interfaces, in *Human-Computer Interaction in the New Millennium*, Addison-Wesley. pp. 579–601, 2001.

31 Want, R., K.P. Fishkin, A. Gujar, and B.L. Harrison. Bridging Physical and Virtual Worlds With Electronic Tags. Proc. *CHI: Human Factors in Computing Systems*. ACM Press. pp. 370–77, 1999.