HUMAN–COMPUTER INTERACTION, 2009, Volume 24, pp. 315–366

# Toolkit Support for Integrating Physical and Digital Interactions

Scott R. Klemmer, James A. Landay

## Summary:

The authors do an extensive review of physical/digital interaction platforms and make a toolkit, Papier-Mache, to more easily integrate several types of physical and digital interactions.

## Principles:

### Input mode vs input action (2)
An elegant architecture should separate mode (WIMP*, gesture, speech) from action (select, create, or move).
*windows, icons, menus, pointer

### Support for ambiguous input and for mediating the input (2.1)
Provide support for input that requires interpretation and methods for intervening between the recognizer and the application to resolve the ambiguity.

Ambiguity is an essential property of recognition-based input support. Ambiguity information is maintained in the toolkit through the use of hierarchical events.

### Categorization of dynamic systems (2.2)
#### spatial applications
users collaboratively create and interact with information in a Cartesian plane

#### topological applications
use the relationships between physical objects to control application objects

#### associative applications
physical objects serve as an index or physical hyperlink to digital media

#### forms applications
provide batch processing of paper interactions

### Event notifications (3.3)
Basic events hinge on notifying objects of the presence, absence, and modification of physical objects.

### Event categorizations (3.3)
#### binary events
Buttons, switches, discrete events

#### continuous scalar events
Planar position, orientation, size

#### rich capture events
Audio recording, video, photo

## Highlights

### 2.2. Characterizing Physical Interaction Designs

We conducted a literature survey of existing systems employing passive, untethered input from paper and other everyday objects. To concentrate on this aspect of physical interaction design, we limited the purview of this survey—and the subsequent toolkit—in two important ways. First, this taxonomy omits interfaces that employ powered sensing and actuation, such as haptic feedback. Second, this taxonomy omits 3D sensing. These constraints offer a coherency of user experience interests, making it easier to compare the systems. In addition, our intuition was that these other areas have design requirements—such as the low-latency needed by force-feedback haptics—that would require different architectural support. Several of the taxonomies and toolkits mentioned earlier in this section target these other aspects of physical interaction design.

Figure 3. The marble answering machine (Ishii & Ullmer, 1997), an associative TUI, uses marbles as a physical index to recorded answering machine messages. Left: Bishop's original sketch, redrawn by the author. Right: Bishop's prototype where resistors are embedded in marbles.

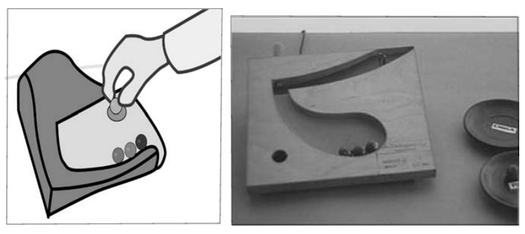
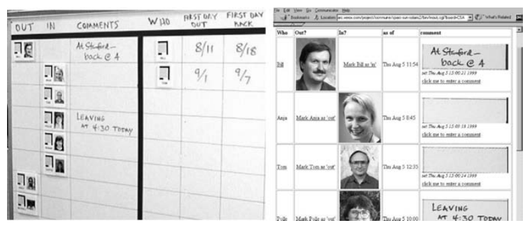
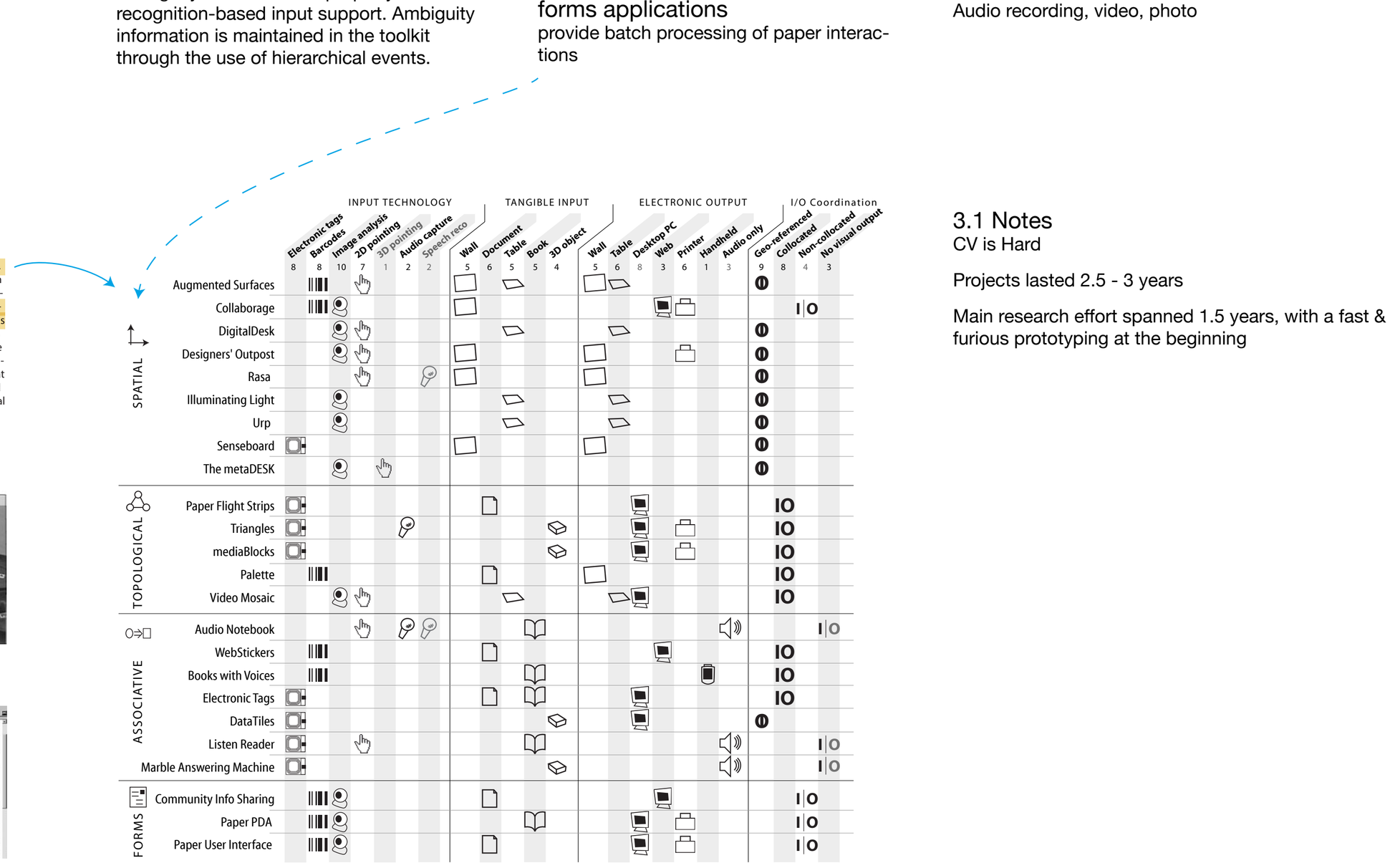


Figure 2. Collaborage (Moran et al., 1999), a spatial TUI where physical walls such as an in/out board (left) can be captured for online display (right). Reproduced with permission.




## 3.1 Notes
CV is Hard

Projects lasted 2.5 - 3 years

Main research effort spanned 1.5 years, with a fast & furious prototyping at the beginning

## 3. FIELDWORK INSPIRING PAPIER-MÂCHÉ

### 3.1. Team, Process, and Goals

In each of the three projects that employed computer vision, the team included a vision expert. Even with an expert, writing vision code 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. In addition, in two of the nonvision projects, the person who developed the tangible input was different from the person who developed the electronic interactions. In the remaining cases, the developers all had a substantial technical background and worked on both the physical and electronic portions. We speculate that if tools with a lower threshold were available to these individuals, then a larger portion of the team may have contributed to functioning prototypes, rather than just conceptual ideas.

Often, the interviewees redesigned aspects of their system architectures to support a wider range of behavior. This refactoring reflects positively on the developers; advocates of agile programming methods argue that software architectures should be simple at first, becoming more complex only as needed (Beck, 2000). This heavy refactoring also reflects negatively on the current state of software tools in this area. Much of the modularity that our interviewees introduced could be more effectively provided at the toolkit level, and indeed is supported in Papier-Mâché. Examples include changing the type of camera, switching between input technologies, or altering the mapping between input and application behavior.

We also spoke with the interviewees about their user experience goals. At a high level, they offered goals like "technology should make things more calm, not more daunting" and people are "torn between their physical and electronic lives, and constantly trying work-arounds." The primary motivation our interviewees had for building a tangible interface was the desire for a conceptual model of interaction that more closely matched user's behavior in the real world, often as one interviewee described it, "trying to avoid a computer."

### 3.5. Language, Architecture, and Reuse

Our interviewees used several different programming languages: C++ (three), Java (two), Prolog (one), Director (one), Visual Basic (one), and Python (one). Two of the non-Java teams have since switched to Java. Eight of the nine interviewees used a Windows PC as their development platform.

Most of the interviewees chose a programming language based on one particular requirement. They requirements cited were as follows:

1. Technology integration: Sometimes, the decision was made to ease integration with a particular piece of input technology, for example, the Designers' Outpost vision system was built in C++ because the OpenCV library it used was in C.
2. Library support: The majority of our interviewees chose a language based on the library use it facilitated. Two developers chose the Windows platform and a Visual Studio language specifically for their easy interoperability with Microsoft Office. A third interviewee was constrained to the Windows platform for integration reasons, and chose Director for its rapid development capabilities. Two of our interviewees decided on Java because of its rich 2D graphics libraries.
3. Developer fluency: For one interviewee, C++ was chosen "because [the lead software developer] knew it well." The same fluency sentiment was expressed differently by another interviewee that C++ "was our language of the time. Now we're Java."
4. Rapid development: One interviewee told us that "I used Python. This language make prototyping fast and easy. It can be too slow at times, but not too often thanks to Moore's law." Another interviewee, previously mentioned, settled on Adobe Director for rapid development reasons.

These four reasons—technology integration, library support, developer fluency, and rapid development—informed our choice of Java as a programming language.

### 4. THE PAPIER-MÂCHÉ ARCHITECTURE

Our literature survey, experiential knowledge, and fieldwork data show that a toolkit for tangible input should support

- Techniques for rapidly prototyping multiple variants of applications as a catalyst for iterative design
- Many simultaneous input objects
- Input at the object level, not the pixel or bits level
- Heterogeneous classes of input
- Uniform events across the multiple input technologies, facilitating rapid application retargeting
- Classifying input and associating it with application behavior
- Visual feedback to aid developers in understanding and debugging input creation, dispatch, and the relationship with application behavior

Figure 4. The inheritance hierarchy for physical input devices. Each device class encapsulates a physical input. The InputDevice is a marker interface: it is an interface class that contains no methods. Classes implement the marker interface to denote that they represent a physical device.

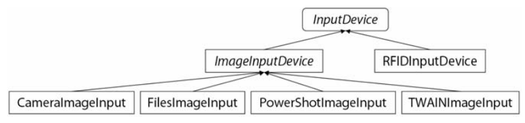



Figure 5. The inheritance hierarchy for PhobProducers. Producers are paired with InputDevices; they take input from a device and generate PhobEvents. The abstract PhobProducer base class manages the event listeners and the production of events.

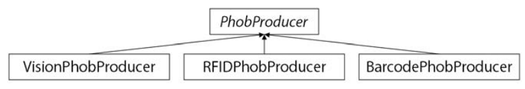



Figure 6. The inheritance hierarchy for factories: objects that create AssociationElts from Phob input. The top level is the AssociationFactory interface. The middle level is the DefaultAssociationFactory abstract class; this class provides the ability to be VisuallyAuthorable and the ability to serialize to XML using JAXB.

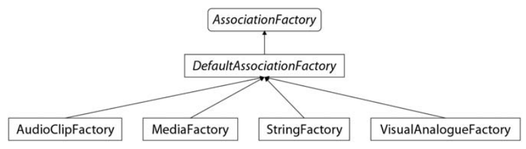



Figure 7. The inheritance hierarchy for associations. Associations are the elements in the Papier-Mâché architecture that input is bound to. These elements can either be nouns or actions. The Papier-Mâché library includes five common media manipulation actions, and four common types of nouns.

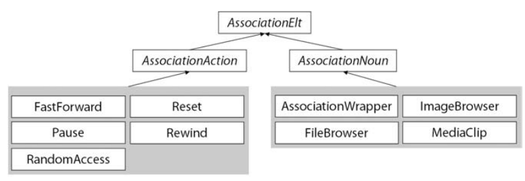
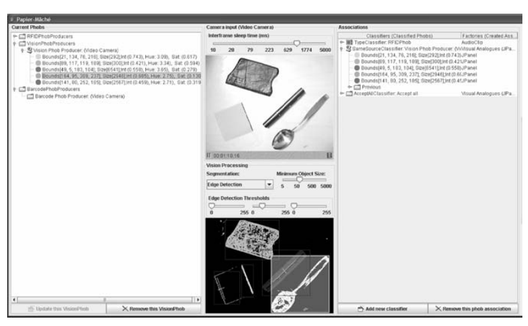



Figure 8. The monitoring window. In the first column, each current object appears in the hierarchy beneath the producer that sensed it. The second column displays the vision input and output. The third column displays classifiers (in this figure, RFID tags are associated with audio clips, and vision objects with graphical analogues).



### 3.2. Acquiring and Abstracting Input

A general theme among interviewees was that acquiring and abstracting input was the most time consuming and challenging piece of development. This is not, 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. These novel input technologies, especially vision, do not always function perfectly. We found that consequently, it is important to design a system where occasional errors do not prevent the system as a whole from functioning, and to provide feedback so that users can diagnose and help recover from system errors. An interviewee explained, "The sensing hardware is not perfect, so sometimes we had to change interactions a bit to make them work in the face of tracking errors." This error-aware design of physical interfaces is similar to the techniques used for voice user interface design, where limiting grammar size and providing confirmation feedback help systems minimize errors, and help users diagnose and recover from errors when they do occur.

A general theme among interviewees was that acquiring and abstracting input was the most time consuming and challenging piece of development. This is not, 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. These novel input technologies, especially vision, do not always function perfectly. We found that consequently, it is important to design a system where occasional errors do not prevent the system as a whole from functioning, and to provide feedback so that users can diagnose and help recover from system errors. An interviewee explained, "The sensing hardware is not perfect, so sometimes we had to change interactions a bit to make them work in the face of tracking errors." This error-aware design of physical interfaces is similar to the techniques used for voice user interface design, where limiting grammar size and providing confirmation feedback help systems minimize errors, and help users diagnose and recover from errors when they do occur.

### 5. EVALUATION

Although there has certainly been prior work on evaluating software tools, this area is more limited than might be expected, perhaps because, as Détienne (2001) wrote,

> The dominant problems have been perceived as technical rather than as related to the usability of the systems. The introspective approach, which is the common approach in this field, carries the illusion that usability problems are automatically handled: tool developers will use their own experience as the basis for judging the usefulness of the tools they develop. (p. 118)

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.

Three other Berkeley projects have used Papier-Mâché. The first is ObjectClassifierViews (De Guzman et al., 2003), which provides a set of graphical user interface dialogs that allow users to create classifiers and modify their parameters. This work inspired us to integrate their code into Papier-Mâché and to provide a mechanism for saving applications created visually.

The second is All Together Now (Lederer & Heer, 2004; see Figure 14), an awareness tool where the locations of individuals in a space are captured through computer vision and presented abstractly on a Web page. Remote individuals can "interact" with the local individuals by placing a marker of themselves on the space. Prior to All Together Now, the Papier-Mâché library only included edge detection, a stateless vision technique. The complexity of this scene and the low fidelity of the camera make stateless techniques impractical. Lederer and Heer implemented the background subtraction algorithm to overcome this. We incorporated their background subtraction code into the Papier-Mâché library. This experience showed us that it is possible for individuals interested in "getting under the hood" to change the vision algorithms used by Papier-Mâché and that its overall architecture is modular enough to easily accommodate new algorithms.

Figure 14. ATN captures a bird's-eye video feed of the physical space (left), locates people using computer vision (middle), and displays local actors' positions (orange) in a virtual space (right) shared with remote actors (green). Non-participating remote actors are placed in an observation deck. Each remote actor's circle is marked with a yellow core in his personal view. (Picture on right is annotated for grayscale printers). Image from Lederer and Heer (2004).