International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

Design and Evaluation of an Event Architecture for Paper UIs: Developers Create by Copying and Combining

Ron B. Yeh, Scott R. Klemme, Andreas Paepcke

Stanford University HCI Group, Computer Science Department Stanford, CA 94305-9035, USA [ronyeh, srk, paepcke]@cs.stanford.edu

ABSTRACT

This paper explores architectural support for interface ensembles comprising pens, paper, and PCs. We show how the event-based approach common to GUIs can apply to augmented paper, and describe additions to address paper's distinguishing characteristics. Most notably, we introduce techniques for integrating interactive and batched input handling, coordinating interactions across devices, and rapidly debugging applications that integrate paper and computation. To understand the developer experience of this architecture, we deployed the toolkit to 17 teams at another university for six weeks. Static analysis of the participants' code provided insight into the appropriateness of events for paper (*e.g.*, programmers relied on debugging output to track event flow). We also distilled usage patterns, revealing that programmers composed simple gesture han- dlers from ink measurements. This desire for informal interactions inspired us to include abstractions for recogni- tion. This study has implications beyond paper—designers of graphical tools can provide visualizations for event flow, and examine API usage to inform toolkit revisions.

. **Keywords** ;Toolkits, augmented paper, design tools, device ensembles.

INTRODUCTION

Recent research has introduced techniques for augmenting paper with computation and interaction. A primary attrac- tion of designing augmented paper interactions is their embrace of existing practices, particularly in mobile, infor- mal, and collaborative settings (*e.g.*, [18, 41]). However, we do not know which techniques will help developers create these systems. This raises several questions. What aspects of graphical UI architectures can be adopted—or borrowed

Figure 1. PaperToolkit provides support through an event-driven model, output to devices, and debugging techniques. Users have created tools for many tasks, including web design (*left*) and music composition (*right*). Going beyond the retrieval and form-filling tasksshown in prior paper + digital work, these apps explore real-time control of screen elements and recognition of informal input.

wholesale—for creating these paper applications? How do differences in interacting with each domain suggest the need for distinct approaches to tools? Finally, what kinds of applications are developers interested in creating, and *what do they do in practice*? This paper aims to provide insights into these questions; we hope the approach and findings we present will

International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

also provide value for other areasof ubiquitous computing.

Augmented paper interactions provide a quiet, flexible input interface for documenting information in domains from biology (*e.g.*, [27, 44]) to design and music (see Fig- ure 1). In describing augmented paper interactions, it can beuseful to delineate two distinct approaches. One takes as its starting point the kinds of drawing and writing that users have traditionally engaged in with paper, building interac- tive functionality on top of this. The other begins by regarding the pen as primarily a command-specification device, exploring paper widgets, gestures, and asynchron- ously executed behaviors. In reality, most research and commercial systems draw from both of these approaches. Examples that draw more on the former include techniques for temporally coordinating multiple media—such as Audio Notebook's and LiveScribe's integration of written notes with captured audio [25, 38], Adapx's ruggedized system for capturing field notes [2], A-book's use of a PDA to help organize laboratory notes [27], and ButterflyNet's integra- tion of field observations and photographs [44]. Work that

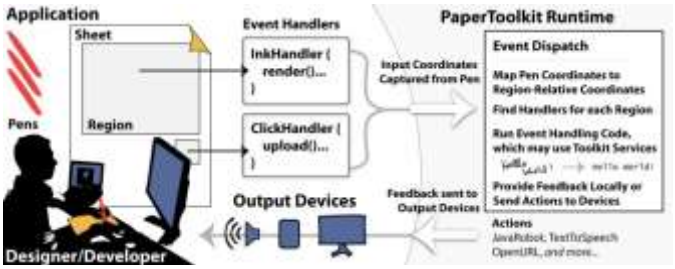


Figure 2. In PaperToolkit, input arrives from pens (*left*) and is sent tohandlers (*middle*). Output is displayed on the local machine or routedto devices (*bottom*). The architecture unifies real-time and batched input by injecting synched data into the event stream. Event dispatchand UI construction are modeled after GUI architectures, to help programmers create paper + digital applications faster.

exemplifies the command-centric approach provides inte- ractions ranging from taps of the pen (*e.g.*, to retrieve scientific citations [32]), through gestures for annotatingand editing printed documents (*e.g.*, [12, 23]), to gesturesfor creating and playing paper-based games [22].

There are many enabling technologies for integrating paper and computation (*e.g.*, [10, 13, 16, 38, 41]). PaperToolkit, our implementation of the ideas described in this paper, is built on top of Anoto's [3]; chosen for its reliability, mobili-ty, high-resolution capture, and ability to distinguish pages. This technology employs a camera, mounted inside the pen and pointed at the tip, to track pen motion across paper (printed with a dot-pattern). This vision-based tracking provides the location, force, and time of each stroke, either in real time (via Bluetooth) or in batched mode (via a wired dock). However, most aspects of PaperToolkit's architec- ture apply to alternate pen technologies.

This paper explores event-based architectures for paper + digital interactions, and introduces PaperToolkit, a manife- station of the ideas. This work offers three contributions:

First, this research builds on prior augmented paper plat- forms [12, 36] that have abstracted development pragmaticssuch as producing Anoto-enhanced paper, acquiring pendata, and rendering captured ink. PaperToolkit is similar to this prior work in its bookkeeping of the correspondence between *interactive* paper elements and their location in the Anoto coordinate space. However, PaperToolkit's architec- ture is more flexible than these prior systems, introducing techniques for integrated real-time and batched input han- dling, coordinated interactions across devices, and rich debugging of augmented paper.

Second, this paper reports on the usage of PaperToolkit,and how it has evolved in response to our findings. We provided the toolkit to a semester-long undergraduate HCI class at another university. The class comprised 69 students in 17 groups, mostly computer science juniors and seniors. The usage patterns we present were distilled through dis- cussions with students and source-code analysis. For example, we learned that much time was spent debugging, so we introduced event replay to streamline the workflow.

Third, the paper contributes a method for user-centered toolkit design through static source-code analysis. Our findings provided ideas for architecture and API revisions, such as support for asynchronous event handling. The findings also offer

International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

insight into how programmers approach new tools, such as copying example code to learn how to construct initial working programs.

The paper is organized as follows. We first introduce the PaperToolkit architecture and describe how applications arecreated with it. We then describe findings about the tool- kit's usage, how it evolved in response to those findings, and additional design implications. We next describe this paper's relationship with prior work, and close by suggest- ing opportunities for future research.

THE PAPERTOOLKIT ARCHITECTURE

PaperToolkit addresses the problem of creating, debugging, and deploying paper + digital applications. In these interfac- es, one or more people use digital pens and paper to captureand organize information, and issue commands to a com- puter via pen gestures and paper widgets. Visual or audio feedback is presented to the user on a nearby PC or hand- held device (see Figure 2). Alternatively, a user may work without a PC nearby; his pen input is batched for later processing. Paper interfaces come in many forms, includingdatasheets for scientists, notebooks for designers, and large maps and posters for engineers. The question is how devel- opers program the input handling and feedback for the UI.

PaperToolkit helps programmers accomplish this by provid-ing methods to create paper forms, abstractions to handle multi-device events, and techniques to develop and debug faster. The abstractions are distributed across seven main concepts, summarized in the following table:

	Concept	Examples	Useful in Other Apps
Paper UI	Application Runtime and Event Handling	PaperToolkit, ClickHandler	○
	Paper UI Construction and Printing	Sheet, Region, Printer	◒
	Coordinating Ensemble Interactions	Device, OpenURL, LiveWhiteboard	●
Pen	Pen Data Capture, Manipulation, Rendering	Ink, InkRenderer, InkXMLParser	◒
	Ink Stroke Metrics and Recognition	InkUtils, HandwritingRecognizer	●
Tools	Development and Debugging Tools	SaveAndReplay, PaperUIDesigner	○
	Units, Coordinate Conversion, Utilities	Inches, CoordinateConverter	●

While all seven areas address augmented paper applica-tions, several are valuable in other areas; these are noted on the right-hand side of the table. For example, an application that uses a device ensemble but not augmented paper may use the *Actions* architecture to coordinate visual and audio feedback across devices.

Scenario: Designing a Paper-Based Blog

Karen is building a paper-based blogging system. A user writes blog entries with a digital pen, and selects a paper button to wirelessly transmit the entries to a handheld device, which uploads them to a web site.

```
1  public class SimplePaperApp {
2     public static void main(String[] args) {
3        Application app = new Application();
4        Sheet sheet = app.createSheet(8.5, 11);
5        Region region = sheet.createRegion(1, 1, 4, 2);
6        Device remote = app.createRemoteDevice();
7        region.addEventHandler(
8           new ClickAdapter() {
9              public void clicked(PenEvent e) {
10             remote.displayMessage("Pen Tap");11         }
12          });
13       app.run();14    }
15 }
```

Figure 3. This program contains one region on a sheet of paper. When a user taps this *button*, the application sends feedback to a remote device. A GUI programmer would find this approach familiar.Additionally, inter-device interactions are abstracted (lines 6 & 10). The *Device* object handles message passing, without requiring a developer to manually program socket communications.

On a PC, Karen writes a Java program to create a *Sheet* object, a large *Region* to capture the user's handwriting, anda small *Region* to act as the upload button (*e.g.*, the sheet in Figure 2). She adds two event handlers: an *InkHandler* to capture notes, and a *ClickHandler* to detect the pen tap. When Karen prints the paper UI, PaperToolkit augments the interface with the dot

International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

pattern signature. At runtime, the*InkHandler* receives the user's strokes from the pen's wire- less connection. When the user taps the button, Karen's code retrieves the ink strokes, sends it through handwriting recognition, renders it to a JPEG, and uploads the image and text to the user's blog. This programming approach (see Figure 3) builds on the Java Swing [39] and WindowsForms [28] architectures. However, PaperToolkit distin- guishes itself by providing API support for integrating paper tools into mobile and collaborative environments, and techniques to help developers replay test sessions, reducing the time required to create these interfaces.

Feedback in Mobile and Collaborative Environments Because paper is tangible, lightweight, and robust, paper applications lend themselves to mobile and ubiquitouscomputing scenarios. These situations often require devel-opers to handle feedback across multiple users and devices(*e.g.*, [5]). To integrate paper into this device ensemble[34], PaperToolkit provides abstractions to help developerscope with three issues:

First, programmers need to be able to distinguish multiple users, as in Twistr, a game we created where players each use two pens to pick photos from a paper print (see Figure 4). To enable this, PaperToolkit supports multiple pens,providing a *penID* in *PenEvent*s (similar to [9, 14]).

Second, unlike graphical UIs, paper does not provide real- time visual feedback. To provide feedback to end users, programmers can use Java Swing on the computer running the program. This toolkit differs from prior paper + digital platforms as it also integrates with Adobe Flash, to allow artists and designers to design the visual feedback.

Third, scenarios may involve multiple devices. For exam- ple, BuddySketch is an application we built to provide shared paper sketching during video conferencing. The local computer asks its remote peer to update its ink displayas needed. This interaction is accomplished through mobile code [40]. Each computer is a *Device*, and can invoke *Actions* (*e.g.*, *OpenURL*). For example, in Figure 3, lines 6& 10 send feedback to a remote device. Network details are hidden, but behind the scenes, a call to the *Device* serializesan *Action* to XML and sends it to a remote device over TCP. The listening device (running a toolkit program which waits for *Actions*) reconstitutes the object and calls `invoke()`. PaperToolkit provides the most common actions, and the option to pass arbitrary messages. This device abstraction speeds up the prototyping of ensemble interactions.

Interleaving Real-time and Batched Event Handling Paper applications can be used near or away from PCs, so programmers must handle both real-time and batched inte-ractions. Prior tool support allowed developers to access pen data *either* in real-time (via Bluetooth) [36], or inbatched mode (when the user docks his pen) [12, 23]. Our experience shows that often, users would benefit from real-time *and* batched handling of interactions.

For example, a biologist using an augmented notebook [44] is frequently away from a PC. Here, a batched architecture enables the handwritten notes to be processed when the pen is docked at the field station. However, while working in the field, she may use a pen gesture to link a photographshe has just captured to a place in her notebook. When her camera recognizes this action, it provides *immediate* audio and visual feedback to acknowledge the linking gesture. In ButterflyNet, this was accomplished with two applications. The camera software recognized gestures in real-time, logging events to a file. The software that ran on the PC would take these timestamps and align them to the photos. While functional, this fractured implementation has nega- tive impact on code readability and maintainability.

PaperToolkit helps developers co-locate event handling for batched and real-time events. To do this, it provides a flag in *PenEvent*s so that data received through the wireless

International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

Figure 4. Applications we built to explore the design space of paperinterfaces. The Twistr game (*left*) is a tabletop game. Students have used the toolkit to produce research, such as a collaborative designenvironment integrating sketches on paper with manipulations on a digital table [7] (*right*). These two support competition/collaboration with multiple pens (*e.g.*, four pens in Twistr).

connection can be distinguished from data received via the dock. In event handlers, developers check the `isRealTime` flag to provide appropriate feedback. To support this, when a pen is docked, PaperToolkit reads the data and injects events into the application's event stream. Besides the creation timestamp and the `isRealTime` flag, batched andreal-time events behave equivalently. This architectural feature enables developers to present feedback appropriate to each situation, facilitates clearer code organization, and provides developers a way to test in real time applications that will be deployed for batched usage.

Debugging with Save and Replay

PaperToolkit introduces techniques that aid developers in testing applications. First, PaperToolkit enables developers to save and replay user input. This saves time by reducing the need to reproduce input sequences, and produces con- sistent results across trials. It also enables developers to engage in test-driven development, where they write han- dlers to support pre-defined test cases.

As an example of test-driven development, we return to Karen's blogging application. She first runs a skeleton version of the app, where none of the handlers are defined. She uses the paper UI as if it were functional, composing a blog article, performing a gesture to insert a photo, and tapping a button to upload the entry. When she later imple- ments the event handlers (*e.g.*, to recognize gestures), she can replay her test to see the program gradually develop.

PaperToolkit accomplishes save and replay by logging all input at runtime. The developer can select a log file and replay the saved input stream. When she selects a file to replay, the replay manager injects events into the *EventDis- patcher*. The dispatcher relays the events to handlers, as if they had come from a physical pen.

Simulating Printed UIs with Off-the-Shelf Notebooks

Our experiences building paper interfaces (*e.g.*, [43, 44]) taught us that the time required to physically print the interface became a bottleneck to testing. Thus, we can speed up development if we can delay printing until it is absolutely necessary, such as when deploying to an end user. To minimize the need for printing, PaperToolkit provides techniques to test event handlers using a graphical simulator or pre-printed Anoto notebooks.

The graphical simulator is accessed through the tray icon ofthe running application, and can be used with a mouse or Tablet PC stylus. Input is translated into *PenEvents* and sent to handlers. To test with an actual digital pen, the designer uses digital paper. Through the tray menu, she selects a *Region* to test. The system prompts her to define that regionby drawing a rectangle on a patterned page; this binding is saved for future test runs. This technique facilitates rapid prototyping. After defining *Regions*, the designer can cut them out and affix them to prototypes. For example, we have augmented a magazine with patterned paper (see Figure 5). Simulation provides a method for testing early

Figure 5. Designers can create functional prototypes by binding pat-terned pieces of paper to *Regions* containing event handlers, and then attaching them to physical objects such as this magazine. On the left, a reader can write his email to subscribe to the mailing list. On the right, he can draw a path to calculate a ball's elasticity. De- signers can prototype interesting interactions in very little time.

prototypes. Since input is saved, developers can gather user data before implementing any event handlers.

Implementation

This toolkit comprises a substantial research effort that has improved with each iteration. The toolkit is implemented primarily with Java SE 6.0, with pieces built on other plat- forms (242 Java, 21 ActionScript, 14 C#, and 8 JavaScript files).

International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

The runtime receives input from pens, and finds and invokes event handlers attached to regions on the paper interface. Batched pen input is handled through a .NET component, as Anoto synchronization is implemented in native Windows code. For rendering to paper, PaperToolkit uses PS/PDF: paper UIs and dot patterns are rendered using the Java EPS [29] and iText PDF libraries [26]. Handwriting recognition uses Microsoft's Tablet PC recognizer, and the gesture handler uses Wobbrock's $1 recognizer [42].

Exploring the Design Space to Inform Toolkit Design Over the last year, we have revised the architecture to makeit more learnable and extensible. By using the toolkit our-selves, we explored points in the design space, andaddressed early feedback rapidly. We explored threeareas—tables, walls, and notebooks—covering multiplescenarios, since tables and walls are collaborative, whilefield notes are single-user and mobile.

Collaborative Scenarios using Paper on Digital Tables Through three projects with digital tables [9] (~19 Javaclasses each), we learned the importance of supporting*multiple users*, coordinating with external *devices*, andtranslating between *coordinate systems*.

We created a tabletop design environment [7] (see Figure 4,*right*) that used PaperToolkit to capture writing from mul- tiple users, render ink to a canvas, and send drawings to a printer. A second project, by a visiting researcher, captured ink written on sticky notes [17]. It recognized multiple users' handwriting, and associated the annotations to a map displayed on the table. Cross-out gestures deleted annota- tions. These projects supported multiple users, and recognition of handwriting and gestures.

The third project, by a Masters student, explored multiple-device input [6]. For exam- ple, a user can set a pivot point with his pen and rotate/zoom a photo around the pivot with his other hand. PaperToolkit provided the hooks to coordinate the pen while writing, rotating, and zooming. From this, we learned the importance of abstracting input to allow simulation of the pen with a mouse or fingertip. Additional- ly, we facilitated coordinate transformations; while pen input was physically co-located with finger input, each device reported coordinates that had to be reconciled.

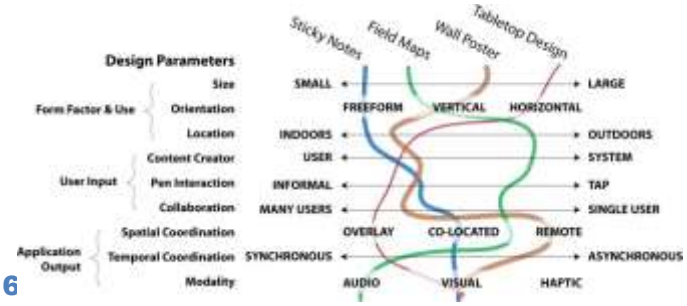
Remote Collaboration around Large Paper Displays

The second area explored large paper interfaces for collabo- ration. We created FeedReader, a wall poster that displayed articles posted to an RSS feed. Multiple users write in com- ments (captured by *InkHandlers*); the ink is transported to aweb site where remote participants can view the discussion. *ClickHandler*s retrieved articles to a nearby display, and the *InkRenderer* created JPEGs for upload to the web. And, as described earlier, we created BuddySketch, which sup- ported video conferencing between scientists by providing real-time sharing of drawings. These projects taught us to coordinate interactions between paper and devices (*e.g.*,triggering remote display of digital ink).

Mobile Field Notes and Maps

In the third area, we looked at how handheld maps could fit into a field biologist's ensemble. For example, users can search GPS-tagged field data with circle gestures, specifying the center and radius of the search in a single motion. The map comprises two Java classes, and communicates with a database over a Web API. We have also explored larger maps. For example, the Audio Guide allows scientists to retrieve audio annotations on a map while out in the field. Apen tap selects a region, and audio notes attached to that region are played to a wireless earpiece the user wears. The mapping projects suggested the need to support GPS trans- formations and gestures.

Figure 6 summarizes the paper + digital design parameters we considered. While the categories are not necessarilyorthogonal (*e.g.*, outdoor use tends to imply smaller sizes), they illustrate the considerations we have encountered.



Index in Cosmos
January 2022Volume 12 Issue 1
UGC Approved Journal

International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

Figure 6. We show how four of the projects we built cover the designspace of paper interactions. One area we did not explore was haptic feedback (*e.g.*, [24]). This design space borrows elements from priortaxonomies (*e.g.*, [11]).

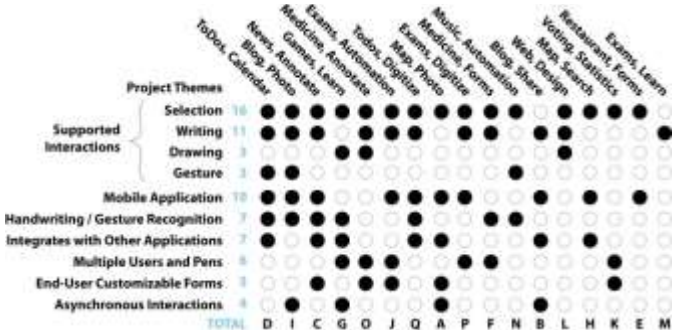


Figure 7. Summary of the 17 projects in this study (A through Q). The projects covered many themes. PaperToolkit supported *selec- tion* (*e.g.*, check a box) and *writing* operations well. However, informal interactions requiring recognition were less common (*e.g.*, draw a musical note). We have since included better support for gesture recognition. Ten projects were mobile, and seven integratedwith existing apps in mash-up fashion. Four supported batched in- put, where ink is processed after the user returns to his PC.

EVALUATION: DEPLOYMENT AND CODE ANALYSIS

We conducted a study with 69 programmers (17 teams), providing the stable core of UI construction, pen input, and event handlers to an undergraduate HCI class at a nearby university (the authors did not teach the class). The students developed interactive paper + digital applications with PaperToolkit; their development began in the eighth weekof a 14-week class. The first author held in-person sessions to answer questions and receive feedback, folding these observations into later iterations of the toolkit.

First, we observed that many developers created applica- tions by mashing-up interactive paper with web services, likely because this leverages pre-built functionality. Project topics included paper-based web design, personal organiz- ers, and sharing tools for news and blogs (see Figure 7). Six teams integrated web applications into their projects, direct-ly scraping HTML, or using established APIs (*e.g.*, Flickr and Google Calendar). One group created a Firefox plug-in. From this, we learned that modern platforms should facili- tate data-transfer with web services, and provide support for connecting to existing desktop applications. PaperToolkit supports this goal by making *Ink* objects available in web- friendly formats (*e.g.*, XML, JPEG, PNG). We have since introduced support for GUI feedback through Adobe Flash.

Second, we learned to unify the model for dispatching synchronous and asynchronous events. The deployed tool- kit handled events when in real-time mode, but only provided access to ink data when in batched mode (reflect- ing Anoto's model). In the end, 13 of 17 projects chose real-time input handling. While real-time input was great for debugging, operating in *batched* mode — where data resides on the pen until it is uploaded—eases the deploy- ment of mobile applications by eliminating the need for a nearby PC. 10 of the 17 teams created mobile applications, and would have benefited from event handling for batched pen input. We have since unified support; real-time and batched inputs now dispatch events identically. If the de-

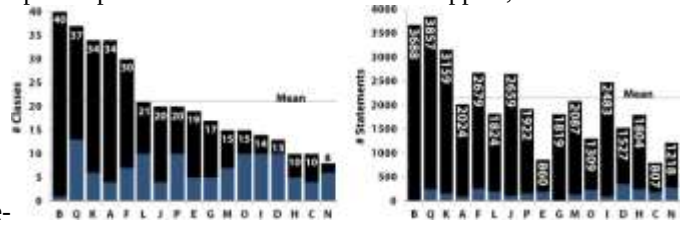


Figure 8. Student teams created substantial paper + digital projectsin six weeks, with the average project comprising 2102 source statements and 21 Java classes. The size of the projects supports the external validity of our research insights. The light blue bars show the portion of projects directly depending on PaperToolkit.

veloper does not need event handling, he can access the entire batch of uploaded ink through a *PenSynchHandler*.

It is notable that 17 teams of students with no prior expe- rience in building paper interfaces were able to build working

International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

projects in less than six weeks. Because Paper- Toolkit built on established GUI conventions, students who had programmed GUIs before could apply their experience. However, many students *learned GUI programming* as part of this introductory course (*e.g.*, a team said that ―none of us had developed event-driven programs prior to this project‖). For these students, the similarity to Swing meant that they did not have to learn two different architectures.

In the remaining sections, we review the insights we achieved by using source-code analysis to inform toolkit design. Examining code produced by developers offers an empirical account of usage patterns and provides insight into usability of the architecture and API. We reviewed the students' 304 source files (~35K source statements or ~51K lines of code with comments) by searching through and reading files and through automatic scripts for calculating statistics (see Figure 8). We recorded observations for each file, and grouped recurring themes. The following sections present these patterns with examples, and introduce impli- cations to inform the design of future pen-and-paper tools.

Composing Ink Operations for Gesture Recognition Developers added value to informal interactions by imple-menting recognition. By understanding how developersapproached gesture recognition, we may be able to gaugethe ceiling of the toolkit. Since ink processing is a coreconcern of paper applications, we gathered data on all

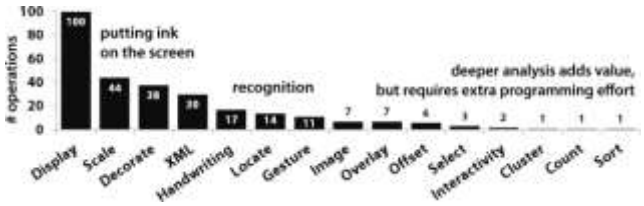
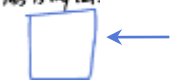


Figure 9. This shows the different operations on ink we discovered,where each operation can consist of multiple statements. While *display* and *scale* operations were common, it took extra effort to compose solutions for recognizing gestures or cluster strokes. Making it easier to explore these operations may raise the tail of thiscurve, and add extra value to the resulting interactions.

instances of ink operations. Figure 9 shows that teams oftendisplay, scale, and decorate ink, but only a handful clus- tered or sorted ink strokes. The deployed toolkit provided handlers for directional gestures, but not a full recognition engine. Thus, only four teams implemented recognition (viaheuristics). This suggests that the value of recognition comes at a considerable cost of time and effort. To lower the barrier, we added a gesture engine [42] and provided extra metrics to facilitate heuristics-based recognition.

For example, four teams composed basic ink operations to recognize higher level ink gestures. *Team D* detected when users crossed out handwritten text, and updated a web planner to reflect the completed task. *Team G* recognized paper-based games (*e.g.*, tic-tac-toe). *Team I* detected boxesusers had drawn in a blog entry, and helped users import photos into those areas. *Team N* recognized handwritten music, including whole, half, quarter, and eighth notes, and translated the composition into MIDI. In the following snippets, we see that heuristics can be robust enough to facilitate gesture recognition. In *Team I*'s project, the user inserts a picture into a blog entry by drawing a square witha single stroke. To determine the bounds, the team iterated through the strokes to find the one with the largest area:

```
getPointForPhotoInsertion() {for (ink : inkList) {
     for (strokes : ink.getStrokes()) {for (s : strokes) {
           if (s.getArea() > maxArea) {maxArea = s.getArea(); boxInkStroke = s;
              boxInk = ink;
              ....
```



Later, they test the box against a size threshold, and posi- tion the photo at its boundaries. Similarly, *Team N* applied heuristic measures to strokes to determine note durations:

```
if (timeOfStroke <= shortTimeThreshold) {
   if (heightOfStroke <= shortHeightThreshold) {out.print("Whole note in Key of ");
   } else {
      out.print("1/2 note in Key of ");
                                    }
                                } else {
```

International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

```
if (firstY > lastY) { // if the note is going upif ((lastY - minY) > 5) { // detect flag
      out.print("1/8th note in Key of ");
   } else { // 1/4 note
....
```

This algorithm compares strokes to temporal and spatial thresholds, and detects their direction. An eighth note is recognized when the last ink samples are written in a direc- tion opposite to the main stem (detecting the note's flag). These examples show how teams approached recognition by composing ink statistics. Heuristics are straightforward to specify and can be robust for many interactions.

We have since added new methods to search and measure digital ink, in addition to a single-stroke gesture recognizer

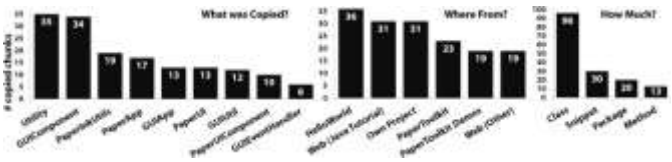


Figure 10. Teams used copy-and-paste to facilitate coding. Devel-opers customized utility functions (*left*), and usually copied code from their own projects and web tutorials (*middle*). Developers would copy an entire class file to get a program working, and then customize it (*right*). This suggests that programmers can benefit from tools that embrace this behavior by tracking code lineage, providing templates, and facilitating refactoring of copied code.

[42]. Developers can combine these measurements to proto-type recognition for their applications, and select a subset of input to be sent to a full recognizer. We plan to further improve support for composing ink operations, selecting strokes in space and time, clustering strokes for calculating size and location, and visualizing the results, so that proto- typing recognition can be approachable.

Creating Interfaces by Example Modification

Developers copy from their own code and from code found on the web to speed up programming. Tools should em- brace this by interleaving examples with documentation, or helping developers visualize what was copied and to where. In particular, we found evidence that developers would copy UI classes, paste them into their project, and then modify the skeletons to grow their interface around the working base. To find out *what* code developers copied, *how much* they would copy, and from *where*, we used a combination of analysis methods. First, we used MOSS [35],a tool traditionally used to detect plagiarism in software, to detect similarities between projects and the toolkit. While MOSS worked for comparing projects fed to it, it was not suited for *locating* clones of code residing on the web. As a result, we manually identified clones in the corpus, noticing that unusual comments and identifiers were effective in pointing out copied code. Once we found a plausible clone candidate, we search the web and the corpus to establish the source. This paper presents the first work that uses static code analysis to study copy-and-paste behavior for assess- ing toolkit usability.

Developers copy to reduce the boilerplate code they need to write, and when encountering problems they (or others) have solved before. Our data show that 41% of the 159 copied chunks directly supported GUIs, and 37% supported paper UIs (see Figure 10, *left*). The clones came from sever-al sources, including a *Hello World* app that they had worked through, the Web (*e.g.*, Java Swing tutorials), and PaperToolkit. In 96 of the 159 instances, developers copied a class file rather than packages, methods, or snippets, and then modified the class to fit their application (*right*). These findings are consistent with earlier studies (*e.g.*, [19, 33]) that discovered that programmers copy blocks of method calls to save time. We find that developers copy boilerplate for constructing UIs, or when working with new APIs (to reduce errors with an unfamiliar framework). We suggest

that tools embrace a developer's approach to learning by copy-and-paste, as this eases API learning. In this vein, we have integrated more example code into the documentation.

Analysis of Usage Reveals Opportunities for Design Examining toolkit usage (see Figure 11) provides insightson which parts of the API might be too verbose (left side) orwhich parts may be too difficult to understand (tail end).The chart shows how many instances of each class weredeclared. The most used classes (*Inches*, *Region* and its subclasses, *Sheet*) are for UI construction. If we concentrateon making this part of the API more clear and concise, we

International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

might reduce the number of bugs developers encounter,since they spend the most time working with these classes.

Examining the distribution's tail may reveal classes that were difficult to understand, hidden by lack of examples. For example, *BatchedEventHandler* was only used twice, asthe deployed toolkit touted real-time handling. We thus improved batched handling. We suggest that designers can improve toolkits by examining usage, and determining why certain methods or classes are or are not used.

Understanding Event Handlers through Debug Output Developers prolifically used print statements to monitorevent flow in their programs. To discover where printstatements helped most, we searched for instances of debugoutput (as this study was conducted outside our lab, we didnot log interactions with the IDE or debugger). The corpuscontained 1232 debugging statements containing ‑*println*.‖ We examined and annotated them to see *where* the state-ments were located and *what* was printed in each one (seeFigure 12). Overall, 39% of all debug statements werefound inside event handlers: 333 were in GUI handlers, and145 in PaperToolkit handlers. The top three printed valueswere *object instances* particular to the program, *error*messages, and *got here* messages serving only to signal thata code block was reached. When coupled with where state-ments were located, we find that half of all GUI handler*println*s and a third of PaperToolkit handler *println*s were

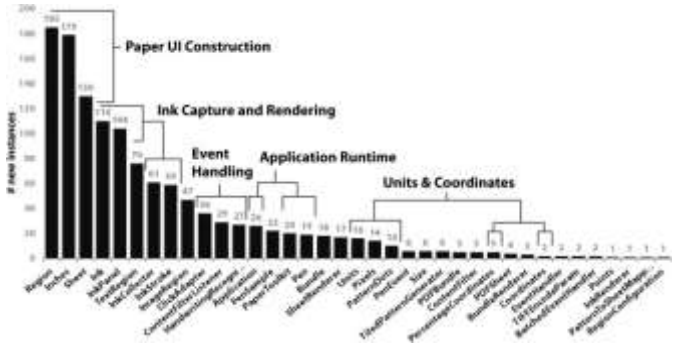


Figure 11. This shows how often teams used the toolkit classes. The heavily used classes are for constructing UIs (*Region*, *Inches*, *Sheet*). Classes for manipulating ink were popular (*Ink*, *InkPanel*, *InkCollector*, *InkStroke*). The seldom used classes were advanced features or that were less well-documented. We can have impact if we improve the API of classes on the left (*e.g.*, by increasing clari- ty). To raise the tail end, we might add documentation or examples.

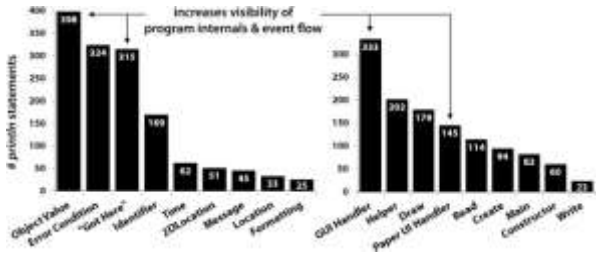


Figure 12. Analysis of the code of 17 projects revealed that most debugging statements (*println*s) helped programmers visualize *object values*. Other statements reveal when the program encoun- ters an error, or gets to a particular location. Developers place mostof the *println*s in handlers (GUI and PaperToolkit). This suggests a need for tools that help monitor objects and handlers at runtime.

got here statements. These served different roles. Some- times, they acted as stubs to track unimplemented handlers:

```
System.out.println("Zoom In");
// TODO Auto-generated Event stub actionPerformed()
```

As the handler was filled in, debug statements are then used to help developers visualize the program internals, and how it responds to pen input. This iterative debugging is referred to as ‑debugging into existence‖ [33]. Overall, this shows that programmers try to visualize their program's (often hidden) behavior to help monitor for bugs.

International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

This suggests that tools can help developers understand *when* events are fired, and *what* the object values are at runtime. A *println* is lightweight compared to a breakpoint or variable watchpoint, and it can signal warnings at run- time without stopping execution. Unlike breakpoints, *println*s are manifest in the code, and are easily shared between teammates. If a tool can maintain this lightweight property, provide the benefits of a debugger, while reducing the time needed to understand event flow, it can substantial-ly speed up the development of applications.

Managing Multiple Coordinate Representations

Finally, we observed two common tasks: converting be- tween different coordinate systems (*e.g.*, from paper to GPS latitude/longitude) and mapping stroke locations to different semantic representations (*e.g.*, musical tones). In this study, two projects used maps, and needed methods such as:

```
convertPaperToEarthCoordinates() getGoogleMapTileCoordinate(lat, lon, zoomlevel)getStreetIntersection(penX,
penY, pageNum)
```

With paper UIs, developers often deal with coordinate conversions (in GUIs, coordinates are all in screen space). With maps, we have paper coordinates (*e.g.*, inches), the device's screen coordinates (pixels) and world coordinates (GPS). PaperToolkit now supports GPS, and provides a *CoordinateConverter* for generic mappings.

Conversions do not always produce numerical results. For example, `getStreetIntersection()` invokes a database query that returns the names of cross streets in San Francisco. Likewise, the music project interpreted ink coordinates as locations on a staff, converting them into musical notes.

Even in non-mapping projects, developers need to scale, translate, and rotate coordinates to accomplish tasks such as aligning ink to images (*Team O*'s medical imaging) or processing ink correctly (*Team M*'s flashcards, to handle when users wrote upside down on the back of a card).

DESIGN IMPLICATIONS

From our observations and source-code analysis, we have distilled implications to inform the design of paper + digital applications and tools.

Interaction Language—to add value to their applications, developers support informal input and gestures. However, this presents problems when an end-user wants to learn to operate the UI. GUIs have built up a language of interaction through widgets, but paper UIs have not. Some patterns translate well (*e.g.*, paper buttons respond to a pen taps). To specify a location on a map, however, a user might tap, circle, or draw an ×. Tools should support gestures, and help to embed textual or graphical directions into the UI to communicate the gestures (*e.g.*, Palm OS's Graffiti manual).

Ordering Constraints—a related issue lies in enforcing interface constraints. In the class projects, students used text labels to direct the user's interaction. In a GUI, developers can gray out a component when it is not appropriate, direct-ing users toward an order of interactions. On paper, one can never prevent a user from checking a box or turning the page. Developers must account for this intrinsic limitation by providing clear directions, and dealing with incomplete or out-of order input. Tools should help determine the validity and completeness of input before further processing (*e.g.*, by using state machines for event handling, as in [4]).

Enhancing Debugging—a barrier a developer faces when iterating designs is the speed at which she can test. In our study, students reported that they seldom iterated their paper UIs, as it took long to print. For paper UIs, printing slows the workflow; thus, we minimized the need to print. Repeated testing was also laborious, so we allowed design- ers to replay input, to reduce the time to locate and fix bugs. Tools should speed up debugging to enhance the workflow.

Visualization Tools—we suggest that visual tools can help programmers debug *textual* programs by making hidden aspects of the application visible. Like all platforms that introduce new constructs, PaperToolkit has the drawback in that learning the patterns requires expertise (*e.g.*, compared to creating spreadsheets). In the past, visualizations have helped people understand algorithms and data structures (*e.g.*, [37]). In the future, visualizations might help pro- grammers understand event flow at runtime. Additionally, programmers copied code to create paper UIs; we now provide a visual UI builder.

RELATED WORK

This research builds upon prior studies of software engi- neers' development and debugging practices, and earlier work in user interface architectures and tools.

International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

Supporting Existing Programming Practices

The literature on development practices shows that copy- and-paste and example modification are common tech- niques among software engineers. Rosson and Carroll studied four programmers and found that they benefited from having working examples they could modify to in- clude in their own project [33]. Other studies indicate that programmers use copy-and-paste to reduce typing, and ensure that the easy-to-forget details (*e.g.*, method ordering)are correct. For example, Kim *et al.* studied expert pro- grammers and found that copy-and-paste was used to save time when creating or calling similar methods [19]. Later, LaToza *et al.* found that modifying example code was one of *several* types of code duplication, which can cause prob- lems when fixing bugs and refactoring [21]. Our results support these earlier findings, contributing from a much larger corpus, and finding that developers tend to copy from*their own* code (earlier projects, or places in the same project), probably because these snippets have a high prob- ability of working, and are easy for them to understand. We also find that programmers copy when they need to learn a new API, such as PaperToolkit's.

The earlier studies have inspired a number of programming environments that map to the way programmers (and non- programmers) think about software [30]. For example, WhyLine [20] provided a timeline view for inspecting how events produced behaviors in an animation. PaperToolkit's replay was inspired by this timeline, which allowed pro- grammers to scroll back to visualize hidden dependencies.

User Interface Software Architectures and Tools

Overall, we learned that the GUI model for programming largely works for programming paper UIs. PaperToolkit's approach was inspired by GUI architectures, borrowing the basic ideas of components, layout, event handling, and extensibility from toolkits like Java Swing [39], Windows Forms [28], and SubArctic [15]. PaperToolkit extends this to include ensemble interactions across augmented paper and digital devices, and support for asynchronous event handling. PaperToolkit has also explored XML representa- tions of the paper UI, inspired by the movement to better separate the view from event handling, as seen in [1, 31].

PaperToolkit also distinguishes itself from existing tools for*paper* interfaces. It combines real-time and batched event handling into a single programming model, and foregrounds abstractions for ensemble interactions, mobility, and gesture and handwriting recognition. Anoto's SDK [3] enables developers to access pen data, but does not support event handling. Cohen *et al.*'s work [2, 8] combines input with speech commands. PADD integrates annotations on a physi- cal document back into the digital one [12]. iPaper maps pen input to objects stored on the remote iServer [32, 36], providing media retrieval and event handling through its active components. iPaper is the platform most related to our own; PaperToolkit contributes by integrating batched and real-time input, supporting debugging with event rep-

lay, and not requiring a database for the retrieval of media. Additionally, PaperToolkit's *Actions* hide network details that iPaper exposes, such as HTTP requests. Our methodo- logical contribution beyond the prior platforms is our evaluation and iteration of the abstractions, informed by deployment and code analysis.

CONCLUSIONS AND FUTURE WORK

Through the deployment, evaluation, and improvement of the toolkit, we have learned that an event-driven approach goes a long way to providing a useful platform for paper applications. Added support for multi-device communica- tion, unifying batched and real-time event handling, ink processing, and rapid debugging helps to provide a low barrier for entering this space. Our results can have impact on tools for GUI programming. For example, we found it valuable to combine evidence from long-term use with static analysis of source code to inform toolkit design. We suggest that tool designers adopt these techniques. GUI platforms can also benefit from support for integrating web services and mobile devices, consistent with today's trends.

However, there remain opportunities for research. First, we would like to involve non-programmers (*e.g.*, designers) during development. One line of future work would be to provide tools to specify interactions by example, and visua- lizations to support user testing. Second, we find that programmers frequently need to learn new toolkits. We will examine how visualizations can help developers understand a toolkit's internals, to speed up development. Finally, in today's paper applications, if the user needs to update his paper UI, he must print out a new copy. In the future, we will support the scheduling of *automatic updates*. The paperUI will be the view as in MVC, but with a slow refresh rate.

PaperToolkit is open-source. The code, examples, and documentation are located at http://hci.stanford.edu/paper.

ACKNOWLEDGMENTS

We thank the students, and Maneesh Agrawala, David Sun, and Gerald Yu for making the deployment a success. Joel Brandt, Marcello Bastéa-Forte, and Jonas Boli contributed code, while others have provided feedback as users. Many anonymous reviewers provided insightful comments. This work was supported by NSF IIS-0534662, and Intel and Nokia hardware. Conclusions expressed are those of the authors and do not necessarily reflect those of the NSF.

REFERENCES

Abrams, M., C. Phanouriou, A. L. Batongbacal, S. M. Wil- liams, and J. E. Shuster. UIML: An Appliance-Independent XML User Interface Language. In Proceedings of *The EighthInternational World Wide Web Conference*, 1999.

Adapx, *Mapx and Penx*, 2007. http://www.adapx.com

Anoto AB, *Anoto Technology*, 2007. http://www.anoto.com

Appert, C. and M. Beaudouin-Lafon. SwingStates: Adding State Machines to the Swing Toolkit. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 319–22, 2006.

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 Com-puting Systems*. pp. 537–44, 2003.

Bastéa-Forte, M., R. B. Yeh, and S. R. Klemmer. Pointer: Multiple Collocated Display Inputs Suggests New Models for Program Design and Debugging. *UIST Extended Abstracts (Posters)*, 2007.

Bernstein, M., A. Robinson-Mosher, R. B. Yeh, and S. R. Klemmer. Diamond's Edge: From Notebook to Table and Back Again. *Ubicomp Extended Abstracts (Posters)*, 2006.

Cohen, P. R. and D. R. McGee. Tangible Multimodal Interfac-es for Safety Critical Applications, *Communications of the ACM*, vol. 47(1): pp. 41–46, 2004.

Dietz, P. and D. Leigh. DiamondTouch: A Multi-User TouchTechnology. *UIST: ACM Symposium on User Interface Soft-ware and Technology*. pp. 219–26, 2001.

EPOS, *EPOS Digital Pen*, 2007. http://www.epos-ps.com

Foley, J. D. and V. L. Wallace. The Art of Natural Graphic Man-Machine Conversation. *IEEE* **62**(4). pp. 462–71, 1974.

Guimbretière, F. Paper Augmented Digital Documents. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 51–60, 2003.

Heiner, J. M., S. E. Hudson, and K. Tanaka. Linking and Messaging from Real Paper in the Paper PDA. *UIST: ACM Symposium on User Interface Software and Technology*. pp.179–86, 1999.

Hourcade, J. P. and B. B. Bederson, *Architecture and Imple-mentation of a Java Package for Multiple Input Devices (MID)*. Technical Report, University of Maryland 1999. http://www.cs.umd.edu/hcil/mid

Hudson, S. E., J. Mankoff, and I. Smith. Extensible Input Handling in the subArctic Toolkit. *CHI: ACM Conference onHuman Factors in Computing Systems*. pp. 381–90, 2005.

IBM Pen Technologies, *CrossPad and TransNote*, 2007.http://researchweb.watson.ibm.com/electricInk

Jiang, H., R. B. Yeh, T. Winograd, and Y. Shi. DigiPost: Writing on Post-its with Digital Pens to Support CollaborativeEditing Tasks on Tabletop Displays. *UIST Extended Abstracts(Posters)*, 2007.

Johnson, W., H. Jellinek, L. K. Jr., R. Rao, and S. Card. Bridging the Paper and Electronic Worlds: The Paper User In-terface. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 507–12, 1993.

Kim, M., L. Bergman, T. Lau, and D. Notkin. An Ethnograph- ic Study of Copy and Paste Programming Practices in OOPL. *International Symposium on Empirical Software Engineering*.pp. 83–92, 2004.

Ko, A. J. and B. A. Myers. Designing the Whyline: A Debug- ging Interface for Asking Questions About Program Behavior. *Proceedings of the 2004 conference on Human factors in computing systems*. pp. 151-58, 2004.

LaToza, T. D., G. Venolia, and R. DeLine. Maintaining Men- tal Models: A Study of Developer Work Habits. *International Conference on Software Engineering*. pp. 492–501, 2006.

LeapFrog Enterprises, *FLY Pentop Computer*, 2007.http://www.flypentop.com

Liao, C., F. Guimbretière, and K. Hinckley. PapierCraft: A Command System for Interactive Paper. *UIST: ACM Sympo- sium on User Interface Software and Technology*. pp. 241–44,2005.

Liao, C., F. Guimbretière, and C. E. Loeckenhoff. Pen-top Feedback for Paper-based Interfaces. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 201–10,2006.

Livescribe Inc., *Livescribe*, 2007. http://www.livescribe.com

Lowagie, B. and P. Soares, *iText Java-PDF Library*, 2007.http://www.lowagie.com/iText
Mackay, W. E., G. Pothier, C. Letondal, K. Bøegh, and H. E. Sørensen. The Missing Link: Augmenting Biology Laboratory Notebooks. *UIST: ACM Symposium on User Interface Soft- ware and Technology*. pp. 41–50, 2002.
Microsoft, *Windows Forms*, 2007.http://www.windowsforms.net
Mutton, P., *Java EPS Graphics2D*, 2007.http://www.jibble.org/epsgraphics
Myers, B. A., J. F. Pane, and A. Ko. Natural Programming Languages and Environments, *Communications of the ACM*, vol. 47(9): pp. 47–52, 2004.
Nichols, J., *et al.* Generating Remote Control Interfaces for Complex Appliances. *UIST: ACM Symposium on User Inter-face Software and Technology*. pp. 161–70, 2002.
Norrie, M. C., B. Signer, and N. Weibel. Print-n-Link: Weav- ing the Paper Web. *DocEng: ACM Symposium on Document Engineering*, 2006.
Rosson, M. B. and J. M. Carroll. The Reuse of Uses in Small- talk Programming. *ACM Transactions on Computer-Human Interaction* **3**(3). pp. 219–53, 1996.
Schilit, B. N. and U. Sengupta. Device Ensembles. *Computer*
37(12). pp. 56–64, 2004.
Schleimer, S., D. S. Wilkerson, and A. Aiken. Winnowing: Local Algorithms for Document Fingerprinting. *SIGMOD: ACM International Conference on Management of Data*. pp.76–85, 2003.
Signer, B., *Fundamental Concepts for Interactive Paper and Cross-Media Information Spaces*, Unpublished PhD, ETH Zu-rich, Computer Science, Zurich, 2006. http://people.inf.ethz.ch/signerb/publications/signer16218.pdf
Stasko, J., J. Domingue, M. H. Brown, and B. A. Price, *Soft-ware Visualization: Programming as a Multimedia Experience*: MIT Press. 550 pp. 1998.
Stifelman, L., B. Arons, and C. Schmandt. The Audio Note- book: Paper and Pen Interaction with Structured Speech. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 182–89, 2001.
Sun Microsystems, *Swing*, 2007.http://java.sun.com/javase/6/docs
Thorn, T. Programming Languages for Mobile Code, *ACMComputing Surveys (CSUR)*, vol. 29(3): pp. 213–39, 1997.
Wellner, P. Interacting With Paper on the DigitalDesk, *Com-munications of the ACM*, vol. 36(7): pp. 87–96, 1993.
Wobbrock, J. O., A. D. Wilson, and Y. Li. Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User In-terface Prototypes. *UIST: ACM Symposium on User InterfaceSoftware and Technology*, 2007.
Yeh, R. B., J. Brandt, J. Boli, and S. R. Klemmer. Interactive Gigapixel Prints: Large, Paper-based Interfaces for Visual Context and Collaboration. *Ubicomp Extended Abstracts (Vid-eos)*, 2006.
Yeh, R. B., *et al.* ButterflyNet: A Mobile Capture and Access System for Field Biology Research. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 571–80, 2006.