# iStuff Mobile: Rapidly Prototyping New Mobile Phone Interfaces for Ubiquitous Computing

**Rafael Ballagas, Faraz Memon, René Reiners, Jan Borchers**
Media Computing Group
RWTH Aachen University
Aachen, Germany
{ballagas, borchers}@cs.rwth-aachen.de, {faraz.memon, rene.reiners}@rwth-aachen.de

**ABSTRACT**
iStuff Mobile is the first rapid prototyping framework that helps explore new sensor-based interfaces with existing mobile phones. It focuses on sensor-enhanced physical interfaces for ubiquitous computing scenarios. The framework includes sensor network platforms, mobile phone software, and a proven rapid prototyping framework. Interaction designers can use iStuff Mobile to quickly create and test functional prototypes of novel interfaces without making internal hardware or software modifications to the handset. A visual programming paradigm provides a *low threshold* for prototyping activities: the system is not difficult to learn. At the same time, the range of examples built using the toolkit demonstrates a *high ceiling* for prototyping activities: the toolkit places few limits on prototype complexity. A user study shows that the visual programming metaphor enables prototypes to be built faster and encourages more iterations than a previous approach.

**Author Keywords**
design, rapid prototyping, mobile phone, cell phone, sensor networks, Quartz Composer, visual programming.

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

## INTRODUCTION

The mobile phone is the first truly pervasive computer, making it an excellent interface for ubiquitous computing applications [2]. However, mobile phones are not built to suit research needs, and the mobile phone hardware is difficult to extend because of its commercial packaging. Kangas et al. [17] describe experience with iterative user-centered design when developing mobile applications:


*CHI 2007*, April 28–May 3, 2007, San Jose, California, USA.


**Figure 1. Back view of a mobile phone augmented with a Smart-Its sensor board in iStuff Mobile. The sensors can be attached to the phone in whatever position the designer finds most appropriate. The pictured Smart-It contains a 3D accelerometer, microphone, and sensors for light, pressure, temperature, and voltage.**

> *The most important aspect of the design process is to provide the user with the real usage context. For mobile phones this means users need to be able to touch the buttons and see software that feels like it is actually working.*

To address this need, iStuff Mobile focuses on building low-cost *functional* prototypes. Interaction designers and researchers can use this framework to create low-fidelity prototypes of applications that combine sensor-enhanced mobile phones and ubiquitous computing spaces. Testing functional prototypes with users allows design flaws to be identified sooner, before significant cost is invested in development. Additionally, the framework encourages exploration of many design alternatives due to the ease with which prototypes can be modified or interactive components exchanged. By decreasing the amount of time, money, and effort to produce functional prototypes, the framework encourages more iterations in the design process, which has been shown to increase the quality of user interfaces [23].

Many promising mobile phone interactions in ubiquitous computing require augmenting the mobile phone handset with additional sensors (e.g., [10, 12, 26]). However, electronically integrating new hardware into existing mobile phones is a large obstacle for most researchers and interaction designers. iStuff Mobile provides access to sensor-enabled mobile phones to those that have little electronics knowledge, by augmenting the mobile phone with externally attached hardware, such as Smart-Its [8] sensor net-

work modules (see Fig. 1). The loose collection of components is then made to interoperate with an intuitive visual programming environment. iStuff Mobile is intended for prototyping activities. After the design has been explored using iStuff Mobile, the final form factor can be refined for production with integrated sensors and optimized applications.

The main contribution of iStuff Mobile is introducing the *first* toolkit for rapid prototyping of new physical interactions with *existing* mobile phones (requiring no internal hardware changes). This radically simplifies prototyping UIs combining mobile phones with ubiquitous computing elements.

The primary parts of this contribution are concepts, not implementations: (1) the idea to create a toolkit that enables such prototyping with unmodified phone hardware and applications, (2) a software architecture that makes this possible, and (3) an appropriate feature set for such a toolkit.

The implementation-oriented parts of the contribution serve as proof of the above concepts. They are:

- a background cell phone application that allows functional prototypes to be built with *any* foreground application, even those built in,
- an extension of Apple's Quartz Composer[1] that makes it suitable for physical prototyping, and
- a sensor network proxy to configure sensors without having to change their embedded software.

To ease dissemination, iStuff Mobile concentrates on software and uses off-the-shelf hardware. With hardware toolkits becoming mature and commercialized, we propose moving past building yet another one, and using them as a layer to build software upon instead. This software, to avoid reinventing the wheel, also builds on top of existing, robust or even commercialized, software components such as iStuff [3], the Event Heap [14], and Quartz Composer. iStuff by itself did not support cell phones, visual programming, or sensor networks.

## RELATED WORK

### How to Evaluate Toolkits

Myers et al. [22] identify several characteristics, or "themes", for evaluating toolkits. In particular, the *threshold* describes how difficult it is to learn to use the system. The *ceiling* describes how much can be accomplished using the system.

Conducting a formal evaluation of a toolkit or software framework is an extremely difficult task. The qualities that are important for toolkits, such as development effort, are difficult to measure directly. Evaluating a software framework has issues similar to evaluating middleware. Edwards et al. [7] point out that although we have good techniques for designing and evaluating interactive applications, we are lacking well formed techniques for designing and evaluating the infrastructure to support application development. Klemmer et al. [18] provide a detailed discussion of this topic, and describe their various approaches to evaluate the Papier-Mâché toolkit. One approach is to measure the efficiency (e.g., development time, or lines of code) of developers while using a toolkit, but efficiency is also related to the quality of the resulting prototype, making these metrics difficult to isolate.

### Physical Hardware Toolkits

As traditional paper prototypes for desktop GUIs prove less adequate for the ubicomp domain [21], a variety of hardware toolkits have emerged in recent years to help prototype physical interactions. BOXES [13] provides a prototyping solution using common household items, to simplify early stage exploration of form factor. Toolkits like Phidgets [9], Teleo[2], Calder [19], VoodooIO [27] and Smart-Its[3] [8] provide a set of reusable hardware components with accessible APIs to reduce the barriers of physical device prototyping. d.tools [11] provides a set of software tools in addition to hardware components to support the full range of design, testing, and analysis activities in an iterative design cycle. Like iStuff [3], d.tools [11] also provides an extensible software framework that allows for multiple hardware platforms to be combined during prototyping activities. iStuff Mobile builds on top of this previous work; it provides a prototyping solution that simplifies exploring new interactions that combine existing mobile phones with many of the above hardware component toolkits in ubiquitous computing application scenarios.

### Software prototyping environments

iStuff Mobile is built on top of the iStuff [3] framework. A key component of the iStuff framework is the Patch Panel [5]. It intercepts and rewrites messages to allow otherwise incompatible components to communicate over a network. Its strength is that mappings between inputs and outputs can be specified dynamically at run-time, making it easier to modify a design on-the-fly. To date, the mappings in the Patch Panel are primarily specified through a light scripting language introduced in [5].

This work introduces a new interface for physical prototyping by extending Apple's Quartz Composer visual programming environment. Quartz Composer provides a cable patching metaphor to lower the threshold for prototype construction. It is very similar to Max/MSP[4] and LabVIEW[5] in that it targets developers, researchers, and interaction designers. They differ in that Max/MSP's strength is audio processing, LabVIEW is geared towards electrical signal analysis, and Quartz Composer focuses on interactive multimedia and 3D rendering. Quartz Composer is a live editor, whereas Max/MSP and LabVIEW have separate edit and run modes. d.tools [11] supports visual programming through a visual

[1] http://developer.apple.com/documentation/GraphicsImaging/Conceptual/QuartzComposer/qc_intro/chapter_1_section_1.html

[2] http://www.makingthings.com/teleo.htm

[3] For Smart-Its hardware, see http://www.particle-computer.net

[4] http://www.cycling74.com/products/maxmsp

[5] http://www.ni.com/labview/

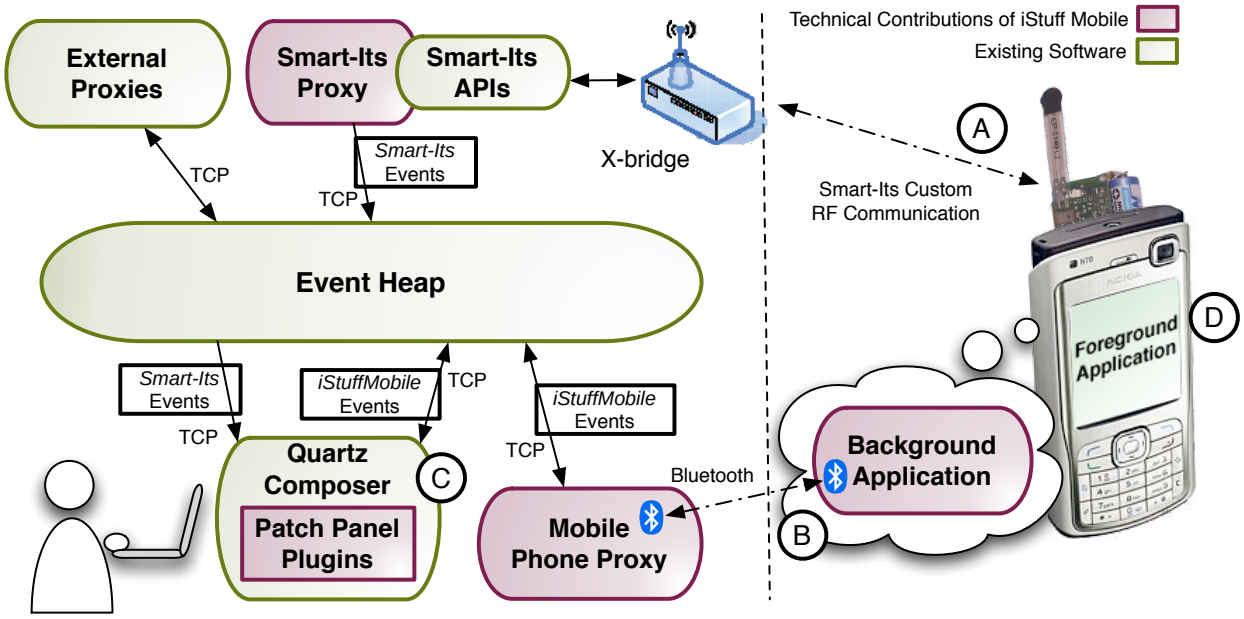


**Figure 2. The iStuff Mobile architecture. (A) When the phone is moved, for example, the Smart-Its sensor board transmits the resulting sensor data wirelessly to the Smart-Its X-bridge. Our Smart-Its Proxy collects that data from the X-bridge over ethernet, and encapsulates it into *Smart-Its* events. (B) Mobile phone input events (such as key presses) are intercepted by the background application and passed to the Mobile Phone Proxy over a Bluetooth connection, and the proxy encapsulates the data into *iStuffMobile* events. (C) The Quartz Composer GUI is extended with special plugins for the Patch Panel to transform input events into desired output events. (D) For mobile phone output, the Mobile Phone Proxy listens for *iStuffMobile* events, and passes the resulting commands to the mobile phone background application over the Bluetooth connection. The background application then either executes the command directly, or forwards it to the foreground application as appropriate.**

statechart editor; the Calder toolkit [19] supports the Macromedia Director[6] development environment; Phidgets [9] and Teleo provide hooks to work with Max/MSP and Adobe Flash[7] as development environments.

### Mobile phone interface prototypes

The TEA project [26] uses a predecessor of the Smart-Its platform to demonstrate mobile phone context interactions. Harrison [10] and Hinckley [12] built custom PDAs and mobile phones with integrated sensor hardware. These projects broke new ground and demonstrated a broad vision for what types of interfaces are possible for mobile phones, but the focus was the interfaces, not the development process. iStuff Mobile focuses on providing a reusable prototyping framework for research and design scenarios like these.

### Mobile phone toolkits

d.tools [11] allows designers to rapidly prototype handheld devices including mobile phones. Its strength is the ability to explore different mobile phone form factors and sensor placements. However, its lack of support for critical phone functionality, such as voice calls, and its wired sensors currently limit the toolkit's ceiling for many mobile phone application scenarios. Topiary [20] is a toolkit for building Wizard of Oz prototypes of location-based applications on existing handheld devices. While valuable for location-based scenarios, it does not address other types of physical sensors, and does not simplify the construction of functional prototypes. iStuff Mobile is the first toolkit to allow rapid prototyping of functional physical sensor-based interactions with existing mobile phones.

[6]http://www.adobe.com/products/director/
[7]http://www.adobe.com/products/flash/flashpro/

## ISTUFF MOBILE ARCHITECTURE

iStuff Mobile is designed as a compound prototype architecture [1] where part of the software is distributed across separate computers. This compound architecture, shown in Fig. 2, allows interface designers to prototype interactions that may be beyond the capabilities of current mobile phone hardware. In addition, this architecture provides communication capabilities necessary for ubiquitous computing application scenarios. The disadvantage of the compound architecture is that it spatially restricts experiments. While a direct communications channel between the sensors and the phone (e.g., through a Bluetooth connection) may be more efficient and less spatially restrictive, this would eliminate the prototyping benefits gained from using the Quartz Composer visual interface, which allows the relationships between user activity and application feedback to be changed at run-time using a comfortable desktop GUI. The delay between user action and application feedback is small enough to easily maintain causality [6] and support a wide range of tasks including continuous pointing tasks which require a latency much less than 100ms.

iStuff Mobile is built on top of the iStuff [3] framework. The benefit of this decision is that iStuff, designed for ubiq-

uitous computing scenarios, already supports a variety of hardware component toolkits such as Phidgets and Teleo. The relevant components in the system can be distributed on different computers across a room, and they communicate through the Event Heap infrastructure, a tuplespace with publish-subscribe semantics [14]. This indirect communications model allows clients to communicate without an explicit rendezvous. As an illustration, consider a presentation controller that sends events to the Event Heap. The presentation software subscribing to the event can be seamlessly moved to a different machine, or a second instance of the presentation software can be used without making any changes to the way the presentation controller posts events. For components that are not designed to communicate with the Event Heap, such as Smart-Its and the mobile phone, iStuff employs a proxy strategy, where an external process communicates directly (e.g., through custom wireless protocols or application hooks) with the devices and sends or receives events on their behalf. This proxy strategy promotes extensibility and simplifies incorporating additional hardware components to the library of reusable building blocks.

#### Mobile Phone application support

The iStuff Mobile architecture divides the mobile phone application into two parts. The *foreground* application is what the user interacts with during testing. The *background* application, provided by iStuff Mobile, is designed to simplify the work of the interaction designer creating the functional prototype. It is not directly visible to the user.

Designers can remotely execute commands on the phone by sending *iStuffMobile* events to the desktop-based Mobile Phone proxy, which relays the commands to the background application on the phone via a Bluetooth connection (see Fig. 2). The background application relays the commands to the foreground application or the operating system as appropriate. The background application can also intercept user actions, such as key presses, from the foreground application, which are relayed to the proxy over the Bluetooth connection and subsequently posted as events on the Event Heap. The prototype implementation of the background application on the mobile phone was designed to include the following feature set. It does not cover the entire design space of interaction possibilities, but it does enable a wide range of interesting interactions, and the architecture encourages expansion to include more features.

1. *Bluetooth Communication*: communicate with the proxy through a low-latency wireless communications channel.
2. *Sound Playback*: trigger available sounds to be played and stopped.
3. *Vibrator Control*: trigger the vibrator to start and stop.
4. *Key Capture*: intercept key events from the foreground application and relay them to the proxy for processing.
5. *Foreground Application Key Simulation*: pass key events to the foreground application.
6. *Launch External Application*: launch any application on the mobile phone.
7. *Profile Control*: programmatically change the ring profile of the mobile phone.
8. *Backlight*: control the backlight programmatically.
9. *Run Application in Background*: send the current foreground application to the background.
10. *Camera Control*: use the camera on the mobile phone for taking pictures, videos, or interactions using motion estimation such as the Sweep technique [4].

We have built a prototype implementation of the background application using the Symbian Series 60 operating system. Our analysis shows that Windows Mobile 5.0 SmartPhone Edition would be a good candidate for porting the iStuff Mobile background application because it shares many of the same capabilities as Symbian Series 60. Java 2 Micro Edition, on the other hand, is currently not a candidate platform as the background application would be lacking critical functionality. We believe that it is possible to port the background application to Linux-based mobile phone platforms such as the Motorola E680i, but open source development efforts on these phones are still in their early stages.

The *foreground* application is the application the user sees and interacts with. iStuff Mobile is designed to be used with any foreground application and communicates primarily through system events (e.g., key presses). Designers are expected to prototype their own mobile phone application using rapid prototyping solutions such as static images, Flash Lite, or a scripting language like Python. Alternatively, designers have the option to program their own application using Java[8] or native code. Lastly, designers can prototype interactions with existing foreground applications that come with the mobile phone, such as its Address Book or Calendar, despite the fact that these applications were not explicitly designed to accommodate new interaction techniques.

#### Sensor network support

Sensor network modules have been shown to be valuable tools in rapid prototyping scenarios [8]. Sensor network applications require low-cost, low-power, wireless sensors. These requirements are pushing devices to be cheaper, smaller, and more widely available commercially, all of which are advantageous for a reusable prototyping framework.

For the initial version of iStuff Mobile, we chose to support the Smart-Its [8] sensor network platform. Smart-Its provides a remote procedure call interface that allows reconfiguration of sensors without modifying the code on the embedded "particle" sensor boards. In iStuff Mobile, we leveraged this capability to develop a cross-platform GUI that combines configuration of sensors, reception of sensor data, and posting of sensor data onto the Event Heap. Each Smart-Its module comes with an array of sensors (3D accelerometer, microphone, and sensors for light, pressure, temperature and voltage). The GUI allows designers to rapidly configure

[8] http://java.sun.com/j2me/

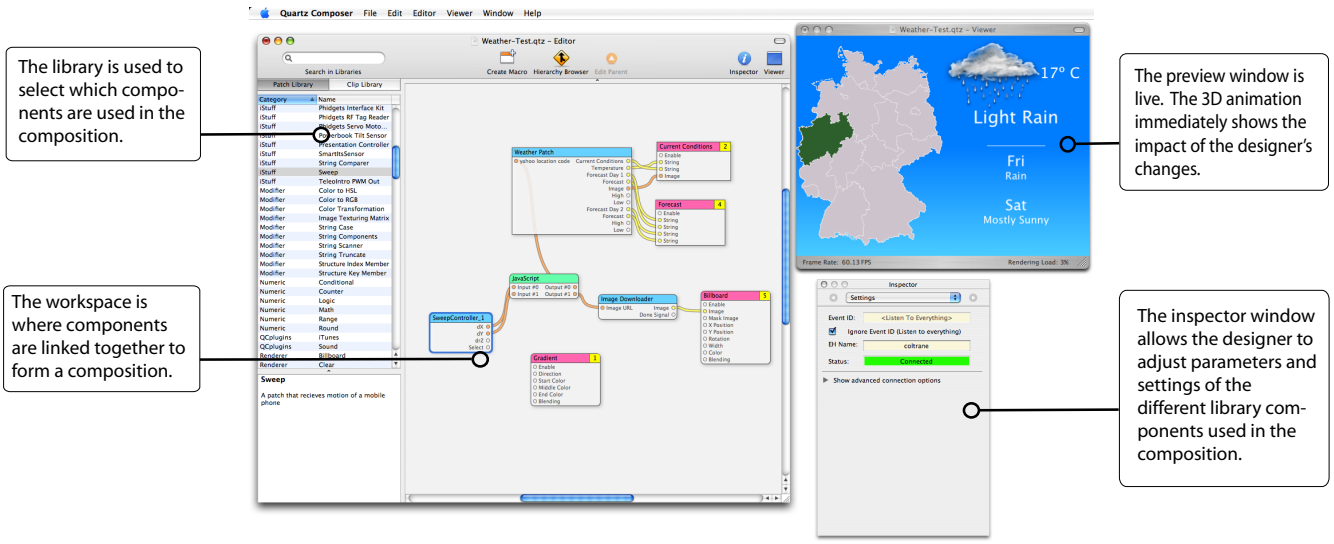


**Figure 3. Apple's Quartz Composer is a visual programming environment designed to support rapid creation of 3D interactive visualizations. We have extended it to support prototyping physical user interfaces. This screenshot shows the development of a weather application for a large public display in a train station. The user can navigate through the different regions of the map by waving their phone through the air using the Sweep [4] technique, and the corresponding weather data is updated live through RSS feeds.**

Smart-Its to activate the appropriate sensors to work with the iStuff framework in a particular prototyping scenario.

### Rapid Prototyping through Visual Programming

Quartz Composer is a visual programming environment (see Figure 3) that is part of Apple's freely available Xcode development environment. It was introduced with Mac OS X 10.4 "Tiger". It uses a cable patching metaphor to establish data and control flow between different components, establishing a composition. The editor is live, and changes made in the workspace are immediately functional without any compilation steps. In iStuff Mobile, we have extended the Quartz Composer environment to enable prototyping of physical interfaces. We added library components for each of the iStuff proxies and new data processing modules that are particularly useful in physical prototyping scenarios.

iStuff Mobile is designed so that other sensor network platforms can be easily substituted in place of Smart-Its. In order to incorporate a different sensor network platform, all that needs to be implemented is a new Event Heap proxy to send and receive events on the sensor module's behalf, and a new Quartz Composer plugin to enable control of the information flow in the visual programming environment.

## RECREATING SEMINAL MOBILE PHONE INTERACTIONS

To demonstrate the utility of the framework, we have used it to recreate several mobile phone interactions discussed in previous literature. Harrison et al. [10] introduced a tilt-scrolling interaction for mobile devices. The implementation consisted of a PDA augmented with an accelerometer and pressure sensors. To activate tilt-scrolling, the user squeezes the sides of the device with her thumb and forefinger. The more the user tilts, the faster the device scrolls.

### Working with iStuff Mobile

To give the reader a feel for what it is like to work with the iStuff Mobile framework, we present a typical work session scenario based on the tilt-scroll example above:

> Tom is a UI designer who wants to experiment with the tilt-scroll interaction introduced by Harrison et al. [10]. To set up his prototyping environment, Tom runs the Proxy Manager on his desktop computer that is part of the iStuff framework. It provides a front end to quickly launch specific proxies on his desktop computer. The Proxy Manager discovers an Event Heap running on the local network upon launch. Using the Proxy Manager, Tom launches the Smart-Its Particle Framework proxy and the iStuff Mobile phone proxy that mediates communication between the phone and the Event Heap.
>
> Tom then grabs a Smart-Its Particle sensor board from a charging station, and uses the Particle Framework proxy to discover the particle, activate its X/Y/Z accelerometer sensors, and then start posting events from the sensor board to the Event Heap. He tapes the Smart-It to the back of the mobile phone. Then he launches the iStuff Mobile background application on the mobile phone and uses it to establish a Bluetooth connection to the iStuff Mobile phone proxy on his desktop computer. Next, he launches the native Contacts application on the phone so that he has a lot of data to scroll through.
>
> Now Tom is ready to map the tilting motion to control scrolling. He launches Apple's Quartz Composer with the iStuff extensions installed. He types "Sensor" in the library search window and selects the "SmartItsSensor" module to import it into his workspace. He then types "Phone" into the library search window and

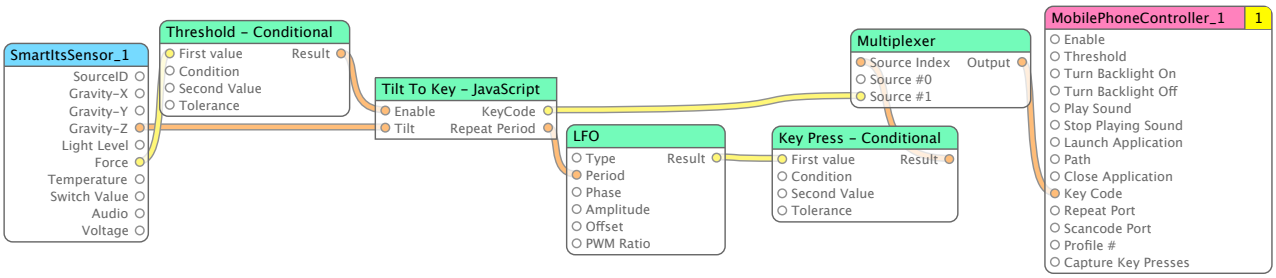


**Figure 4. The Quartz Composer implementation of the tilt-scrolling interaction from [10]. Squeezing input is measured by the "Force" node from the *SmartItsSensor_1* and is tested with a simple threshold. The result is passed to the *Tilt To Key - JavaScript*, which maps various tilts in the Z-direction of the gravity sensor to different key codes and key repeat rates. The outputs from that JavaScript node include "KeyCode", which represents the appropriate key (up or down arrow) depending on the current tilt, and "Repeat Period", which specifies how fast the *LFO* (low frequency oscillator) node should operate. For this scenario, larger tilt is mapped to faster repeat rates. The *Key Press - Conditional* changes the oscillator to function like a binary clock, regularly switching between 0 and 1. "Source #0" (which defaults to 0) represents no key pressed, and "Source #1" represents the key specified from the JavaScript node. The key is then passed to the *MobilePhoneController_1* to forward to the mobile phone. The naming convention of the iStuff Mobile related nodes corresponds to the name of the device being controlled. (*_1* helps distinguish multiple devices of the same type.)**

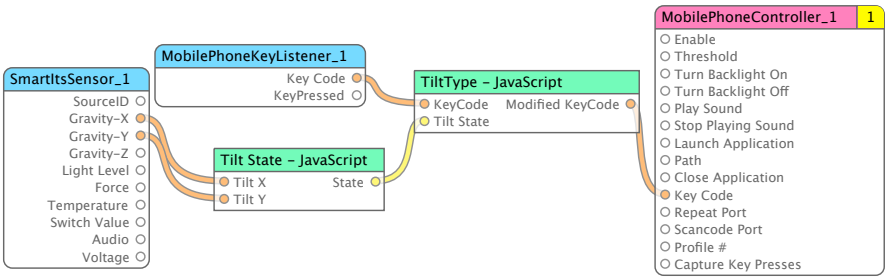


**Figure 5. The TiltText [28] technique maps numeric keys to different characters based on the tilt of the device.**

selects the "Mobile Phone" module. He can't link the accelerometer directly to the mobile phone because there needs to be some simple interpretation of the sensor data to map sensor data thresholds to individual key presses. He decides to use the built-in JavaScript module to accomplish this task.

Tom doesn't know exactly what sensor values to use for the thresholds so he uses native Quartz Composer modules to display his sensor readings directly to the live preview window. He observes the sensor data as he tests different tilt positions with the phone. He settles on some sensor thresholds and writes a small JavaScript that reacts to sensor thresholds by sending directional arrow key presses to the phone. He links the output of the JavaScript module to the Mobile Phone module. Now, tilting the phone triggers a single key press. He's pleased with his progress, but he wants the scrolling to continue as long as the phone is tilted and the rate of scrolling should change based on the degree of tilting. He does this by adding LFO and Multiplexer modules (as shown in Fig. 4), finishing his prototype.

We were also able to recreate some context-aware interactions described in the literature. Schmidt et al. [26] and Hinckley et al. [12] describe a scenario where the mobile phone ring tone profile is automatically switched to vibrator-only when the mobile phone is in the user's hand, since an audio notification is unnecessary in that situation. This interaction was recreated using the Smart-Its pressure sensor to detect when the user was holding the phone. Pressure sensor activity triggers a command to the mobile phone proxy to switch the ring tone profile, and inactivity triggers the command to switch the ring tone back.

As another example, the TiltText [28] technique simplifies text entry using a numeric keypad by adding tilting. Tilting the phone to the left activates the first letter, tilting upward activates the second letter, tilting to the right activates the third letter, tilting downward activates the fourth letter (applicable for keys '7' and '9') and no tilt activates the standard numeric character. This technique has been shown to be faster than MultiTap and comparable to dictionary-based techniques. Again, we were able to quickly recreate this interaction using the iStuff Mobile framework (see Fig. 5). This example is a good illustration of how capable Quartz Composer is of expressing state-based interfaces. In the implementation, the tilt state is determined through the *Tilt State - JavaScript* node and passed to the *TiltType - JavaScript* node, which modifies the key presses based on the current state. Note that since there are 5 different states for 12 different phone keys, this would be cumbersome to model as a state machine, but it is fairly easily modeled using Quartz Composer.

## UBIQUITOUS COMPUTING PROTOTYPING SCENARIOS

To show how the framework goes beyond the localized sensor-based interactions described so far, we will now

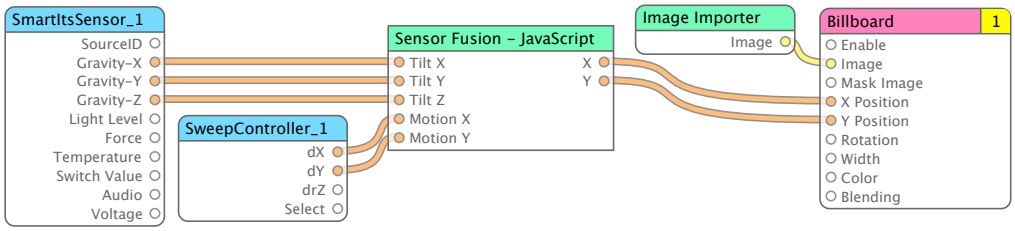


**Figure 6. The Quartz Composer implementation for combining accelerometer data with camera-based motion detection to improve motion detection accuracy. The *Sensor Fusion - JavaScript* node implements the algorithm to combine the sensor values in a meaningful way. The JavaScript logic can be modified at run-time to test and refine the sensor fusion strategy. The standard *Billboard* node of Quartz Composer displays an image to the screen (e.g., a cursor). The output of the sensor fusion algorithm in the JavaScript node controls the position of the billboard on the screen.**

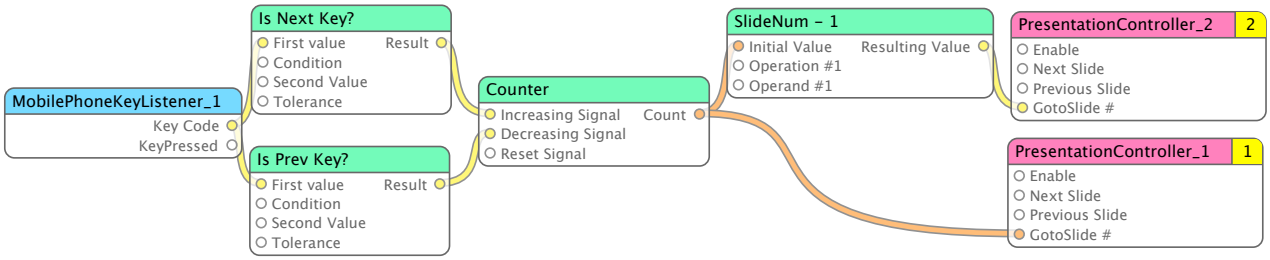


**Figure 7. The Quartz Composer implementation for a multi-screen presentation application. On the far left, the *MobilePhoneKeyListener_1* node receives the key presses from the iStuff Mobile Proxy. The two nodes on the far right are iStuff modules to control two instances of the same PowerPoint presentation, each running on a different computer in an interactive workspace. No JavaScript nodes are required for this composition.**

demonstrate how it simplifies prototyping entire ubiquitous computing scenarios.

### The mobile phone as a ubiquitous input device

The idea of using the mobile phone as an input device for ubiquitous computing application scenarios is very compelling because the phone is almost always with us [24]. One technique that has demonstrated some potential for this use is the Sweep [4] interaction technique, which uses camera-based motion estimation to allow the phone to be used as a relative pointing device for public displays. The motion estimation algorithm on the low power mobile processor is not perfect and suffers from some estimation errors. We used the iStuff Mobile framework to combine accelerometer data with the camera information to improve the motion estimation, as shown in Fig. 6. This allows the mobile phone to serve as a more accurate pointing device, for example when interacting with public displays. The synergies of choosing Quartz Composer as a visual programming environment are demonstrated in this scenario because Quartz Composer makes authoring visually compelling interactive graphics very simple. This means that the iStuff Mobile input techniques can directly drive these 3D visualizations and the framework provides end-to-end prototyping assistance.

Mobile phones have also emerged as popular presentation remote controls, due in part to the success of tools like Salling Clicker[9]. But ubicomp environments like interactive workspaces [15] have multiple screens that can be taken advantage of to enhance the presentation. As a proof of concept, we developed a multi-screen presentation interface (see Fig. 7). In this scenario, one screen is showing the current slide of the presentation, while the second screen is showing the presentation history. By pressing a key on the mobile phone, the user is remotely controlling a series of PowerPoint applications to create this effect. These presentations are controlled through simple proxies running on different machines in the interactive workspace. Each proxy listens for events with different names (e.g. *PresentationController_1* ) so that they can be individually controlled.

[9]http://www.salling.com/Clicker/

### Using ubiquitous resources as an interface to the phone

Ubiquitous computing environments such as interactive workspaces [15] are rich in input and output capabilities, including touch sensitive wall-sized displays, and interactive tabletop displays. PointRight [16] demonstrates a system that lets users redirect the mouse and keyboard input to the different computers in the room. We wanted to demonstrate a system that would also let users redirect their keyboard input to a mobile phone. The resulting prototype is shown in Fig. 8.

Using an almost identical configuration, we were able to prototype a scenario where the user could dictate text to the phone using continuous speech recognition. Mobile phones are years away from having the processing ability to support such continuous speech recognition, but iStuff Mobile enabled us to create a functional prototype today.

## EVALUATION

New toolkits are typically validated by the breadth of coverage in the designs they support [3, 9, 18] and their ability to recreate important interactions from the literature more easily [18]. Similarly, the high ceiling of iStuff Mobile is demonstrated by the range of prototypes it enables.

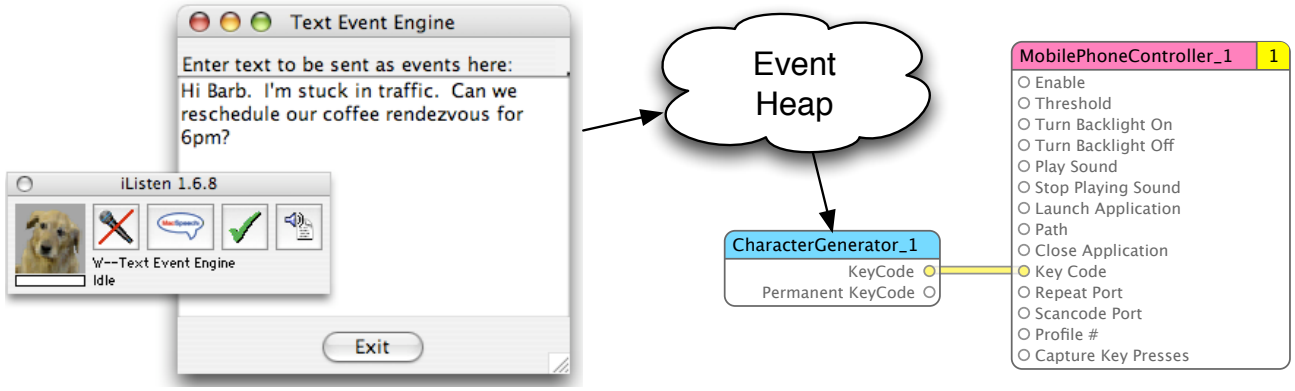


**Figure 8. (Left) Our Text Event Engine is a Java application that produces *Character* events for each key entered in the textbox. The window floating above it belongs to iListen, a commercial application that supports continuous speech recognition (dictation) on Mac OS X. In this example, the user is dictating an SMS message. (Right) System key events are recognized by the *CharacterGenerator_1* and transferred to *MobilePhoneController_1*. This composition can alternatively be used to allow users to type messages onto their mobile phone using a standard keyboard on their desk.**

The rest of the evaluation is designed to show that iStuff Mobile has a low prototyping threshold. In this evaluation, we examine the efficiency of prototyping with the iStuff Mobile toolkit using development time and number of prototype iterations as the primary metrics. In the case of Quartz Composer, the lines of code metric is less relevant since only a small part of the modeling is done textually in JavaScripts.

In order to experimentally evaluate iStuff Mobile, we chose to compare the effectiveness of the new visual programming paradigm to the established Patch Panel scripting language introduced in [5]. This comparison was chosen to reduce the risk of task bias inherent in the comparison of different toolkits: in our evaluation, the underlying toolkits and hardware are the same.

To further reduce task bias, we chose to examine four different design problems, each making use of different hardware components.

1. **Multi-screen presentation**. Participants were asked to prototype a presentation interface that used the mobile phone to control a presentation across several different screens. The station was equipped with a mobile phone and two remote machines connected to large LCD displays. The remote machines were running PowerPoint presentation software and connected to the design station via ethernet.

2. **Tilt-to-scroll**. Participants were asked to prototype a mobile phone interface that allows scrolling through large lists of data by tilting the phone similar to [10]. The station was equipped with a mobile phone and several Smart-Its sensors.

3. **Handheld music player**. Participants were asked to prototype a new handheld music player, including song selection and volume control functionality. The station was equipped with a variety of Phidgets sensors, and the iTunes software music player. This task was chosen to provide some level of comparability to d.tools and BOXES, where the portable music player serves as a prominent example.

4. **Remote steering**. Participants were asked to design a remote control mechanism for a model boat, where physical input would control an electronic motor to steer a boat rudder. The station was equipped with Phidgets sensors for input and Phidgets servo-motors and cardboard to build a low-fidelity boat prototype.

The test consisted of 16 participants (10 male, 6 female) from a computer science course. On average, the students were in their 4th year of university studies. All students had completed lectures on Human Computer Interaction and were familiar with the fundamentals of interaction design, prototyping, and iterative design.

The test was structured such that the group of participants received training before the design exercises with the following time schedule:

*0:00 – 0:30*: Introduction to iStuff prototyping principles
*0:30 – 1:00*: Training for Prototyping Environment 1
*1:00 – 1:30*: Design Task 1
*1:30 – 2:00*: Design Task 2
*2:00 – 2:30*: Training for Prototyping Environment 2
*2:30 – 3:00*: Design Task 3
*3:00 – 3:30*: Design Task 4

The test was designed as a within group study, with special care taken to avoid learning effects both in terms of the tasks completed and the prototyping environments used. The 16 participants were split up into 2 shifts of 8 and further split into 4 teams of 2 participants. During each shift, there was one team at each test station attempting one of the 4 design tasks described earlier. After each task, the teams rotated to a different station such that each team performed the design tasks in a different order. The first shift of 4 teams completed their first 2 design tasks using the Quartz Composer GUI,

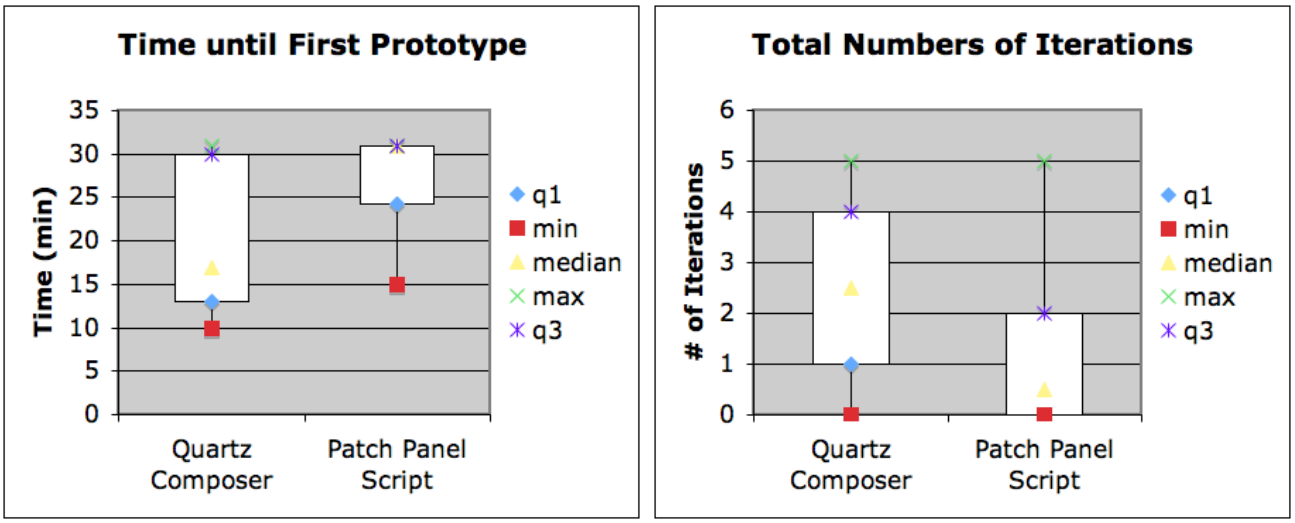


**Figure 9. Results show that Quartz Composer is significantly faster and enables significantly more iterations than the Patch Panel scripting language.**

then their last 2 tasks with the Patch Panel script. The second shift of 4 teams used the prototyping environments in the opposite order. Participants were instructed to stop after 30 minutes, regardless of whether or not they had a functional prototype ready.

We measured the time it took for the participants to build their first functional prototype, and the number of iterations completed in the time allotted. For this study, we define functional prototype as an artifact that successfully implements at least a portion of the functionality described in the design task. We define a design iteration as a full DIA cycle (design, implement, analyze). Every time the participants created a functional prototype, analyzed the problems, and made changes to the design we counted it to be a design iteration.

## RESULTS

Results from the experiment are shown using box-plots in Fig. 9. In the time measurement plot, if a group was unable to complete a prototype in the allotted 30 min., we assigned a time of 31 minutes to maintain the visual integrity of the box-plots, but omitted these data points during statistical analysis. The average time for the first functional prototype was 19.6 minutes for the Quartz Composer GUI, and 27.6 minutes for the Patch Panel script. However, we are unable to show that this difference is statistically significant because of the large number of incomplete first iterations for the Patch Panel script. Participants were able to complete at least one iteration of the test in the time alloted 81% of the time with Quartz Composer, compared to only 31% with the Patch Panel script ($p < 0.05$, Fisher's test). Participants were also able to complete an average of 2.5 iterations using the Quartz Composer GUI compared to 0.9 iterations using the Patch Panel scripting language ($p < 0.05$, Student's t-test). The combined results show that the Quartz Composer GUI is significantly faster than the Patch Panel script in building prototype iterations. Qualitative results from the study can be found in [25].

Clearly, there are disadvantages to this experimental design since the comparison is limited and doesn't reflect the range of prototyping approaches that exist today. The main takeaway is that the study demonstrates a low prototyping threshold: with very little training, users could often build early prototypes in less than 30 minutes. The Patch Panel scripting language serves as a familiar point of reference to interpret the results.

## CONCLUSIONS

The quality of a user interface tends to increase with more iterations in the design process, motivating the need for rapid prototyping solutions. The iStuff Mobile framework is the first to simplify the exploration of new mobile phone interactions in ubiquitous computing environments. We have demonstrated how it can be used to easily recreate seminal sensor-enabled mobile phone applications, and to significantly simplify the integration of mobile phone interactions into ubiquitous computing scenarios. Our evaluation shows that the visual programming environment allows prototypes to be built faster and encourages more design iterations to be performed. By making this tool available as open source[10] to the interaction design and research community, we hope to advance the pace of innovation and improve the quality of interface designs in ubiquitous computing.

## REFERENCES

1. Abowd, G. D., Hayes, G. R., Iachello, G., Kientz, J. A., Patel, S. N., Stevens, M. M., and Truong, K. N. Prototypes and paratypes: Designing mobile and ubiquitous computing applications. *IEEE Pervasive Computing* (2005), 67–73.

2. Abowd, G. D., Iftode, L., and Mitchell, H. Guest editors' introduction: The smart phone–a first platform for pervasive computing. *IEEE Pervasive Computing 4*, 2 (2005), 18–19.

[10]http://istuff.berlios.de

3. Ballagas, R., Ringel, M., Stone, M., and Borchers, J. iStuff: A Physical User Interface Toolkit for Ubiquitous Computing Environments. *Proc. CHI '03*. ACM Press (New York, NY, USA, 2003), 537–544.

4. Ballagas, R., Rohs, M., Sheridan, J. G., and Borchers, J. Sweep and Point & Shoot: Phonecam-based interactions for large public displays. *Extendend abstracts of CHI '05*. ACM Press (New York, NY, USA, 2005), 1200–1203.

5. Ballagas, R., Szybalski, A., and Fox, A. Patch Panel: Enabling Control-Flow Interoperability in Ubicomp Environments. *PerCom '04. Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications*. IEEE (Orlando, FL, USA, 2004).

6. Card, S. K., Newell, A., and Moran, T. P. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Inc., 1983.

7. Edwards, W. K., Bellotti, V., Dey, A. K., and Newman, M. W. The challenges of user-centered design and evaluation for infrastructure. *Proc. CHI '03*. ACM Press (New York, NY, USA, 2003), 297–304.

8. Gellersen, H., Kortuem, G., Schmidt, A., and Beigl, M. Physical Prototyping with Smart-Its. *IEEE Pervasive Computing 3*, 3 (2004), 74–82.

9. Greenberg, S., and Fitchett, C. Phidgets: Easy development of physical interfaces through physical widgets. *Proc. UIST '01*. ACM Press (New York, NY, USA, 2001), 209–218.

10. Harrison, B. L., Fishkin, K. P., Gujar, A., Mochon, C., and Want, R. Squeeze me, hold me, tilt me! An exploration of manipulative user interfaces. *Proc. CHI '98*. ACM Press/Addison-Wesley Publishing Co. (New York, NY, USA, 1998), 17–24.

11. Hartmann, B., Klemmer, S. R., Bernstein, M., Abdulla, L., Burr, B., Robinson-Mosher, A. L., and Gee, J. Reflective physical prototyping through integrated design, test, and analysis. *Proc. UIST '06*. ACM Press (2006), 299–308.

12. Hinckley, K., and Horvitz, E. Toward more sensitive mobile phones. *Proc. UIST '01*. ACM Press (New York, NY, USA, 2001), 191–192.

13. Hudson, S. E., and Mankoff, J. Rapid Construction of Functioning Physical Interfaces from Cardboard, Thumbtacks, Tin Foil and Masking Tape . *Proc. UIST '06*. ACM Press (2006), 289–297.

14. Johanson, B., and Fox, A. The Event Heap: A Coordination Infrastructure for Interactive Workspaces. *WMCSA '02: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*. IEEE (2002), 83.

15. Johanson, B., Fox, A., and Winograd, T. The interactive workspaces project: experiences with ubiquitous computing rooms. *IEEE Pervasive Computing 1* (2002), 67–74.

16. Johanson, B., Hutchins, G., Winograd, T., and Stone, M. PointRight: experience with flexible input redirection in interactive workspaces. *Proc. UIST '02*. ACM (2002), 227–234.

17. Kangas, E., and Kinnunen, T. Applying user-centered design to mobile application development. *Commun. ACM 48*, 7 (2005), 55–59.

18. Klemmer, S. R., Li, J., Lin, J., and Landay, J. A. Papier-mâché: toolkit support for tangible input. *Proc. CHI '04*. ACM Press (New York, NY, USA, 2004), 399–406.

19. Lee, J. C., Avrahami, D., Hudson, S. E., Forlizzi, J., Dietz, P. H., and Leigh, D. The Calder toolkit: wired and wireless components for rapidly prototyping interactive devices. *DIS '04: Proceedings of the 2004 Conference on designing interactive systems*. ACM Press (New York, NY, USA, 2004), 167–175.

20. Li, Y., Hong, J. I., and Landay, J. A. Topiary: a tool for prototyping location-enhanced applications. *Proc. UIST '04*. ACM Press (New York, NY, USA, 2004), 217–226.

21. Liu, L., and Khooshabeh, P. Paper or interactive?: a study of prototyping techniques for ubiquitous computing environments. *Proc. CHI '03*. ACM Press (New York, NY, USA, 2003), 1030–1031.

22. Myers, B., Hudson, S. E., and Pausch, R. Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact. 7*, 1 (2000), 3–28.

23. Nielsen, J. Iterative user-interface design. *Computer 26*, 11 (1993), 32–41.

24. Patel, S. N., Kientz, J. A., Hayes, G. R., Bhat, S., and Abowd, G. D. Farther than you may think: An empirical investigation of the proximity of users to their mobile phones. *Proc. Ubicomp '06*. Springer (2006), 123–140.

25. Reiners, R. The Patch Panel GUI: A Graphical Development Environment For Rapid Prototyping Interfaces For Ubicomp Environments. Master's thesis, RWTH Aachen University, Aachen, Germany, 2006.

26. Schmidt, A., Aidoo, K. A., Takaluoma, A., Tuomela, U., Laerhoven, K. V., and de Velde, W. V. Advanced interaction in context. *HUC '99: Proceedings of the 1st international Symposium on Handheld and Ubiquitous Computing*. Springer-Verlag (London, UK, 1999), 89–101.

27. Villar, N., Gilleade, K., Raymundy-Ellis, D., and Gellersen., H. The VoodooIO gaming kit: a real-time adaptable gaming controller. *ACE '06: Advances in Computer Entertainment*. ACM Press (New York, NY, USA, 2006).

28. Wigdor, D., and Balakrishnan, R. TiltText: using tilt for text input to mobile phones. *Proc. UIST '03*. ACM Press (New York, NY, USA, 2003), 81–90.