Amarino: A Toolkit for the Rapid Prototyping of Mobile Ubiquitous Computing

Bonifaz Kaufmann and Leah Buechley
MIT Media Lab
High-Low Tech Group
Cambridge, MA 02139, USA
bonifaz@mit.edu, leah@media.mit.edu

ABSTRACT
Ubicomp applications increasingly involve smart phones that control or communicate with embedded systems. Compelling examples in this space include tangible interfaces, environmental sensor networks, game controllers and automated homes. Across research, design, and hobbyist communities there is clearly a desire to build applications that involve combinations of mobile and non-mobile technologies. However, constructing these applications is a laborious process that requires considerable breadth and depth of expertise in programming, electronics, industrial and interaction design.

Amarino is a toolkit that enables the rapid prototyping of such applications by connecting the Android operating system to the Arduino microcontroller platform. It consists of an Android application, an Arduino library, and a collection of documentation and examples. This suite of tools allows users to: 1) access Android events (ie: compass orientation, accelerometer data, and text messages received) and send them to Arduino microcontrollers without doing any Android programming, and 2) quickly develop Android applications that receive data (ie: environmental sensor data) from (and send data to) Arduino microcontrollers. This paper introduces Amarino and presents the results of a preliminary user study.

Categories and Subject Descriptors
H.5.2 [**Information Interfaces and Presentation (e.g., HCI)**]: User Interfaces – *Haptic I/O, Input devices and strategies, Prototyping.*

General Terms
Design, Human Factors

Keywords
Mobile Devices, Android, Arduino, Toolkit, Microcontroller, Interfaces, Communication, Tangible, Mobile Computing, Smart phones, Mobile Phones, Wearables

1. INTRODUCTION
When Weisner introduced "ubiquitous computing", he motivated his discussion by articulating problems with current computing paradigms. "Even the most powerful notebook computer, with access to a worldwide information network, still focuses attention on a single box" [18]. Strikingly, this criticism can be applied to most of today's technology, particularly smart phones. Smart phone "boxes" engross and isolate us with their power to capture our attention, a problem that has negative social and physical consequences. Ambient and tangible interfaces can solve many of these problems—they can leverage a wide range of our existing physical skills and can convey information without demanding all of our visual and cognitive attention [12]. But, while tangibles have many advantages, they aren't as common as traditional GUI systems in part because they are difficult to design and construct [16, 17]. Ambient interfaces to smart phones are especially rare, perhaps because they are particularly challenging to build [2].

Amarino aims to facilitate the development of tangible interfaces to smart phones by eliminating several of the developments steps that are required to build them. In particular, Amarino allows Android-based mobile phones to communicate seamlessly with Arduino-based microcontrollers. It also allows tangible developers who would like to employ phone-based sensing in their projects to do so without engaging in any mobile device programming. The toolkit consists of two components, an Android [3] application, which runs on a mobile device, and a software library for the Arduino [5], which runs on a tangible device. Central to each of these is a communication protocol that allows a developer to focus on the behavior of his or her project instead of low level communication details. The current implementation takes advantage of most phones' built in Bluetooth radios and assumes that communication takes place over Bluetooth.

Before we describe the system in detail, we will describe an example that highlights Amarino's most important features. Figure 1 shows a picture of this project—an RGB LED lamp (similar to the Ambient Orb [1]) that can be controlled by a smart phone. To change the color of the lamp, the phone is placed on a flat surface and rotated—the lamp's color changes in response to changes in the phone's compass heading.

The process we used to construct the lamp illuminates the affordances and benefits of Amarino. In the first step, the Arduino that controls the lamp was programmed to receive data from Amarino. In this stage, we specified the type of information that would be sent by the phone—compass data—and programmed the Arduino to translate this data into lamp color.

Once the Arduino was programmed, we could control the lamp directly from the Android Amarino application. The Amarino interface allows users to choose from a list of built-in Android

MobileHCI'10, September 7–10, 2010, Lisbon, Portugal.
ACM 978-1-60558-835-3/10/09.

events (including accelerometer readings, compass readings, and call received events) to send to Arduinos. We selected compass events from the list and then also used the Amarino interface to discover the Bluetooth enabled lamp and connect to it. Once we connected, compass data was continually sent from the phone to the lamp and we were able to use the phone as a remote control color mixer.

Figure 1: A multicolor lamp controlled by an Android phone.

What is noteworthy about this example is that we *did not need to do any Android programming* to build and control the lamp. Only the Arduino was programmed. Amarino allowed us to use the phone as an input device without doing any mobile development. Amarino has many additional features, but this example highlights what we believe is its most important contribution—enabling developers to very quickly and easily create tangibles that are controlled by mobile devices.

2. RELATED RESEARCH

Though our introduction focused on the difficulties of developing projects that span the mobile and tangible domains, several researchers have built compelling projects at this intersection. We take inspiration from this previous work, which both illustrates the potential of the area and illuminates the limitations of current development tools.

Among the interesting mobile-tangible projects are systems that involve wearables, games, and environmental sensing. For example, *WeWrite* [20]—which employs mobile phones and the LilyPad Arduino [7]—enables users to send text messages to a T-shirt that then displays them on a built-in LED display. Another wearable-based project, called *united-pulse* [19], uses electronic rings, heart rate monitors and smart phones to communicate heart rate across distance. The rings communicate with the phone and monitors to enable two people to feel each other's pulses remotely. A less poetic, but perhaps more useful application, *eye-q* consists of a small display mounted on the inside of a pair of glasses that delivers peripheral visual cues to its wearer about phone events [8]. Other applications include games that motivate their players to be physically active [9], and environmental sensor networks which rely on mobile and tangible nodes that communicate with one another (cf *MobSens* [13]) .

In the realm of toolkits, there are many systems that aim to make working with hardware easier. In addition to Arduino [5], the best examples include Phidgets [10], D.Tools [11], Basic Stamps [6], and Lego Mindstorms [14]. Similarly, there are several tools that enable developers to write applications for mobile phones—Apple's iPhone development kit [4] being a particularly nice example. However, Amarino is the first fully realized kit that brings these worlds together.

3. APPROACH

After surveying related work and reflecting on our own experiences, we identified three different kinds of expertise that are required to produce smart phone/microcontroller projects:

- Developing smart phone applications
- Implementing communication protocols
- Building tangible devices

Each of these tasks requires different skills including application programming, microcontroller programming, electrical engineering, industrial design, and networking. Most developers, and especially novices, have experience in one or two of these domains, but rarely in all of them. We realized that most of the work required to implement a particular project is application dependent and therefore difficult to simplify or eliminate. However, it was clear that we could develop a standardized communication protocol. We also observed that many projects, like *eye-q* [8], were using tangibles as ambient output devices for simple phone events and felt that this presented another compelling opportunity for standardization. Amarino was thus designed to eliminate smart phone development and communication protocol development from the work process as much as possible; our goal was to enable developers to focus their attention on building the tangible devices.

To achieve our goals we chose to leverage two powerful existing technologies—the Arduino microcontroller platform and the Android mobile operating system. Both of these are mature systems with active user communities. Both are also open source and well documented—attributes that made them easy to access and customize for our purposes.

We wanted our application to be as useful as possible and accessible to a broad audience. In approaching this challenge, we were particularly inspired by the Arduino project, which has a large community following that includes both novice and expert developers. The Arduino community has become a vibrant, creative and largely self-supporting entity with documentation, hardware extensions, software libraries, and curricula increasingly produced by community members. Motivated by this example, we identified three additional goals for our project. To facilitate adoption by a wide range of users and ongoing community development Amarino should be:

- Easy to use (to support novice developers)
- Extendible (to support expert developers)
- Open source (to support community extension)

Throughout our development process we were guided by these aims and philosophies, which we will highlight as we describe our system and user experiences.

4. AMARINO

As illustrated in Figure 2, Amarino is a system that allows Android based smart phones to communicate with Arduino microcontrollers. It consists of two components: an Android application called "Amarino" and an Arduino library called "MeetAndroid".

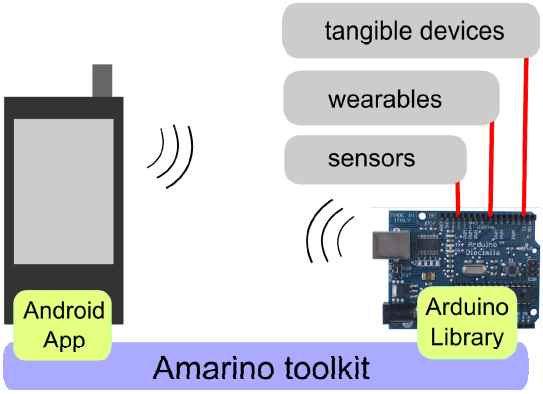

Figure 2: Amarino

Amarino, the Android application, sends event data from phones to Arduinos. It consists of a graphical user interface that allows users to: select built-in phone events to send to specific Arduinos; create collections of events that are associated with specific Arduinos; monitor data being sent from the phone; and manage Bluetooth connections. In its more advanced mode, Amarino can also receive data and pass it along to other Android applications.

On the Arduino side, the MeetAndroid library enables developers to associate Arduino functions with Android events and extract data attached to events. For projects that require two-way communication, MeetAndroid also provides functions to send data from microcontrollers to phones.

The remainder of this section will discuss Amarino and MeetAndroid in detail.

4.1 Android Interface

When a user starts the Amarino application, he is welcomed by a main screen like the one shown in Figure 3. This screen is the control center for managing all of the events, connections, and data flows being handled by Amarino. As can be seen in the image, it is broken into four quadrants:

- Bluetooth Manager
- Monitoring
- Settings
- Event Manager

Each of the four quadrants corresponds to one of Amarino's core functionalities. (In addition to these four areas there is a connect/disconnect button in the middle of the screen whose functionality we will describe shortly.)

Figure 3: Main screen

4.1.1 Bluetooth Manager

Clicking on the Bluetooth Manager quadrant opens up a window (Figure 4) that enables the user to manage Bluetooth devices—which, for our purposes, usually correspond to Arduinos. Within this module, users can discover nearby Bluetooth devices and pair with them. This is a process that must be undertaken once for each Bluetooth enabled Arduino before any communication can happen between it and the phone.

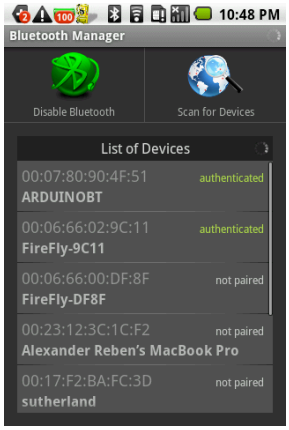

Figure 4: Bluetooth Manager

Since Bluetooth is a peer-to-peer communication protocol, Amarino only communicates with one Bluetooth device (one Arduino) at a time. From now on we will refer to the Arduino that is currently paired with the phone as the *active* Arduino. Amarino also remembers which device it was most recently connected to, and clicking the connect/disconnect button on the main screen (Figure 3) will cause the application to connect to or disconnect from this device.

4.1.2 Monitoring

Clicking on the monitoring quadrant opens a window (Figure 5) that displays the streams of data being sent and received by Amarino. This window also allows users to send arbitrary characters to the active Arduino.

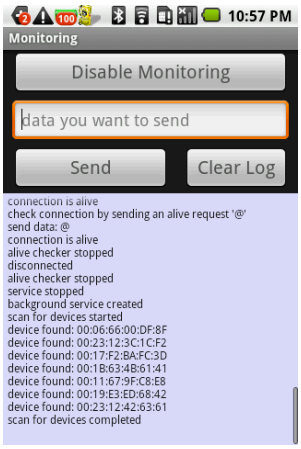

Figure 5: Monitoring

4.1.3 Settings

The settings window allows the user to customize the toolkit's behavior. Here users can change the rate at which event data is sent and choose whether phone to Arduino Bluetooth connections should be permanent (for high data flow applications) or intermittent (for occasional event sending).

4.1.4 Event Manager

The Event Manager, shown in Figure 6, is the central component of the Amarino interface. It allows users to create collections of events that they wish to send to particular Arduinos. A collection is a set of events with a user-defined name that is associated with a specific Bluetooth device (Arduino). The *active collection* is the collection associated with the last active device or the collection that is explicitly set to be the active collection.

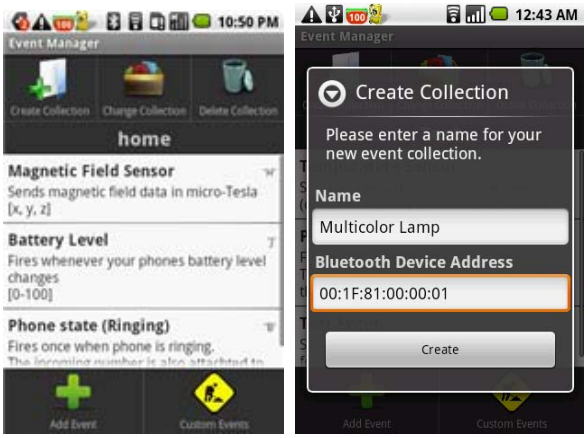

Figure 6: Left: the Event Manager showing the active collection "home". Right: the Create Collection window

To create a new collection of events, the user clicks on the Create Collection button at which point she is prompted to enter a name for her collection and an address of a Bluetooth device, as shown in the right hand image of Figure 7. Once the user creates the new collection, it becomes the active collection and she can use the Add Event button to choose events for it.

When a user opens the Event Manager, she is presented with a window like the one shown in the left side of Figure 6, which shows information about the currently active collection including its name—in this case, *home*—and the list of events in the collection—which in this case includes *Magnetic Field Sensor* and *Battery Level*. To add an event to the home collection, the user clicks the Add Event button, which brings up a list of all available Android events. All of the events that are included in the Amarino distribution are shown in Table 1.

Table 1: List of all built-in events in Amarino

Built-in Event	Description
Test Event	Sends an empty message every 5 secs
Time Tick	Fires each minute
Phone state (Ringing)	Fires once when phone is ringing [incoming number attached]
Phone state (Idle)	Fires once when phone is idle
Phone state (Offhook)	Fires once when phone is hung up
Compass	Sends heading in degrees [0-359]
Orientation Sensor	Sends orientation data in degrees [azimuth, pitch, roll]
Accelerometer Sensor	Sends acceleration data in m/s^2 [x,y,z]
Magnetic Field Sensor	Sends magnetic field data in micro-Tesla [x,y,z]
Temperature Sensor	Sends temperature in Celsius degree
Light Sensor	Sends ambient light level in SI lux units
Battery Level	Fires whenever the phone's battery level changes [0-100]
Receive SMS	Fires when an SMS has been received
Earthquake	Fires when an earthquake occurs somewhere on earth with magnitude [0-10]

Other buttons within the Event Manager include the Delete Collection button, which deletes the current collection, and the Change Collection button, which allows the user to choose which collection is active.

Users can also create custom events using the Custom Events button. We will describe this functionality in more detail in Section 4.4. Once created, custom events can be added to collections in the same way as preinstalled events.

Event data that is sent from Amarino is structured so that it can be easily identified and parsed by Arduinos. Each event type is associated with a single alphanumeric character that uniquely identifies the event. For example, the Battery Level event is associated with the character 'J'. Event data packages contain the identifier character followed by the relevant event data. A Battery Level event, for example, would consist of the character 'J' followed by a number from 0-100 that indicates the phone's current battery level.

4.2 Architecture

Amarino's architecture, shown in Figure 7, has five components: a Graphical User Interface (GUI), Background Service, Bluetooth Handler, IntentEventMapper and Database. The Background Service is at the heart of Amarino. It listens for sensor and phone state information, generates events, sends events, and maintains Bluetooth connections. The Bluetooth Handler implements low-level access to the Bluetooth functions of the operating system. The IntentEventMapper converts Android "intents" to Amarino events. Finally, the database stores all of the collection, event, and Bluetooth device history and the GUI makes all of this functionality accessible to the user.

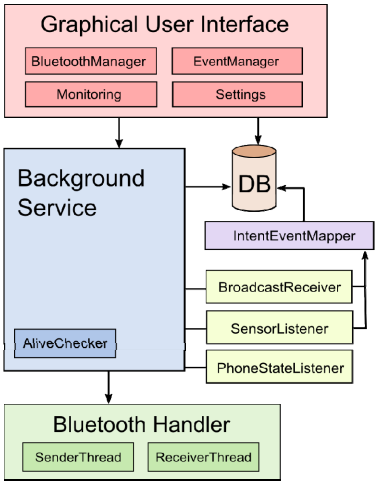

Figure 7: Architecture of the Android application

4.3 MeetAndroid Library

The second major component of the Amarino toolkit is the Arduino library, MeetAndroid. This library provides a set of functions for receiving Amarino events and sending data from Arduinos to Amarino.

To receive data from Amarino, an Arduino programmer first uses the registerFunction() library component to "register" functions in his Arduino code. This determines which functions will be used to parse specific Amarino events. To employ event data in an Arduino program, the *meetAndroid.receive()* function is used to check for incoming event data. This data is then examined and passed to the appropriate registered function. To parse event data, the MeetAndroid library provides additional helper functions that can be employed in the user-written registered functions. For example, the *meetAndroid.getInt()* function retrieves an integer value from an event package. A listing of all MeetAndroid functions is shown in Table 2.

The library also provides *send()* functions that allow the user to send data to Amarino.

Table 2: The MeetAndroid library

MeetAndroid
+library_version() : int +MeetAndroid(H_voidFuncPtr err) +MeetAndroid() +registerFunction(void(*)(uint8_t, uint8_t),uint8_t) : void +unregisterFunction(uint8_t) : void +receive() : bool +void send(char) : void +void send(const char[]) : void +void send(uint8_t) : void +void send(int) : void +void send(long) : void +void send(long, int) : void +void send(double) : void +void sendln(void) : void +flush() : void +bufferLength() : int +getBuffer(uint8_t[]) : void +getInt() : int +getLong() : long +getFloat() : float +getDouble() : double +getIntValues(int[]) : void +getLongValues(long[]) : void +getFloatValues(float[]) : void +getDoubleValues(double[]) : void

The final important component of the library is a series of example Arduino programs that demonstrate different Amarino functionality. These examples include programs that illustrate how to parse and use Android phone events (like ringing and hang-up events) and sensor events (like compass and accelerometer readings) as well as programs that employ all of the helper functions shown in Table 2 and programs that show how data can be sent from Arduinos to Android phones.

4.4 Using Amarino to Create Custom Android Applications

Though the primary purpose of Amarino is to enable users to prototype complete systems without doing any smart phone programming, Amarino also provides assistance for Android development. On the Android side, the Amarino application can be run as a background service that listens for events generated by any Android application and passes them on to Arduinos.

To make use of this functionality, a user would develop a custom Android application that broadcasts event information to the Android operating system via Android intents. Amarino can then listen for this information; within the Amarino application, the user would use the Custom Event Manager to create an event that corresponds to the intent by supplying an event name, a description, the name of the intent and information about the data attached to the intent. Once the custom event has been created, it is handled like a standard event, both within Amarino and by the MeetAndroid library—it will appear in Amarino's event list and users can add it to any collections of events that will be sent to Arduinos. Figure 8 shows a flow diagram that outlines the steps in this process.

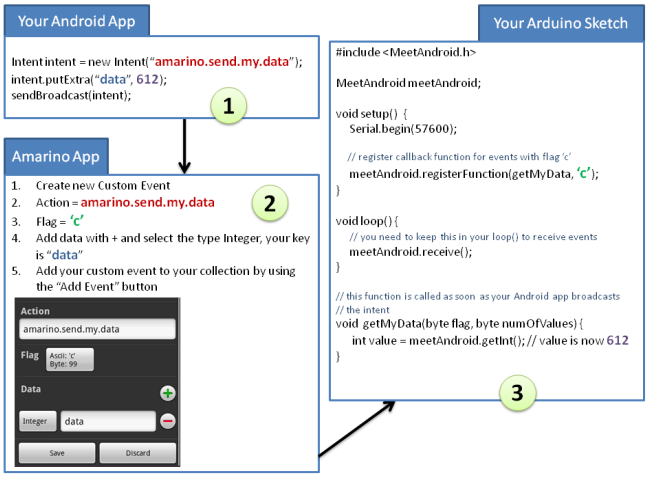

Figure 8: Sending data from a custom Android application to Arduino via Amarino

In addition to receiving intents from Android applications and turning them into events to send to Arduino, Amarino can also do the inverse: receive data from Arduinos and broadcast this information to other Android applications via intents. To accomplish this task, a user would first create a custom Android application for receiving/visualizing the Arduino data. Then she would create an Amarino collection with the same name as this application. Once Amarino is activated as a background service, the connection to the Arduino is established, and the custom application is run, Amarino passes along all received Arduino data as intents to the custom Android application.

Amarino also provides a transparent way for custom Android applications to connect to Arduinos. To initiate a connection, the third-party Android application has to send a specific intent: `sendBroadcast (new Intent("amarino.CONNECT");` This will immediately start Amarino as a background process and Amarino will begin trying to connect to the device associated with its current active collection. The same mechanism can be used to disconnect the phone from the Arduino upon quitting the custom application.

In Section 5 we will discuss examples that demonstrate the full range of functionality that Amarino facilitates.

4.5 Documentation

To introduce users to our toolkit we also developed a set of tutorials and references to accompany it. All of this information is published on the Amarino website. The website includes a section that explains how to download and install Amarino, a section that details all of the functionality of the Amarino Android application, and a selection of in-depth tutorials that walk users through specific application scenarios. In particular, there is a tutorial that describes how to build the RGB LED lamp we described in the introduction and a tutorial that details how to develop an Android application that plots sensor data received from Arduinos on phone screens. The website also includes links to all of the source code for the project.

5. AMARINO EXAMPLES

We have constructed a number of examples that rely on Amarino at our lab. We will describe three of them in this section. First, we will return to the multicolor lamp application from the introduction and discuss extensions of that project. Then we will discuss two examples that involve wearables: CallMyShirt, which highlights phone to Arduino communication and Workout, which highlights Arduino to phone communication.

5.1 Multicolor lamp

The multicolor lamp example we introduced earlier consists of a tricolor LED that is controlled by an Arduino. In the physical lamp, the Arduino, the LED, and a battery are mounted inside a custom lampshade we constructed from plywood and textile composites. As we described earlier, the lamp is controlled by rotating a phone. In the earlier example, we described how a user could control the color of the lamp via Amarino. In this scenario, though the user is able to control the lamp with a phone, there is no information on the phone that indicates how this control works or communicates the current state of the lamp.

We developed an extension to this basic example that consists of a special Android application that enriches the interaction. It provides feedback about the current state of the lamp and a subtle indication of how the compass interface works via a round color wheel like the one shown on the left in Figure 9. To indicate color, the current color of the lamp is darkened—in Figure 9, the lamp is an orange-ish red—and the circular color layout provides an intuitive indication of what will happen when the phone is rotated clockwise or counter clockwise. The lamp's current color and brightness are also written in text at the bottom of the interface window.

Developing a custom Android application also allowed us to add more complex interactions to our interface. For example, we added functionality that allows a user to tap the screen to freeze the lamp at a particular color. If a user tapped the screen on the left of Figure 9, the lamp would freeze red and the user could then move her phone around without affecting the color. A second tap would unfreeze the color. An extended finger press to the color screen opens up a new window, like the one shown on the right in Figure 9 that allows the user to control the brightness of the lamp instead of its color, again via phone rotation. Though we could have theoretically implemented the same interactions with Amarino, the chain of interactions would be very difficult to follow without the graphical queues on the phone.

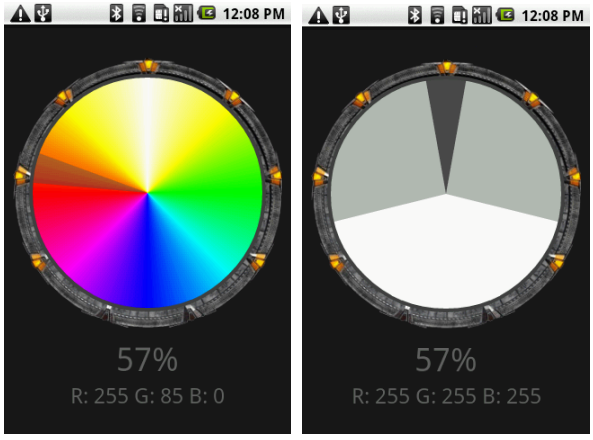

Figure 9: Multicolor lamp, custom Android interface.

To develop this example, we created custom events that were sent from our custom application to Amarino—which runs as a background service companion to our app—and then on to the Arduino. Within Amarino we defined two custom events, one for changing light intensity and one for changing color. On the Arduino side, we wrote and registered functions to respond to these events.

5.2 CallMyShirt

Our second example, CallMyShirt, uses Amarino to communicate phone events to a wearable device. This example provides another case study that involves no mobile phone application development. The wearable, shown in Figure 11 is a shirt with 10 luminescent pads on its front.

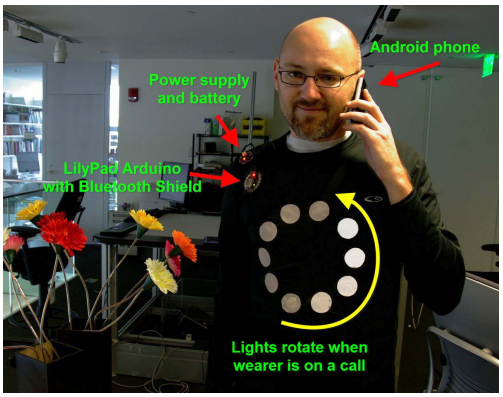

Figure 10: CallMyShirt

These individually controllable pads are used to display information about the phone's current state. When the phone is ringing, all of the pads blink on and off. If a call is accepted, the lights rotate in a circle until the user hangs up. When the phone is idle, the shirt generates random patterns. The shirt can also be used as an ambient display for phone battery state. In this mode, each pad corresponds to 10% of the current battery capacity.

The shirt was constructed from a LilyPad microcontroller and 10 LilyPad LEDs that were sewn into the shirt with conductive thread. The microcontroller is connected to a Bluetooth shield, which facilitates communication with the phone.

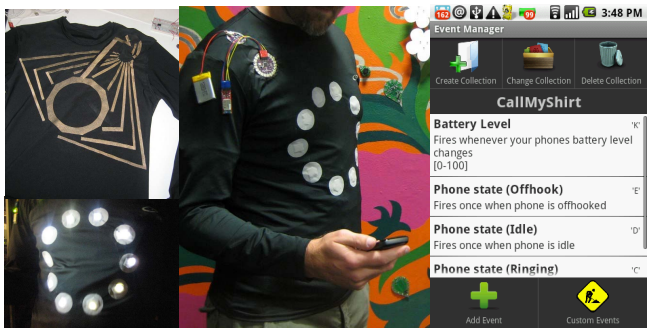

Figure 11: The inside of CallMyShirt with conductive traces (top left), all LEDs light up (bottom left), testing all functions before sewing remain parts on the shirt (middle) and the corresponding event collection (right)

This example highlights a very important feature of Amarino that we have not yet mentioned. The wearer of this shirt would likely want to use other Android applications or would want to put the phone in his pocket while the shirt was "running". Amarino supports this style of interaction. Once Amarino establishes a connection to an Arduino, a user can close the Amarino application. The connection is maintained and events are sent through Amarino's background process until the Arduino connection is lost or some specific action is taken. This enables Amarino to operate without interfering with regular phone operation.

5.3 Workout

Our final example, "Workout", was developed to help exercisers keep track of their routines.

Figure 12: Left: the Armband. Right: the Workout interface.

It uses a motion sensing wearable and an Android application that allows wearers to visualize and analyze their activity level. Our wearable is a knitted armband, shown on the left in Figure 11 that contains an embedded stretch sensor. The sensor was constructed from a piezo resistive yarn knit into the fabric of the armband. When placed over an elbow, the armband can reliably detect when the elbow is bent. A LilyPad Arduino—again sewn into the band with conductive thread—reads sensor data from the band and relays it to a phone via Bluetooth.

The Android Workout application plots the raw sensor data it receives, extracts a number-of-arm-bends count from this data, and displays this count to the wearer. A snapshot of this application in action is shown on the right in Figure 11. In this example, Amarino is used to receive data from the arm band and pass it along to the workout application.

It is worth (anecdotally) noting that we produced this complete prototype, both the wearable and the application, in a few hours. Amarino (and Arduino) allowed us to develop and demonstrate our Workout proof of concept much more quickly than we would have been able to without it.

6. EVALUATION

To evaluate our toolkit, we hosted a one day workshop for a group of 13 engineering and design students, 10 male and 3 female. All were in their 20s or 30s. Our group included undergraduate, masters, and PhD students. All students had some previous programming experience. 12 had previous Arduino programming experience and 5 had previous Android programming experience.

Each student was given an Android phone, an Arduino board, and access to a range of electronic components like LEDs and motors. The workshop began with a short introduction to Arduino and Android. Then the participants were asked to go independently through one of our online tutorials. This tutorial explains how to install all of the necessary software and establish the connection between the phone and the Arduino. The end result of this tutorial is an Amarino collection that sends a time tick event every 5 seconds and an Arduino-connected LED that blinks in response to this event.

All students were able to complete the online tutorial successfully within 45 minutes. As a second challenge, the students were asked to experiment with Amarino's inbuilt events. Most students successfully used the compass or accelerometer events to change the intensity of an LED.

The third challenge was to write a simple Android application. We guided the students through developing an Android interface with a single button that was used to switch an Arduino-connected LED on and off.

After completing this exercise, students were allowed to build their own applications. Two students produced particularly interesting prototypes in this session. One student wrote a program that activated the phone's vibration motor whenever a light sensor attached to the Arduino was covered with the hand.

The most compelling example however was from a student who used the Android's built-in speech recognition engine to build a voice-activated light. This student wrote an Android application that sent out an event message whenever a user said the word

"light". His Arduino was programmed to toggle an LED each time it received this message.

To evaluate the effectiveness of our toolkit and workshop, we conducted a short survey. In this survey all respondents rated the kit easy to use and the documentation helpful though several asked for additional tutorials and examples. We have since incorporated many of these suggestions into our documentation. All respondents also said that they were planning to use the kit in own projects. Though it is nearly impossible to draw any specific conclusions from this preliminary data, for us the experience verified the kits basic utility and usability and provided us with promising avenues for future development.

7. CONCLUSION AND FUTURE WORK

We developed Amarino to empower developers to work easily with more of the artifacts in our environments—from phones to furniture to clothing. We focused on keeping our tools easy to use so that they would be accessible to novices but also easily extendible so that they would be useful for experts.

In our future work, we would like to provide users with access to higher level events, like gestures [15][14] and spoken words for example. We would like to add support for communication via wireless LAN—this functionality could be particularly useful in the home environment. We would also like to investigate a location detection feature that would automatically activate particular event collections based on current location. Finally, we want to construct more prototypes to explore the seamless integration of smart phones into our personal environment.

8. ACKNOWLEDGMENTS

Thanks to Martin Hitz, David Mellis, Hannah Perner-Wilson, Emily Lovell, and all of our workshop participants for their contributions and conversation. This work was funded in part by the MIT Media Lab consortium.

9. REFERENCES

[1] Ambient Orb. http://www.ambientdevices.com/

[2] Amft, O. and Lukowicz, P. 2009. From Backpacks to Smart phones: Past, Present, and Future of Wearable Computers, *IEEE Pervasive Computing*, vol. 8, no. 3, pp. 8-13, July-September, 2009.

[3] Android. http://www.android.com/

[4] Apple. iPhone SDK. http://developer.apple.com/iphone/

[5] Arduino. http://www.arduino.cc

[6] Basic Stamp. http://www.parallax.com/Sharlin, E., Watson, B., Kitamura, Y., Kishino, F., and Itoh, Y. 2004. On tangible user interfaces, humans and spatiality. *Personal Ubiquitous Comput*. 8, 5 (Sep. 2004), 338-346.

[7] Buechley, L., Eisenberg, M., Catchen, J. and Crockett, A. 2008. The LilyPad Arduino: Using Computational Textiles to Investigate Engagement, Aesthetics, and Diversity in Computer Science Education. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI)*, (Florence, Italy, April 2008), pp. 423-432.

[8] Costanza, E., Inverso, S. A., Pavlov, E., Allen, R., and Maes, P. 2006. eye-q: eyeglass peripheral display for subtle intimate notifications. In *Proceedings of the 8th Conference on Human-Computer interaction with Mobile Devices and Services* (Helsinki, Finland, September 12 - 15, 2006). MobileHCI '06, vol. 159. ACM, New York, NY, 211-218.

[9] Fujiki, Y., Kazakos, K., Puri, C., Buddharaju, P., Pavlidis, I., and Levine, J. 2008. NEAT-o-Games: blending physical activity and fun in the daily routine. *Comput. Entertain*. 6, 2 (Jul. 2008), 1-22.

[10] Greenberg, S. & Fitchett, C., 2001. Phidgets: easy development of physical interfaces through physical widgets. In Proceedings of the ACM symposium on User interface software and technology (UIST). pp. 209–218.

[11] Hartmann, B. et al., 2006. Reflective physical prototyping through integrated design, test, and analysis. In Proceedings of the ACM symposium on user interface software and technology (UIST). pp. 299–308.

[12] Ishii, H. and Ullmer, B. 1997. Tangible bits: towards seamless interfaces between people, bits and atoms. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, United States, March 22 - 27, 1997). S. Pemberton, Ed. CHI '97. ACM, New York, NY, 234-241.

[13] Kanjo, E., Bacon, J., Roberts, D., and Landshoff, P. 2009. MobSens: Making Smart Phones Smarter. *IEEE Pervasive Computing* 8, 4 (Oct. 2009), 50-57.

[14] Lego Mindstorms. http://mindstorms.lego.com/

[15] Liu, J., Wang, Z., Zhong, L., Wickramasuriya J., and Vasudevan, V. 2009. uWave: Accelerometer-based personalized gesture recognition and its applications, *Pervasive Computing and Communications*, 2009. PerCom 2009. IEEE International Conference on , vol., no., pp.1-9, 9-13 March 2009

[16] Shaer, O. and Jacob, R. J. 2009. A specification paradigm for the design and implementation of tangible user interfaces. *ACM Trans. Comput.-Hum. Interact.* 16, 4 (Nov. 2009), 1-39.

[17] Shaer, O., Leland, N., Calvillo-Gamez, E. H., and Jacob, R. J. 2004. The TAC paradigm: specifying tangible user interfaces. *Personal Ubiquitous Comput.* 8, 5 (Sep. 2004), 359-369.

[18] Weiser, M, The computer for the twenty-first century. *Sci. Am*, Sept. 1991, 94-104.

[19] Werner, J., Wettach, R., and Hornecker, E. 2008. United-pulse: feeling your partner's pulse. In *Proceedings of the 10th international Conference on Human Computer interaction with Mobile Devices and Services* (Amsterdam, The Netherlands, September 02 - 05, 2008). MobileHCI '08. ACM, New York, NY, 535-538.

[20] Winkler, T., Ide, M., Wolters, C., and Herczeg, M. 2009. WeWrite: 'on-the-fly' interactive writing on electronic textiles with mobile phones. In *Proceedings of the 8th international Conference on interaction Design and Children* (Como, Italy, June 03 - 05, 2009). IDC '09. ACM, New York, NY, 226-229. (Cambridge, United Kingdom, February 16 - 18, 2009). TEI '09. ACM, New York, NY, 323-330.