The Interaction Engine

Nikolas Martelaro, Wendy Ju, and Mark Horowitz

Abstract The Interaction Engine is a framework for prototyping interactive, connected devices based on widely available single-board Linux computers. With microcontrollers, networking, and modular open-source software, these modules enable interaction modalities such as audio, video, tangible, and digital interfaces to be embedded into forms that go beyond traditional computing. In this paper, we outline the hardware and software components that make up the general Interaction Engine framework and discuss its benefits for interaction designers. We provide an illustrative case study of the Interaction Engine in use. We ran workshops to introduce designers to the Interaction Engine framework and we describe the projects where they subsequently employed Interaction Engines to understand issues and opportunities presented by this model. In describing the framework and case studies, we intend to shift designer's thinking of computer as product to computer as material to create new interactive devices.

1 Introduction

We are entering a golden age for interaction design. Concepts that have long existed as mere visions, like Vannevar Bush's Memex (Bush 1989), Marc Weiser's Tabs, Pads and Boards (Weiser 1999), and Apple's Knowledge Navigator (Sculley 1987) are suddenly a commercial reality. High-end commercial products such as the Nest thermostat, Drop Cam security camera, Amazon Echo interactive radio, and Phillips Hue light are inspiring designers to contemplate what else is made possible with networked interactive hardware. What is most exciting is that the basic components that make such concepts possible are accessible to savvy designers, not just seasoned engineers (Hartmann and Wright 2013).

Just as the interaction design community's adoption of embedded microcontrollers has driven a breakthrough in physical computing (Banzi and Shiloh 2014), enabling artists and designers "more creative thought and further iterations" (Gibb 2010), the adoption of lightweight computation platforms can power a similar

N. Martelaro (✉) • W. Ju • M. Horowitz
Center for Design Research, 424 Panama Mall, Stanford, CA, 94305-9035, USA
e-mail: nikmart@stanford.edu; wendyju@stanford.edu; horowitz@stanford.edu

H. Plattner et al. (eds.), *Design Thinking Research*, Understanding Innovation,
DOI 10.1007/978-3-319-60967-6_8

revolution of multimodal connected devices. Various efforts have sought to introduce such platforms to the tangible, embedded, and embodied interaction design community (Berdahl and Ju 2011; Kubitza et al. 2013; Overholt and Møbius 2013; Vandevelde et al. 2015; Martelaro et al. 2016). The emerging design pattern uses networked single-board Linux computers, such as the Raspberry Pi or BeagleBone, with embedded microcontrollers, PC peripherals, open-source software, and connection to cloud-computing services to enable designers to more easily explore a wide variety of novel, connected, post-PC devices (Kuniavsky 2010). Although this framework enables the conception of new interactive devices, there are many challenges that can keep designers from adopting these new tools. We aim to help designers shift their thinking, from seeing computers as product to recognizing computers as malleable materials for enabling interaction.

In this chapter, we articulate the common framework underlying these platforms to allow interaction designers to understand their power and capabilities. We give a name to this computing framework—the Interaction Engine—and detail the various subsystems that make up the platforms and their common interactive functions. We present a case study on the design of an expressive sofa to elaborate on the use of the Interaction Engine in practice. We then describe and detail a series of workshops we led to introduce designers to the Interaction Engine. Finally, we discuss the implications of the Interaction Engine on interaction design thinking and outline a set challenges to address that will help further adoption of this framework within the interaction design community.

2 The Interaction Engine Framework

The name Interaction Engine harkens back to the early twentieth century, when a revolution in increasingly inexpensive and widely available powered motor technology enabled a wide variety of new products from the automobile to the home kitchen stand mixer. An Interaction Engine can similarly be used as a computational material (Landin 2005) to make an object connected and interactive, providing interfaces to the immediate physical world, computation capability, and connection to the larger digital world. One can think of this framework as an embeddable composite material for new device design (Vallgårda and Redström 2007) consisting of a single-board operating system based computer, microcontroller, open-source software, and connection to cloud computing services (Fig. 1).

2.1 Single Board, Operating-System-Based Computer

A single-board Operating-System-based computer serves as the foundational computing unit of the Interaction Engine. These systems are small, inexpensive and accessible to anyone who knows how to use a desktop computer. Currently, the

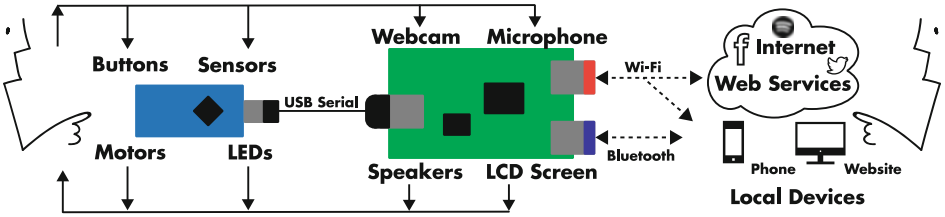

Fig. 1 The Interaction Engine, enabling physical, digital, and networked interactions with people, cloud, and devices

HCI community is using the Raspberry Pi running some version of Linux as its favored default platform; however, for specific applications, another system may be preferred. These single-board Linux computers (SBLC) are being updated on a regular basis; it makes sense to size the specific board to the complexity of the computation in each application. The provision of a standardized Linux OS provides commonality in a diverse and rapidly changing market, while giving designers the ability to choose systems based on computing power, peripherals, and form factor, without much switching cost.

In our design work, we have implemented Interaction Engines using five different systems, ranging from the single-core Raspberry Pi to the 8-core ODROID XU-3. While there are many options for SBLCs available today, most systems are based around ARM microprocessors and provide access to common peripherals such as USB, Ethernet, and audio/video input and output. With the power of a microprocessor, Linux OS, and USB, capabilities such as audio playback, audio recording, graphical display, video recording, networking, and data processing applications are enabled through high-level software. This extensibility can allow designers to quickly explore new, multi-modal interactions by building upon the huge variety of open source software. Designers can speed their development and try out new interactive devices in less time, allowing for more design iteration and exploration. Additionally, since SBLCs provide a low cost and common OS interface, designers can simultaneously bring up many parallel prototypes, eventually leading to better designs (Dow et al. 2010).

2.2 Physical Computing Interface

A microcontroller unit (MCU) often serves as an interface to the immediate, physical world. Although most SBLCs feature General Purpose IO ports (GPIO) which can act as the physical computing interface, these interfaces are not standardized. Often, the embedded Linux community uses separate MCU's for robust, standardized physical computing (Russell 2012; Palazzetti 2015). Although different MCUs have widely varying peripherals such as ADCs, PWM, and various communication interfaces, one critical MCU feature within an Interaction Engine is

some form of connectivity and communication, such as serial over USB, with the upstream OS-based computer. We have found the ability to decouple the physical interface from the processing capability to be valuable for modularity, extensibility, and upgradeability. This benefits the design process as designers can develop stable physical interfaces and interchange them with different SBLCs allowing them to use SBLCs that are best for their interactions. In our own projects, we have often added features that require greater computational power or different input/output capabilities in the middle of the project and can simply plug our physical computing interface into a new computer and port the high-level software in little time.

2.3 Connection to the Cloud

While some Interaction Engines operate in a stand-alone autonomous mode, most modern-day devices rely on some degree of external communication. Most SBLCs come with Ethernet ports for wired networking. While this may be useful for fixed exhibits, wireless networking is more flexible. USB Wi-Fi and Bluetooth modules are widely-supported, low-cost options for adding connectivity to local devices such as phones, tablets, and other computing systems. This is done using a built-in web server, which can mediate communication between devices while also providing its own rich multimedia web-interface.

The network interface also allows the Interaction Engine to gather data from the cloud, outsource high-computation tasks, store information remotely, and interact with Web services such as Twitter, Spotify, Amazon AWS, or Internet-of-Things platforms. This connectivity effectively embeds these remote services as material inside an object (Vallgårda and Sokoler 2016). For designers, this is one of the most enabling aspects of the Interaction Engine as the SBLC allows them the use of mature and robust interfaces for allowing designers to interact with the cloud. This provides designers with a huge area of exploration for embodied, tangible interaction with digital services and data by extending potential designs beyond the physical and local world.

2.4 Open-Source Linux Operating System

Embedded Linux has become a popular choice for an operating system in these integrated systems amongst designers. Real-time operating systems (RTOS), like TinyOS (Levis et al. 2005) or Windows CE, have been popular amongst product engineers and are useful for real-time task scheduling and execution (Tan and Nguyen 2009). These RTOSes are used in small multimedia devices such as Siftables (Merrill et al. 2007). However, many of these RTOSes are proprietary, only support the C programming language, and have a steep learning curve. Other

embedded OSes, such as Android and iOS, are intended for specific hardware; this hardware is optimized around phone and tablet applications and can be difficult to interface with external microcontrollers.

We feel that, for designers, the use of open source Linux operating systems is a major boon to an Interaction Engine. Designers are free to develop using a variety of high-level applications and programming languages, and to draw upon shared code for low-level functions. These are free and robust in Linux and are not available at all for proprietary real-time operating systems. These tools can greatly speed development of networked and data-intensive applications while allowing designers to rapidly iterate on the interactions they wish to create (Roman et al. 2003). Additionally, this code is often highly portable to new systems, allowing designers to share and extend each other's work. The open code base, shared libraries and code modules, and easier learning curve make Linux-based platforms more ideally suited to the rapid and iterative design practice of interaction designers (Raghavan et al. 2005).

The use of an operating system also provides designers with many standard solutions for common computing tasks, which are not available on lower level microcontroller-based systems. For example, almost all high-level programming languages provide serial interfaces for communicating with MCUs. Many robust, free, open source messaging frameworks exist for real-time web communication, such as WebSockets, ZeroMQ, and MQTT. Aside from communication, most Linux operating systems support a wide array of programming languages for application development, such as Python and NodeJS. This allows designers to add capabilities such as audio processing, digital display, and computer vision using publicly available software modules (Stankovic et al. 2005). As such, the designer can now utilize the capabilities of the entire OS as a design material.

3 Prior Work

Recently, we have seen systems that incorporate single-board computers in interactive design concepts spanning full-body gesture input devices (Chan et al. 2015), wearable polygraphs (Charlesworth et al. 2015), voice-enabled, social museum exhibits (Marshall et al. 2015), body-controlled, networked musical input devices (Tahiroğlu et al. 2015), and Twitter-powered, energy and environment obsessed talking radios (Gaver et al. 2015). The common framework of single-board computers and microcontrollers in these embedded design prototypes gives designers a huge boost in computing power, easy connection to the web, and leverages open-source hardware and software modules for rapid prototyping and design iterations. This allows designers to focus on creating interactions rather than optimizing for computational limitations.

We see these new systems as the latest in an evolution towards increasing computation within interactive objects. We review this development in relation to four prevalent models for physical computing systems: (1) *standalone* systems

that run solely on microcontrollers, (2) *tethered* systems where a microcontroller is constantly connected to a multimedia PC, (3) *integrated* systems with a single-board PC integrated, often with a microcontroller, into the designed object, and (4) *eMbedded-Gateway-Cloud* where ultra-low power devices communicate with cloud services via a user's phone. Within each of these models, we take a design-centered perspective, examining key projects, design tools, and design patterns.

3.1 Standalone Systems

Standalone systems based on microcontroller units (MCUs) sense and interact with the world and have limited communication with other microcontroller based objects. Cheap microcontrollers such as the PIC enabled designers to create simple interactive objects such as Resnick's *Cricket* toy blocks and balls (Resnick et al. 1998), Frei et al.'s *curlybot* (Frei et al. 2000) drawing toy robot, Beigl et al.'s sensorized and networked Mediacup (Beigl et al. 2001), and Buechley's LED Game-of-Life tank top (Buechley et al. 2006). The BASIC Stamp was also heavily used as an embedded platform, particularly in education for creating simple robots with basic sensing and actuation capabilities (McComb 2003) and as a basis for early physical computing curriculum (O'Sullivan and Igoe 2004). Although these early systems were enabling, "working with microcontrollers required navigating a steep learning curve. Microcontrollers were general-purpose industrial components, designed to be a starting point for electrical engineers to design complex circuits, not a plaything for artists" (Townsend 2013). This lead to Wiring (Barragán 2004) and subsequently Arduino-based (Mellis et al. 2007) microcontroller development platforms which were specifically targeted for artists and designers and featured well-designed interfaces and useful abstractions. These systems made programming standalone systems more accessible to a wider group of designers. This led to a myriad of new opportunities for interaction designers to explore, such as interactive dresses, gloves, bags, and textiles (Buechley and Hill 2010) using the Lilypad Arduino (Buechley et al. 2008), interactive pop-up books (Qi and Buechley 2010), new tools for computer education (Millner and Baafi 2011), and even Do-It-Yourself cell phones (Mellis and Buechley 2014).

3.2 Tethered Systems

Although microcontrollers enable the creation of many new interactive physical objects, more opportunities are possible when the microcontroller is connected to a multimedia PC (O'Sullivan and Igoe 2004). On their own, microcontrollers have limited processing, connectivity, and interactivity, requiring the use of expensive specialty electronics for functions such as sound generation and network connectivity. Throughout O'Sullivan and Igoe's "Physical Computing," it is recommended

that many functions such as sound and speech input and output, graphical display, video, and networking are best left to a multimedia computer while the microcontroller handles physical input and output. This pattern enabled such work as Rozin's "Wooden Mirror" (Rozin 1999), with physical wooden pixels actuated by a microcontroller communicating with a multimedia computer running computer vision code. This microcontroller-plus-PC pattern was also used by those in the new instruments for musical expression (NIME) community to make novel physical music controllers that worked in concert with audio synthesized on computers using software such as MAX/MSP and PureData (Wilson et al. 2003).

This model is used by many Physical Computing toolkits, such as Phidgets (Greenberg and Fitchett 2001), iStuff (Ballagas et al. 2003), calder (Lee et al. 2004), and d.tools (Hartmann et al. 2006). These were developed to provide designers with modular physical prototyping interfaces to control multimedia computer interfaces. These systems helped designers to focus more on the software to create interaction rather than on hardware design.

3.3 Integrated Systems

One of the challenges of the tethered systems pattern is the divide between the physical computing and multimedia computing interface. In practice, designers often use their own computers as the multimedia PC in these tethered designs. In exhibit design, for example, the expense of whole computers can easily be justified as a project expense. Exhibits such as "Dacha Digital Murals, 2008" (Aminzade et al. 2008) incorporate microcontrollers and PC's to create interactive physical experiences with digital light and sound. Often, exhibit designers have used small form factor PC's like the MacMini to accomplish this (Harris-Cronin 2008). Dedicated low-cost computers, such as netbooks, can provide designers with the flexibility of a general purpose computer (Sipitakiat and Blikstein 2013). This was also a short trend within the robotics community (e.g. Willow Garage 2011).

Overall, however, the cost of integrating a traditional computer into each design limits exhibition and long-term deployments, makes replication and scaling difficult, and discourages exploration with new libraries and software packages. Specific toolkits have been developed to address these limitations, including Bug Labs (Gibb 2009), Chumby (Huang 2008), .NET Gadgeteer (Villar et al. 2012), and Kinoma Create (Marvell 2016). These toolkits integrate pre-built hardware and software for physical input/output, driving displays, and networking. Although these systems are excellent for enabling rapid prototypes, they can limit the design opportunities to the specific functional modules provided with the toolkit as well as the form of objects (Mellis et al. 2013).

Nowadays, the design community has converged upon the use of small, low-cost single-board computers, such as the Raspberry Pi and BeagleBone, to provide the features of full multimedia computers, while enabling the computer to be dedicated

and integrated into design prototypes. Exhibit designers, for instance, have begun opting for low-cost open source single-board computers and microcontrollers, such as a Raspberry Pi together with an Arduino, in their work (Langer and Alderman 2016; Borthwick 2016). Notable examples using this pattern include "I have a message for you" (Output Arts 2012), a replica of Turing's Delilah speech scrambler, and "Pianette/Sound Fighter," where two pianos were transformed into Street Fighter game controllers. Their smaller size allows them to be embedded seamlessly in smaller objects rather than hidden behind tables and walls. Their lower cost and relatively low power consumption compared to laptops allow them to be used in mobile devices such as The Workers "After Dark" live streaming telepresence robots designed for late night exploration of museums via the web (The Workers 2015).

Platforms such as Satellite CCRMA [for creating new physical music devices with advanced onboard sound generation (Berdahl and Ju 2011)] and UDOO [for creating interactive home automation systems (Palazzetti 2015)] are similar to tethered systems but are able to exploit their low cost and small size, to eliminate the separation between physical and digital computing. Although this difference is subtle, the ability for a full computer to be embedded into a designed object changes how designers think and create new devices; this integrated systems model is the basis for the Interaction Engine.

3.4 eMbedded-Gateway-Cloud

Lest the progression from standalone to tethered to integrated systems seem like the inevitable progress of things, we would like to point out that there is a viable competing model to the Integrated System model. The eMbedded-Gateway-Cloud (MGC) model describes the design pattern of ultra-low power, application specific embedded devices communicating with cloud-based data storage and computing service through a user's smartphone via Bluetooth Low-Energy (BLE) (Hartmann and Wright 2013). This computing model enables a class of connected devices with strict power requirements requiring only limited sensing and actuation. Multimedia interactions are displayed on a phone or website. Examples of products using this model include the FitBit activity tracker (Fitbit 2016) and Amazon Dash Button (Amazon 2016a). These objects often employ advanced microcontrollers such as the ARM Cortex-M series (ARM 2016) with ultra-low power modes where the device spends most of the time in standby and then wakes up periodically to sense, send, and receive data. Devices to help prototype these systems by combining a low power microcontroller and BTLE interface include the TI SensorTag (Texas Instruments 2016), Lightblue bean (Punch Through Design 2016), and mBed nRF51822 (mBed 2016). Early toolkits such as Amarino (Kaufmann and Buechley 2010) and Dandelion (Lin et al. 2010) supported the use of smartphones as gateways for physical hardware. More recently, cutting-edge systems such as fabryq

(McGrath et al. 2015) have further enabled designers employing the MGC model by providing a single development interface for using the phone to connect low-power physical hardware to the cloud.

There are also a number of products, such as Tessel (Technical io 2016), Particle (Spark Labs, Inc. 2016), and Electric Imp (Electric Imp, Inc. 2016), which combine powerful microcontrollers and integrated Wi-Fi to allow direct connection between physical hardware and the web. They effectively bypass the Gateway in the MGC model, but the bulk of the computation and modeling occurs in the cloud rather than at the device or node which the user is close to.

The MGC model is best suited for limited-interaction objects such as the Amazon Dash button, whereas the Interaction Engine's integrated model is similar, but focuses on a different class of untethered devices with more computational and multimedia capabilities. The Integrated Systems model is a framework for prototyping embedded objects, more akin to the Amazon Echo (Amazon 2016b) or Jibo home robot (Jibo, Inc. 2016) than to smart dust.

4 Sofabot Case Study

The SofaBot ($190 without sofa) is part of a larger investigation into how everyday objects can be made expressive and interactive. We used the sofa project to explore both benefits and challenges of using the Interaction Engine framework. This project used an ODROID U3 (Hardkernal 2016a) running a NodeJS web-server to serve a webpage directly from the sofa. The SofaBot webpage pulls a live video stream from a USB webcam using 'mjpg-streamer' (Redmer and Stoeveken 2016) an open source command line tool for lightweight video streaming. The page also reports whether someone is sitting on the couch using analog force sensitive resistors connected via an Arduino Nano. Keystrokes on the webpage allow a person to remotely control the sofa via websocket messages. The webserver in turn sends simple serial messages to the Arduino, which controls 12 V motors that move the sofa. All systems are battery powered, making the unit entirely mobile (Fig. 2).

During our development, we planned to use a Wi-Fi enabled microcontroller such as the Particle, but found that our need for a live webcam stream would require the use of a dedicated IP-camera and would not allow for future development of on-board computer vision-based interaction. We then opted to use a BeagleBone Black, as it incorporated both a Linux OS and well documented physical I/O in the same package. Unfortunately, we found development difficult due to known Wi-Fi issues based on the specific Linux kernel used as well as interference from the HDMI port's ground plane (DiCola 2015). We then used an ODROID C1 (Hardkernal 2016b) and developed all the control software to get the sofa moving in under a week, but began having power issues—the board periodically shut down.

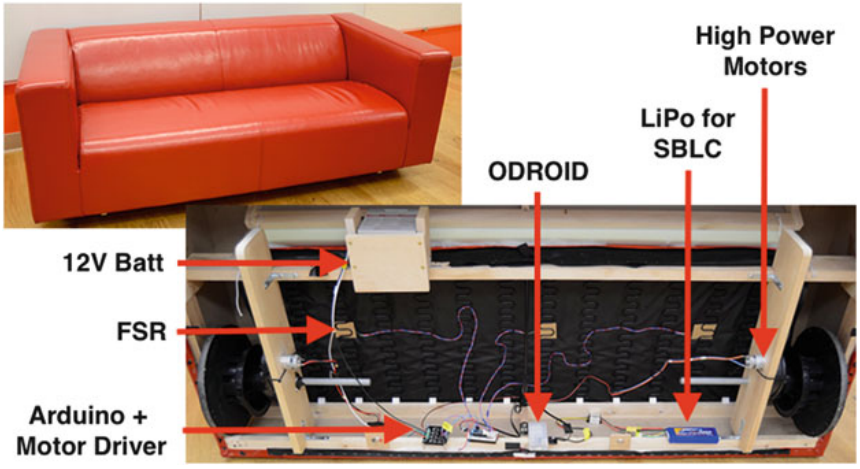

Fig. 2 SofaBot: A web connected sofa built using an Interaction Engine

Fortunately, we simply swapped in the more stable ODROID U3, downloaded the control software from our GitHub repository, and were back up and running within 15 minutes.

This process paints a realistic picture of developing new interactive devices with single-board computers. These computers are often not drop in replacements for laptops and push the designer to better understand the underlying technology. While this may add complexity, and distract from the design of the sofa's interactions, it requires the designer to understand the computer more as a material with specific properties that can be both advantageous or inhibiting to a design. In many ways, the computing elements were treated just like the motors. The sofa's motors were taken from a children's ride-on toy car and were chosen by decomposing the toy into the materials that enable movement of heavy objects, a common strategy for design prototyping (Sheppard 1992). However, many designers often "black box" the computer. By mentally deconstructing the computing systems into separate physical and digital input/output components, we could choose and swap computers based on their properties.

The SofaBot project also highlights strengths of the Interaction Engine framework, showing how existing products can quickly be networked and controlled via the web using a composite of computing tools. When we needed additional features, it was easy for us to switch from one SBLC ODROID board to another, and doing so required no changes to the motor driver circuitry or the microcontroller hardware or firmware. This would have been far more difficult if we had used something like the Arduino Yun or Beaglebone. This modularity has also benefited similar furniture projects in our lab with hardware and software quickly ported to other single-board computers such as the Raspberry Pi.

5 Exploring Interaction Engines with Designers

After our own experience developing the Interaction Engine framework, we were interested in how designers with less experience with embedded systems could benefit from it, what projects they might take on and what challenges they might encounter. We developed and ran four workshops to expose designers to the Interaction Engine's conceptual framework while also providing a concrete set of tools and examples for creating operating system enabled physical hardware. These workshops lasted from 3–6 hours and covered the basics of using single-board Linux computers, microcontrollers, and web software to connect physical hardware to a digital, web-based interface. We collected data and observations from these workshops to improve the framework and to enable new design opportunities.

5.1 Workshop Participants

We held two workshops with 16 master's design students (8 students per session) at an art and design college, a third workshop with 25 Masters and PhD students in Product Design, Mechanical Engineering, Computer Science, and Communications at a research university, and a fourth workshop at an HCI conference with seven teachers or researchers within academic design departments. In our workshops, we found that while participants often had some experience with physical computing or web-programming, most did not have experience integrating single-board Linux computers, microcontrollers, and web software to create physical/digital interactive devices.

5.2 Workshop Tutorial

At the start of our workshops, we always introduced the Interaction Engine Framework, using the image shown in Fig. 1. We described the parts in their toolkit as instances of the microcontroller or SBLC in the diagram but emphasized that these could be swapped for alternatives, based on the design. To introduce designers to working with an Interaction Engine, we developed the equivalent of a "Hello, World!" example to help designers understand the role of each of the Interaction Engine components and to illustrate physical and digital interfaces communicating over a network. Our example, called "Hello, You!" consisted of a physical button which could control the color of a web page and a web page button that could turn an LED light off and on. The tutorial walked designers through interacting with a single-board Linux computer using a console over secure shell (SSH), communicating between the computer and microcontroller using serial USB, and setting up a web server with real-time network communication to create a digital

interface that gives someone remote, wireless control of the physical hardware. We designed our tutorial to highlight the specific properties of each computing system and describe the system as a general design pattern, a useful method for teaching novice designers (Chung et al. 2004).

5.3 Our Interaction Engine "Untoolkit"

We developed our workshop toolkit based on Mellis et al.'s conception of an "untoolkit" should "frame the technology for a target audience," "leverage existing hardware and software," and "provide paths for further exploration" (Mellis et al. 2013). We created a kit of parts specifically targeted at interaction designers, leveraging widely available open source tools already common within the interaction design curriculum (Igoe 2014). Our kit included:

- **Single-Board Linux Computer:** Raspberry Pi 2B
- **Microcontroller:** Arduino Pro Micro (ATMega 32u4)
- **Wi-Fi:** Edimax USB Wifi Adapter (802.11n)
- **Web-server software:** NodeJS (Express.js)
- **Web communication:** WebSockets (SocketIO.js)

We selected the Raspberry Pi 2B for its low cost, reasonable computing power, and plentiful documentation and examples. We also chose the Arduino Pro Micro for its wide community support, as well as for its size and varied input and output options NodeJS was selected as the web server framework for its relative ease of use, low system requirements, and its large repository of libraries including `Express.js` for web applications, `SocketIO.js` for real-time WebSocket communication, and `serialport.js` for communication between the single-board computer and microcontroller. Although we developed our own simple set of examples for creating a NodeJS web server with real-time communication to and from an Arduino, this pattern is common in the open-source maker community (Waldron 2012; Hennigh-Palermo 2015; Van Every 2015). Overall, our goal in selecting these specific components was to provide designers with a set of representative tools they could continue using after the workshop.

5.4 Workshop Findings

After our workshops, we interviewed the designers about what they learned. We have compiled our findings into a set of emergent themes.

5.4.1 Getting on the Network

Due to network security at the schools we held the workshops at, we brought our own Wi-Fi router and pre-configured the Linux OS, via a text file, to automatically connect using the `occi` tool from Adafruit (Adafruit 2016). This, in turn, helped ease the transition for participants using their devices at home. Still, many participants found booting into the GUI environment easier to set up the Wi-Fi. The difficulty of getting on the network echoes our experience with other connected development boards.

5.4.2 Understanding Technologies

Many of the designers in our workshops commented on finally understanding what a server actually does. Setting up their own server with communication between a web page and hardware helped designers to understand how the internet functions, helping to demystify web services. Designers also appreciated learning about common technologies they had heard about such as NodeJS, Raspberry Pi and Arduino but were not sure how they worked or could be used for interaction design.

5.4.3 Text-Based Interaction

Using SSH was new for most designers. We often introduced the topic by saying "we are now going back to a way of programming from the 80's." Although it is possible to develop directly on the Raspberry Pi using a GUI, we felt it was important to introduce designers to developing using a text-based console common across many development boards. For code editing, we showed students how to use 'nano' but also enabled the built-in SAMBA file server on the Raspberry Pi so they could edit code on their laptops with a modern text editor. Still, many designers felt wanting for a more graphical development environment.

5.4.4 Distributed Development

Development with an Interaction Engine involves at least three different computers (MCU, server, and client) and requires a shift from programming a single device to programming a distributed system. The framework helped designers manage these messages. One designer stated that they used the framework to envision the code running on each device and arranged their code editor windows from left to right to think spatially about messages moving back and forth between devices. Whereas current day interfaces for writing code are well suited for programming one device, next generation coding interfaces within Interaction Engines might address the flow *between* devices.

5.4.5 Code Stitching

After understanding the back-end messaging and system coordination, many designers across our workshops were excited to connect their physical hardware to beautiful, interactive web user interfaces such as d3.js (Bostock) for visualizing live data from physical sensors or p5.js (McCarthy 2016) for Processing-like interactive multimedia. However, we found that integrating these toolkits was often more difficult than expected. This is similar to problems we have seen from students trying to stitch together many hardware modules on the Arduino. Future systems may help by better supporting module based development.

5.4.6 Demystifying the Computer

The Interaction Engine framework helped students to develop a better understanding of the properties of the computer as a tool for interaction design. Our modular approach helped students understand each part of the system. Feedback from students included:

> Yes—it seems less like black magic and more like, oh yeah, that's just some hardware and some code.
>
> It gave me a better understanding of exactly what's going on inside various interactive objects.

This helped boost designer's confidence toward using these systems:

> It made me realize that I could program them – they seemed less intimidating.
>
> It expanded my comfort range in terms of what I might tackle as a project.
>
> The fact that I was able to do it so quickly, and know that I can use the interaction engine to do it again if I need to gives me a bit of confidence that I might be able to do it in the future.

6 Interaction Engines in the Wild

After our workshops, we followed-up with some designers who decided to employ the Interaction Engine pattern to their own projects. Looking at three of these projects—a networked plant monitoring system, a computer vision enabled art installation, and a web-controlled interactive robot—we discuss the reflections of designers working with an Interaction Engine.

6.1 Networked Plant Monitoring

One designer, a self-taught electronics maker with a formal arts background, is developing a system to allow indoor farmers to collect data about individual plants.

His system uses multiple radio-frequency (RF) based sensor nodes to monitor individual plant data such as soil moisture and temperature. He uses a tablet with an RF unit and RFID reader to tap the plants and bring up data about them from a central server. The user can then add notes about the plant using the tablet and save this information back to the server. For his prototype, he connected the RF-based sensors back to an RF Arduino and the Raspberry Pi. He built an application using the built-in examples on the Interaction Engine that serves a tablet based web page and manages the data for the plants.

By using an Interaction Engine, this designer stated that "*it saved my prototype.*" Although he had the physical sensors and RF working, he did not have a central data server or graphical interface. He first attempted to use Bluetooth and Processing but found that the code libraries were out of date. He then built a simple web application using the Interaction Engine in about 5 hours over two working sessions the night before a project demonstration. This example shows how an Interaction Engine can extend previously built hardware, adding new functionality in little time (Fig. 3).

During the demonstration, the designer said that his main feedback was why his prototype was not in the hands of real users. He felt that his ability to create such a high fidelity, interactive prototype spoke to the power of the Interaction Engine pattern. Had the system been less functional he would have received more feedback about building a better system. Instead, those giving feedback could focus on assessing the design concept rather than on the technology itself.

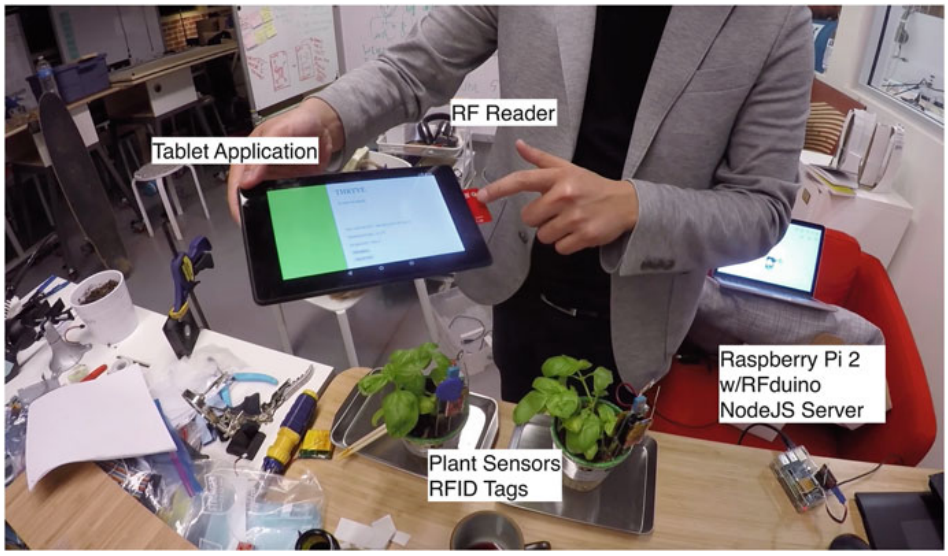

Fig. 3 Networked plant monitoring. The system uses an Interaction Engine model to integrate RF sensor nodes with a data server and tablet based web interface

6.2 Computer Vision Enabled Art

A design student employed an Interaction Engine for their Senior Art Thesis. The project consisted of our Interaction Engine toolkit running a computer vision program to identify faces and highlight them while writing "*I like it when you watch*" underneath them on a projected screen. Once a face was detected, the Interaction Engine controlled a prosthetic attached on a motor driven by a printer belt drive.

This designer extended the capabilities of their Interaction Engine by installing OpenCV (Itseez 2016), an open source computer vision library, and Processing for visual computing. They used the vision data from a Processing based program to control the motor drivers attached to an Arduino microcontroller. Using the Interaction Engine enabled the designer to fully embed the computing elements into the art exhibit. Additionally, the designer could remotely edit and update the software interactions without needing to disassemble part of the exhibit to access the computer. This improved development time and allowed the designer to make more changes (even from their home) to the interaction of the piece without altering hardware.

6.3 Web-Controlled Interactive Robot

An interaction design student built an interactive robot for use in child-child emotion regulation studies based on an Interaction Engine. The robot was designed to be wirelessly controlled from a self-served web interface, allowing a remote wizard to interact with children playing a game. The control page allows the wizard to control the head of the robot (two degrees of freedom, rotate and tilt), view a live webcam stream from the robot's point of view, and play non-verbal sounds (beeps and chimes) via a speaker. The robot was developed using a NodeJS web application communicating with an Arduino over USB. All components ran off a 16,100 mAh USB battery pack.

While developing the robot, the designer needed to make a conceptual shift from their knowledge of Arduino to an understanding of how the server enabled the connection between the Arduino and the web. This process took some time, however, after one debugging session, the designer made a realization the server was what allowed anyone on the web page to control the Arduino. After understanding the importance of the communication, the designer could greatly accelerate her work and begin focusing on the robot's behavior and appearance.

Later the designer shipped the robot to another designer who had not taken our workshop but did have experience with Linux and NodeJS. This designer was able to get the system running within a day and further extended the robot capabilities by adding the cloud-based IBM Watson text-to-speech engine (IBM 2016) to have the robot speak typed messages.

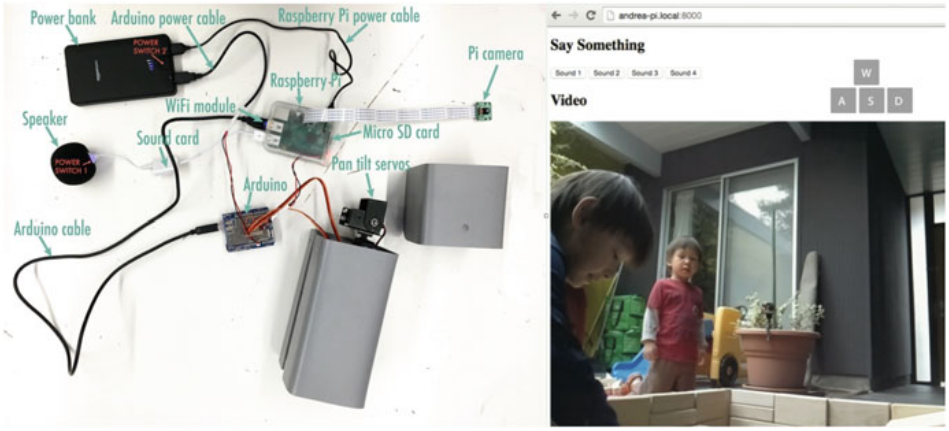

Fig. 4 Ella: A web controlled, interactive robot for child-child interaction built using an Interaction Engine

Overall, using an Interaction Engine allowed the designer to select appropriate software and hardware components for their robot in a piecemeal fashion. This case study also highlights the challenges in shifting to a framework that requires the coordination of many devices and the mental shift a designer must make to effectively use this framework. Additionally, the use of the single-board Linux computer and networking capabilities allowed another designer to easily and quickly extend the functional capabilities using cloud-based software (Fig. 4).

7 Discussion

Users and designers often employ vastly different mental models (Norman 2013), so the transition of viewing computers as products to materials can be challenging (Tullio et al. 2007). We found that the Interaction Engine framework was helpful for shifting the designer's mental model of the computer from a product to a material.

We found that the low cost of the Interaction Engine helped designers to reconsider the way they thought about how computers could be used–they were less concerned about leaving the computer in an exhibit, for example, and less hesitant to connect experimental components or to run new software. This is important, as trepidation about the preciousness of computers can inhibit designers from exploring new uses for them (Acholonu 2012).

The form-factor of the single-board computer itself was a revelation. We learned that many designers had been told that their cellular phones were powerful computers, but that they had assumed this was metaphorical; the discussion about what SBLCs do helped them realize that there really was a computer inside their phones. This, we feel, helped designers to understand the computer as a material rather than as a product.

The interface for the Interaction Engine also helps to shift the designer's mental model. We pushed our designers to interact with their Interaction Engines remotely via a text-based console rather than a GUI, for the pragmatic purpose of not requiring an external mouse, keyboard, and monitor. Most of our designers had never interacted with a computer in this way. Although being introduced to command-line interfaces was challenging for the designers at first, this interface helped designers understand how a computer works underneath the familiar graphical user interfaces.

With its mix of physical computing, operating system based computing, and connectivity, the Interaction Engine allows designers to explore a wide, new range of interactions once limited to desktop computing in a far wider range of user contexts. The components and more importantly, the framework, help to create a cognitive tool that helps support and guide designer's thinking toward new opportunities (Derry 1990). This conversion is not without it challenges, as the designer must begin to think deeply (Jonassen 1994) about what it means for a computer to be a material.

As everyday objects become more imbued with computation and begin to communicate with other devices both near and far, it will become more important for designers to understand how new computing tools can enable new designs. Understanding the various sub-systems within computing systems will allow designers to better appropriate new technologies as they become available (Kay 1998). Although it may be inefficient for final products, the *bricolage* approach encouraged by the Interaction Engine framework allows designers to more easily prototype interactive objects using different connected modules; this may be different from what is preferred for expert electrical engineers or computer programmers (Stiller 2009).

8 Future Work

Although single-board Linux computers and microcontrollers are becoming more usable as design materials, there are still many areas for improvement.

Getting on the Network Getting on the network is still one of the most difficult parts of setting up an Interaction Engine. Though we have used some tools to make this easier, there are opportunities for new design patterns to connect devices.

Modern Interfaces For many designers, Unix and command-line interfaces can have a steep learning curve. While we felt this was pedagogically beneficial, updated graphical interfaces and setup tools will enable wider access to the Interaction Engine's capabilities.

Managing Messages Coordinating messages between numerous computing units can be hard. Improved tools for managing interactions between MCU, SBLC, and the cloud can help simplify development. Additionally, message management systems can be standardized to work across hardware types allowing for sub-component upgradeability.

Developing for Multiple Devices Jumping between devices can be conceptually difficult as projects become larger. New work in high-level languages for writing code across multiple devices and services like Ravel (Riliskis and Levis 2014) and fabryq (McGrath et al. 2015) will help improve the development process.

Cloud Based Computing Modules As machine learning and artificial intelligence systems become more advanced, it will be important to develop usable interfaces for designers to access these capabilities. Having an OS with standard web communication will allow a new class of interaction design modules to be created and used on a wide range of devices.

9 Conclusion

As computing trends shift and the nature of interactive devices add more networked and operating system based technologies, it will be more important for designers to understand the underlying capabilities of these systems. In understanding the components and conceptual framework of the Interaction Engine pattern, we feel that designers will more readily be able to utilize existing and emerging technologies to their fullest advantage to create better-designed products.

Acknowledgements We would like to thank the following people for their help with this work: Michael Shiloh, Victor Chahuneau, Tom Igoe, Bjoern Hartmann, Rob Semmens, Dickson Chow, Phillip Dupree, Noam Zomerfeld, Carey Smith, Nathan Seidel, Bill Verplank, CCA & the Stanford Design Group.

References

Acholonu, U. C. (2012). *Techonology [sic] bricolage exploring the tension between convention and innovation when problem-solving with technology*. Stanford University.

Adafruit. (2016). *Adafruit-occidentalis*. Retrieved August 1, 2016, from https://github.com/adafruit/Adafruit-Occidentalis

Amazon. (2016a). *Amazon dash button*. Retrieved July 31, 2016, from https://www.amazon.com/Dash-Buttons/b?ie=UTF8&node=10667898011

Amazon. (2016b). *Amazon echo*. Retrieved July 30, 2016, from https://www.amazon.com/Amazon-Echo-Bluetooth-Speaker-with-WiFi-Alexa/dp/B00X4WHP5E

Aminzade, D., Hartmann, B., Doorley, S., et al. (2008). *Interactive digital murals at San Jose Museum of Art.*

ARM. (2016). *Cortex-M Series*. Retrieved July 31, 2016, from http://www.arm.com/products/processors/cortex-m/index.php

Ballagas, R., Ringel, M., Stone, M., & Borchers, J. (2003). iStuff: A physical user interface toolkit for ubiquitous computing environments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 537–544). New York: ACM.

Banzi, M., & Shiloh, M. (2014). *Make: Getting started with Arduino: The open source electronics prototyping platform*. Sebastopol: Maker Media.

Barragán, H. (2004). *Wiring: Prototyping physical interaction design*.

Beigl, M., Gellersen, H.-W., & Schmidt, A. (2001). Mediacups: Experience with design and use of computer-augmented everyday artefacts. *Computer Networks, 35*, 401–409. doi:10.1016/S1389-1286(00)00180-8.

Berdahl, E., & Ju, W. (2011). Satellite CCRMA: A musical interaction and sound synthesis platform. In *Proceedings of the International Conference on New Interfaces for Musical Expression* (pp. 173–178).

Borthwick, M. (2016). *Building museum exhibits with open hardware: Raspberry Pi & Arduino*.

Bostock, M. *D3.js—Data-driven documents*. Retrieved July 26, 2016, from https://d3js.org/

Buechley, L., Eisenberg, M., Catchen, J., & 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* (pp. 423–432). New York: ACM.

Buechley, L., Elumeze, N., & Eisenberg, M. (2006). Electronic/computational textiles and children's crafts. In *Proceedings of the 2006 Conference on Interaction Design and Children* (pp. 49–56). New York: ACM.

Buechley, L., & Hill, B. M. (2010). LilyPad in the wild: How hardware's long tail is supporting new engineering and design communities. In *Proceedings of the 8th ACM Conference on Designing Interactive Systems* (pp. 199–207). New York: ACM.

Bush, V. (1989). As we may think. In *Perspectives on the computer revolution* (p. 49).

Chan, L., Hsieh, C.-H., Chen, Y.-L., Yang, S., Huang, D.-Y., Liang, R.-H., et al. (2015) Cyclops: Wearable and single-piece full-body gesture input devices. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (pp. 3001–3009). New York: ACM.

Charlesworth, T., Ford, H., Milton, L., Mortensson, T., Pedlingham, J., Knibbe, J., et al. (2015). TellTale: Adding a polygraph to everyday life. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems* (pp. 1693–1698). New York: ACM.

Chung, E. S., Hong, J. I., Lin, J., Prabaker, M. K., & Landay, J. A. (2004). Development and evaluation of emerging design patterns for ubiquitous computing. In *Proceedings of the 5th conference on designing interactive systems: processes, practices, methods, and techniques* (pp. 233–242). New York: ACM.

Derry, S. (1990). Flexible cognitive tools for problem solving instruction. In *Annual meeting of the American Educational Research Association, Boston, MA*.

DiCola, T. (2015). *Setting up WiFi with BeagleBone Black|Adafruit Learning System*. Retrieved July 30, 2016, from https://learn.adafruit.com/setting-up-wifi-with-beaglebone-black/overview

Dow, S. P., Glassco, A., Kass, J., Schwarz, M., Schwarz, D. L., & Klemmer, S. R. (2010). Parallel prototyping leads to better design results, more divergence, and increased self-efficacy. *ACM Transactions on Computer-Human Interaction TOCHI, 17*, 18. doi:10.1145/1879831.1879836.

Electric Imp, Inc. (2016). *Electric Imp—Electric imp hardware.* Retrieved July 31, 2016, from https://www.electricimp.com/platform/hardware/

Fitbit. (2016). *Fitbit official site*. Retrieved July 31, 2016, from https://www.fitbit.com/

Frei, P., Su, V., Mikhak, B., & Ishii, H. (2000). Curlybot: Designing a new class of computational toys. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 129–136). New York: ACM.

Gaver, W., Michael, M., Kerridge, T., Wilkie, A., Boucher, A., Ovalle, L., et al. (2015). Energy babble: Mixing environmentally-oriented internet content to engage community groups. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (pp. 1115–1124). New York: ACM.

Gibb, A. (2009). Bug labs: Hacks and apps. *Linux J, 2009*, 4.

Gibb, A. M. (2010). *New media art, design, and the Arduino microcontroller: A malleable tool*. Pratt Institute.

Greenberg, S., & Fitchett, C. (2001). Phidgets: Easy development of physical interfaces through physical widgets. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology* (pp. 209–218). New York: ACM.

Hardkernal. (2016a). ODROID U3. Retrieved July 30, 2016, from http://www.hardkernel.com/main/products/prdt_info.php?g_code=g138745696275

Hardkernal. (2016b). ODROID C1. Retrieved July 30, 2016, from http://www.hardkernel.com/main/products/prdt_info.php?g_code=G141578608433

Harris-Cronin, S. (2008). *High resolution video playback*.

Hartmann, B., Klemmer, S. R., Bernstein, M., Abdulla, L., Burr, B., Robinson-Mosher, A., et al. (2006). Reflective physical prototyping through integrated design, test, and analysis. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology* (pp. 299–308). New York: ACM.

Hartmann, B., & Wright, P. K. (2013). Designing bespoke interactive devices. *Computer, 46*, 85–89. doi:10.1109/MC.2013.274.

Hennigh-Palermo, S. G. (2015). *p5bots*. Retrieved July 21, 2016, from https://github.com/sarahgp/p5bots

Huang, A. B. (2008). Chumby: An experiment in hackable pervasive computing. *IEEE Pervasive Computing, 7*, 55–62.

IBM. (2016) watson-developer-cloud/text-to-speech-nodejs. Retrieved July 26, 2016, from https://github.com/watson-developer-cloud/text-to-speech-nodejs

Igoe, T. (2014). *Lab: Serial communication with Node.js—ITP physical computing*.

Itseez. (2016). OpenCV|OpenCV. Retrieved July 26, 2016, from http://opencv.org/

Jibo, Inc. (2016). *Jibo—The worlds first social robot*. Retrieved July 31, 2016, from https://www.jibo.com/

Jonassen, D. H. (1994). Technology as cognitive tools: Learners as designers. *ITForum Paper, 1*, 67–80.

Kaufmann, B., & Buechley, L. (2010). Amarino: A toolkit for the rapid prototyping of mobile ubiquitous computing. In: *Proceedings of the 12th International Conference on Human Computer Interaction with Mobile Devices and Services* (pp. 291–298). New York: ACM.

Kay, A. (1998). *Alan Kay on messaging*.

Kubitza, T., Schmidt, A., Pohl, N., Petrelli, D., Dingler, T., & Dulake, N. (2013). Tools and methods for creating interactive artifacts. In *Proceedings of the 8th International Conference on Tangible, Embedded and Embodied Interaction* (pp. 385–388). New York: ACM.

Kuniavsky, M. (2010). *Smart things: Ubiquitous computing user experience design*. Amsterdam: Elsevier.

Landin, H. (2005). Fragile and magical: Materiality of computational technology as design material. In *Proceedings of the 4th Decennial Conference on Critical Computing: Between Sense and Sensibility* (pp. 117–120). New York: ACM.

Langer, M., & Alderman, J. (2016). Open hardware belongs in your museum. In *Museums and the Web 2016*. Los Angeles, CA.

Lee, J. C., Avrahami, D., Hudson, S. E., Forlizzi, J., Dietz, P. H., & Leigh, D. (2004). The Calder Toolkit: wired and wireless components for rapidly prototyping interactive devices. In *Proceedings of the 5th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques* (pp. 167–175). New York: ACM.

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., et al. (2005). TinyOS: An operating system for sensor networks. In: W. Weber, J. M. Rabaey, E. Aarts (Eds.), *Ambient intelligence* (pp. 115–148). Berlin: Springer.

Lin, F. X., Rahmati, A., & Zhong, L. (2010). Dandelion: A framework for transparently programming phone-centered wireless body sensor applications for health. In *Wireless health 2010* (pp. 74–83). New York: ACM.

Marshall, M. T., Dulake, N., Petrelli, D., & Kockelkorn, H. (2015). From the deposit to the exhibit floor: An exploration on giving museum objects personality and social life. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems* (pp. 1917–1922). New York: ACM.

Martelaro, N., Shiloh, M., & Ju, W. (2016). The interaction engine: Tools for prototyping connected devices. In *Proceedings of the TEI '16: Tenth International Conference on Tangible, Embedded, and Embodied Interaction* (pp. 762–765). New York: ACM.

Marvell. (2016). *Kinoma Create*. Retrieved July 31, 2016, from http://kinoma.com/create/
mBed. (2016). *Nordic nRF51822*. Retrieved July 31, 2016, from https://developer.mbed.org/platforms/Nordic-nRF51822/
McCarthy, L. (2016). p5.js (Computer Program).
McComb, G. (2003). *Robot builder's bonanza* (3rd ed.). New York: McGraw-Hill.
McGrath, W., Etemadi, M., Roy, S., & Hartmann, B. (2015). Fabryq: Using phones as gateways to prototype internet of things applications using web scripting. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (pp. 164–173). New York: ACM.
Mellis, D., Banzi, M., Cuartielles, D., & Igoe, T. (2007). Arduino: An open electronic prototyping platform. In *Proceedings of CHI*.
Mellis, D. A., & Buechley, L. (2014). Do-it-yourself cellphones: An investigation into the possibilities and limits of high-tech diy. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1723–1732). New York: ACM.
Mellis, D. A, Jacoby, S., Buechley, L., Perner-Wilson, H., & Qi, J. (2013). Microcontrollers as material: Crafting circuits with paper, conductive ink, electronic components, and an "untoolkit." In *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction* (pp. 83–90). New York: ACM.
Merrill, D., Kalanithi, J., & Maes, P. (2007). Siftables: Towards sensor network user interfaces. In: *Proceedings of the 1st International Conference on Tangible and Embedded Interaction* (pp. 75–78). New York: ACM.
Millner, A., & Baafi, E. (2011). Modkit: Blending and extending approachable platforms for creating computer programs and interactive objects. In *Proceedings of the 10th International Conference on Interaction Design and Children* (pp. 250–253). New York: ACM.
Norman, D. A. (2013). *The design of everyday things: Revised and expanded edition*. New York: Basic books.
O'Sullivan, D., & Igoe, T. (2004). *Physical computing: Sensing and controlling the physical world with computers*. Boston, MA: Course Technology Press.
Output Arts. (2012). I have a message for you . . .
Overholt, D., Møbius, N. "DZL". (2013). Embedded audio without beeps: Synthesis and sound effects from cheap to steep. In *Proceedings of the 8th International Conference on Tangible, Embedded and Embodied Interaction* (pp. 361–364). New York: ACM.
Palazzetti, E. (2015). *Getting Started with UDOO*. Birmingham: Packt.
Punch Through Design. (2016). *LightBlue Bean*. Retrieved July 31, 2016, from https://punchthrough.com/bean
Qi, J., & Buechley, L. (2010). Electronic popables: Exploring paper-based computing through an interactive pop-up book. In *Proceedings of the Fourth International Conference on Tangible, Embedded, and Embodied Interaction* (pp. 121–128). New York: ACM.
Raghavan, P., Lad, A., & Neelakandan, S. (2005). *Embedded linux system design and development*. Boca Raton: CRC Press.
Redmer, A., & Stoeveken, T. (2016). MJPEG-STREAMER.
Resnick, M., Martin, F., Berg, R., Borovoy, R., Colella, V., Kramer, K., et al. (1998). Digital manipulatives: New toys to think with. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 281–287). New York: ACM Press/Addison-Wesley.
Riliskis, L., & Levis, P. (2014). Ravel a framework for embedded-gateway-cloud applications. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems* (pp. 344–345). New York: ACM.
Roman, M., Al-Muhtadi, J., Ziebart, B., Campbell, R., & Mickunas, M. D. (2003). System support for rapid ubiquitous computing application development and evaluation. In *System Support for Ubiquitous Computing Workshop (UbiSys' 03) in conjunction with UbiComp*.
Rozin, D. (1999). Wooden mirror.
Russell, S. C. (2012). Pi and Arduino in action. *The MagPi*, 4–6.
Sculley, J. (1987). *Odyssey: Pepsi to apple, a journey of adventure, ideas, and the future*. New York: Harper & Row.

Sheppard, S. D. (1992). Mechanical dissection: An experience in how things work. In *Proceedings of the Engineering Education: Curriculum Innovation & Integration*, January 6–10.

Sipitakiat, A., & Blikstein, P. (2013). Interaction design and physical computing in the era of miniature embedded computers. In *Proceedings of the 12th International Conference on Interaction Design and Children* (pp. 515–518). New York: ACM.

Spark Labs, Inc. (2016) Particle. In *Particle*. Retrieved July 31, 2016, from https://www.particle.io/prototype

Stankovic, J. A., Lee, I., Mok, A., & Rajkumar, R. (2005). Opportunities and obligations for physical computing systems. *Computer, 38*, 23–31. doi:10.1109/MC.2005.386.

Stiller, E. (2009). Teaching programming using bricolage. *Journal of Computing Sciences in Colleges, 24*, 35–42.

Tahiroğlu, K., Svedström, T., & Wikström, V. (2015). NOISA: A novel intelligent system facilitating smart interaction. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems* (pp. 279–282). New York: ACM.

Tan, S. L., & Nguyen, B. A. T. (2009). Survey and performance evaluation of real-time operating systems (RTOS) for small microcontrollers. *IEEE Micro*. doi: 10.1109/MM.2009.56.

Technical io. (2016). *Tessel*. Retrieved July 31, 2016, from https://tessel.io/

Texas Instruments. (2016). *Simplelink SensorTag*.

The Workers. (2015). *After dark* (Artwork).

Townsend, A. M. (2013). *Smart cities: Big data, civic hackers, and the quest for a new utopia*. New York: WW Norton & Company.

Tullio, J., Dey, A. K., Chalecki, J., & Fogarty, J. (2007). How it works: A field study of non-technical users interacting with an intelligent system. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 31–40). New York: ACM.

Vallgårda, A., & Redström, J. (2007). Computational composites. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 513–522). New York: ACM.

Vallgårda, A., & Sokoler, T. A. Material strategy: Exploring material properties of computers. *International Journal of Design*. Retrieved August 2, 2016, from http://www.ijdesign.org/ojs/index.php/IJDesign/article/view/628/309

Van Every, S. (2015). p5.serialport (Computer Program). Retrieved July 21, 2016, from https://github.com/vanevery/p5.serialport

Vandevelde, C., Vanhoucke, M., & Saldien, J. (2015). Prototyping social interactions with DIY animatronic creatures. In *Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction* (pp. 509–512). New York: ACM.

Villar, N., Scott, J., Hodges, S., Hammil, K., & Miler, C. (2012). .NET gadgeteer: A platform for custom devices. In *Pervasive computing* (pp. 216–233). Berlin: Springer.

Waldron, R. (2012). rwaldron/johnny-five (Computer Program). Retrieved July 21, 2016, from https://github.com/rwaldron/johnny-five

Weiser, M. (1999). The computer for the 21st century. *SIGMOBILE Mobile Computing and Communications Review, 3*, 3–11. doi:10.1145/329124.329126.

Willow Garage. (2011). Turtlebot. In *Turtlebot*. Retrieved January 4, 2017, from http://www.turtlebot.com

Wilson, S., Gurevich, M., Verplank, B., & Stang, P. (2003). Microcontrollers in music HCI instruction: Reflections on our switch to the Atmel AVR platform. In *Proceedings of the 2003 Conference on New Interfaces for Musical Expression* (pp. 24–29). Singapore: National University of Singapore.