WATCHCONNECT: A Toolkit for Prototyping Smartwatch-Centric Cross-Device Applications

Steven Houben [1,2]
[1] Pervasive Interaction Technology Laboratory
IT University of Copenhagen
Rued Langgaardsvej 7, Copenhagen, DK
s.houben@ucl.ac.uk

Nicolai Marquardt [2]
[2] University College London
UCL Interaction Centre / ICRI Cities
Gower Street, London, UK
n.marquardt@ucl.ac.uk

ABSTRACT
People increasingly use smartwatches in tandem with other devices such as smartphones, laptops or tablets. This allows for novel cross-device applications that use the watch as both input device and output display. However, despite the increasing availability of smartwatches, prototyping cross-device watch-centric applications remains a challenging task. Developers are limited in the applications they can explore as available toolkits provide only limited access to different types of input sensors for cross-device interactions. To address this problem, we introduce *WatchConnect*, a toolkit for rapidly prototyping cross-device applications and interaction techniques with smartwatches. The toolkit provides developers with (i) an extendable hardware platform that emulates a smartwatch, (ii) a UI framework that integrates with an existing UI builder, and (iii) a rich set of input and output events using a range of built-in sensor mappings. We demonstrate the versatility and design space of the toolkit with five interaction techniques and applications.

Author Keywords
Smartwatch; Toolkit; Cross-Device Interaction; Rapid Prototyping; Gestural Interaction; Interface Design

ACM Classification Keywords
H.5.2. Information Interfaces. User Interfaces – input devices and strategies, prototyping.

INTRODUCTION
Smartwatches give people lightweight and immediate access to messages, notifications, and other digital data while on the go. While already powerful as standalone devices, the capabilities of smartwatches increase significantly when used in tandem with other devices that people carry, such as their phones or tablets, which allows for novel cross-device interaction techniques (e.g. [7,24]). However, so far there are only a relatively small number of explorations into *watch-centric, cross-device interaction techniques*. Building and exploring cross-device interaction techniques and applications is a difficult task, as most existing development kits have only limited support for input gesture recognition, different sensor hardware configurations, rapid interface designs, or cross-device connectivity and transfer of information.

CHI 2015, April 18 - 23 2015, Seoul, Republic of Korea

ACM 978-1-4503-3145-6/15/04...$15.00
http://dx.doi.org/10.1145/2702123.2702215

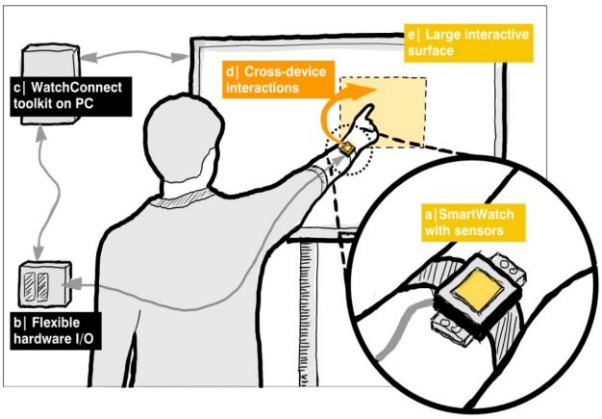

Figure 1. *WatchConnect* toolkit consists of (a) wired prototyping smartwatches with sensors through a (b) flexible and extendable hardware layer, (c) a software development platform providing user interface components and a rich set of input and output events and gestures, facilitating (d) cross-device interactions with (e) other interactive surfaces.

To bridge the gap between concept design and full implementation, we introduce *WatchConnect*, a rapid prototyping toolkit for watch-centric cross-device interaction techniques and applications (Figure 1). The toolkit provides (i) a modular and extendable hardware platform that emulates a smartwatch, (ii) a runtime system and user interface components that support quick prototyping of watch interfaces using an existing UI framework, and (iii) a rich set of input and output events and gestures using a range of built-in sensor mappings and simulators. The contribution of this paper is a *novel approach for rapidly prototyping and designing smartwatch-centric cross-device applications and interaction techniques, using simulated hardware and software building blocks.*

In this paper we first sample key related work and introduce the design of the *WatchConnect* toolkit. We proceed with the details of the architecture and components of the toolkit. Next, we demonstrate the versatility and generality of the toolkit by implementing five applications using only the basic building blocks of the toolkit. We conclude this paper with a discussion and reflection on the design and features of the toolkit, compared to other approaches.

RELATED WORK

WatchConnect builds on work on interaction techniques for smartwatches, cross-device setups, and toolkit designs.

Smartwatch Interactions

Most smartwatches allow for *touch* input. Ashbrook et al. [1] explored interaction techniques for round touch-enabled watch faces. Facet [22] allows for multi-screen interactions by expanding the watch to multiple touch-enabled watch faces arranged as a bracelet. Later, Duet [7] introduced a set of cross-device interaction techniques using both the touch screen and sensors of the watch. TouchSense [18] expanded the touch bandwidth of a watch screen, by augmenting the human finger with an IMU. Finally, Mayer et al. [24] employed the touch screen of a watch to interact with objects in the environment. A number of other approaches moved touch interaction to the *bevel* and *band* of the watch face. Blasko et al. [4] support bidirectional strokes on the frame of the watch providing tactile feedback. Oakley et al. [29] expanded this idea to the side of the bevel providing high resolution capacitive input. Xiao et al. [38] moved away from a static bevel and introduced mechanical input such as panning, twisting, tilting and clicking the bevel. Watchit [33] is the first approach that moves touch interaction and scroll gestures to the wristband. More recently, Funk et al. [8] explored using the wristband for touch-enabled text entry. Finally, Abracadabra [12] is a system that supports *above* the device interaction using a magnetic input sensor.

Other systems expanded interaction with a smartwatch by using the arm or hand for gesture or touch input. One of the first explorations into smartwatches, was Gesturewrist [35], augmenting a watch with sensors to allow for hand gesture and arm posture recognition. Gesture Watch [19] augments a watch face with sensors for the detection of swipes gestures above and around the watch. Similarly, the Haptic Wristwatch [32] allows for detection of gestures such as covering the watch, turning the bevel, or swipe over the watch. AugmentedForearm [30] took this concept further, stretching the touch display of the watch across the entire forearm. Knibbe et al. [20] augmented the watch with proximity and acoustic sensors to detect hand postures and multi-finger interactions. Finally, Skin buttons [21] project touch-enabled interface elements on the skin. Other approaches include interaction with the back of a small display [3], and with small spatial aware displays such as Siftables [25].

Cross-Device Interaction Techniques

Cross-device interaction techniques have been explored in a wide range of other device configurations. Pick and Drop [34] introduced cross-device direct manipulation. Hinckley et al. [17] allow users to bump devices together into a single workspace, using synchronized gestures. Another approach is to stitch devices together to allow for cross-device pen input [16]. Hardy et al. [11] proposed to use the back of the phone to select and interact with information on a large display. Similarly, PhoneTouch [36] allows users to interact with an interactive surface, using their phones as a personal device to configure or change the interaction with the surface. Cross-device interaction techniques were described in function of proxemics in the *gradual engagement pattern* [23]. Only recently, systems explicitly used smartwatches for cross-device interaction. Duet [7] introduced a number of interaction techniques and gestures to support distributed interaction between a watch and smartphone. Mayer et al. [24] proposed "*user interfaces beaming*" to interact with objects that are in the focus of a head-mounted display. SleeD [40] uses a sleeve display for interaction techniques distributed between the sleeve and a large interactive wall display.

Toolkits and Programming Interfaces

In recent years, a number of novel cross-device interface design toolkits have been proposed to mitigate the engineering challenges in building distributed interfaces. HydraScope [14] supports multi-surface interfaces by transforming and synchronizing existing web-based applications. Conductor [10] is a prototyping framework that allows for the construction of cross-device applications and provides task-, session-, and information-management. Panelrama [39] is a web-based toolkit for DUIs that supports built-in UI synchronization across devices by allowing developers to specify the suitability of groups of UIs (or *panels*) that are used by an algorithm to automatically distribute panels across devices. XDStudio [27] is a GUI builder that supports interactive development of cross-device interfaces through the simulation of devices, or by actual on-device authoring. The Tandem Browsing Toolkit [15] is a proxy-based online multi-display application toolkit that provides developers with a declarative framework to define multi-device web pages. XDKinect [28] is a cross-device interface toolkit that uses a Kinect depth camera to mediate interaction between different devices. The toolkit allows for proxemic-aware interaction, body tracking and multi-modal input. Finally, PolyChrome [2] is a toolkit for multi-device collaborative applications that provide support for concurrency management. A small number of commercial application programming interfaces (APIs), such as the Pebble [41], Sony SDK [42] or Apple's WatchKit [43] are available for developers.

These toolkits and APIs, however, are designed for existing hardware platforms and interfaces and provide no support for novel hardware designs, custom sensor mappings or watch-specific cross-device interfaces. Although they provide means to synchronize UIs and events, using custom hardware or designing specific gestures and postures would still require substantial engineering. While still possible to build single smartwatch applications (as seen in the related work), the challenges to build those prevent rapid prototyping and experimentation [9]. Existing commercial watch APIs require proprietary hardware and lack support for rapid prototyping of cross-device applications. In contrast, *WatchConnect* provides holistic support for the entire prototyping cycle including (i) hardware design, abstraction and mapping, (ii) built-in machine learning and gesture recognition, (iii) distributed user interface and event systems, and (iv) a high level visual programming framework and tools.

INTERACTION SPACE

To summarize the challenges of supporting interaction between a watch and an interactive surface, we present an overview on the *interaction space* that emerges when connecting the input and output space of both the watch and surface.

Watch Input Space

Prior work shows that the sensors built into smartwatches provide three interaction spaces:

W1: On the watch interaction. A watch allows for direct interaction through physical contact with the device. Users can touch the screen of the watch [1,7,18,22,24], grab and interact with the bevel of the watch face [4,29,38] for discrete touch input, or interact with a touch-enabled wristband to provide continuous input [8,33]. Combining these different modalities into one watch design provides users with a very rich input device that allows for combinations of screen, bevel and strap input.

W2: Above the watch interaction. Users can perform gestures with the non-watch-arm in the three dimensional space above the watch. Although proximity sensors and depth cameras are becoming increasingly popular, only Abracadabra [12] currently supports above the watch interaction. However, a watch equipped with distance sensors or light sensors, that are frequently used to support mid-air gestures such as in SideSight [5], can provide both continuous and discrete input. This allows for a range of gestures above the watch such as covering the watch face, hovering and holding above the edges of the watch, zooming by moving the hand closer and away from the watch face or simply using the measured distance as discrete input.

W3: Interaction via internal sensing. Integrated watch sensors can provide data on the acceleration and orientation of the device that allow for a wide range of both implicit and explicit gestures. Implicit gestures can be used to, e.g., automatically turn the watch screen on or off depending on the orientation of the watch. As demonstrated by Duet [7], TouchSense [18] and GestureWrist [35], explicit gestures allow users to switch interaction modes or express hand posture and gestures. A high granularity of input allows one to use the watch as a game controller or to express different input forms with the watch hand. Similar to other interaction spaces, the integrated sensors support both continuous and discrete input.

Interactive Surface Input Space

When wearing a watch to interact with *another touch screen* – for example a tablet or a digital whiteboard – the setup has three basic input spaces (informed by [37]):

S1: Interaction Connector Point. Because the watch hand can be recognized using the built-in sensors (as demonstrated in Duet [7]), it can be used to identify the user and to connect a specific user session to the interactive display. Identifying the user behind a touch input, as done by Schmidt et al. [36] using a mobile phone, allows applications and interaction techniques to incorporate user specific functionality, to personalize the user interface or to use the input for authentication.

S2: Interaction Collision Plane. When touching the external touch screen with the watch hand, a two dimensional input space is created that is merged with the normal touch-based input space. The screen can differentiate between touches performed with the watch hand and non-watch hand. This allows applications and interaction techniques to consider bimanual input in which specific modalities or functionality is assigned to a specific hand. Furthermore, the built-in sensors allow the screen to detect touches from the watch hand with a higher degree of granularity, thus allowing for the detection of, e.g., back of the hand, knuckle or nail touches [7].

S3: Interaction Volume. The orientation and acceleration of the watch hand can be used for expressive input, adaptive user interfaces or even mid-air gestures. Furthermore, by combining three-dimensional spatial interaction with touches from the non-watch hand, applications and interaction techniques can support advanced scenarios. Examples include navigation in three-dimensional applications, game input, gestural interaction, and gradual transitions of UI elements between devices [23].

Joint Output Space

When using the watch and interactive surface, the combination of both displays creates an output space that can be used in three configurations:

O1: Output on interactive display. The output of the interaction technique or application is shown only on the display of the interactive screen, and not on the watch. This configuration can support scenarios in which the watch is used purely as an input sensor (such as, e.g., detecting how the watch hand is touching the screen [7]) or when user-specific personalized user interface elements are shown on the display [37] based on touch input.

O2: Output on watch display. The output of the interaction with the interactive display is only shown on the small watch display. This setup can be used to provide a private or contextual view (such as, e.g., a peephole metaphor on a static map) of the data shown on the interactive surface [37].

O3: Output distributed across displays. The output or feedback of the interaction between both devices is distributed or shared across both displays [37]. This configuration allows for scenarios in which both the interactive display and the watch display are updated to reflect or visualize cross-device interactions.

Temporal Synchronized Interaction

User actions in this interaction space combine input and output spaces of both devices. By performing temporally sequenced touches, postures and gestures, users can express input and interact with the dual setup. *Temporal interactions*

provide users with a fine-grained distributed interaction framework. Interaction designers can combine touches, postures or gestures in any arbitrary sequence. *WatchConnect* is designed to support the prototyping of temporal interactions.

TOOLKIT

To mitigate the challenges in designing and prototyping watch-centric cross-device interaction, we present the *WatchConnect* toolkit. The major goal of the toolkit is to provide *a fast event-driven platform* for *rapid prototyping* of *watch-centric cross-device interaction techniques and applications*. The *WatchConnect* toolkit is composed of two parts: (i) a flexible and extendable hardware platform that emulates a smartwatch, and (ii) a software platform providing user interface components and a rich set of input and output events and gestures, based on default sensor mappings. The toolkit is integrated with an existing visual user interface design tool (WPF Visual Studio) to support a rich set of existing UI components and framework, and existing platforms for rapid hardware prototyping (Phidgets [44] and Arduino [45]). In this section, we provide an overview of the architecture and components of the toolkit.

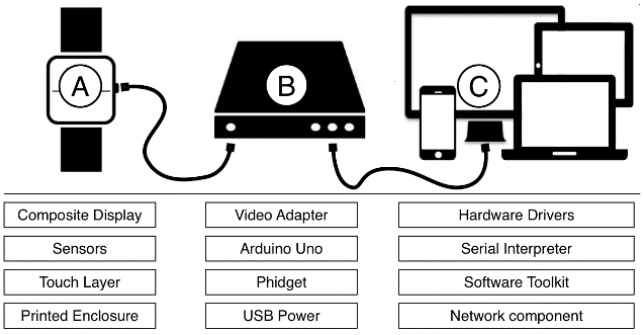

Figure 2. An overview of the *WatchConnect* toolkit.

Hardware

The *WatchConnect* toolkit is built around a wired *prototyping watch*, a smart watch emulator (Figure 2A and Figure 3) that is composed of a miniature display, a number of touch and motion sensors, and a microprocessor integrated into a form factor that *resembles a smart watch*. Using a physical cable, the *prototyping watch* is connected to the base station (Figure 2B) which converts and sends the data from the sensors and screen over a USB cable to the development computer (e.g., tablet or a large interactive surface) that runs the emulator software as well as the main toolkit (Figure 2C).

The default *prototyping watch* (Figure 3) is built around the Arduino platform and contains a light sensor, two infrared proximity sensors, an 8 channel capacitive touch sensor, a six-axis MEMS motion tracker (gyro + accelerometer), an RGB led, a flexible force sensing potentiometer and a 2 inch TFT display. The hardware components are soldered on a PCB, which slides into the 3D printed enclosure that is mounted on a wristband. Because of this setup, developers can easily extend the design with additional sensors, reconfigure the layout of the sensors or even redesign the existing watch hardware. Although the default watch uses Arduino, the toolkit also supports Phidgets to allow for fast plug and play prototyping (but somewhat bulkier components) without the need to write code for the hardware emulator.

The base station, which is connected to the development computer device using a USB and a VGA cable, consists of an Arduino microprocessor, a Phidgets interface kit, a USB power supply and VGA to component converter. All sensors are connected to either the Arduino or Phidget interface kit, which push the sensor readings over a serial protocol to the master device. The VGA converter converts the screen output from the development computer into a component signal, which is shown on the miniature display. To allow the software toolkit to analyze sensor data, a structured data exchange protocol is used which is composed of three parts: (i) a header that describes the sensor, (ii) the body that contains the sensor readings and (iii) the closing symbol that signifies the end of a package.

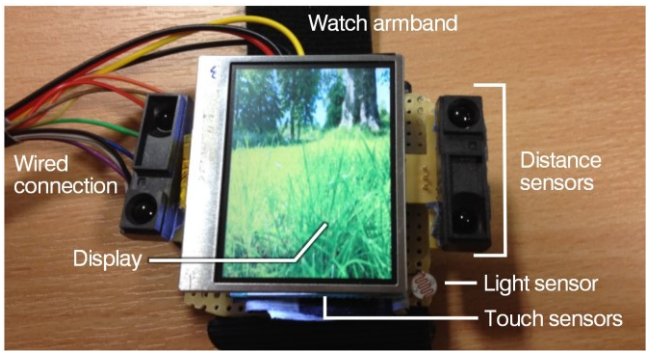

Figure 3. The Arduino-based watch probe with sensors.

Software

The toolkit's software architecture consists of six modules (Figure 4): (i) an *interface* library that includes a runtime and UI framework for the watch, (ii) an *input* library, providing touch, gesture and tracking input, (iii) a *hardware* and (iv) *processing* layer that abstracts the hardware and machine learning into events and gestures, (v) a *network* library that wraps REST and web socket services around the watch runtime and (vi) a *tools* library that provides applications to inspect and calibrate raw sensor data. In this section, we provide more details on the different modules.

Toolkit.Interface

The *WatchRuntime* is the central object of the toolkit in which all other toolkit components are merged into a single runtime environment that is used by developers to create a new watch application. The runtime, that can be configured and setup by using the *WatchConfiguration*, initiates all input, sensor management, processing and output into a watch window. When the hardware base station and watch probe are connected, the *Window Manager* of the runtime will push the watch window to the probe. If no hardware is connected, the runtime launches a native window to show the output. Watch applications can be designed using standard c# Windows Presentation Foundation (WPF) components in Visual Studio and the Expression Blend UI designer.

The only requirement for compatibility with *WatchConnect* is that watch applications are designed as user controls that inherit from the WatchVisual class, provided by the toolkit. Applications can be added and launched in the runtime by simply adding them as a new visual. Internally, the runtime manages all applications using a *WatchManager* that provides developers with a basic operating system-like environment to swap out watch applications. Through the *WatchRuntime,* the developer can access high level abstract gesture, touch and tracking events, which can easily be integrated into the interface design. Although we expect that most developers' needs reside in this high level abstraction space, we will later show how developers can make use of all lower level layers, right down to the hardware.

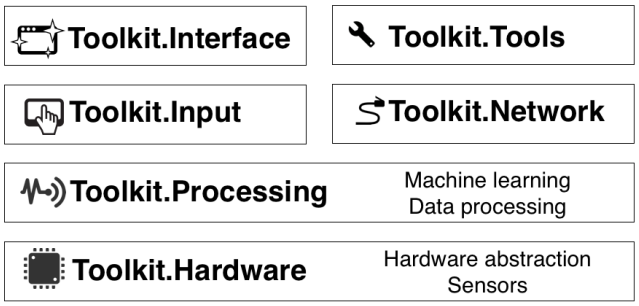

Figure 4: The architecture of the *WatchConnect* software.

Toolkit.Input

The input module provides three built-in input managers: a touch manager, gesture manager and tracker manager. First, the *TouchManager* encapsulates all "*on the device touch sensors*" - such as the *SlideTouch* device for the wristband, and a BevelTouch device - and presents the developer with high-level events including *TouchDown*, *TouchUp* and *TouchDoubleTap* events but also more complex and specialized events such as, e.g., *SliderDown*, *SliderUp* or *BevelMultiGrab*. Second, the *GestureManager* encapsulates "*above the device sensors*" – such as a light sensor and infrared proximity sensors, and tracks the internal state of the sensors using configurable thresholds and gesture detection algorithms to detect higher level gestures and postures. The abstract gesture events accessible in this manager include *SwipeLeft*, *SwipeRight*, *HoldLeft*, *HoldRight* and *Cover*. Finally, the *TrackerManager* encapsulates all the "*interaction via internal sensors*" such as the accelerometer, gyroscope and magnetometer. Similarly to the other managers, the *TrackerManager* monitors the internal state of the sensors and presents the developer with abstract high-level gestures or postures. These include an abstract IMU representation that includes the raw acceleration data, the world acceleration data, the angular motion, magnetic data and the yaw, pitch and roll. Using machine learning methods (defined in the *Toolkit.Processing* module), the manager can also detect gestures and postures that are defined by the developer who can provide training data and labels to the manager through the *WatchConfiguration*. This data can be collected using the data capture tool (provided in the *Toolkit.Tools* module). The gesture detection events provide users with the detected label as well as the probability and score of the detection. All three managers provide access to raw fused sensor data and can easily be extended by developers who can add new sensors, create new events or even add new managers (e.g., for "*on the skin sensors*"). Finally, each manager has a built-in simulator that allows developers to trigger events using simulated input such as a 3D controller or simulated data.

Toolkit.Hardware

The toolkit operates using an abstract HardwarePlatform, which can be an Arduino, Phidget or any other hardware platform that supports the *WatchConnect* protocol. The hardware module provides low level plug and play serial port management and allows managers to hook into the serial data loop to filter for specific data packets. Individual sensors are created and initiated in the managers, but use the packet definition to internally update their values. Although the toolkit supports a wide variety in sensors, there are four high-level abstract sensors: touch sensor, multi-touch sensor, proximity sensors and an IMU. These can represent a wide range of low level sensors ranging from flexible linear force resistant potentiometers to multi-channel capacitive sensors, various types of IMUs, and light and distance sensors. Although the managers provide high level events, developers can add custom lower level events directly to the sensor in order to listen or monitor changes in the internal values. Every sensor instance has an internal dynamic event mechanism that allows programmers to define events with a custom condition, which is checked and triggered from the internal value update function. This is achieved by allowing developers to inject methods into the execution body of the sensor. Finally, if new sensors are added to the setup, developers can add a new and custom hardware packet listener to the managers. This packet listener can be included in an existing manager, a newly defined manager or be used directly inside the existing watch application setup.

Toolkit.Processing

To support gesture, posture and pattern recognition, the Toolkit.Processing module provides a number of machine learning algorithms and data structures, that are built using the Accord framework and are integrated into the toolkit. The processing module includes a dynamic decision tree generator and a dynamic time warping (DTW) template engine that both use the training data and labels provided in the WatchConfiguration. The toolkit will use the training data to generate internal structures that are used by the managers to match recorded templates (e.g., for "*above the device*" proximity sensors) or monitor for gestures inside a time window.

Toolkit.Network

To allow multiple watches (connected to the same or multiple master computer devices) to interact with each other, the toolkit includes a network module that provides a websocket service that wraps the WatchRuntime and exposes all events over a real-time data connection. The module also includes *Bonjour Discovery* services to allow for zero-configuration networking support and broadcasting of watch addresses. It

also provides a number of abstractions to distribute and share descriptions of the watch applications. The network module distributes watch data, meaning that each watch renders the data locally if new data is received from other watches.

Toolkit.Tools

To support developers in debugging and using the software framework, the toolkit includes a number of tools. First, the InputVisualizer provides developers with a number of visualizations that present the raw sensor data and allow for the testing of the machine learning and pattern matching data. Second, the DataRecorder provides a visual interface to record sensor data. Developers can select the data, sample rate and file location of the captured data. The recorder also allows developers to label the data as it is being recorded.

CROSS-DEVICE INTERACTION TECHNIQUES

To demonstrate the *functionality* and test the *feasibility* and *applicability* of the toolkit for the design of cross-device applications, we present five different interaction techniques implemented in realistic applications. The implementations of all applications and techniques use only the standard toolkit components, events and machine learning of the toolkit and do not include any specialized code or external tools. Table 1 provides an overview of the applications (with lines of code), and how the applications utilize the input and output of the interaction space. All applications were designed using a drag and drop editor for all UI elements, with minimal background code to link the UI to the underlying toolkit through high level objects and events. Although all applications are demonstrated on a laptop with interactive touchscreen, these techniques and applications are also usable and suitable for tablets and large horizontal or vertical surfaces. The purpose of these example applications is to demonstrate the types of advanced applications that can be constructed using only default components of the toolkit. Although these applications can be built using other methods (as demonstrated in [7,24,40]), these include custom hardware design, machine learning and other advanced computer science skills that many interaction designers do not have.

Application 1: Data Transfer

One of the core problems in multi-device information spaces is the fast, intuitive and easy transfer of files and resources across different devices [23]. A body of previous work (e.g., [23,34,36]) has explored how information can be seamlessly transferred across devices. These techniques can be expanded to smartwatches that have the potential to become *wearable mediating storage devices* that allow users to easily move their personal information to any display or device on hand. The *touch and swipe* technique allows users to connect their smartwatch to a display and use a mid-air swipe gesture to send information to the display. Users first touch the display with the watch hand to create a connection between the two devices. After the watch hand touch is recognized and the user touches an empty space, the user interface reveals a colored rectangle that is filled up over a period of two seconds. The color represents the active information on the

		Watch			Screen			Output		
Application	Lines of Code	W1: Touch	W2: Above	W3: 3D Move	S1: Identify user	S2: Multi touch	S3: 3D Space	O1: W feedback	O2: S feedback	O3: Distr. UI
Data transfer	164	●	●		●	●		●	●	●
Privacy	50			●	●			●	●	●
Navigation	98	●	●		●			●	●	●
Reading	132	●		●	●	●			●	●
UI distribution	47				●		●		●	

Table 1. Five example applications with their lines of code and how they use the interaction space.

watch, and the filling of the rectangle visualizes the time window in which the user can perform gestures to move the resource to the display. If the time window passes, the system dismisses the watch connection and treats the touch as a normal touch input. If the user performs a left to right swipe during the time window, the resource on the watch is sent to the display and shown as a touch-enabled resource (Figure 5A-B). To select which resources to send to the display, the user can use the wristband touch sensor to scroll between the different resources stored on the watch.

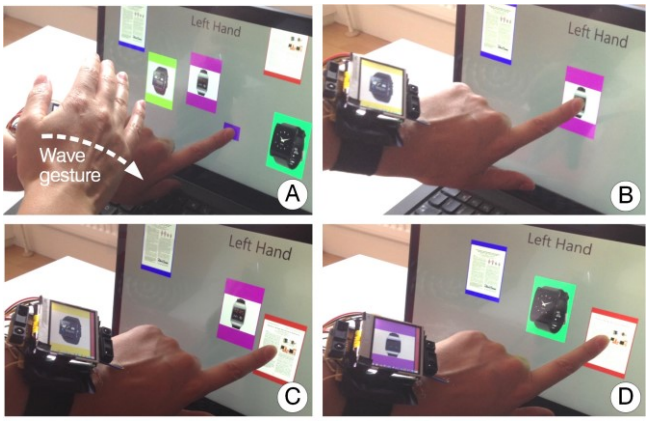

Figure 5. Users can perform gestures to move information between the display and the watch (A-B). The UI reveals part of the resource in the form of a color and shape (C-D).

If the user touches an existing resource on the display, the watch will update the UI to reveal that the watch can receive the resource, by showing a colored border on the right side of the watch (Figure 5 C-D). If the user performs a right-to-left swipe during the reveal time window, the resource is removed from the display and sent to the watch. When interacting with the resources on the surface, the UI can distinguish between left hand and right hand touches. As a consequence the UI only offers time windows to send information between devices, if a touch is linked to the watch hand.

The application leverages the entire software stack of the *WatchConnect* toolkit and was built in only 164 lines of code in a single class. It uses the built-in gesture recognizer to detect the watch hand. The UI elements are simply relocated between the watch runtime and the full screen application.

The different type of touch inputs (watch hand, non-watch hand) are channeled through events and coupled directly to the UI. The input layer on the watch automatically captures touch input on the bevel and updates the UI on the watch.

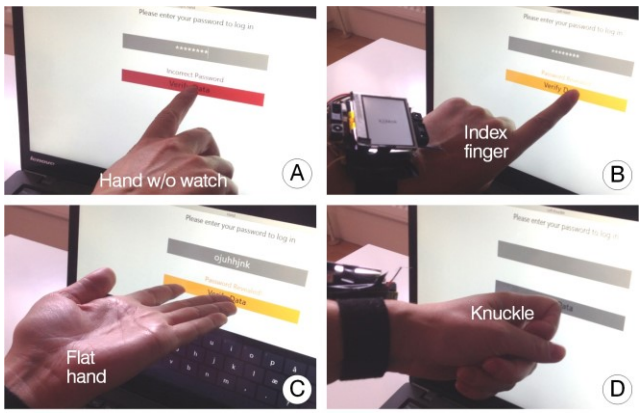

Figure 6. Users log in with the non-watch hand (A) or use the index finger or flat hand to show the password on the watch or screen (B-C). Users reset the password using the knuckle (D).

Application 2: Privacy and Password Access

The next technique facilitates access to highly private data such as passwords, bank account data or personal email. The *pose and touch* interaction technique provides users with a rich set of interactive capabilities to enter or correct a password field. Similar to the previous technique, the watch is paired to the display by touching the screen. However, in this case, the screen will monitor the posture of the hand at the moment of touching the screen. This means that the screen cannot only detect if the watch hand is touching the screen but also with which part of the hand (similarly to [7,13]). Touching the button with the non-watch hand validates the password and provides appropriate feedback (Figure 6A). The user can reveal the content of the hidden password field on the watch display, by touching the button with the index finger of the watch hand (Figure 6B), or on the touchscreen, by touching the button with the flat watch hand (Figure 6C). Users can reset the password field by touching the button with the knuckle of the watch hand (Figure 6D).

The application uses the gesture recognizer to distinguish between four different hand postures. Each posture is pushed to the UI as a different event, allowing the UI code to simply switch states and push the correct UI to the watch runtime or full screen application. This example was built in 50 lines of code and allows developers to focus only on the UI.

Application 3: Supporting Map Navigation

Interacting with maps often requires users to modify the view, find a location, or start route planning. Most maps currently provide little support for using additional devices to expand or distribute the view on the map. The *touch and push* interaction technique allows users to modify a custom secondary view on the display of the smartwatch, while using the interactive touchscreen for an overview of the general environment they want to explore. After touching the screen with the watch hand, the maps on both displays are synchronized (Figure 7A). The watch map has a default zoom level that is twice that of the main map. This allows users to quickly glance at the watch for more details as they explore the map. By touching the bevel of the watch, users can zoom in and out of the customized view, or toggle the watch map between a satellite view or the traditional map view (Figure 7C). When users interact with the map on the touchscreen, the map on the watch follows the movement, thus keeping both views synchronized. When users explore the customized view on the watch in more detail, they can synchronize the main map to that of the watch by using the *touch and swipe* gestures (Figure 7D). Finally, for selecting small targets, such as placing pushpins or route marks, the display of the watch can be used as a scope to zoom and find the exact location (Figure 7B). The user can touch the screen and press the left bevel of the watch to mark the point on the main map.

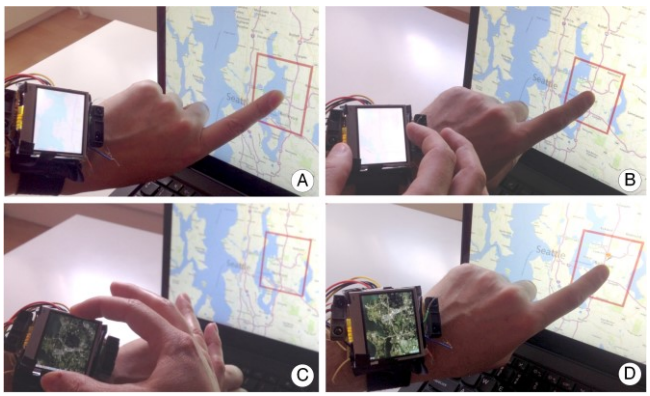

Figure 7. The watch screen shows a mini map (A), and allows users to zoom (B), change view (C) or mark locations (D).

This example was built in 98 lines of code, and utilizes the input layer to channel input from above and on the watch probe to the main interface on the surface. The application uses the temporal events to synchronize views between the watch runtime and main application, but integrates with a standard Bing maps component available in WPF / C#.

Application 4: Support Active Reading Applications

With the increasing availability of touch-enabled devices, active reading applications integrate new forms of touch-based interaction. The *gesture and touch* interaction techniques support a range of input techniques designed to create a fluid active reading application. In this application, the non-watch hand is used for *passive* browsing and reading, while the watch hand is used for *active* editing. Users can simply scroll through the text by performing on-screen swipe gestures using the non-watch hand (Figure 8A). By touching the bevel of the watch, users can browse through the menu items, thus, changing the selected option, which determines the effect of touching the screen with the watch hand (Figure 8B). The finger of the watch hand thus becomes *a reconfigurable instrument* that can be used for basic annotation with a black pen (Figure 8C), painting with a translucent brush (Figure 8D), marking text with a yellow marker (seen in Figure 8E)

or as an eraser. Users can use the knuckle of the watch hand to select and copy text to a clipboard (Figure 8F).

This example was built in 132 lines of code, and again leverages the ML and processing features of the toolkit to augment a basic e-reader with advanced gestural interactions. The recognizer of the toolkit channels the recognized labels through events to the UI, which can simply be updated. Similar to all other examples, the UI itself is designed using drag and drop WPF C# components available in the Visual Studio IDE. *WatchConnect* simply connects the gestures and sensors of the watch to the already existing UI components.

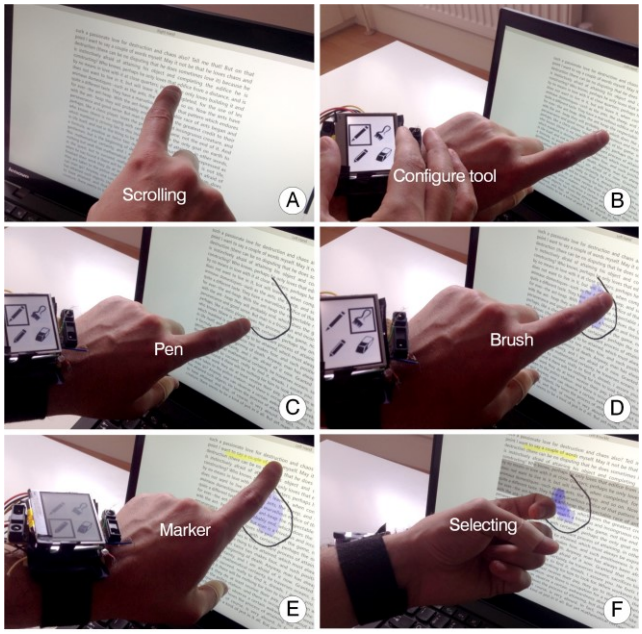

Figure 8. Users can browse text with the non-watch hand (A), configure the watch hand (B) into a pen (C), pencil (D), markers (E) or use the knuckle to select text (F).

Application 5: User Interface Beaming

One important research challenge in cross-device information spaces is how user interface elements can be seamlessly moved between different connected devices. Prior work has proposed the notion of "*user interface beaming*" [24] for mixed reality environments, or the flashlight metaphor [6] for transferring user interface elements from one device to another. Smartwatches can play a mediating role in *defining*, *exchanging* and *using* interface components or data. In this *touch and beam* technique, a UI element is initially only shown on a watch. After connecting the watch to an interactive surface by touching the display, the user interface is sent to that bigger display, to provide a bigger space for the output and utilize the potentially more advanced features provided by that device. E.g., an incoming phone call on a smart watch (Figure 9A) is simply transferred to a bigger display with better sound and camera by touching the display and connecting the watch. The hand acts like a flashlight that beams the interface on a larger canvas (Figure 9B), thus, increasing the interaction space for the user interfaces.

This example was built with 47 lines of code and uses the layout engine of the watch runtime to relay UI components based on synchronized event triggers. Designers do not need to define a multi-device context but can simply rely on the toolkit to move UI elements between the watch runtime and the main surface application.

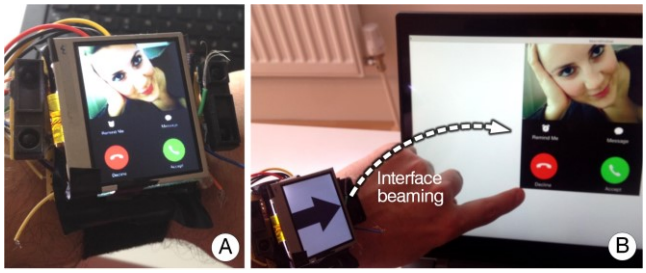

Figure 9. UI elements (A) can be beamed to the surface (B).

DISCUSSION

Designing, prototyping and testing cross-device interaction techniques with smartwatches is a complex task. To mitigate these challenges, we introduced *WatchConnect*, which uses a watch prototyping emulator to provide developers with a platform for the rapid design and prototyping of cross-device interaction techniques. Developers can create their own sensor and hardware configurations and use the software framework for easy and fast access to those hardware designs. In this section, we thematically compare *WatchConnect* to other approaches using Olsen's framework [31].

Problem Not Previously Solved

Commercial smartwatch APIs (such as [41,42,43]) provide limited support for existing hardware and single screen user interfaces. These APIs are designed to provide a *path of least resistance* towards specific UIs, but are not designed to explore novel interaction techniques and alternative designs. In contrast, *WatchConnect* allows designers to experiment, build and evaluate a range of different hardware designs, interaction techniques and gestural cross-device applications without any knowledge on distributed computing, hardware development and interfacing, data processing and sensor fusion, machine learning and networked setups. With the use of smartwatches and other mobile and wearable devices, providing tool support for designing UIs across an ecology of devices, becomes increasingly important and relevant.

Earlier cross-device UI toolkits – such as HydraScope [14], Conductor [10], or XDStudio [27] – lowered the threshold for developing applications spanning the ecology of devices. *WatchConnect* builds on top of these toolkits, and extends this work with a specialized support and focus on smartwatch specific interaction techniques, support for diverse hardware platforms, custom sensor mappings or creation of watch-centric gestural interactions. The toolkit also draws from previous work in watch-centric cross-device applications (such as Duet [7], UI Beaming [24] and SleeD [40]) to generalize these approaches and allow for rapid prototyping of complex cross-device interaction techniques using different display sizes and novel sensor input.

Reduce Solution Viscosity

Compared to other methods to develop watch-centric cross-device applications, *WatchConnect* dramatically reduces development viscosity [31] by providing a flexible architecture that allows for expressive leverage. The example applications demonstrate the range of scenarios that are supported by the toolkit. By moving all complex processes into high-level objects and events, designers can create complex gestural interactions with little overhead. However, designers with expert skills can leverage the toolkit and access, modify and create complex low level sensor mappings, custom hardware protocols and advanced machine learning approaches. Furthermore, the layered architecture allows for potential replacement of the UI layer with another existing cross-device toolkit (such as HydraScope [14], Conductor [10], or XDStudio [27]) to leverage existing multi-device features, while still using the watch-centric features of *WatchConnect*.

Empowering New Design Participants

Without adequate toolkit support, exploring watch-centric cross-device systems remained the domain of designers with highly specialized computer science skills. This is reflected in the very few watch-centric cross-device systems so far, and the number of simulation techniques used (e.g., using smartphones as watch proxies or relying on Wizard of Oz approaches). *WatchConnect* focuses in particular on making this emerging technology accessible to new, non-expert programmers. Complex applications and interaction techniques that support gestures, postures and multi-device synchronization can be designed in a short period of time without in-depth knowledge of distributed computing or machine learning. The toolkit lowers the threshold [26] for beginning the exploration of cross-device smartwatch applications, and it allows designers to focus their efforts on creative design solutions [9] for the actual cross-device user experience and interface design. In particular, the software abstracts sensor input from *above*, *on* and *in* the watch into abstract high-level events and objects for easier configuration and use. While an in-depth study of developers applying the toolkit in practice is part of our future work, we see *WatchConnect* as a fundamental step towards rapid iterative smartwatch prototyping, facilitating the exploration of novel cross-device behaviors.

Power in Combination

WatchConnect combines distributed UIs, machine learning, hardware management, data processing and simulation into one toolkit. Each building block is highly decoupled, allowing for the design of new layers or the inclusion of other approaches. The basic building blocks of *WatchConnect* can be used in complex temporal and spatial sequences that provide a power by combination [31]. *WatchConnect* integrates with C# / WPF to support a major and stable development platform that provides designer-level abstractions and a flexible and broad UI framework with numerous tools and libraries [9]. Once the design of the interaction technique or application transcends the prototyping phase and is verified and validated, designers can move the design to more permanent platforms, using standard SDKs and commercial hardware.

Generality

Using five example applications that include both novel and replications of state of the art interaction techniques, we demonstrate the versatility and generality of the toolkit, as well as the expressivity of the building block components [31]. The fundamental limitation but also strength of this toolkit is that it is based around a watch prototyping emulator and not a real – and wireless – watch. We argue that for the rapid prototyping and creative stage of the design process this is an acceptable trade-off. It brings the advantage that developers are not bound by existing hardware limitations or existing device designs, but can design and use their own setup to develop compelling and forward looking cross-device interaction techniques. We expect that in the near future more accessible smartwatch hardware platforms will emerge and future versions of the toolkit could support some of these smartwatches for prototyping. Furthermore, an in-depth testing of the toolkit and its expressive power with developers can provide us further insights into the prototyping process with smartwatch cross-device applications.

CONCLUSION

WatchConnect allows for rapid prototyping of smartwatch-centric cross-device applications and interaction techniques, using custom hardware designs and a software framework that removes complex machine learning, sensor fusion and hardware management into high level objects and events that are integrated with an existing drag and drop UI framework. The toolkit reduces development complexity and lowers the threshold for developers to design for complex device ecologies using a smartwatch as mediating instrument. The toolkit allows for future explorations of a wide range of novel multi-device interaction techniques, hardware designs and collaborative multi-surface environments. Future work includes integrating *WatchConnect* with other cross-device toolkits and smartwatch platforms to support existing frameworks and a wider set of setups and development platforms.

ACKNOWLEDGMENTS

This work was supported by the EU Marie Curie Network iCareNet under grant number 264738 and ICRI Cities. Thanks to Michael Nebeling, Sarah Gallacher and our anonymous reviewers for their feedback and helpful suggestions for the improvement of the manuscript.

REFERENCES

1. Ashbrook, D., Lyons, K., and Starner, T. An investigation into round touchscreen wristwatch interaction. *Proc. of ACM MobileHCI'08*.
2. Badam, S.K. and Elmqvist, N. PolyChrome: A Cross-Device Framework for Collaborative Web Visualization. *Proc. of ACM ITS'14*.
3. Baudisch, P. and Chu, G. Back-of-device interaction allows creating very small touch devices. *Proc. of ACM CHI'09*.
4. Blasko, G. and Feiner, S. An interaction system for watch computers using tactile guidance and bidirectional segmented strokes. *Proc. of IEEE ISWC 2004*.

5. Butler, A., Izadi, S., and Hodges, S. SideSight: multi-touch interaction around small devices. *Proc. of ACM UIST'08*.
6. Cao, X. and Balakrishnan, R. Interacting with dynamically defined information spaces using a handheld projector and a pen. *Proc. of ACM UIST'09*.
7. Chen, X., Grossman, T., Wigdor, D.J., and Fitzmaurice, G. Duet: exploring joint interactions on a smart phone and a smart watch. *Proc. of ACM CHI'14*.
8. Funk, M., Sahami, A., Henze, N., and Schmidt, A. Using a touch-sensitive wristband for text entry on smart watches. *ACM CHI'14 EA*.
9. Greenberg, S. Toolkits and interface creativity. *Multimedia Tools and Applications*, (2007).
10. Hamilton, P. and Wigdor, D.J. Conductor: enabling and understanding cross-device interaction. *Proc. of ACM CHI'14*.
11. Hardy, R. and Rukzio, E. Touch & interact: touch-based interaction of mobile phones with displays. *Proc. of ACM MobileHCI'08*.
12. Harrison, C. and Hudson, S.E. Abracadabra: wireless, high-precision, and unpowered finger input for very small mobile devices. *Proc. of ACM UIST'09*.
13. Harrison, C., Schwarz, J., and Hudson, S.E. TapSense: enhancing finger interaction on touch surfaces. *Proc. of ACM UIST'11*.
14. Hartmann, B., Beaudouin-Lafon, M., and Mackay, W.E. HydraScope: creating multi-surface meta-applications through view synchronization and input multiplexing. *Proc. of ACM PerDis'13*.
15. Heikkinen, T., Goncalves, J., Kostakos, V., Elhart, I., and Ojala, T. Tandem Browsing Toolkit: Distributed Multi-Display Interfaces with Web Technologies. *Proc. of ACM PerDis'14*.
16. Hinckley, K., Ramos, G., Guimbretiere, F., Baudisch, P., and Smith, M. Stitching: pen gestures that span multiple displays. *Proc. of ACM AVI'04*.
17. Hinckley, K. Synchronous gestures for multiple persons and computers. *Proc. of ACM UIST'03*.
18. Huang, D.-Y., Tsai, M.-C., Tung, Y.-C., et al. TouchSense: expanding touchscreen input vocabulary using different areas of users' finger pads. *Proc. of ACM CHI'14*.
19. Kim, J., He, J., Lyons, K., and Starner, T. The gesture watch: A wireless contact-free gesture based wrist interface. *Proc. of IEEE ISWC'07,*.
20. Knibbe, J., Martinez Plasencia, D., Bainbridge, C., et al. Extending interaction for smart watches: enabling bimanual around device control. *ACM CHI'14 EA*.
21. Laput, G., Xiao, R., Chen, X., Hudson, S.E., and Harrison, C. Skin buttons: cheap, small, low-powered and clickable fixed-icon laser projectors. *Proc. of ACM UIST'14*.
22. Lyons, K., Nguyen, D., Ashbrook, D., and White, S. Facet: a multi-segment wrist worn system. *Proc. of ACM UIST'12*.
23. Marquardt, N., Ballendat, T., Boring, S., Greenberg, S., and Hinckley, K. Gradual engagement: facilitating information exchange between digital devices as a function of proximity. *Proc. of ACM ITS'12*.
24. Mayer, S. and Sörös, G. User Interface Beaming - Seamless Interaction with Smart Things using Personal Wearable Computers". *Proc. of IEEE BSN 2014*.
25. Merrill, D., Kalanithi, J., and Maes, P. Siftables: towards sensor network user interfaces. *Proc. of ACM TEI'07*.
26. Myers, B., Hudson, S.E., and Pausch, R. Past, present, and future of user interface software tools. *TOCHI '00*.
27. Nebeling, M., Mintsi, T., Husmann, M., and Norrie, M. Interactive development of cross-device user interfaces. *Proc. of ACM CHI'14*.
28. Nebeling, M., Teunissen, E., Husmann, M., and Norrie, M.C. XDKinect: development framework for cross-device interaction using kinect. *Proc. of ACM EICS'14*.
29. Oakley, I. and Lee, D. Interaction on the edge: offset sensing for small devices. *Proc. of ACM CHI'14*.
30. Olberding, S., Yeo, K.P., Nanayakkara, S., and Steimle, J. AugmentedForearm: exploring the design space of a display-enhanced forearm. *Proc. of ACM AH'13*.
31. Olsen Jr, D.R. Evaluating user interface systems research. *Proc. of ACM UIST'07*.
32. Pasquero, J., Stobbe, S.J., and Stonehouse, N. A haptic wristwatch for eyes-free interactions. *Proc. of ACM CHI'11*.
33. Perrault, S.T., Lecolinet, E., Eagan, J., and Guiard, Y. Watchit: simple gestures and eyes-free interaction for wristwatches and bracelets. *Proc. of ACM CHI'13*.
34. Rekimoto, J. Pick-and-drop: a direct manipulation technique for multiple computer environments. *Proc. of ACM UIST'97*.
35. Rekimoto, J. Gesturewrist and gesturepad: Unobtrusive wearable interaction devices. *Proc. of IEEE ISWC'01*.
36. Schmidt, D., Chehimi, F., Rukzio, E., and Gellersen, H. PhoneTouch: a technique for direct phone interaction on surfaces. *Proc. of ACM UIST'10*.
37. Schmidt, D., Seifert, J., Rukzio, E., and Gellersen, H. A cross-device interaction style for mobiles and surfaces. *Proc. of ACM DIS'12*.
38. Xiao, R., Laput, G., and Harrison, C. Expanding the input expressivity of smartwatches with mechanical pan, twist, tilt and click. *Proc. of ACM CHI'14*.
39. Yang, J. and Wigdor, D. Panelrama: enabling easy specification of cross-device web applications. *Proc. of ACM CHI'14*.
40. Von Zadow, U., Büschel, W., Langner, R., and Dachselt, Raimund. SleeD: Using a Sleeve Display to Interact with Touch-sensitive Display Walls. *Proc. of ACM ITS'14*.
41. Pebble. http://developer.getpebble.com.
42. Sony Watch. http://developer.sony.com/.
43. Apple Watch. http://apple.com/watch/.
44. Phidgets. http://phidgets.com.
45. Arduino. http://arduino.cc.