Connichiwa – A Framework for Cross-Device Web Applications

Mario Schreiner
HCI Group
University of Konstanz
mario.schreiner@uni-konstanz.de

Harald Reiterer
HCI Group
University of Konstanz
harald.reiterer@uni-konstanz.de

Roman Rädle
HCI Group
University of Konstanz
roman.raedle@uni-konstanz.de

Hans-Christian Jetter
Intel ICRI Cities
University College London
h.jetter@ucl.ac.uk

CHI'15 Extended Abstracts, Apr 18-23, 2015, Seoul, Republic of Korea
ACM 978-1-4503-3146-3/15/04.
http://dx.doi.org/10.1145/2702613.2732909

Abstract
While Mark Weiser's vision of ubiquitous computing is getting closer to reality, a fundamental part of it - the interconnection of devices into a "ubiquitous network" - is not achieved yet. Differences in hardware, architecture, and missing standardizations are just some reasons for this. We think that existing research is not versatile enough and too tailored to either single applications, hardware, or location. We contribute Connichiwa – a versatile framework for creating web applications across multiple devices. We base Connichiwa on four key goals: integration of existing devices, independence of network infrastructure, versatility of application scenario, and usability of its API. Connichiwa runs web applications on off-the-shelf consumer devices. With no external dependencies, such as a server, it enables a great variety of possible scenarios. We tested the technical feasibility of Connichiwa in seven example applications and plan to evaluate the framework and the usability of its API in a one-week Hackathon.

Author Keywords
ubiquitous networks; framework; web; cross-device

ACM Classification Keywords
D.3.3. [Programming Languages] Language Constructs and Features – Frameworks.

Figure 1: Example application. A video player distributing the video across multiple devices and playing it back synchronously.

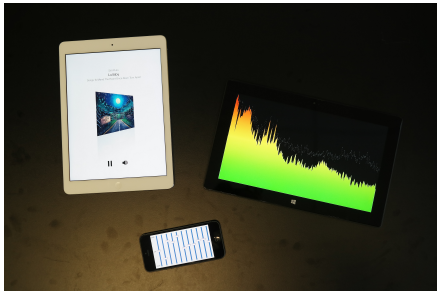

Figure 2: Example application. A music player with music metadata on the left device, a synchronized visualization on the right device and an equalizer on the smartphone. Sound output can be redirected to any device.

Motivation

With more and more devices of different sizes, shapes and capabilities, reality gets closer to Mark Weiser's vision of ubiquitous computing [13]. But while modern devices are interconnected in many ways, Weiser's envisioned "*ubiquitous networks*" [13] are still far from becoming a reality. Devices have only limited awareness of each other's presence and still fail in working together to enable cross-device interactions.

As work-in-progress, we contribute Connichiwa (jap. こんにちは (konnichi wa), good day + engl. connect), a framework for creating web applications across multiple devices. Applications work on off-the-shelf consumer devices. Connichiwa runs *local web applications* on one of the participating devices, without requiring an existing network or an internet connection. Connichiwa does not require an instrumented environment or external hardware, and is therefore a versatile tool for developers. We showcase Connichiwa with seven example applications, such as a distributed video player (Figure 1) or a cross-device music application (Figure 2). We also tested applications outdoor over an ad hoc network. Connichiwa's JavaScript API is designed to give easy access to common functions like device detection and connection, but at the same time is flexible enough to allow even complex applications.

In recent years, enabling interactions across devices is a hot topic in research and commercial products alike. One trend is to move features "into the cloud", which means that user data is stored online, accessible by internet-enabled devices. Examples for such applications are Dropbox, Google Docs or iTunes Match. But while cloud services "*enable between device information access, [they] fail to take advantage of another growing trend: the development of experiences which cross devices*" [4]. In 2014, Apple introduced Continuity, allowing users to move their current application state (such as an open document) instantly between devices. But again, this does not allow for simultaneous use of devices, sharing of resources (such as display space or computational power) or cross-device interactions.

To tackle the latter, research has looked into different sensing technologies to enable (fluent) cross-device interaction. For instance, by employing computer vision [1,2,11,12] or developing custom sensing hardware [5,6,8]. In the examined research, communication between devices is most commonly realized through a shared dedicated remote server [1,2,4,6,7,11,12,14]. Recently, short-range wireless technologies (NFC and Bluetooth) have emerged and seem a promising candidate for ad-hoc device communication [10]. A common issue with most solutions, however, is that they require extensive preparation of devices or the environment. Further, they are rarely designed to be versatile and mostly tailored to a single application, hardware, or location.

We believe a versatile framework for cross-device interaction can push ubiquitous computing to the next level. It could also lead to proliferation of cross-device interaction when giving researchers the ability to implement cross-device applications without the need to solve aforementioned technical difficulties.

Key Goals

We identified four key goals that are essential for a tool such as Connichiwa:

Integration of existing devices
A wide variety of consumer devices are part of everyday life. Connichiwa aims to support these devices without the need to augment them with additional hardware, markers or tags. By this, we lower the threshold for both developers implementing cross-device applications and end-users employing these applications.

Independence of network infrastructure
Cross-device interaction is most commonly realized using a remote server to communicate among devices. This requires set up and maintenance of such a server. Further, devices have to either reside in the same local network as the server or the server is reached over the internet, requiring a permanent internet connection, being subject to delay and bandwidth limitations and posing a security issue for sensitive data. A direct connection that does not require external hardware broadens the scope and design space of applications (e.g., for the use in outdoor scenarios).

Versatility of application scenario
Connichiwa wants to enable novel cross-device applications. Devices should be restricted as little as possible in regard to their number, size, location or relative position. For example, we do not want to restrict usage of Connichiwa to a room that is particularly instrumented. Instead we want to support a great variety of scenarios, for example usage across multiple rooms or outdoor use.

Usability of API
The importance of a well designed API cannot be overestimated as "*this can have a tremendous impact on the final product as well as the efficiency of the development process*" [3]. Connichiwa wants to hide implementation details and provide high-level functions to quickly achieve common tasks ("low threshold" [9]) and at the same time be flexible enough to implement complex applications ("high ceiling" [9]).

System Design
In order to achieve the four key goals, a number of system design decisions were made:

Local Web Applications
An early decision was to base the framework on web technologies. Web applications are designed to run on many devices, adapt to different screen sizes, resolutions, and input modalities and work on desktop and mobile devices. Modern web technologies (HTML5, CSS3) further have a large standardization and acceptance across consumer devices [15].

To achieve network independence, Connichiwa runs *local web applications*. A native helper application automatically runs a webserver on-demand on one of the joined devices. Other devices can then access the webserver through a shared network (such as an existing Wi-Fi network) or an ad hoc network. This eliminates the need for a remote server, keeps communication local and the communication delay to a minimum.

Detection & Connection
The native helper application is further responsible for automatic detection of and connection to other devices. This is achieved using Bluetooth Low Energy. Bluetooth pings are sent out and picked up to detect nearby devices. To connect another device, the IP address of the local webserver is sent over Bluetooth, which

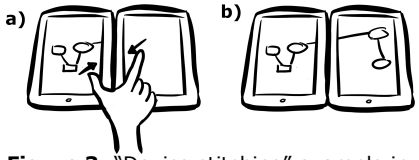

Figure 3: "Device stitching" example in Connichiwa. (a) Relative device positions are determined with a synchronous gesture. (b) The application uses this information to show content across both devices.

```
CW.onDetect = function(device) {
  device.connect();
}

CW.onConnect = functon(device) {
  device.show(moreInfoPanel);
}

CW.onMove = function(device) {
  if (device.distance < 1.0) {
    device.replace(
      moreInfoPanel,
      expandedInfoPanel
    );
  }
}
```

Listing 1: Example code of detection, connection and usage of a remote device with Connichiwa. Detected devices are instantly connected, each connected device shows a UI panel. Devices closer than 1 meter expand their UI panel.

enables the other device to access the received IP in a web view. Note that devices can also join manually through any standard web browser.

JavaScript API

The web application is notified about device detection through JavaScript events (Listing 1, `CW.onDetect`). It can request a connection and is informed when it was successfully established (Listing 1, `CW.onConnect`). The technical details - such as the Bluetooth communication or the kind of network - are completely hidden from the developer. Connichiwa automatically established a websocket connection and offers JavaScript functions to exchange information or send and respond to remote events. Developers can easily manipulate the remote devices' Document Object Model (DOM) to show, change, or hide content. Connichiwa approximates the distance between devices using the Bluetooth signal strength and reports distance changes to the application (Listing 1, `CW.onMove`).

Current State

Connichiwa is work-in-progress and currently in active development. A usable version can be downloaded at our website[1] along with installation instructions and first steps. Based on the current version, we implemented seven example applications to test our key goals and run a technical feasibility study.

Integration of existing devices

Connichiwa is currently able to run local web applications on iOS devices. Fully automated detection of and connection to other devices currently works between iOS and Mac OS X devices running the Connichiwa application. Porting of Connichiwa to other platforms in the future can enable additional operating systems. A large variety of devices can already join through any standard web browser. In an example application that distributes a high-resolution image amongst different devices, we joined iOS devices, a Microsoft Surface Pro 2 running Windows 8 and different Android devices (Figure 4). The application has also been tested on Mac OS X and Linux computers.

To determine each device's segment of the image, Connichiwa supports "device stitching". A synchronous gesture on two devices determines their relative positions (Figure 3) and adjusts display content accordingly. Currently, Connichiwa supports pinching gestures [10], but other synchronous gestures can be implemented in the future. Connichiwa also compensates for differences in pixel density and device rotation. The framework supports manual unstitching by the application or automatic unstitching when a device is moved, using the device's integrated accelerometer.

Versatility of application scenario

Number of devices, arrangement and orientation are not predefined in Connichiwa. Aforementioned image stitching application has been tested with up to eleven devices and with arbitrary device arrangements and different device orientations. Hardware and size do not matter – the application has been run on a 4" iPhone as well as a 55" Microsoft Perceptive Pixel without problems. Example applications have been implemented for mobile touch devices and desktop systems, and have also been tested on an Android-based photo camera (Samsung Galaxy NX) or TVs and

[1] Connichiwa on GitHub – http://www.connichiwa.info

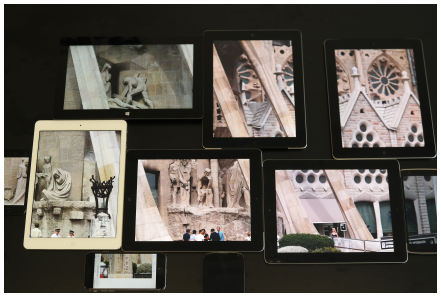

Figure 4: Example application. A high-resolution image is distributed across a large number of devices. Panning gestures are synchronized across all devices.

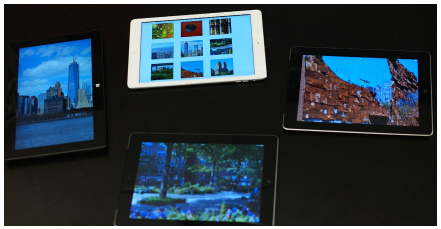

Figure 5: Example application. A photo viewer that shows a photo library on one device, and selected photos are pushed to remote devices.

projectors with attached computers. Devices are further not restricted to a location and do not rely on external hardware. They can lie on a table, hang on a wall or be carried around by people. For example, we implemented a photo viewer where one device displays a photo library and selected photos are pushed to the personal devices of other users (Figure 5).

Independence of network infrastructure

Connichiwa is able to run over an existing Wi-Fi network, ad hoc Wi-Fi network, or Bluetooth Personal Area Network. The Connichiwa application automatically determines the best connection. We have tested outdoor usage with an example document viewer (Figure 6). The application displays a document and parts of the document can be highlighted. The highlights are synchronized across joined devices. This application has been tested in a park using a manually created ad hoc Wi-Fi and a Bluetooth network. Automated creation and joining of ad hoc networks is currently being implemented. API limitations of iOS currently restrict automated networks to iOS and Mac OS X devices, but this limitation might be lifted in the future.

Usability of API

Connichiwa's JavaScript API currently allows web developers to be notified of nearby, connected and lost devices. It further approximates device distance based on the Bluetooth signal strength, which has been tested in an example application where each device shows its distance to each other nearby device. Developers can connect devices, push content to connected devices and execute custom JavaScript on them. Messages can be sent and event handlers can be installed to respond to messages. As seen in Listing 1, common functions like device detection, connection and pushing content are currently handled with a few lines of code. The example applications show that the API is flexible enough to support a variety of applications.

Limitations & Future Work

Current Technical Limitations

Currently, automated detection, connection, and automated ad hoc networks only work between iOS and Mac OS X devices. Porting the Connichiwa application to additional platforms can enable new operating systems. Further, there are technical limits on memory usage and speed of the local webserver created by iOS, in particular limiting certain applications that rely on large assets such as videos. With a different web server implementation, these restrictions can hopefully be lifted in the future.

API Evaluation

After the technical implementation and the remaining API functions are implemented, we will conduct a study to evaluate Connichiwa's API usability. The study will be designed as a one-week Hackathon. During the Hackathon, we will give the framework to a number of developers, give them an introduction into Connichiwa and its API and then ask them to design and implement either an application of their choice or one of several provided examples. Developers will be interviewed and asked to take down any questions, problems, missing functionality and other feedback regularly. Based on those notes and the created applications, we hope to uncover problems with the API or the framework. The evaluation will also reveal how developers adopt to the possibilities enabled by Connichiwa. The framework and API will then be reworked based on our findings.

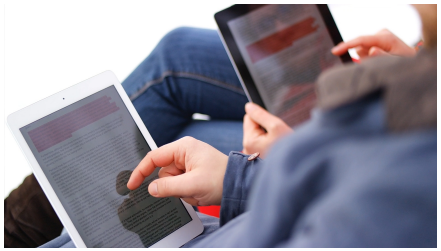

Figure 6: Example application in an outdoor scenario. A document viewer where paragraphs can be highlighted and highlights are synchronized to other devices.

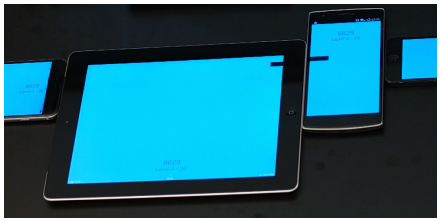

Figure 7: Example application. A snake-like game where the snake will run towards a device edge and players have to add new devices to expand the playfield. If the snake runs into the edge of the playfield the game is lost.

Conclusion

Connichiwa is a versatile tool that lowers the threshold for developers to create cross-device applications. To achieve this, we defined four key goals: integration of existing devices, versatility of application scenario, independence of network infrastructure and usability of the API. We implemented seven example applications to test the first three goals. Connichiwa currently supports modern off-the-shelf consumer devices and applications can run over existing or ad hoc networks. Devices are further not limited by location, number, size, or technical capabilities. No external hardware is required, enabling applications to run anywhere. Some technical limitations remain that we hope to lift in the future. We further plan to study the usability of Connichiwa's API in a one-week Hackathon. The API will be reworked based on the developer's feedback.

References

[1] Ballendat, T., Marquardt, N., and Greenberg, S. Proxemic Interaction: Designing for a Proximity and Orientation-aware Environment. *In Proc. of ITS '10*, ACM (2010), 121–130.

[2] Dearman, D., Guy, R., and Truong, K. Determining the Orientation of Proximate Mobile Devices Using Their Back Facing Camera. *In Proc. of CHI '12*, ACM (2012), 2231–2234.

[3] Gerken, J., Jetter, H.-C., Zöllner, M., Mader, M., and Reiterer, H. The Concept Maps Method As a Tool to Evaluate the Usability of APIs. *In Proc. of CHI '11*, ACM (2011), 3373–3382.

[4] Hamilton, P. and Wigdor, D.J. Conductor: Enabling and Understanding Cross-device Interaction. *In Proc. of CHI '14*, ACM (2014), 2773–2782.

[5] Holmquist, L.E., Mattern, F., Schiele, B., Alahuhta, P., Beigl, M., and Gellersen, H.-W. Smart-Its Friends: A Technique for Users to Easily Establish Connections Between Smart Artefacts. *In Proc. of ICUC '01*, Springer-Verlag (2001), 116–122.

[6] Huang, D.-Y., Lin, C.-P., Hung, Y.-P., et al. MagMobile: Enhancing Social Interactions with Rapid View-stitching Games of Mobile Devices. *In Proc. of MUM '12*, ACM (2012), 61:1–61:4.

[7] Maekawa, T., Uemukai, T., Hara, T., and Nishio, S. A Java-Based Information Browsing System in a Remote Display Environment. *In Proc. of CEC '04*, IEEE Computer Society (2004), 342–346.

[8] Merrill, D., Kalanithi, J., and Maes, P. Siftables: Towards Sensor Network User Interfaces. *In Proc. of TEI '07*, ACM (2007), 75–78.

[9] Myers, B., Hudson, S.E., and Pausch, R. Past, Present, and Future of User Interface Software Tools. *ACM Trans. Comput.-Hum. Interact. 7*, 1 (2000), 3–28.

[10] Ohta, T. and Tanaka, J. Pinch: An Interface That Relates Applications on Multiple Touch-screen by 'Pinching' Gesture. *In Proc. of ACE '12*, Springer-Verlag (2012), 320–335.

[11] Rädle, R., Jetter, H.-C., Marquardt, N., Reiterer, H., and Rogers, Y. HuddleLamp: Spatially-Aware Mobile Displays for Ad-hoc Around-the-Table Collaboration. *In Proc of ITS '14*, ACM (2014), 45–54.

[12] Schwarz, J., Klionsky, D., Harrison, C., Dietz, P., and Wilson, A. Phone As a Pixel: Enabling Ad-hoc, Large-scale Displays Using Mobile Devices. *In Proc. of CHI '12*, ACM (2012), 2235–2238.

[13] Weiser, M. The Computer for the 21st Century. *SIGMOBILE Mob. Comput. Commun. Rev. 3*, 3 (1999), 3–11.

[14] Yang, J. and Wigdor, D. Panelrama: Enabling Easy Specification of Cross-device Web Applications. *In Proc. of CHI '14*, ACM (2014), 2783–2792.

[15] Can I use... Compatibility tables for HTML5, CSS3, SVG and more. 2014. http://caniuse.com/#agents=desktop,ios_saf,op_mini,android,and_chr,ie_mob&show_conc=1 .