Deployable Cross-Device Experiences: Proposing Additional Web Standards

Mario Schreiner
University of Konstanz
78457, Konstanz, Germany
mario.schreiner@uni.kn

Harald Reiterer
University of Konstanz
78457, Konstanz, Germany
harald.reiterer@uni.kn

Roman Rädle
Microsoft Research
Cambridge, UK
t-rorad@microsoft.com

Kenton O'Hara
Microsoft Research
Cambridge, UK
keohar@microsoft.com

Abstract
Cross-device interaction is rarely observed in everyday life and outside of research facilities. In this position paper we explore potential reasons for this shortcoming and discuss why the web is a promising enabling technology for cross-device interactions. We propose a concept for new, cross-device centric web standards that would allow to develop, deploy, and use cross-device applications in everyday life.

Author Keywords
cross-device; api; ad hoc; web; standards; position paper

ACM Classification Keywords
D.3.0 [Programming Languages]: General — Standards

Problem Statement
Interaction across multiple devices is an everyday activity and support for moving information and working between devices is wanted by their owners [6]. Yet, not many real-world applications exist that satisfy this need. Features such as Apple's Continuity allow users to transition a task from one device to another, but such technologies focus on sequential use of devices rather than using them in parallel on the same task and interacting across devices [6]. Even more, a technology standard to find, connect, and interact across nearby devices does not exist, which can be discouraging for developers when designing and imple-

menting cross-device applications. Based on our experiences in developing cross-device technologies [9, 10], we defined four key requirements for future cross-device technologies, which we believe will be essential to enable and deploy cross-device interactions in everyday life.

Low threshold to join: While device augmentation is often utilised to make devices aware of each other [4, 5, 8], the need for special hardware hinders ad hoc cross-device interactions. Software-side setups [3, 7] face similar problems and increase the threshold for joining a cross-device application. We propose the use of off-the-shelf consumer devices to enable everyone to participate in cross-device interaction. Also we believe cross-device technology with minimal to no configuration effort can leap cross-device interactions beyond the scope of research projects.

Independent of location: Augmenting the environment [1, 9, 11] enables accurate tracking of device locations, but confines interaction to a small space (e.g., a table or a room). Such augmentation seems unfeasible for real-world deployment and an approach that enables interaction anywhere and at any time should be sought. Understanding appropriate groupings of devices in cross device interactions is also not something that is bound up entirely in physical characterisations of proximity. How these physical properties map onto social characterisations of proximity is something that also needs consideration in the definition of cross-device interaction. One might be physically close to another person (e.g. on a bus) but not socially proximate. Augmenting physical proximity indicators with social indicators (e.g. presence in contact lists may better support the establishment of appropriate device-to-device connections).

Fluidity in device configurations: Users must rely on whatever devices are currently available in their surrounding. They must be able to join an activity and add or remove devices at any time during a task. This requires cross-device technologies to cope with and support fluid changes of device configurations. There also needs to be good intelligibility of device availability to facilitate socially appropriate configuration of device assemblies.

Small development effort: Increased development effort in porting an application to all major operating platforms can lead to developers not adapting a new technology, which in turn will make it difficult to establish cross-device applications for everyday use. Developers must be able to build cross-device experiences for users and adapt to the multitude of platforms without heavy migration efforts. APIs like Microsoft UWP, Mono, PhoneGap, or Cordova are already targeting this issue and translate a shared code base into devices' respective native language. However, this still requires deployment of native apps on the devices.

Based on our experience [9, 10], we believe that web technologies can fulfil these requirements in the future. In this position paper, we want to discuss the benefits and shortcomings of state-of-the-art web technologies in regard to cross-device interaction. We further present a concept for how current web standards could be extended to establish them for real-world cross-device application development.

Cross-Device Web APIs

The modern web stands out with its massive availability and large standardisation across consumer devices [2]. At first glance, this seems to make the web an ideal choice for cross-device development with web applications running on mobile and desktop systems, TVs, gaming consoles, digital cameras, watches, across different hardware, screen sizes, and input modalities. Most modern device have a built-in web browser, making software installation obsolete. Development effort is minimised due to standard-

```
window.ondevicenearby =
function(device)
{
  if
  (
    device.clientWidth >= 1920
    &&
    device.clientHeight >= 1080
    &&
    device.memory.total >= 2048
    &&
    device.supports("camera")
  )
  {
    device.connect();
  }
};

window.ondeviceleave =
function(device) {
   //do cleanup here
};
```

Listing 1: Example code of enter and exit events of nearby devices. `.connect()` will request a peer-to-peer connection.

```
window.ondeviceconnect =
function(device)
{
  device
   .getElementById("content")
   .innerHTML = "Hello!";
};
```

Listing 2: Device objects allow to perform actions on a device such as remotely updating content.

ised markup, styling, and scripting languages (HTML, CSS, JavaScript). User interface presentation and interaction consistent across different devices, making web development an appealing underlying technology for many companies and researchers. Notably, many applications formerly developed for particular platforms are transitioning to the web (e.g. Microsoft Office, Spotify, Mendeley, Apple Mail).

Despite these advantages of web technologies, combining devices remains a technical challenge. Web communication is based on central, remote servers, and information about devices in close physical proximity is not available, which would be necessary for nearby devices to join ad hoc in the formation of co-located communities of devices. We therefore propose a JavaScript API to access local communication technologies such as Bluetooth, NFC, and Wi-Fi Direct. This would allow web applications to detect nearby devices and establish a local peer-to-peer communication channel (see Listing 1). By using already available technologies, augmentation of devices is not necessary and the approach is independent of the devices' current location.

With devices connected in such a manner, the available device information should be extended to allow applications to adapt to different device configurations. As of now, JavaScript can access only limited information about devices, such as resolution or operating system. Other information such as physical screen size, input capabilities, available memory, or attached hardware are not discoverable through an API. This, however, will be necessary to determine the role of devices in a task [12]. For example, a presentation software needs to find nearby large screens for displaying the presentation, and handheld devices for displaying controls. Attendees could get annotation abilities on stylus-enabled personal devices. Therefore, browsers should make device information accessible to web applications. Of course, owners of devices should be able to opt out from broadcasting such information.

In addition, web languages should be extended for multi-device support. We propose enter and exit events for devices in close, physical and social, proximity (see Listing 1). Complexity should be reduced by encapsulating devices in JavaScript *device objects*. A device object represents a `window` object of a close device and is passed as a parameter to the enter and exit events. This, for instance, would allow developers to gather information about remote devices (see Listing 1) or to execute commands on remote objects (see Listing 2). Internally, the device object would handle serialising, transmitting, parsing and executing commands, completely transparent to the developer. Due to the asynchronous nature of network communication, such function calls require callbacks or JavaScript *Promises*. For instance, `device.getElementById()` would not instantly return a DOM element but rather a Promise that is resolved when the element was received on the caller's end. Multi-device concepts can also be applied to CSS through new media queries (see Listing 3), allowing developers to adjust styles based on the number or features of remote devices.

In a prototype framework [10], we showed the feasibility of these concepts. During a small scale user study, we handed the framework to developers, including data synchronisation, reactive templating, and position detection via cross-device gestures. Developers adopted quickly to the framework and responded positively to the new cross-device interaction possibilities.

Limitations

In order for the given concept to be feasible for real-world deployment, this standard must be proposed to the W3C, accepted, and implemented by all major browser vendors.

```
@media
  (device-role: main)
  and
  (min-number-of-devices: 2)
{
  #main-content
  {
    display: block;
  }
}

@media
  (device-role: sidebar)
  and
  (min-number-of-devices: 2)
{
  #sidebar {
    display: block;
  }
}

@media
  (any-device-role: video)
{
  #button-airplay {
    visibility: visible;
  }
}
```

Listing 3: Example of multi-device CSS support. Features such as the total number of devices or the device role - set via JavaScript - are available to adjust styles.

Still, there are some limitations to the proposed concept:

Communication over the Internet: A stable Internet connection to retrieve assets from and store data on a remote server is still required. Technologies such as the HTML5 ServiceWorker API or the HTML5 IndexedDB API could be powerful and decentralised alternatives to remote servers.

Performance of local communication: Local communication, e.g. over Bluetooth tends to be slower than Wi-Fi or GSM. Eventually, such communication might not satisfy the performance needs of applications. A solution could be communication over a shared Wi-Fi network or offering developers the ability to take control over the communication using an on-device or remote WebSocket server.

Security: Security is of large concern on the web, and some features proposed must be implemented with care to ensure the protection of private data and user data.

Conclusion

We believe web technologies could provide a new standard for cross-device development. Their availability, standardisation, and support by the majority of devices and operating systems is appealing to application developers and users. In order to enable interaction with nearby devices in everyday life, methods for detection of and communication with devices in their vicinity are needed. The web languages must be extended for APIs that allow developers to easily interact with other devices, e.g. exchanging data and remotely updating content. If the web standard is extended in such a way, we believe cross-device experiences will find their way out of the labs into our everyday life.

References

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

[2] caniuse.com. 2015. Can I use - Web Feature Availability Overview. http://caniuse.com//#feature_sort=usr_score. (2015). Online - Accessed: 19th August 2015.

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

[4] L. E. Holmquist, F. Mattern, B. Schiele, P. Alahuhta, M. Beigl, and H.-W. Gellersen. 2001. Smart-Its Friends: A Technique for Users to Easily Establish Connections Between Smart Artefacts *(UbiComp '01)*. Springer.

[5] D.-Y. Huang, C.-P. Lin, Y.-P. Hung, T.-W. Chang, N.-H. Yu, M.-L. Tsai, and M. Y. Chen. 2012. MagMobile: Enhancing Social Interactions with Rapid View-stitching Games of Mobile Devices *(MUM '12)*. ACM.

[6] T. Jokela, J. Ojala, and T. Olsson. 2015. A Diary Study on Combining Multiple Information Devices in Everyday Activities and Tasks *(CHI '15)*. ACM.

[7] A. Lucero, J. Holopainen, and T. Jokela. 2011. Pass-them-around: Collaborative Use of Mobile Phones for Photo Sharing *(CHI '11)*. ACM.

[8] D. Merrill, J. Kalanithi, and P. Maes. 2007. Siftables: Towards Sensor Network User Interfaces *(TEI '07)*.

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

[10] M. Schreiner, R. Rädle, H.-C. Jetter, and H. Reiterer. 2015. Connichiwa: A Framework for Cross-Device Web Applications *(CHI EA '15)*. ACM.

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

[12] J. Yang and D. Wigdor. 2014. Panelrama: Enabling Easy Specification of Cross-device Web Applications *(CHI '14)*. ACM.