XD-MVC: Support for Cross-Device Development

Maria Husmann
Department of Computer Science
ETH Zurich
husmann@inf.ethz.ch

Moira C. Norrie
Department of Computer Science
ETH Zurich
norrie@inf.ethz.ch

Abstract
Cross-device applications are still rarely encountered in the wild. In order to facilitate application development, we introduce XD-MVC, an open-source cross-device framework based on web technologies. XD-MVC is lightweight and integrates with MVC frameworks. Most modern browsers are supported and client devices require no installation. XD-MVC provides connection management, data-synchronisation, and support for UI distribution based on device types and roles. Developer needs are addressed with defaults for common tasks and powerful customisation mechanisms.
If available, peer-to-peer communication is used to achieve fast data-synchronisation between devices, but a hybrid architecture offers a client-server fallback to ensure that a wide range of devices can be supported. As a first evaluation, several applications have been built on top of XD-MVC, using both its JavaScript API and the provided integration with the Polymer MVC framework.

Author Keywords
cross-device;framework;web;peer-to-peer.

ACM Classification Keywords
H.5.2 [Information interfaces and presentation (e.g., HCI)]: User Interfaces - Input devices and strategies

Figure 1: Example application. A gallery application distributed over three smartphones.

Figure 2: Example application. The same gallery application distributed over a smartphone and a TV.

Motivation

Despite a lot of interest in academia, there are still relatively few cross-device applications available to the public. In part, this could be caused by the fact that developing them is challenging due to the lack of available tools as many of the research projects that do offer support are not accessible to developers (e.g. Panelrama [8], Conductor[3]). Two notable exceptions are Polychrome [1] and Connichiwa [7]. Our goal is to support developers in the development process from prototyping through implementation to testing and debugging. We have addressed the first part in MultiMasher [4], a tool for cross-device mashups which could be used for prototyping, and XDStudio [6], a GUI builder for cross-device applications. As a next step, we have created XD-MVC, a framework for implementing web-based cross-device applications. XD-MVC is an open-source project and has been published on Github[1].

System Design

Unlike Connichiwa [7], XD-MVC requires *no installation* on client devices and runs in most modern browsers, however it does depend on network infrastructure. While similar in architecture to Polychrome [1], XD-MVC is not limited to visualisation applications. Besides this *versatility* in terms of application domain, we wanted to give the developers the choice of which specific technologies and UI frameworks to use. Working with familiar technologies could lower the threshold to developing cross-device applications. Thus, we paid attention to making XD-MVC *lightweight*. XD-MVC can be used in combination with MVC frameworks[2] and we provide an integration with the Polymer[3]. We chose Polymer for our proof-of-concept implementation because it is a future-facing framework that supports interesting concepts such as two-way databinding, encapsulation and specification of custom tags. XD-MVC consists of a lower level pure JavaScript API and a higher level API based on Polymer. As argued by Myers et al. [5], UI frameworks should aim for a low threshold and a high ceiling. In order to achieve a low threshold, we provide *defaults* for common use cases in the Polymer API. On the other hand, to keep the ceiling high, control options and means for *customisation* are offered to the developers in the JavaScript API. A developer opting for a different MVC framework, or none at all, can build directly on the JavaScript API (Fig. 3). In order to achieve *fast communication* between devices, we have experimented with a peer-to-peer architecture. Performance tests revealed large performance gains when devices were co-located and the server remote when compared to a client-server architecture. However, not all modern browsers support the necessary WebRTC[4] protocol yet. We thus built XD-MVC with a hybrid architecture where devices preferably communicate peer-to-peer, but use client-server communication as a fallback.

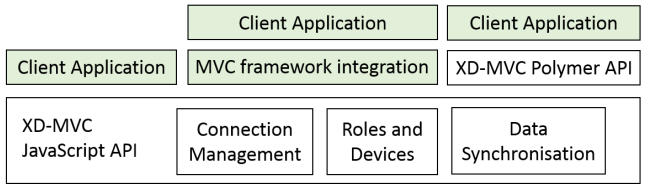

Figure 3: XD-MVC framework structure

XD-MVC Concepts

The framework conceptually consists of three main parts, namely *connection management*, *data synchronisation* and *UI distribution*.

[1] https://github.com/mhusm/XD-MVC
[2] See http://todomvc.com/ for a comprehensive list of frameworks.
[3] https://www.polymer-project.org
[4] http://www.webrtc.org/

```
<xdmvc-connection
  server="xdmvc.
    ethz.ch"
  peerport="9000"
  socketport="3000"
  ajaxport="9001"
  reconnect
  architecture="
      hybrid">
</xdmvc-connection>
```

Listing 1: Declaratively adding connection management using the Polymer API. All attributes have default values and need not be specified.

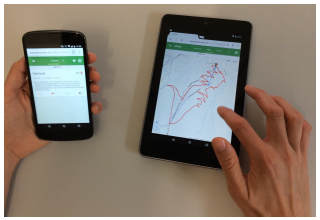

Figure 4: Example application. A bike trip planning application that displays information on a selected route across multiple devices. The map is displayed on the largest device.

Connection Management

Each device is identified by a device ID either generated by the system or set by the developer. To group two devices together, one of them specifies the ID of the other in a connection request. If either device is already connected to a group of other devices, the groups will be merged.

The framework allows the device ID and the connections to be made persistent. Upon loading the web application, the device will attempt to reconnect to all previously connected devices, thus reducing the often tedious pairing process. This behaviour may not always be desired and can be changed or disabled, but has been invaluable during the development process of the showcase applications.

Data Synchronisation

Conceptually, there is a single model that is shared among all connected devices. Depending on the application scenario, it may be desirable to not only synchronise the domain model, but also the state of the view, for example, the content of an input field. In XD-MVC, this requires that a model of the view is defined, referred to as viewmodel in a variation of the MVC pattern. The data synchronisation module of XD-MVC is agnostic to the kind of data that is shared. The developer can specify any object to be shared among the devices—it makes no difference whether the object represents a model or a viewmodel.

XD-MVC's API provides both defaults and options for customisability. While, per default, the framework observes the objects for changes, the developer can also explicitly push updates. This can be useful when the objects are not fully under the control of the developers. In XD-MVC, synchronisation is based on a publish-subscribe mechanism. Each device publishes changes of its objects to devices that subscribed. By default, the changes will be integrated into the local version of the object, however, in the changed event-handler, the developer can specify other actions. The event-handler not only reports the changes to the object, but also which devices sent the changes. In addition to sending changes to connected devices, they can also be sent to the server where they can be made persistent.

UI Distribution

UI distribution mostly concerns the views, which in turn are highly dependent on the specific MVC frameworks used. XD-MVC provides a query mechanism for devices as a basis for the distribution, so that distribution can be based on the type of the local device as well as the devices which are connected to it. In addition, XD-MVC supports roles. Both [2] and XDStudio [6] introduce the notion of user roles which relates to the tasks carried out by users. In XD-MVC, we interpret the terms more loosely and simple consider it additional information associated with a device. While it can relate to users, a device can assume multiple roles simultaneously and roles can be created and changed dynamically at run-time. When a role is added to a device, this information is propagated to all connected devices so they can adapt accordingly. We implemented a publish-subscribe API for role changes. In addition to this event-based mechanism, the current state of roles in the group can also be queried at any time. The query API can be used to check the roles of the local device (List. 2) as well as the number of roles (or devices) of a given type to which it is connected.

```
<template is="dom-if" if="{{roles.
    isselected.owner}}">
  <gallery-element>
  </gallery-element>
</template>
```

Listing 2: Specifying the UI distribution by querying roles in a Polymer application. The Gallery elment will only be displayed if the local device has the owner role.

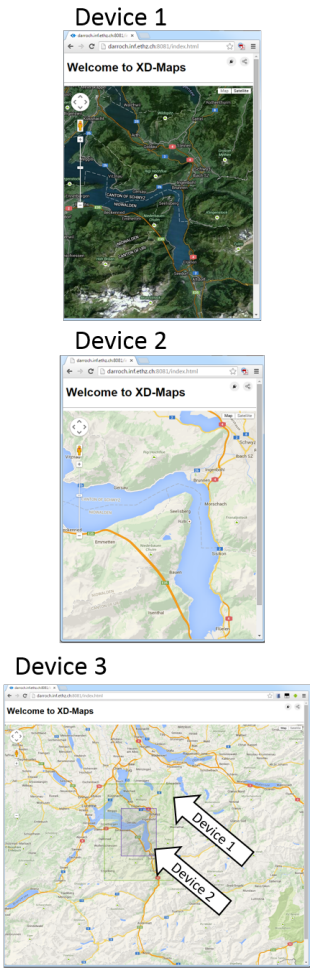

Figure 5: Example application. A maps application. Device 1 and 2 synchronise the map centre while device 3 shows their viewports.

Applications

Several applications have been built based on XD-MVC. XDMaps is a maps application that allows users to explore a location in multiple views (e.g. street map and satellite). It also includes an overview mode that visualises the viewports of the collaborating devices on a map. XDMaps builds on the JavaScript API. XDGallery is an application for easily showing pictures to friends using whatever devices are at hand, such as smartphones but also large screens such as smart TVs. A student used AngularJS[5] on top of XD-MVC to build a slideshow application. A group of four students used XD-MVC to build an application for planning bike trips. The application distributes itself over the available devices taking into account the screen size of the devices. For example, map information will be shown on a larger screen while a short description of a bike route will be displayed on smaller screens. XDBike and XDGallery haven been built using the Polymer API.

Conclusion

With XD-MVC, we lay the foundation of our goal to make cross-device application development easier. However, during the development of the applications, we noticed that further support is needed. We are currently building a set of tools for testing and debugging cross-device applications. Cross-device applications are typically built to support a large number of device configurations which is challenging to test as not all devices may even be available to the developer and switching between configurations could be cumbersome. In parallel, we are working on a tool for analysing usage data of deployed cross-device applications. The goal is to gain insights into how users actually use cross-device applications in order to optimise and improve them.

[5] https://angularjs.org/

REFERENCES

1. Sriram Karthik Badam and Niklas Elmqvist. 2014. PolyChrome: A Cross-Device Framework for Collaborative Web Visualization. In *Proc. ITS*. ACM, 109–118. DOI: http://dx.doi.org/10.1145/2669485.2669518
2. Luca Frosini and Fabio Paternò. 2014. User Interface Distribution in Multi-Device and Multi-User Environments with Dynamically Migrating Engines. In *Proc. EICS*. ACM, 55–64.
3. Peter Hamilton and Daniel J. Wigdor. 2014. Conductor: Enabling and Understanding Cross-Device Interaction. In *Proc. CHI*. ACM, 2773–2782.
4. Maria Husmann, Michael Nebeling, Stefano Pongelli, and Moira C. Norrie. 2014. MultiMasher: Providing Architectural Support and Visual Tools for Multi-Device Mashups. In *Proc. WISE*. Springer, 199–214.
5. Brad A. Myers, Scott E. Hudson, and Randy F. Pausch. 2000. Past, present, and future of user interface software tools. *ACM TOCHI* 7, 1 (2000), 3–28. DOI: http://dx.doi.org/10.1145/344949.344959
6. Michael Nebeling, Theano Mintsi, Maria Husmann, and Moira C. Norrie. 2014. Interactive Development of Cross-Device User Interfaces. In *Proc. CHI*. ACM, 2793–2802.
7. Mario Schreiner, Roman Rädle, Hans-Christian Jetter, and Harald Reiterer. 2015. Connichiwa: A Framework for Cross-Device Web Applications. In *Proc. CHI EA*. ACM, 2163–2168. DOI: http://dx.doi.org/10.1145/2702613.2732909
8. Jishuo Yang and Daniel Wigdor. 2014. Panelrama: Enabling Easy Specification of Cross-Device Web Applications. In *Proc. CHI*. ACM, 2783–2792.