Heterogeneous Device Hopping

Bridging the Mobile Cross-Platform Gap via a Declarative Query Language

Sanchit Chadha, Antuan Byalik, and Eli Tilevich

Software Innovations Lab
Virginia Tech, Blacksburg, VA 24061, USA
{schadha,antuanb,tilevich}@cs.vt.edu

Abstract

A typical mobile user employs multiple devices (e.g., a smartphone, a tablet, wearables, etc.). These devices are powered by varying mobile platforms. Enabling such cross-platform devices to seamlessly share their computational, network, and sensing resources has great potential benefit. However, sharing resources across platforms is challenging due to a number of difficulties. First, the varying communication protocols used by major mobile vendors tend to overlap minimally, making it impossible for the devices to communicate through a single protocol. Second, the host platforms' underlying architectural differences lead to drastically dissimilar application architectures and programming support.

In this demo, we present *Heterogeneous Device Hopping*, a novel approach that systematically empowers heterogeneous mobile devices to seamlessly, reliably, and efficiently share their resources. The approach comprises 1) a declarative domain-specific language for device-to-device communication based on the RESTful architecture; 2) a powerful runtime infrastructure that supports the language's programming model. In this demo, we show how our approach can be used to implement a multi-device animation across heterogeneous nearby devices. The animation starts on one device and moves across the device boundaries, irrespective of the underlying mobile platform.

Categories and Subject Descriptors D.2.7 [*Software Engineering*]: Distribution, Maintenance, and Enhancement

General Terms Languages, Experimentation

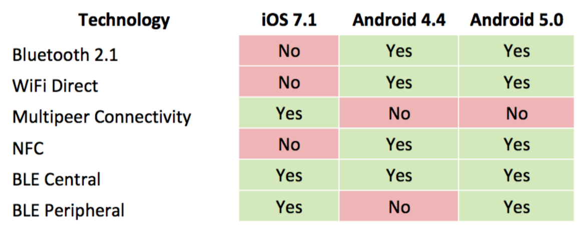
Keywords Mobile Applications, Near Field Resource Sharing, Domain Specific Languages, Runtime

1. Introduction

The modern mobile application market remains fragmented with multiple competing platforms and application architectures. Additionally, the average mobile user enjoys a wide range of commonly available device types, ranging from smartphones and tablets to smart watches and other wearables. With no single, dominant mobile software vendor, these devices may be powered by one of the major platforms, including Android, iOS, and Windows Phone. The fleet of available devices carries a wide range of functionality distributed unevenly among devices. A tablet may be more suitable for a computationally expensive operation than a smartphone, if prolonging battery life is a chief concern. Similarly, a smart watch would be far more appropriate to continuously retrieve heart-rate information with its built-in sensor than a tablet without this functionality. In essence, resource sharing facilitates access to functionality not easily supported on the host device, due to either hardware or software resource limitations.

Sharing heterogeneous mobile resources efficiently is strewn with difficulties, chief among which is a lack of a common communication protocol. Figure 1 shows the various primary communication protocols, available on the latest iOS and Android platforms. As the latest Android-based OS, Android 5.0 is yet to be fully embraced by the mobile community. Indeed, the current most common Android version is KitKat (4.4) [1]. This adoption delay complicates heterogeneous resource sharing—Figure 1 shows that only Android 5.0 support both BTLE Central and Peripheral alongside iOS, thus being the only cross-platform communication protocol. The primary mobile platforms' architectural and programming model differences requires a nuanced handling for heterogeneous devices to seamlessly share their resources.

In this demonstration, we highlight the main design features of our nascent project, whose overriding goal is to address the challenges outlined above. Our approach—*Heterogeneous Device Hopping*—enables the mobile software developer to express inter-device communication logic declaratively, by means of a domain-specific language. The language is supported by a runtime system that bridges over the panoply of dissimilarities of the major mobile platforms, while also providing fault tolerance.



Technology	iOS 7.1	Android 4.4	Android 5.0
Bluetooth 2.1	No	Yes	Yes
WiFi Direct	No	Yes	Yes
Multipeer Connectivity	Yes	No	No
NFC	No	Yes	Yes
BLE Central	Yes	Yes	Yes
BLE Peripheral	Yes	No	Yes

Figure 1. iOS vs. Android Connectivity Protocols

2. Technical Approach

At the core of *Heterogeneous Device Hopping* is the Resource Query Language (RQL), a declarative, platform-independent programming language based on the RESTful architecture. Its runtime

SPLASH Companion'15, October 25–30, 2015, Pittsburgh, PA, USA
© 2015 ACM. 978-1-4503-3722-9/15/10...$15.00
http://dx.doi.org/10.1145/2814189.2814191

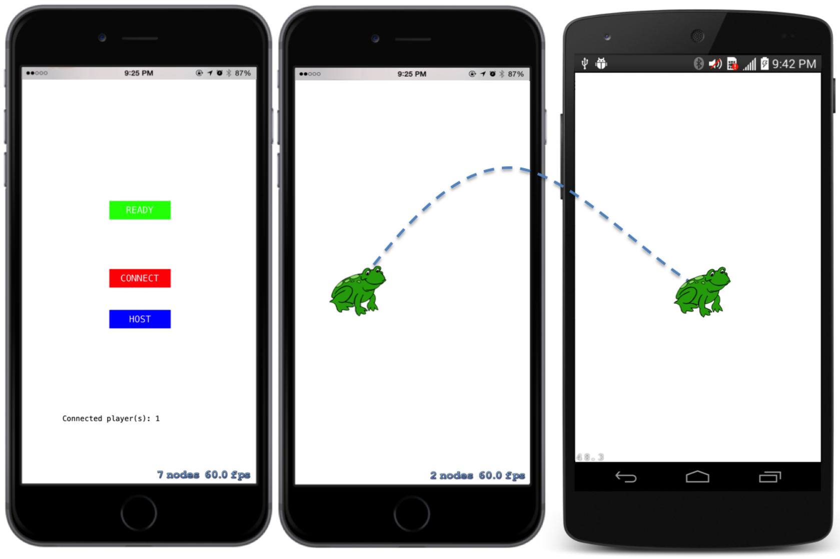


Figure 2. From left to right: Start/setup screen for host, iOS initial position, frog sprite "hopped" across to Android screen

system frees the mobile programmer from the need to write low-level, platform-specific code to enable cross-device communication. We describe RQL and the runtime in turn next.

2.1 RQL Design

RQL, a platform-independent, domain-specific language for the sharing of resources across heterogeneous devices, is based on the RESTful architecture [3], which solves many of the toughest challenges that arise when engineering robust heterogeneous distributed systems. Hence, RQL follows the nouns/verbs paradigm: nouns express the requested resources; verbs express the actions performed on these resources. In *Heterogeneous Device Hopping*, the nouns are characteristics that identify the host or peripheral device while the four verbs are: (1) `pull` data, (2) `push` data, (3) `delegate` work and await result, and (4) `bind` to a *resultCharacteristic* and receive updates.

Figure 3 displays the RQL command, which initiates the process shown in Figure 2. Specifically, the RQL *bind* verb applied to the *any:game/result* noun exemplifies the core operational procedure of RQL. In this instance, the RQL command leverages the heterogeneity of the `bind` verb when applied to a noun prefixed with "any", which directs the runtime to find *any* available nearby resource, as opposed to a particular device or location.

```
remoteServiceConnection.sendRQL("bind any:game/result");
```

Figure 3. RQL command sent to host

2.2 Runtime Infrastructure

The runtime infrastructure is centered around two major tasks: (1) process RQL requests, both incoming, from the 3rd-party application, and outgoing, to the runtime located on the accessed nearby device, and (2) handle the appropriate communication details to send/receive the requested data specified by the RQL requests. Additionally, the extensible design of the runtime makes it possible to handle the communication protocols irrespective of the host platform. Serving as a virtual layer on the communicating hosts, the runtime provides a level of abstraction making it possible to express remote requests in a high-level, declarative fashion, without polluting the code with convoluted fault-tolerance logic.

3. Demonstration Plan

This interactive demonstration illustrates a representative case of heterogeneous mobile devices pulling their resources to accomplish a common task. In particular, we show a multi-device animation, in which a sprite commences its journey on one device, and upon reaching the edge of the device's screen, moves over to the next device, thus creating the notion of inter-device hopping. Although animations like this can be implemented by writing low-level, platform-specific code, in this demonstration we show how this non-trivial functionality is implemented by means of a couple of lines of RQL and its platform-specific bindings.

In the demonstrated case, the host mobile device is an iOS-based device with the devices connecting to it being either Android or iOS. More specifically, we used two equivalent game development toolkits for iOS [2] and Android [4], respectively to create an animation for both platforms. The main thrust of the demo will show a hopping frog jumping across device screens as shown in Figure 2. The demonstration will support as many heterogeneous devices as we choose to engage for the demo. Furthermore, removing a device while the journey is in progress would cause the animated character to jump to the next available device. A possible secondary thrust would be enabling attendees to play the classic game of pong wirelessly across several heterogeneous devices.[1]

References

[1] C. B. Bautista. Android Lollipop is only on 5.4 percent of devices, and KitKat is Still the Most Popular, 2015.

[2] developer.apple.com. About Sprite Kit, 2015.

[3] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[4] https://code.google.com/. Cocos2d-android, 2015.

[1] This research is supported in part by the National Science Foundation through Grant CCF-1116565.