Supporting Heterogeneous Device Development and Communication

Sanchit Chadha

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Eli Tilevich, Chair
Alla Rozovskaya
Kurt Luther

December 8, 2015
Blacksburg, Virginia

Keywords: Mobile Computing, Code Synthesis, Heterogeneous Distributed Runtime, Domain-Specific Language Design

Supporting Heterogeneous Device Development and Communication

Sanchit Chadha

(ABSTRACT)

To increase market penetration, mobile software makers support their popular applications on all major software platforms, which currently include Android, iOS, and Windows Phone. Although these platforms often offer a drastically different look and feel, cross-platform applications deliver the same core functionality to the end user. Maintaining and evolving such applications currently requires replicating all the changes across all supported variants, a laborious and intellectually taxing enterprise. The state-of-the-practice automated source translation tools fall short, as they are incapable of handling the structural and idiomatic differences of the software frameworks driving major mobile platforms.

In addition, popular mobile applications increasingly make use of distributed resources. Certain domains, including social networking, productivity enhancement, and gaming, require different application instances to continuously exchange information with each other. The current state of the art in supporting communication across heterogeneous mobile devices requires the programmer to write platform-specific, low-level API calls that are hard not only to develop but also to evolve and maintain.

This thesis reports on the findings of two complementary research activities, conducted with the goal of facilitating the development and communication across heterogeneous mobile devices: (1) a programming model and runtime support for heterogeneous device-to-device communication across mobile applications; (2) a source code recommendation system that synthesizes code snippets from web-based programming resources, based on the functionality written for Android or iOS and vice versa. The conceptual and practical advancements of this research have potential to benefit fellow researchers as well as mobile software developers and users.

This thesis is based on research accepted for publication to SPLASH'15 [25] and GPCE'15 [22].

This research is supported by the National Science Foundation through Grant CCF-1116565

Dedication

I dedicate my thesis to my parents, close friends and most importantly my late grandfather, Dr. K.L. Sahni

He has always showered me with wisdom and this journey would not have been possible without his blessings.

"Om Sai Ram"

Acknowledgments

"Success is not a destination, it is a journey"

Although only my name is listed as the main author for this thesis, there is no way I would have been able to finish the production of this manuscript without the support, wisdom, blessings, help, and love from so many people that I have encountered on this journey. Without their motivation and advice, I would not be the man that I am today. I wish to take a moment to acknowledge and thank the people who were responsible for making my journey through higher education a most cherishing and rewarding experience.

First and foremost, I would like to thank the two most important people in my life. My parents, Anu Chadha and Salil Chadha. I literally would not be here without you two. There are not enough words to describe how grateful I am for the upbringing that you have provided me. The sacrifices you have made for me to get the best education wherever we lived and to keep me motivated on the path to success will never be forgotten. Without your blessings and wisdom, I would have given up a long time ago. Thank you for being with me every step of the way, for putting your faith in me and trusting me that I would make the right decisions, but also always being there when I was lost. I appreciate the cultural values and etiquettes that you have taught me and that have allowed me to travel this far in life. Dad, you have always been my pillar of support and I can't thank you enough for the wisdom and life lessons that you have instilled upon me. The quote at the beginning of this page was written on your resumé almost 15 years ago and I will never forget it and its meaning. Mom, you have been ever so loving, caring, supportive and selfless—I trust you more than anyone else and I thank you for raising me to be your loyal son. There is so much more to say, but just know that I will always make you both proud.

To my late grandfather, Dr. K.L. Sahni, I know you are looking down at me with your beautiful and wise gaze and I hope that I have made you proud. My first experience with research and scientific papers was by looking at your PhD dissertation as a young boy. I know that your scholarly achievements and strive for excellence are the reasons that I have been able to succeed in this journey and I thank you for listening to me attentively and always caring about my education. I wish that God hadn't taken you so soon because I would have loved to share my experiences in higher-education and gain even more wisdom from you.

To my brother from another mother, Asad Jafree. There are people in one's life that leave an impression so positive and impressive that it is impossible to forget them. You are one of those people Asad. Our friendship, rather our brotherhood, is something that I am genuinely proud and grateful for. Supportive, selfless, kind, humble and generous are some adjectives that come to mind when talking about you. You have always been there for me and I couldn't have asked for a more fulfilling friendship than yours. Thank you for everything and I look forward to what the world will bring for us.

I would like to thank my roommate and closest friend since the beginning of our college careers, Divit Singh. Your constant pursuit for excellence and competitive edge were my motivations to succeed and I am so thankful to have the honor of being your best friend and roommate. We have shared so many unforgettable experiences at Virginia Tech and I thank you for making them even more entertaining with your quick wit and humor. I have enjoyed these years with you and can't wait to see what the future holds for us! I would also like to take this opportunity to thank your parents. They have treated me like their son and I am very grateful for their love and support.

If there was one man who was responsible for igniting the pursuit for research in me, it would be my advisor Dr. Eli Tilevich. My academic relationship with you began in the Fall of 2012 while I was involved in undergraduate research. We immediately clicked because of a mutual Soviet background and a passion for novel research in the exciting field of mobile software engineering. Soon after, you demonstrated your impressive wealth of knowledge, experience and absolute command of the English language. You have a gift for turning lackluster sentences into brilliant scriptures and I have the utmost respect and admiration with the way you handle writing and editing technical publications. You have taught me to stay dedicated, responsible and independent by giving me several deadlines and objectives to accomplish and then let me roam free, trusting me that the task at hand would be delivered by the deadline. It is not easy to establish such a trusting relationship and I thank you from the bottom of my heart for allowing me to work with you. I will never forget our "crunch-time" late nights for paper submissions (especially OOPSLA'15) and our intense table tennis matches after you exponentially improved your skills (demonstrating another one of your strengths). Thank you for your everlasting support, guidance and trust in me. You play an incredible part in making me a better student and person, and I owe my post-graduate accomplishments to you.

I would also like to express my gratitude to my committee members Dr. Alla Rozovskaya and Dr. Kurt Luther. Dr. Rozovskaya, thank you for your support and expertise in the completion of this thesis, I look forward to our future publications. Dr. Luther, you were the most engaging professor of my graduate career and I really enjoyed attending both of your classes. Your discussions and projects were intellectually stimulating and I am delighted to say that I will be applying your web development techniques for my job. Thank you both for understanding and resolving the scheduling conflicts for this thesis defense.

My transition from pursuing an undergraduate degree to a graduate degree with the 5 year

BS/MS program would not have been possible without the support and guidance from Dr. Calvin J. Ribbens. You have been extremely influential in my academic success and I thank you for all the help that you have provided me. Thank you to Mrs. Terry Arthur for providing me with excellent advice during my undergraduate career. Dr. Osman Balci, thank you for your support and belief in my abilities—your mobile software engineering class was an amazing learning experience and my love for iOS and Swift is attributed to you. Mrs. Christina Lockette, I thank you from the bottom of my heart for always believing in me and being proud of my accomplishments ever since you were my 1st grade teacher.

A special thanks to Antuan Byalik for providing a bulk of the research material for this thesis and for being my benchmark for estimated time till thesis completion. Thanks to Melanie Sandler for being such an amazing friend, giving me great advice and for always supporting my ambitions. Thank you to Josh Orrick for keeping me motivated by demonstrating your excellence in education and for always being open to talk about anything ... and I mean *anything*. Thanks to my freshman year hall mates Ben Tronrud, Jordan Miller and Takumi Gillen—you guys have been great friends and I will forever cherish the memories that we have made. Last but not least, I want to thank all of my other friends and family members who have pushed me to succeed and took a part in my happiness and achievements.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Objective and Overview

The primary objective of this thesis is to address the challenges imposed on modern software developers by the fragmentation of the mobile development space, with several major platforms, including Android, iOS, and Windows Phone, competing to dominate market share. Shrewd mobile software vendors strive to support popular applications on all major platforms, so as to maximize market penetration. On each supported platform, an application replica essentially delivers identical functionality, albeit within the conventions and format of the platform on which it is running.

1.1.1 Central Challenges

The goal of this thesis is to address the following central challenges and provide a feasible solution for overcoming them:

1. What programming model is required to support mobile software developers charged with the challenge of providing heterogeneous device-to-device communication across applications?

2. What runtime support must be provided for the aforementioned programming model to ensure efficiency and robustness in the face of network volatility, device mobility, and energy constraints?

3. How can one effectively use vast web-based programming resources, utilizing existing developing efforts on the Android platform, to identify which code snippets are most suitable for performing the same programming task on the iOS mobile platform, and vice versa?

1.1.2 Heterogeneous Distributed Applications

Technology	iOS 9.1	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 1.1: Communication Protocols Supported by Android and iOS Platforms.

The average mobile user enjoys a wide range of commonly available device types, ranging from smartphones and tablets to smart watches and other wearables. 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. Mobile developers have great potential to design novel mobile applications that share resources to facilitate access to functionality not easily supported on the host device, due to either hardware or software resource limitations.

Despite their huge potential to open up an entirely new array of mobile capabilities, heterogeneous distributed mobile applications remain difficult to design and maintain. One of the issues standing in the way of developing these applications is a lack of a common communication protocol. Figure 1.1 shows the various primary communication protocols, available on the latest iOS and Android platforms. As the latest Android-based OS, Android 5.0, is being embraced by the mobile community, the next version, 6.0, is already rolled out. It would be unrealistic to expect all Android users to suddenly switch to the platform's latest version. This adoption delay complicates device-to-device communication—Figure 1.1 shows that only Android 5.0 and up support both Bluetooth Low Energy (BLE) Central and Peripheral alongside iOS, thus currently being the only cross-platform communication protocol.

Even putting the lack of a common standard communication protocol aside, relying on low-level network APIs to implement heterogeneous device-to-device functionality results in brittle, difficult-to-understand and maintain communication code that is bound to become obsolete and non-functional after an update to the next platform version. Hence, the mobile platforms' architectural and programming model differences require a nuanced handling for

heterogeneous devices to enable them to seamlessly communicate with each other. By raising the level of abstraction for such communication, we formulate a viable approach to addressing the technical challenges discussed above. In addition, this approach offers solutions to the mainstay complications of distributed computing, such as handling partial failure.

1.1.3 Reducing Programmer Effort

Having overcome the challenges of ascertaining the correct program logic and implementation details on the Android mobile platform for example; the developer has no choice but to repeat the same laborious and intellectually tiresome process on the application's counterpart for the iOS mobile platform. The issue at hand is that *the developer is unable to leverage the expertise gained by undertaking a programming task on one platform to facilitate the performance of that same task on other platforms*. What if it were possible to automatically capture the knowledge acquired by working on an application on one platform to lower the effort required to work on the same application on the remaining platforms?

For example, we developed an interactive demo for both the Android and iOS platforms natively. That is, logic and view based code was written specifically for both the Android platform in Java and the iOS platform in Swift. This demo featured animated sprites moving across the screen of an Android device and seamlessly entering the screen of the second iOS device. Engineering this application and functionality required two times the programming effort because one application logic was written in Java for Android and that same logic had to be repeated in Swift for iOS. Although from the end-user's perspective, this application provides identical functionality on both platforms, from the software engineering perspective, implementing the same feature on different platforms requires the use of vastly dissimilar languages, APIs, and software architectures. This approach of designing native mobile applications for two syntactically and architecturally different mobile platforms was an arduous process and is in dire need for reduction in the effort expended by a developer.

1.2 Contributions

In this thesis, we present a programming model and runtime support for mobile applications to fulfill the challenges of heterogeneous device-to-device communication. This programming model is realized as a platform-independent, domain-specific language based on a Representational State Transfer (REST)ful architectural style [36] and allows the mobile software developer to express inter-device communication logic declaratively. The language is supported by a runtime system that bridges over the panoply of dissimilarities of the major mobile platforms (iOS and Android for this thesis), while also providing fault tolerance [35]. To ease portability, the approach performs and is managed at the application level, without modifying operating system libraries. Hence, the programing abstractions for inter-device

communication run on off-the-shelf nearby devices, with the runtime making use of near-field wireless protocols (e.g., Bluetooth LE).

To address the challenge of reducing programmer effort for cross-platform mobile development, we introduce a bidirectional source code recommendation system that synthesizes code snippets from web-based programming resources for both Android/Java and iOS/Swift code blocks. It discovers publicly available code blocks for one mobile platform whose semantics are equivalent to a code block written for another platform. The basic development flow proceeds with capturing the code for a feature at hand on one platform, then the captured code is used to form a web search query to fetch the available posted code on a target platform that implements the same functionality.

1.3 Thesis Roadmap

The rest of the thesis is organized as follows. First, Chapter 2 gives an overview of the terminology and abbreviations used in this thesis. Chapter 3 presents a background of the mobile development environment and why heterogeneous mobile applications are so marketable. Then, Chapter 4 compares the presented approach with several related works which have inspired these concepts. Afterwards, Chapter 5 expounds upon the programming model and runtime support for supporting heterogeneous mobile communication. Chapter 6 provides the proof-of-concept demo's design and implementation details. Switching gears, Chapter 7 discusses the source code recommendation system and its associated intricacies. Chapter 8, delves into the evaluation of this system and the interpretation of the presented results. Finally, Chapter 9 presents my conclusions for this thesis and potential future work directions.

Chapter 2

Terminology

- *distributed system* → software system in which components communicate and coordinate actions by passing messages
- *heterogeneous* → designed for systems with different capabilities in terms of hardware, platform or protocols
- *peripheral* → in the bluetooth low energy protocol, a device that has data that is needed by other devices
- *central* → in the bluetooth low energy protocol, a device that uses the information served up by a peripheral to accomplish a task
- *native resources* → hardware sensors and software libraries customized to the host platform
- *source document* → input code block from any mobile platform
- *target document* → output code block from any mobile platform
- *query* → extracted semantics from an input code block, used to search for the target document's constituent components
- *token* or *element* → any single document element at the level of individual strings

Chapter 3

Background

This chapter provides a background on some of the technologies and the foundations for the research discussed in this thesis. An overview of the mobile development landscape is provided which intends to uncover the similarities and differences between the modern mobile platforms. This transitions into details about the communication protocol used for our runtime's heterogeneous device communication and finally an overview of what recommendation systems are and why they can be useful to the modern mobile developer. This chapter intends to provide a high-level overview on some of these technologies so the reader can follow along the technical ideas presented in the remainder of this thesis.

3.1 Mobile Software Development

The mobile landscape currently features a plethora of available device types ranging from smartphones and tablets to smart watches and other wearable technologies. Each type of these devices carry their own wide range of features that may or not be present in another type of the same device running a different mobile platform. This creates a sort of competition between mobile manufacturers to develop products that outdo one another. Currently, Android, iOS and Windows Phone dominate the marketplace for end-user mobile platforms. When one platform deploys a feature, the other platforms follow suit and develop a similar feature for their devices. In Figure 3.1, one can see the home screens of the three most popular mobile platforms—Android, iOS and Windows Phone. The layouts and aesthetics are fairly different across the platforms, but they all serve one purpose; to organize and display installed applications on the device. Table 3.1 goes into further detail comparing the various features and attributes associated with each of the mobile platforms. One can see that all of these mobile platforms have most features implemented but with their own name branding or twist associated with it. The following sections will discuss these mobile platforms in more detail.

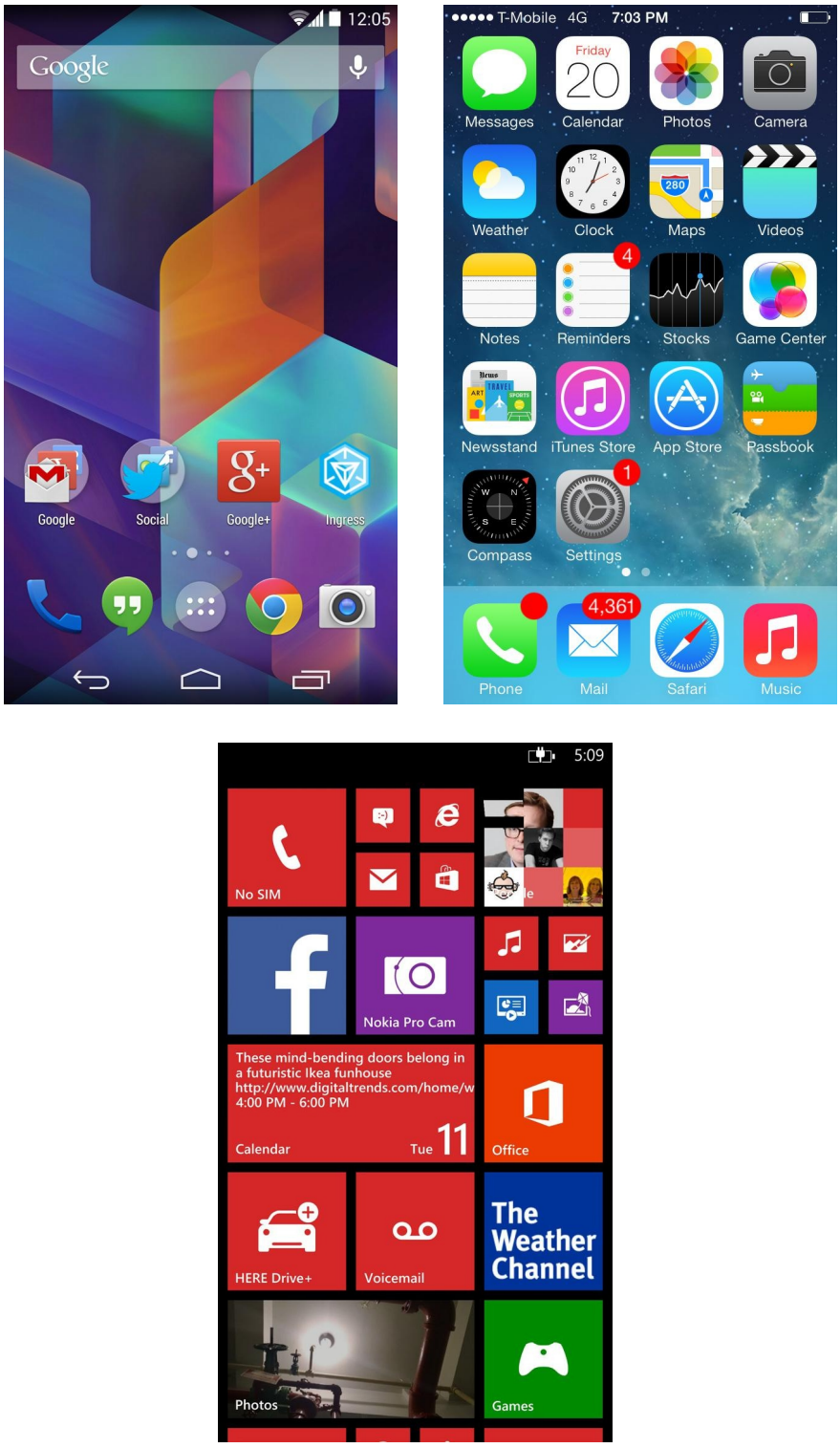

Figure 3.1: Mobile Platform Screenshots: Top Left - Android [104] Digital trends - android screenshot. `http://icdn6.digitaltrends.com/image/android-vs-ios-windows-interface-1-640x1136.jpg`, 2015, Top Right - iOS [105] Digital trends - ios screenshot. `http://icdn6.digitaltrends.com/image/android-vs-ios-windows-interface-3-640x1136.jpg`, 2015, Bottom - Windows Phone [106]Digital trends - windows phone screenshot. `http://icdn6.digitaltrends.com/image/android-vs-ios-windows-interface-2-640x1136.jpg`, 2015. Used under fair use, 2015

Table 3.1: Feature comparison between Android, iOS and Windows mobile platforms

	Android	**iOS**	**Windows Phone**
Affordability	Varies	$200 - $650	Varies
Interface	Minimalist	Bright and Modern	Live Tiles
Apps	1.8 Million	1.5 Million	385,000
App Store Usability	2nd	1st	3rd
Alt App Stores	Yes	No	No
Battery Life	Battery Saving Mode Removable Battery	Optimized OS Detailed Usage	Basic Battery Saver
Updates	Older devices lag	Most devices updated to latest version	Older devices lag
Customizability	Lots of freedom to customize	Somewhat restricted	Color schemes Tile Size
Calls Messaging	Hangouts	iMessage	Skype
Email	Similar Support	Similar Support	Similar Support
Peripherals	MicroUSB standard opens up to a lot of 3rd party vendors	Vast variety of accessories made for iOS	MicroUSB standard opens up to a lot of 3rd party vendors
Cloud Services	Google Drive	iCloud	OneDrive
Photo Backup	Google+ / Photos	iCloud Photo Library	OneDrive Photos
Voice Assistants	Google Now	Siri	Cortana
Connectivity	Bluetooth Bluetooth LE Wi-Fi Direct NFC	Bluetooth Bluetooth LE Multipeer NFC (Apply Pay Only)	Bluetooth Bluetooth LE (8.1 above) Wi-Fi File Sharing NFC
Security	Easy target for malware attacks	Touch ID Regular Security Updates	Not a huge target
Maps	Google Maps	Apple Maps	Bing Maps
Camera	Varies	8MP camera surpasses many regular digital cameras	Varies
Simplicity	Customization options can be overwhelming	Simple to pick up and use	Most readable

3.1.1 Android

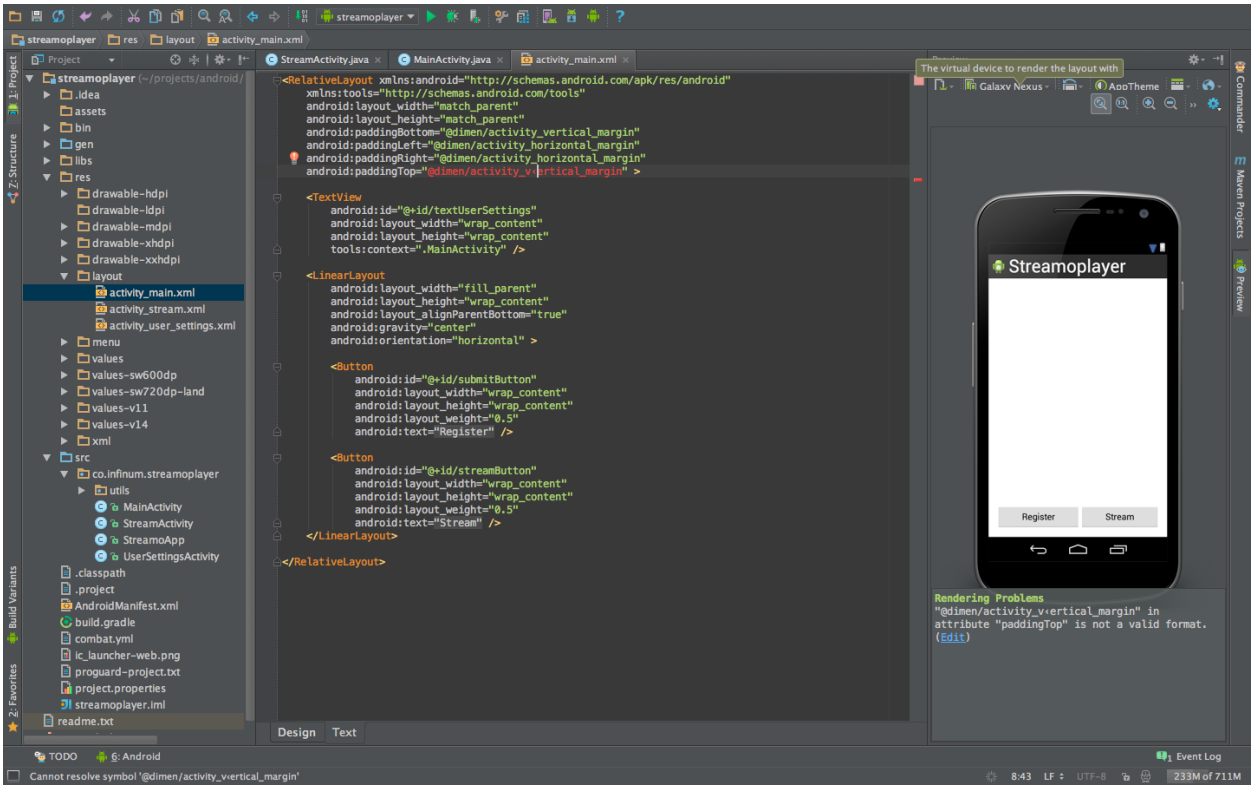

Figure 3.2: Android Studio IDE for developing Android applications [85] — Z. Plesac. Android studio vs. eclipse. `https://s3.amazonaws.com/infinum.web.production/repository_items/files/000/000/168/original/android-studio-3.png?1393599622`, 2013. Used under fair use, 2015

The Android operating system was unveiled in 2007 after Google bought Android Inc. in 2005 [24]. The source-code is available as open-source and free for the public to use and enhance. This philosophy makes Android one of the most customizable operating systems out of the three, but also introduces vulnerabilities due to the open nature of the operating system. Development for this platform is through the Android Standard Library package which is written in Java. Since Java is one of the most popular programming languages to date [56], it makes developing Android applications easier to approach for the modern developer.

Android's mobile market for Android applications, Google Play Store, currently hosts approximately 1.8 million [4] applications for users to download for free or to buy for a nominal one time payment. Android's market share has expanded to a plethora of other devices such as tablets and wearables. The Google Glass headset, introduced in 2013, uses an Android software variant called Glass OS [45] and several Android watches run another variant called

Android Wear [44]. The codebase foundation of these different software variants is still Java, so developing applications that work across different Android devices is a trivial matter for an experienced Java developer. However, complications arise when developing applications for cross-platform purposes.

3.1.2 iOS

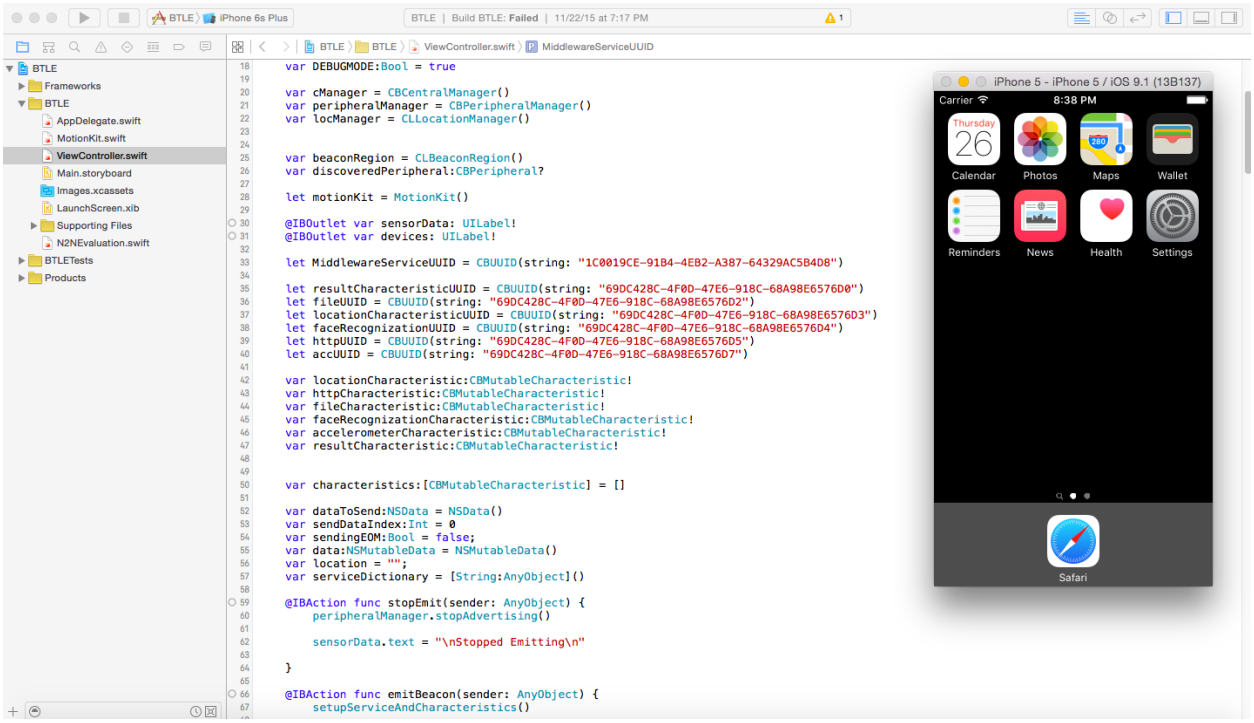

Figure 3.3: XCode 7 IDE for developing iOS applications

The original iPhone was unveiled in January of 2001 along with its new mobile platform, iOS [103]. Originally a closed-source platform written entirely in Apple's in-house programming language, Objective-C [39], iOS was slow to gain popularity with developers due to its hardware restriction of only being able to be compiled on Macintosh computers and operating systems. However, as Mac adoption increased and the iPhone gained mass popularity with the introduction of the Apple App Store in 2008, Objective-C began to be widely adopted and used for creating a variety of different iOS applications.

Just like the Android Standard Library, iOS comes equipped with a plethora of features and APIs under the Core OS [6] and Cocoa Touch [7] libraries. Right behind Google's Play Store, Apple's App Store has 1.5 million applications [99] available to download. Applications submitted to Apple's App Store are regulated to a rigorous review process [10] which means that most, if not all of the available applications are high-quality and safe to use.

As the Apple iOS market share increased, Apple introduced another in-house programming language, Swift [8] to mass excitement and admiration. Swift was designed as a multi-paradigm, object-oriented and functional language to be used for Apple's mobile and other peripheral device platforms. It was also designed to integrate seamlessly with existing Objective-C codebases since several large software applications were still written and maintained in the previous programming language. With Swift 2.0 in 2015, Apple released the operating system as open-source [11] which marked a step in the right direction for cross-platform mobile developers since the hardware restriction of developing on Macintosh computers would be lifted.

3.1.3 Windows Phone

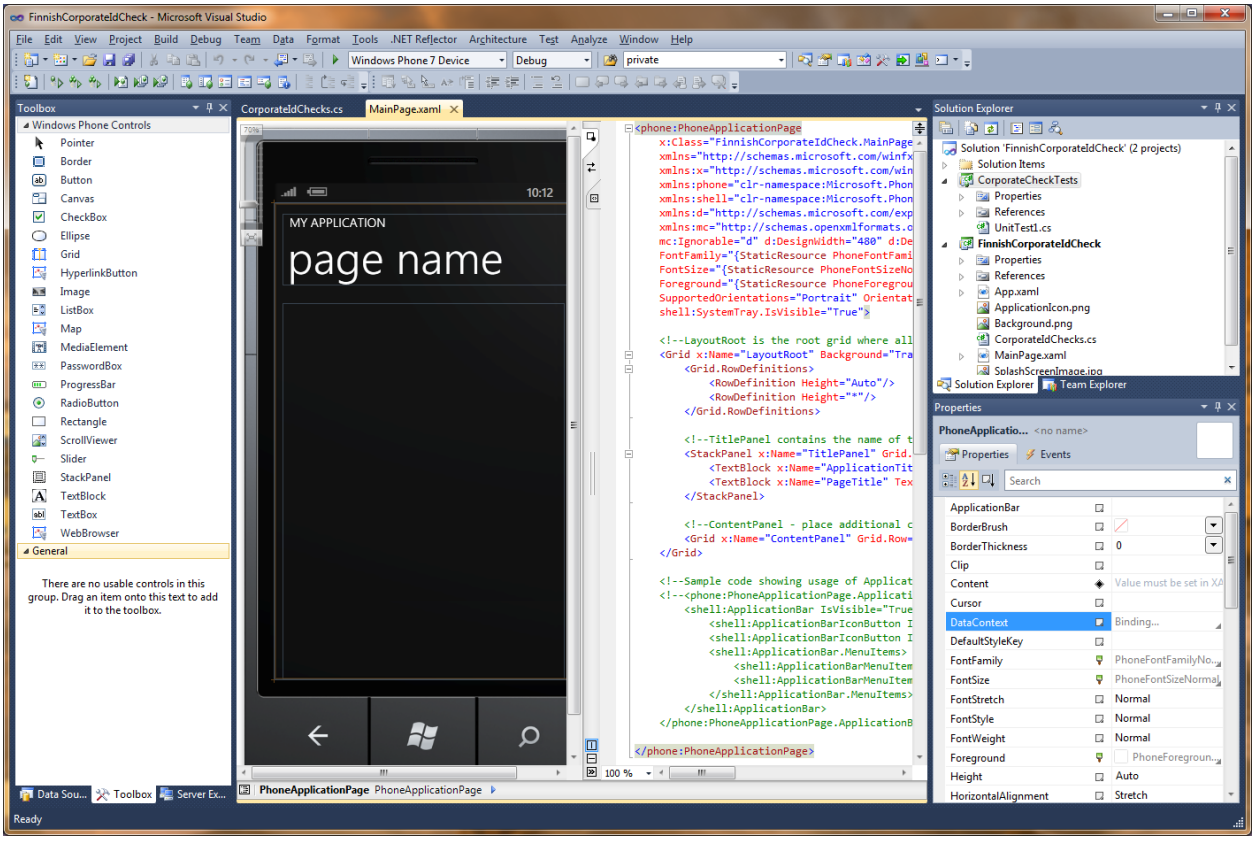

Figure 3.4: Microsoft Visual Studio IDE for developing Windows Phone applications [30] — Developer.com. Submitting your windows phone 7 application to the windows marketplace. `http://solutions.developer.com/mobilephone/images/Figure4-VisualStudioIDE.png`, 2012. Used under fair use, 2015

Compared to iOS and Android, Windows Phone has a very low percentage of market share (2.6% [55]) especially since it's still a young platform first introduced in 2010 [49]. Microsoft

has primarily been a desktop and tablet software provider and has only recently made a push towards the mobile market. Comparitively, the Windows Phone App Store has only 385,000 applications [112] available to download. With the advent of Windows 10 Mobile, the latest iteration of the Windows Phone OS, Microsoft has developed a unified development pattern for developing both mobile and desktop based applications in one go. The Windows Phone platform has a drastically different UI layout, based on tiles, than other more popular platforms. C# and C++ are the primary development languages for Windows Phone and Visual Studio (Figure 3.4) is the popular IDE for development. The focus of this thesis is towards cross-platform communication and automated code synthesis between Android and iOS and vice versa. However, Windows Phone, especially with Windows 10 Mobile, has great potential to share a higher marketshare and thus be supported for cross-platform ease-of-development in the future.

3.2 Communication Protocols

Understanding the various communication protocols offered with modern mobile devices is an essential piece of information to justify our choice of Bluetooth Low Energy for communicating across heterogeneous mobile devices. Figure 1.1 in Chapter 1 shows the different communication technologies or protocols supported by versions of iOS and Android devices. This section will give a brief overview of each of these protocols and explain their advantages and disadvantages for cross-platform communication.

3.2.1 Bluetooth 2.1

Also known as traditional Bluetooth, this protocol is found in almost every modern consumer level electronic device. In 2014, an estimated three billion devices [89] equipped with Bluetooth were shipped for consumer use.

Pros

1. Discovery of other devices in the near vicinity of 30ft
2. Ability to transfer medium sized files (1MB-10MB) fairly quickly
3. Established and widely used technology

Cons

1. Slow transfer rate (3 Mbit/s)

2. Hardware restrictions for Apple devices (Made for iPhone)
3. Drains battery quickly after prolonged use

Bluetooth 2.1 is an established, thoroughly tested and widely adopted technology. However, Apple/iOS devices have a restriction with the type of 3rd party devices that can connect to an iOS device. Apple's Made For iPhone/iPad/iPod (MFi) program [12] restricts only devices participating in this program to be able to connect to an Apple device. Android devices are not a part of this program and therefore are unable to connect or transfer data with an iOS device through Bluetooth 2.1.

3.2.2 WiFi Direct

Also known as Wi-Fi P2P allows devices to easily connect and transfer data with each other without the need for an external wireless access point. Android first introduced this technology with their Android 4.0 (Ice Cream Sandwich) operating system in 2011 [3].

Pros

1. High transfer rate (250 Mbps)
2. Ability to transfer large sized files (10MB-100MB) fairly quickly
3. Long range transfer up to 200 meters possible

Cons

1. Apple devices use their proprietary "Multipeer" connectivity for P2P data transfer
2. Group based connection and relationship. Once host leaves, all connections are dropped
3. New technology, so bugs are common

WiFi Direct was a very promising communication protocol for device-to-device communication because the protocol has a high transfer rate, high bandwidth, high security and long range transmission capabilities which were very favorable for our use cases and motivation behind this portion of thesis. However, the most important feature of supporting heterogeneous devices was not present because Apple uses their proprietary Multipeer connection protocol to establish peer-to-peer (iOS only) communication channels.

3.2.3 Multipeer Connectivity

Multipeer Connectivity, first introduced in iOS 7 in mid-2013 [61], allows iOS devices in close proximity to communicate and transfer data to one another via Bluetooth, WiFi or a combination of both. This technology works really well with devices in the Apple ecosystem, but fails to work with any non-Apple devices such as Android mobile phones.

Pros

1. High transfer rate (250 Mbps)
2. Very stable communication channel with fallbacks and robust error handling
3. Seamless transfer between Apple devices such as iPhones, iPads and Mac Computers

Cons

1. Proprietary technology, restricted to Apple devices

Just like WiFi Direct, Multipeer Connectivity provided several of the required features for our runtime but failed to support heterogenous capabilities since the technology is restricted to Apple devices.

3.2.4 NFC

Near Field Communication, NFC, was first introduced in Android's Gingerbread operating system in late-2010 [51]. NFC roots can be traced to RFID research done as early as 1983. Widely popular now with the introduction of Apple's contact-less payment system, Apple Pay, NFC on iOS devices is restricted to payment services only.

Pros

1. Implemented in contact-less payment systems
2. Secure communication
3. Designed with sharing files and important information between devices

Cons

1. Apple devices restricted NFC to Apple Pay only
2. Extremely short transfer range (<5 inches)
3. Low transfer bandwidth

Despite being restricted by Apple, NFC wasn't the best protocol for consideration because it featured a very low bandwidth and transfer range. Devices had to be in direct contact or in very close proximity of each other to get any sort of data to transfer. These features weren't ideal for the approach that we were seeking.

3.2.5 Bluetooth Low Energy

The holy grail of protocols for heterogeneous device-to-device communication was found in the Bluetooth Low Energy (Bluetooth 4.0) protocol. Also known as BLE, this protocol was first introduced to the original Bluetooth standard in 2010 [21]. Its primary target market is for low energy technologies, particularly in the health and fitness sector where transferring constant, high rate, low bandwidth data such as heart rate is very common.

Pros

1. High transfer rate achieved through short bursts of smaller data size
2. Small overhead
3. Ability for heterogeneous devices to establish a connection and transfer data between each other

Cons

1. Fixed data rate of 1Mbps
2. Short 20 byte packets per transmission
3. Not suitable for high data size transfers

Despite having a very small data bandwidth of 20 bytes, the one quality of being compatible with both iOS and Android devices gave BLE an advantage over any of the other discussed protocols. Even the 20 byte bandwidth limit was overcome by implementing an algorithm

(Figure A.3) that divides data into 20 byte chunks and transfers the chunks one after the other to a target device.

As seen in Figure 3.5, BLE implements a service-characteristic relationship for its core architecture. That is, a BLE-capable device is able to have multiple services that another device can search specifically for and each of those services can have one or more characteristics associated with it that contain the actual data values which can be read or written to by another device. For example, our runtime system implements a `Middleware Service` that the host device specifically searches for in order to find a compatible target device in its vicinity. The `Middleware Service` contains several readable and writable values or characteristics that other devices can access. Some characteristics provide a device-specific resource such as the GPS sensor information for other devices to be able to read from. Other characteristics provide a writable value that can be used by other devices as an input parameter to run a task on the target device. In other words, the host can write a value to a particular characteristic on a target device, and the target device performs an activity or task on that value. The result on the target device can then be written to a writable characteristic on the host device to provide the result for that task.

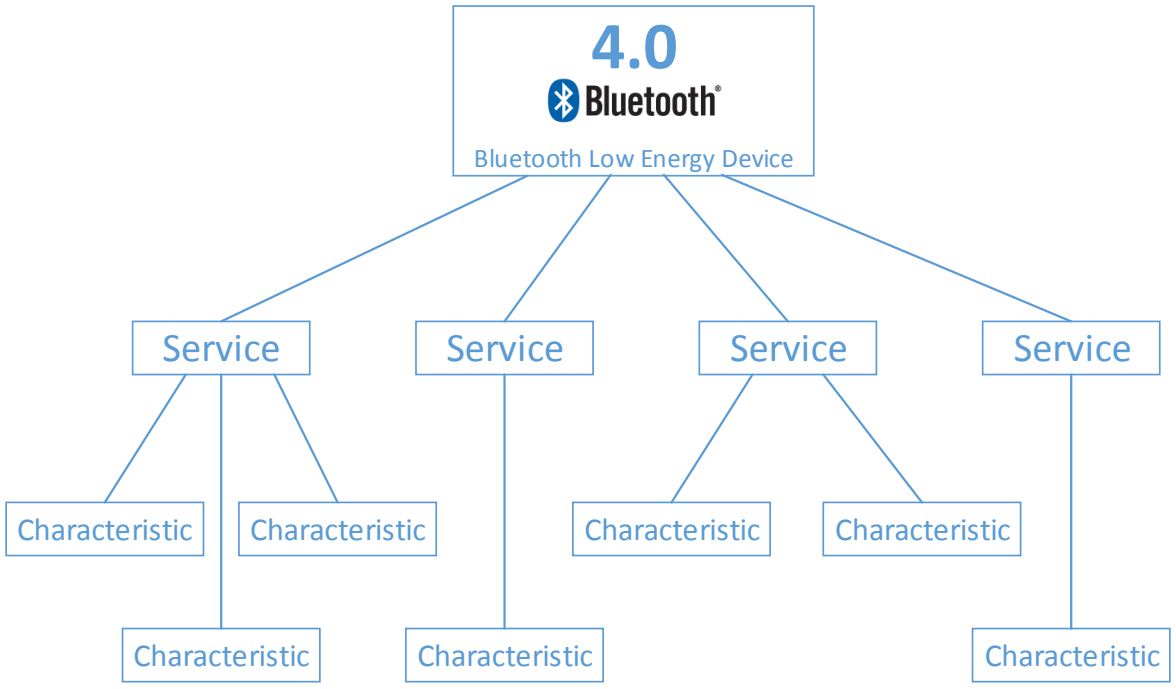

Figure 3.5: BLE Service-Characteristic Relationship

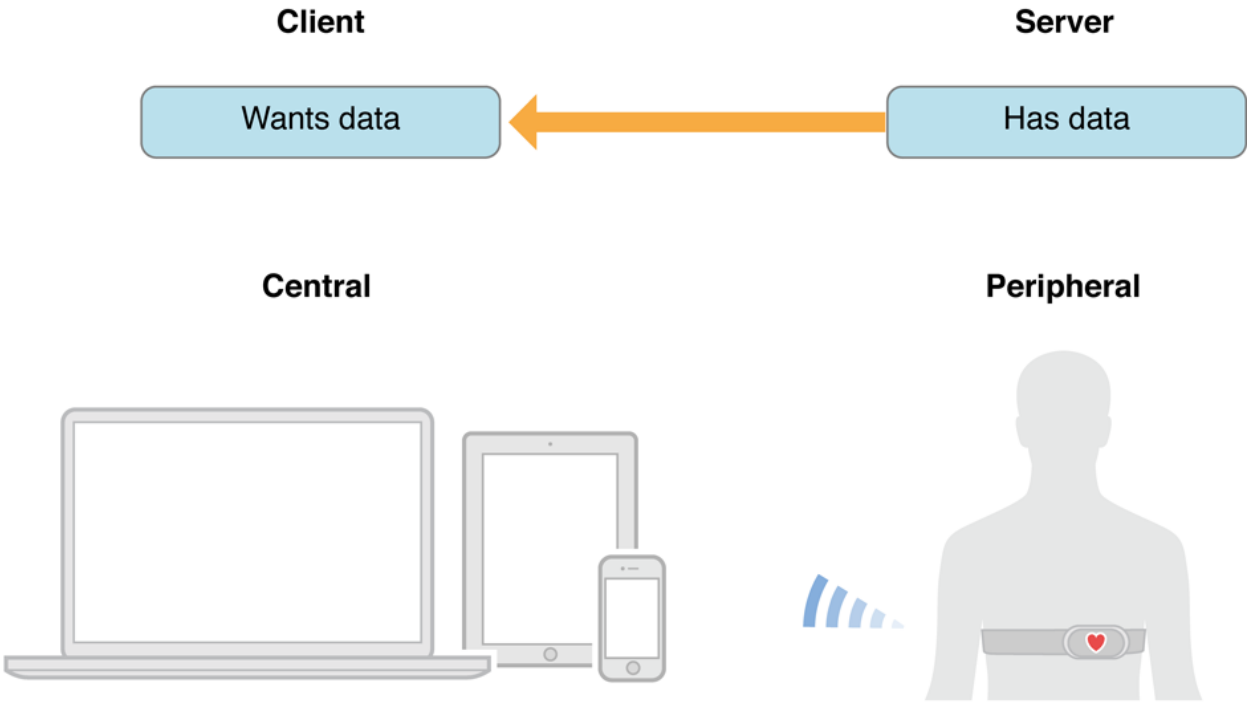

Figure 3.6: BLE Central-Peripheral Relationship. [5] — Apple. Core bluetooth overview. `https://developer.apple.com/library/ios/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/Art/CBDevices1_2x.png`, 2013. Used under fair use, 2015

3.2.6 Central-Peripheral Relationship

Figure 3.6 shows the roles of devices in the BLE architecture. In the traditional client-server model, the client is a device that wants data and a server is a host device that contains the data. Similarly in BLE, the client is known as the `Central` device and the host or server is known as the `Peripheral` device. The central can search for any peripherals that are running a particular service. Once a targeted peripheral is located, a connection is immediately established and the peripheral service's characteristics are revealed. The central device can act upon one of those characteristics based on the task at hand and data transfer is established. These roles can be applied to either iOS or Android devices and therefore makes this the best candidate for heterogeneous device-to-device communication.

3.2.7 Hardware Limitations

There are however some hardware limitations that exist on the Android platform that makes it difficult for both a central and peripheral role to be played by any device. Both central

and peripheral modes are implemented by default in all iOS devices and in Android 5.0 or better devices. However the peripheral or host mode is only supported in the Google Nexus 6 and 9 devices. This limitation constrained our implementation to only treat iOS devices as peripherals and Android devices as centrals when connecting heterogeneous devices. As a result of this limitation, our runtime system supported iOS to iOS communication using BLE and Android to Android communication using Bluetooth 2.1.

Chapter 4

Related Works

This chapter showcases some of the related works that have inspired components in both our programming model with runtime support for heterogeneous mobile device communication and recommendation system. We will discuss how our approaches differs from or improves over these examples.

4.1 Mobile Device Cooperation

The idea of using nearby devices to cooperatively achieve new functionality started from [66], yet the original idea was private data exchange over devices for data sharing and data mining. Along this direction, [67, 54, 81] continued to study how to use the mobility of device users to pass messages.

Besides data sharing, another application category explored the potential of using map reduce on distributed mobile devices to accomplish computationally intensive tasks [74, 32, 96]. However, in their system model, the mobility of devices and the willingness of device owners were not considered.

Recently, to take advantage of device cooperation these three main approaches have been explored:

1. **context-related data sharing:** [100] used microphones of nearby devices to obtain directive audio, while [27] used cameras of nearby devices to obtain video from all observation angles. [69, 71] used various sensors like the microphone and Bluetooth to calculate device owners' interaction with their connections, and their algorithms requires data exchange between nearby phones. [102] used Bluetooth to achieve location related authentication via other devices. [90] enables phone owners to play games interactively using the orientation sensor embedded in their phones.

2. **computational power sharing:** [74, 32, 96] offload computationally intensive tasks to nearby devices. [70] requires all nearby phones to accomplish computationally intensive OCR tasks, so that the latency of the identifying queue could be shortened.

3. **service sharing w.r.t. network services:** [59, 16, 48, 58, 62] considers cooperative videostreaming and content fetching by nearby devices. Cong Shi, [95], with CoAST shows that the cooperative network usage scheduling and prefetching among applications mounted on nearby devices could largely improve the application's user experience. Dimatteo et al. [31] explored how adding very few WiFi access points(AP) and enabling cooperative downloads can greatly improve the overall network throughput capacity. [113] enabled 3G data plan sharing by proposing an incentive strategy and [13] treats phonecalls as a service provided by nearby devices.

This cooperation is not limited to mobile devices only, but can also involve IoT (Internet of Things) tags. Kiryong Ha et al. [46] first proposed offloading tasks to nearby devices (not mobile devices, like a Road Side Unit). [17] uses other nearby devices (like Estimote) to store and forward user messages.

4.1.1 Resource Sharing Methods

Concerning peer-to-peer resource sharing, traditional middleware approaches have been long proposed for infrastructured network. Examples include Open CORBA[72], Globe[107] and JXTA[41]. However, these cooperative middleware approaches disregard the mobility of devices.

An early example of peer-to-peer sharing with consideration for device mobility offers a specialized engine to share content [82]. Other software engineering approaches to enabling mobile device cooperation [2, 60, 64, 92, 78, 79, 33, 23] all provide middleware for sharing resources across devices, with the limitations of being tailored to a single platform and requiring system level modification. Other middleware approaches focus on specific functionality, including face-to-face interactions [94] and cooperative display [18].

Extending the notion of cooperating with devices in one-hop range, MANET further considers using devices through multi-hop wireless communications [26]. LIME[77], TOTA[73], Limone[37], CAST[91], MESHmdl[50], Preom [65], MobiPeer [20], Peer2Me [111], Steam [75], Transhumance [83], QAM [40], MobiCross [28] are all well accepted middleware approaches concentrating on different aspects. [84] attacked the resource sharing problem by providing a platform-independent query language, which however relies on the presence of an HTTP server thus introducing unnecessary latency and overhead.

By contrast, the vision behind our approach is to equip the mobile developer with a high level programming abstraction to help conquer the challenges of heterogeneous device-to-device communication. Unlike the middleware systems above, which provide programming

interfaces to control network topology, network traffic, and peer management, our approach provides a declarative programming interface, with the runtime system hiding much of the complexity of distributed communication and ensuring the quality of service.

4.2 Recommendation Systems

The source code recommendation system is related to those works that synthesize code snippets from web-based programming resources [88, 87, 86, 114] or build an intelligent code search engine [63].

Prompter [88] automatically identifies relevant StackOverflow discussions from a working code context. By providing a user-controlled confidence threshold, it suggests only those discussions that surpass this threshold. Prompter uses the StackOverflow API as the only source for relevant discussion and code snippets. Our approach's larger search space [108] includes Google Search, Google Code, and third-party resources to achieve the level of precision required for cross-language and platform translation.

Some related approaches make use of statically cached programming resources to accelerate data retrieval. For example, the approaches presented in [88, 87] rank output code blocks by normalizing a sigmoid function of the average StackOverflow vote count from a June 2013 static data dump. Selene [101] recommends equivalent code blocks by searching a repository of 2 million example programs to provide usage examples for a given input code block. Sourcerer [15] is another code search engine for a large-scale code repository (SourceForge). Strathcona [52], similarly to the systems above, also uses a repository based search corpus and provides the user with a structural overview of relevant code rather than actual code examples and discussions. A recommendation system developed by Bacchelli et al. [14] utilizes a vector space model with *tf-idf* as its frequency weighting model along with a singular query corpus source of StackOverflow. Similarly to other systems, Seahawk [86] also uses a static and publicly available dump of StackOverflow questions and lacks support for Swift.

StackOverflow receives close to 6000 new questions a day, so a static data snapshot from 2 years ago may be sufficient to mine for information about established language ecosystems and environments. However, the focus of this thesis is mobile computing with rapidly evolving programming environments and language ecosystems. By combining vector space and linear models on live StackOverflow data, our approach can recommend code that is relevant, up-to-date, and geared toward newer languages, such as Swift. Seahawk motivates the necessity of using the static dump to be able to search by means of the *Apache Solr* search system to achieve high performance efficiency. Although it would be unrealistic trying to match the performance efficiency of searching against a static local snapshot, we posit that carefully calibrating weights and feature sets for the *tf-idf* analysis and vector space model not only provide complete and relevant results for both StackOverflow posts and other code sources,

but can also yield performance levels sufficient for practical use. Lastly, Kim et al. [63] focus on providing useful documentation that supersedes standard API usage documentation. Although an important aspect of the developer's ability to create mobile applications can at times include understanding necessary documentation, our approach searches for working code rather its supplemental documentation.

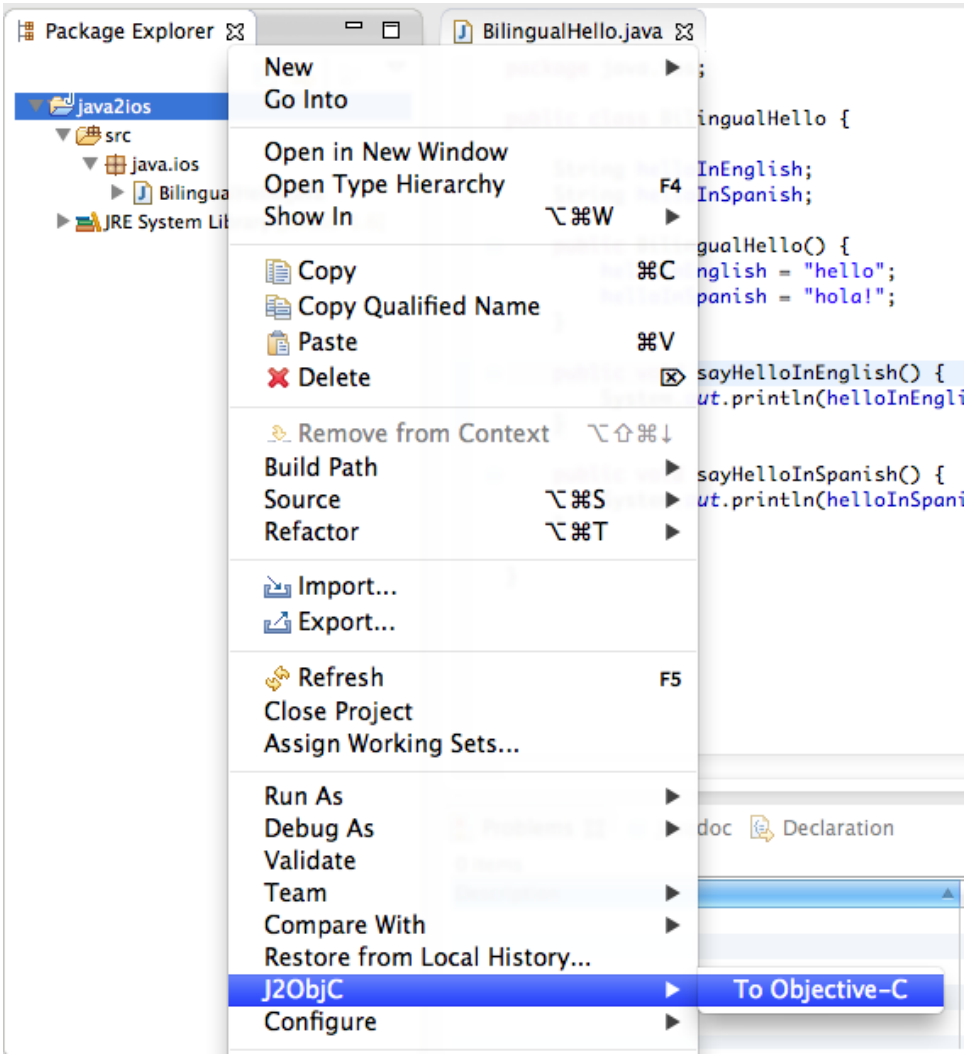

Figure 4.1: J2ObjC use case. [93] — H. Sapkota. j2objc-eclipse-plugin. `http://hemantasapkota.github.io/images/20-img-002.png`, 2015. Used under fair use, 2015

4.2.1 Transpilation

Direct source-to-source compilation is known as transpilation. Transpilation tools have not been developed for adolescent languages such as Swift. Google's J2ObjC [57] is the closest

contender that attempts to translate Java code into Objective-C (a language which was widely used by Apple to develop iOS applications before adopting Swift). Even though J2ObjC is a very powerful and unique transpilation tool, it lacks in several areas where a recommendation system such as ours would shine. J2ObjC lacks support for Android APIs and only focuses on transpiling the model portion of the MVC (Model-View-Controller) architecture. And since the tool strictly transpiles between Java and Objective-C, it lacks the versatility and elasticity provided by our approach which allows any mobile platform language to be translated into another mobile platform language natively. Figure 4.1 shows a sample use of the J2ObjC plugin in the Eclipse IDE [34].

Chapter 5

Supporting Heterogeneous Device Communication

When engineering heterogeneous distributed systems, two primary architectural paradigms are commonly considered: (1) those based on the Remote Procedure Call (RPC) abstraction [19], and (2) those structured around the RepresEntational State Transfer (REST) architecture [36]. The foremost example of applying RPC to heterogeneous distributed computing is the Common Request Broker Architecture (CORBA) [80]. Nevertheless, CORBA has come to under considerable criticism [110, 109] as a programming abstraction that promotes ill-conceived software design and implementation policies, poorly fit for realistic distributed applications. In addition, a typical CORBA software infrastructure is quite heavy weight with substantial runtime libraries that leave a heavy footprint on the memory and energy budgets of the underlying device. Because of these reasons, we decided against pursuing a CORBA-based solution. Rather, our solution is built upon the lessons of arguably the most successful distributed application to date, the World Wide Web, which is based on a RESTful architecture. This thesis explores a RESTful solution to the distributed heterogeneity challenge of the communication between dissimilar mobile devices.

More specifically, we support heterogeneous device communication by means of:

1. a domain-specific language for the mobile developer to implement the communication logic required to support distributed mobile applications across heterogeneous devices
2. a runtime architecture to support the efficient and robust execution of the aforementioned language in the presence of network volatility, device mobility, and energy constraints.

Next, we explain our key design ideas by presenting our domain-specific paradigm, Resource Query Language (RQL).

5.1 Resource Query Language (RQL) Design

RQL is a platform-independent domain-specific language for expressing the communication logic required for heterogeneous mobile devices to interact with each other. The language and its runtime system are based on the RESTful architecture [36], which is known to solve many of the toughest challenges that arise when engineering robust heterogeneous distributed systems. Hence, RQL follows the verb/noun paradigm: verbs express the actions to perform; nouns express the remote target resources upon which the actions are performed.

5.1.1 Example

As a specific example, consider the RQL command, `bind any:game/spriteRatio`, with the verb being `bind` and the noun being `any:game/spriteRatio`. By issuing this command, any source mobile device (on any platform) will register with any target device that is running an application that exposes a functionality identified with the key of `game/spriteRatio`, in which `game` is the type of application and `spriteRatio` the type of data that the application produces. Similarly to the notion of the URL in WWW, nouns in RQL simply identify remotely accessible resources, without constraining the language, the operating system, or even the functionality performed when a client accesses the identified resource. The verbs in RQL will be uniquely tailored toward the realities of mobile communication, with `bind` implying a persistent connection with expected periodic updates, typical for scenarios that come up in distributed mobile games. As per the REST design philosophy, RQL features a limited number of verbs applicable to an infinite number of nouns, with the combination determining the performed functionality.

5.1.2 Verbs

The following verbs have been identified as being the most advantageous to support heterogeneous device-to-device communication:

1. `pull`—transfer the data identified by a given noun to this client device
2. `push`—transfer the data to the target device, as identified by a given noun
3. `delegate`—instruct a remote device to perform some work on your behalf and await the result upon completion
4. `bind`—register to a specified remote channel to receive updates
5. `update`—change a variable value and notify a specific target device or group of devices about the change

Figure 5.1: Defined RQL Verbs [68] — A. Kunwar. Apple launches iphone 6.`http://3.bp.blogspot.com/-kvm1QSUF2CQ/VBBLrVioyiI/AAAAAAAAAlo/rFRkJYnxjTY/s1600/iphone.png`, 2015. [47] — Handy. Samsung galaxy s5. `http://www.handy.bg/web/files/products/201403/1013/3303.png`, 2015. Used under fair use, 2015.

5.1.3 Detailed Language Design

RESTful architectures favor limiting the number of available verbs, and our preliminary work has convinced us of the wisdom of this design principle. With only two verbs, we were able to create fully functional interactive demos. Hence, we decided to fully support only the three other verbs (i.e., `push`, `delegate`, `update`) bringing their total number to five. Only if we discover a truly compelling new recurring scenario requiring an additional verb would we consider growing the language. Hence, this architecture will again explore the veracity of the design principle applying a limited number of verbs to an infinite number of nouns in an emerging application domain.

```
1 <RQL command> := <RQL command>
2 <RQL command> := <verb> <noun>
3 <verb> := pull | push | delegate | bind | update
4 <noun> := [<device description>] : <task description> / [<task parameter>]+
5 <device description> := [<device owner>] | [<device type>]
6 <device owner> := my | group | [a-zA-z0-9]+
7 <device type> := iPhone | iPad | Android | Android_Tablet
8 <task description> := game | file | algorithm | sensor | service
```

Figure 5.2: The preliminary BNF definition of RQL.

Figure 5.2 illustrates a BNF definition of a RQL command. As discussed before, an RQL command consists of a verb and a noun. Currently there are 5 supported verbs. An RQL noun consists of the target device's description, the task's description and the parameters associated with that task. The device's description can consist of either the target device's name or the type of target device to connect to (i.e. iPhone, Android etc.). The task description can be a variety of different tasks, but the most commonly supported tasks are gaming-based tasks, file transfer and algorithm offloading based tasks.

5.1.4 Architecture

The RQL architecture is depicted in Figure 5.3, where the mobile developer is expected to produce the logic for both the source and the target portions of a distributed mobile application. RQL is specific to communication and does not offer any support for developing regular functionality. However, it is worth noting that this is the stated objective of the first research activity that this thesis puts forward. To understand the productivity benefits of RQL, one can see that to implement the communication logic between any pair of mobile devices, the programmer only has to write a string query, albeit submitted to the RQL runtime by means of a platform-specific API (similar to the relationship between SQL and JDBC for Java). The RQL runtime will negotiate a common low-level communication protocol (in our case, Bluetooth LE) across a given pair of devices, perform the required handshaking procedure,

and maintain the communication channel until the command at hand finishes its execution. The runtime is capable of returning error messages to the main application where it is up to the developer to handle the errors and display the proper messages to the end-user.

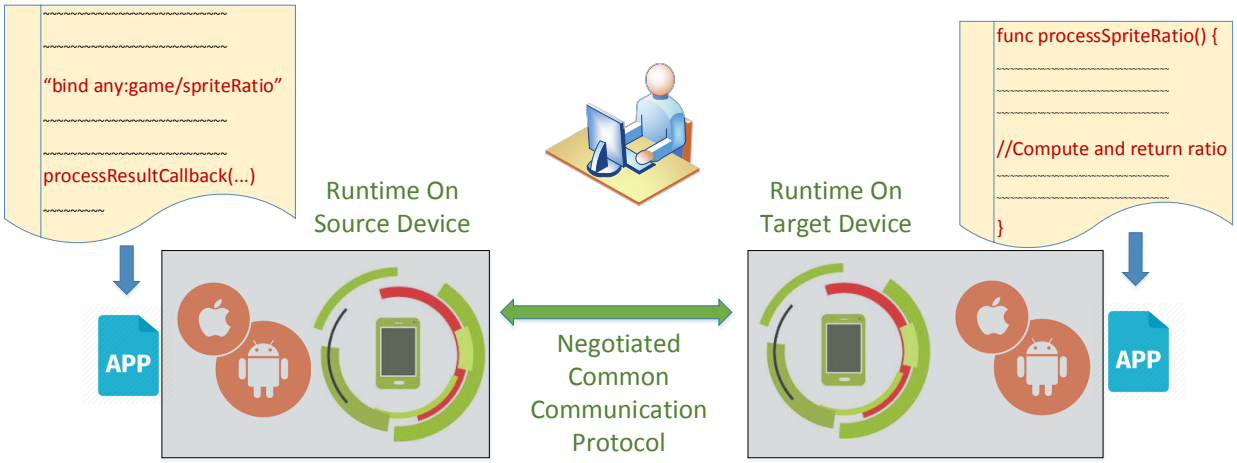

Figure 5.3: The Proposed Architecture

5.2 Runtime Support

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
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.

5.2.1 Android Implementation

On Android, the RQL runtime executes as a background service. Android applications communicate with the runtime by establishing an Android Interface Definition Language (AIDL)

[42] connection, a standard Android mechanism for inter-application/service communication. Our Android API provides methods for sending RQL requests over the established AIDL connection. Upon receiving an RQL request, the runtime immediately returns a unique identifier for that request. The application can then use this identifier to locate the request's results once it has been carried out. The runtime is responsible for several functionalities, including parsing the RQL commands, determining which device should be the target for a given command, controlling the communication with other devices, and receiving the results from target devices. The returned results are made available to mobile applications via a broadcast-based callback mechanism. The unique identifiers must be discarded once the results of the RQL requests associated with them have been received.

In some rare cases, the programming scenario at hand may require that the results of an RQL command be received prior to executing any subsequent program statements. In other words, the RQL command needs to be executed in a blocking fashion. To enable this blocking behavior, albeit ill-advised for performance and fault-tolerance reasons, the programmers can simply add the switch `--blocking` to any RQL command. The runtime processes this directive by finishing the specified command first and returning the results back to the caller.

One peculiarity of BLE communication is that each device can serve either a peripheral or central role, roughly corresponding to the traditional server/client functionalities, respectively. In other words, the peripheral role entails advertising services, each having potentially multiple characteristics, while the central role entails locating or accessing the services of the devices playing the peripheral role. This clear role separation is currently only supported by iOS devices and Android devices with the latest OS distribution (Android 5.0 or above).

To accommodate these distinct roles, the RQL runtime enables mobile devices communicating via BLE to seamlessly process and advertise resources. The runtime keeps track of the available services by means of unique ids (UUIDs). Since the UUIDs are universal and guaranteed to be globally unique by the BLE standard, the RQL runtime can easily keep track of the available services and their associated characteristics. These characteristics can have read, write, and notify properties. Reading allows a central device to read a value, writing allows it to write a new value into the characteristic, and notify allows a central device to subscribe to the target characteristic's value, so any changes to it will cause a notification update on the central device.

The RQL runtime currently provides five peripheral device's characteristics: GPS, accelerometer, file writing, HTTP request, and gaming support. As a means of getting updated values, it provides a special-purpose, subscribable result characteristic. A high level overview of the runtime process in a resource sharing scenario with an Android device being the central and an iOS device being the peripheral would look something like this:

1. The RQL runtime of an Android device first scans for nearby devices, and then scans for the services they are advertising

2. The source device runtime then establishes a connection with the target device utilizing

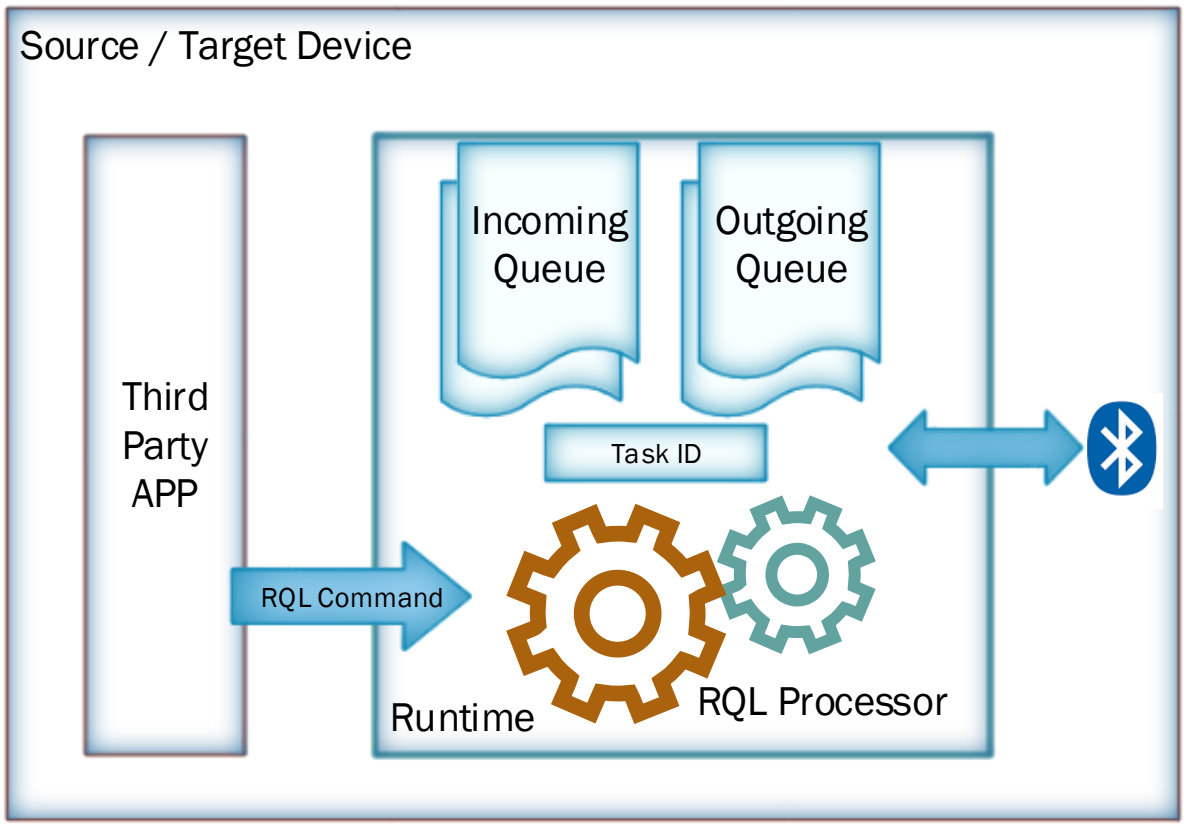

Figure 5.4: Runtime Support

the BLE communication protocol

3. If the RQL verb is "pull", the Android device directly reads a resource or value from the corresponding BLE characteristic on the iOS side

4. If the RQL verb is "push", the runtime sends a resource or file in chunks of 20 bytes (the size limitation of BLE per packet) to the file writing characteristic on the iOS side

5. Otherwise, the RQL verb "delegate" will write a parameter or command to the target device runtime and then receive the results back from the result characteristic once that command has been executed on the target iOS device

The runtime of the central device keeps track of the status of surrounding devices and undergoing tasks. Once the runtime receives a RQL request, it enqueues that request into a task queue and returns a task-id immediately. Meanwhile, the device manager periodically scans the Bluetooth advertisements of surrounding devices to discover the provided services.

When a task is popped from the task queue, the runtime parses the RQL command to decide on which peripheral device's characteristic it should query or write.

The runtime of the peripheral device queues up all received requests. Each item contains the id of that request and the RQL command associated with it. When a RQL request is received it is added to the end of the queue. When the runtime wants to process a request, it removes a task from the head of the queue, ensures that the task is unexpired, and executes it using the designated service.

Figure 5.4 provides a high level overview of the runtime architecture and how the architectural features are identical across both the source and target devices regardless of the mobile software platform that they are running. Figure 5.4 also shows how the runtime will maintain two queues to keep track of the outgoing and incoming tasks for the source and destination devices, respectively.

5.2.2 iOS Implementation

As discussed before, the runtime architecture is designed in the same way regardless of the underlying mobile platform. In our iOS devices, the runtime contains the same RQL features and characteristics that the Android implementation has, however the only difference is exactly how an application running on an iOS device communicates with our middleware that is designed for that platform. Swift and iOS allow the use of the singleton pattern [38] to be used in place of background services since the functionality of creating custom services [9] is not yet supported by iOS. The singleton pattern ensures that only one instance exists for a given class and that a global access point to that instance exists for other classes to use. This idea is identical to that of a service, except that this class is included directly in the iOS source code for a particular application, whereas the service source code for Android is separate from the main application.

Using our runtime architecture as a singleton pattern, an iOS developer can freely and simply add a cross-platform communication feature to their application and supply RQL commands to carry out any sort of resource sharing between two or more devices regardless of their underlying mobile platform. In other words, Android applications communicate with our Android middleware through the AIDL and that middleware establishes a connection via BLE to any device in its vicinity. If the target device happens to be running iOS, then BLE will transfer the RQL command to an entry point in the iOS application's runtime singleton class and that class will then handle the parsing and execution of that command. Once executed, the result will then be sent back to the Android device via BLE and the Android middleware will trigger a broadcast-based callback to notify the Android application and user that the task has been accomplished with the attached result.

5.3 Pilot Programmability Study

To better understand the potential benefits of RQL to the mobile developer, we conducted a pilot user study. We recruited 10 Junior to Senior level CS students from an intermediate Android development class at Virginia Tech. The recruited students were divided into 2 groups: experimental and control, for novice and experienced Android developers, respectively. The experimental group comprised 6 students with no prior experience in Android programming, while the control group comprised of 4 students with several years of Android development experience.

In the beginning, we briefly introduced the main Android communication concepts, such as AIDL services, broadcast receivers, and Bluetooth Low Energy. Then, each group was given 90 minutes to write a distributed mobile application that obtains the GPS sensor reading from an iOS device to an Android device. The experimental group was asked to use RQL, while the control group was asked to use any existing, mainstream Android APIs. The control group was also given an Android BLE sample application [43] as an example from which to draw device-to-device coding idioms.

Table 5.1: Pilot Study Results

Group	Experimental	Control
Android Development Experience	Novice	Intermediate
Students Accomplishing the Task	3	0
Total Students	6	4

Table 5.1 presents the results of the study. To our surprise, none of the students in the control group were able to complete the task successfully, which demonstrates the non-trivial nature of device-to-device communication. The results of the experimental group, armed with RQL, were mixed, with 3 students successfully completing the task, with the remaining 3 giving up before the experiment concluded. Because the group using RQL comprised non-experienced Android programmers, the results above indicate that RQL streamlines the process of implementing device-to-device interactions and can become a pragmatic tool for future applications.

Chapter 6

Proof-of-Concept

This chapter discusses the interactive demos that were developed using a reduced set of verbs. The demos provided a proof-of-concept by displaying an animated object being seamlessly transmitted across cross-platform devices.

6.1 Interactive Demos

A simple RQL prototype implementing the `bind` and `push` verbs proved highly promising, as it enabled several interactive demos [25] and a pilot user productivity study. These interactive demonstrations illustrate a representative case of heterogeneous mobile devices combining their functionality to accomplish a common task.

In particular, we were able to create several multi-device animations, 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. Two equivalent game development toolkits for iOS [29] and Android [53], respectively, were used to create the animations for both platforms. Although one can implement these animations to "hop" between devices by writing low-level, platform-specific code, our demonstrations showcased how a couple of lines of RQL and its platform-specific bindings can support non-trivial communication functionality. Figure 6.1 shows one of the created demos, with a frog hopping across device screens. The host (peripheral) is an iOS-based device with the Android device (central) connecting to it.

One of our other demos was more interactive—the classic game of pong played across heterogeneous devices, as shown in Figure 6.2. The most difficult part of developing the pong demo was actually figuring out the logic of the game and calculating sprite collisions with the paddles and the screen edges. Implementing the touch gestures was also a challenge so that both the pong paddles would follow a user's finger exactly without acceleration. Using

our language and runtime was one of the most trivial parts of this process because all the communication protocol initialization, handshaking and data transfer was handled by the middleware runtime.

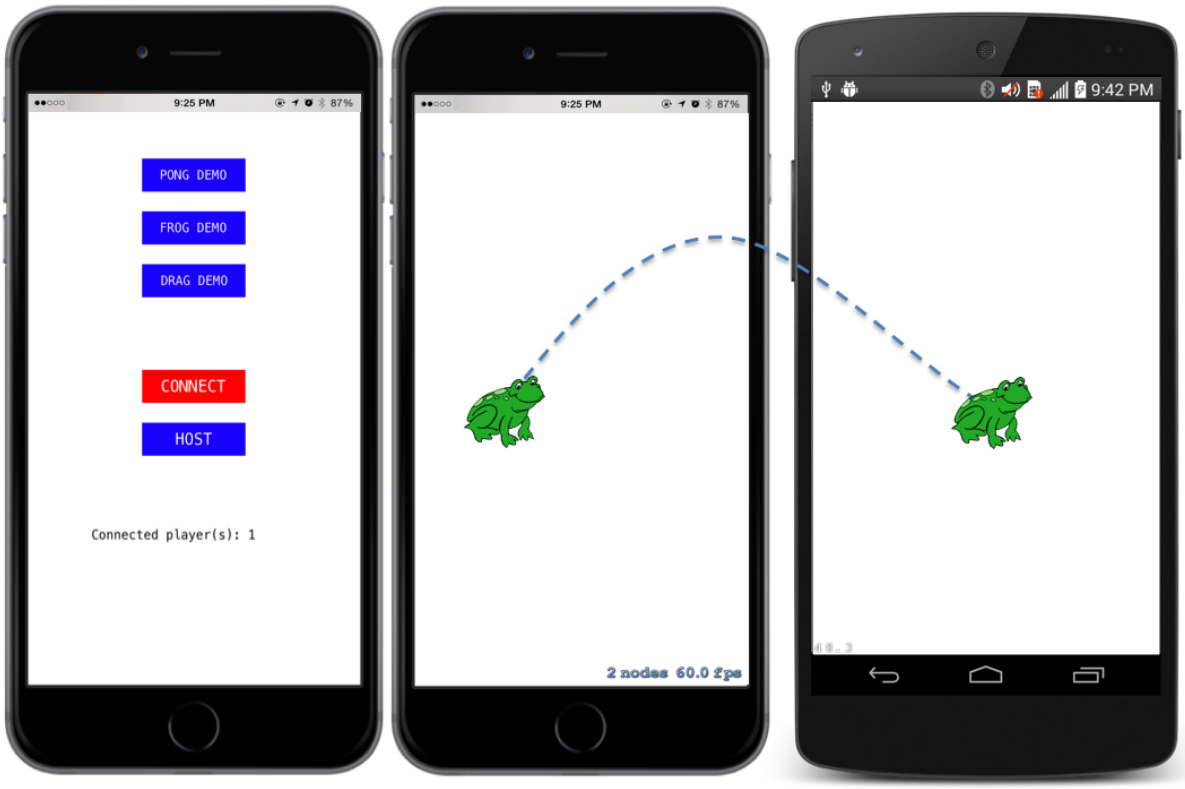

Figure 6.1: Hopping Demo - (L) Setup screen (C) iOS initial position [1] — N. Alderman. The walk game. `http://www.thewalkgame.com/images/5d6f4cc7.iphone-frame.png`, 2015. (R) sprite "hopped" to Android screen [76] — D. T. Milano. Android: Obtaining beautiful screenshots automatically. `http://3.bp.blogspot.com/-xAfNwhMROKg/VT1rhenkPDI/AAAAAAAAMvs/pKX5GS3ux6Q/s1600/0936964802203bc7-com_android_calculator2-com_android_calculator2_Calculator-2015-04-26T18%3A40%3A28.708173.png`, 2015. Used under fair use, 2015

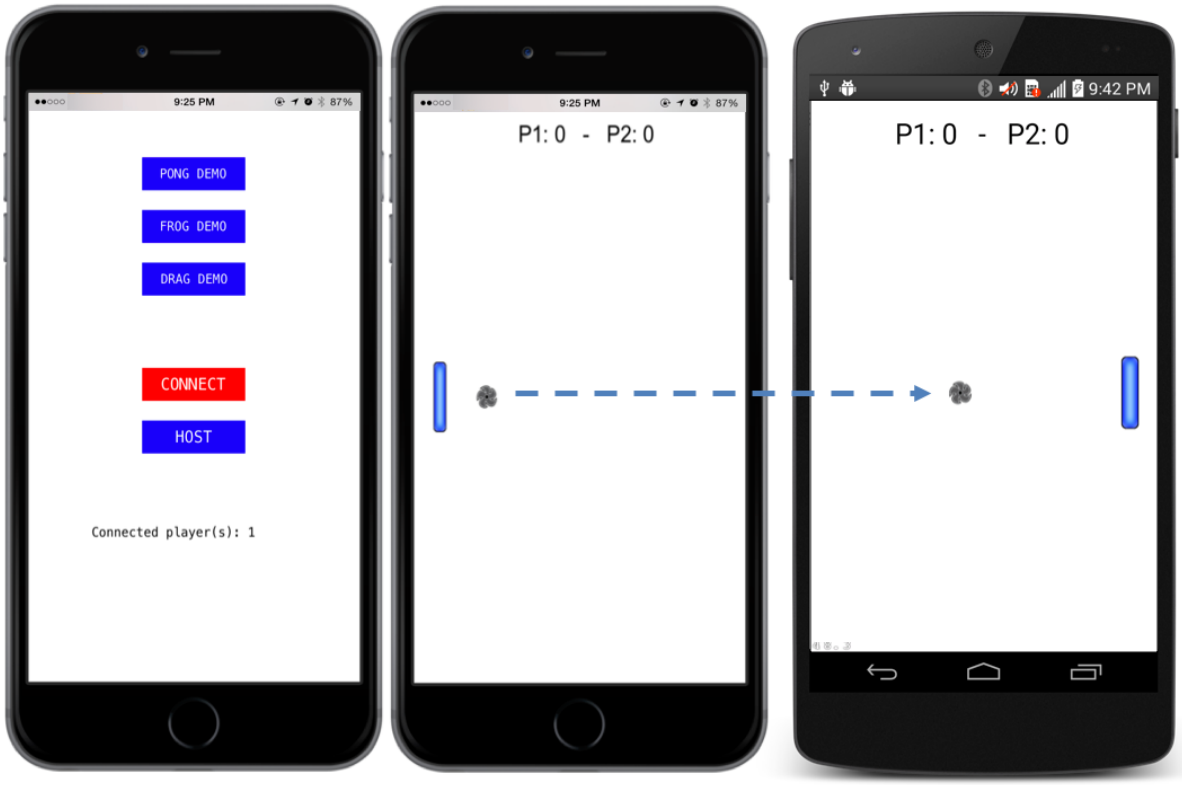

Figure 6.2: Pong Demo - (L) Setup screen (C) iOS initial position [1] — N. Alderman. The walk game. `http://www.thewalkgame.com/images/5d6f4cc7.iphone-frame.png`, 2015. (R) pong ball transferred to Android screen [76] — D. T. Milano. Android: Obtaining beautiful screenshots automatically. `http://3.bp.blogspot.com/-xAfNwhMROKg/VT1rhenkPDI/AAAAAAAAMvs/pKX5GS3ux6Q/s1600/0936964802203bc7-com_android_calculator2-com_android_calculator2_Calculator-2015-04-26T18%3A40%3A28.708173.png`, 2015. Used under fair use, 2015

Chapter 7

Supporting Heterogeneous Device Development

A cross-platform mobile application is supported on more than one mobile platform, with each replica essentially providing equivalent functionality. Currently, the overall development effort required by such cross-platform applications is $E_1+E_2+...+E_p$, where E_i is the required effort on a platform and p is the number of supported mobile codebases. The goal of this approach is to reduce the value of this sum by automating the effort expended on one of the constituent codebases.

To address the 3^{rd} central challenge presented in section 1.1.1, a fellow researcher and I developed a code recommendation system, `Native-2-Native`, that leverages the effort expended on the Android mobile platform to perform equivalent tasks on the iOS mobile platform [22]. Our approach, as presented in Figure 7.1, comprises of a code synthesis algorithm that discovers publicly available Swift code blocks whose semantics are equivalent to the code block written by the developer for the Android platform.

The approach starts with the programmer performing some development task on the Android mobile platform using Java. The resulting code block is used to search the web for the available target platform's (Swift) code for the same development task, so as to avoid replicating the effort expended on performing the development task on the source platform. The search results are filtered by means of a ranking algorithm that applies a high-dimensional feature vector to select the code block whose functionality is the closest to that of the source platform. The best-ranked code candidate is then presented to the mobile developer for inclusion into the target application.

This approach is quite one dimensional in that it only supports code recommendation, translation and synthesis strictly from Java to Swift. The purpose and objective of the following discussion in this thesis is to provide the reader my contributions of:

1. Implementing a bidirectional system that supports both Java → Swift and Swift → Java code translation
2. Providing results and discussing the refined evaluation process for both directions
3. Revealing new insights and patterns with the introduction of another code recommendation direction

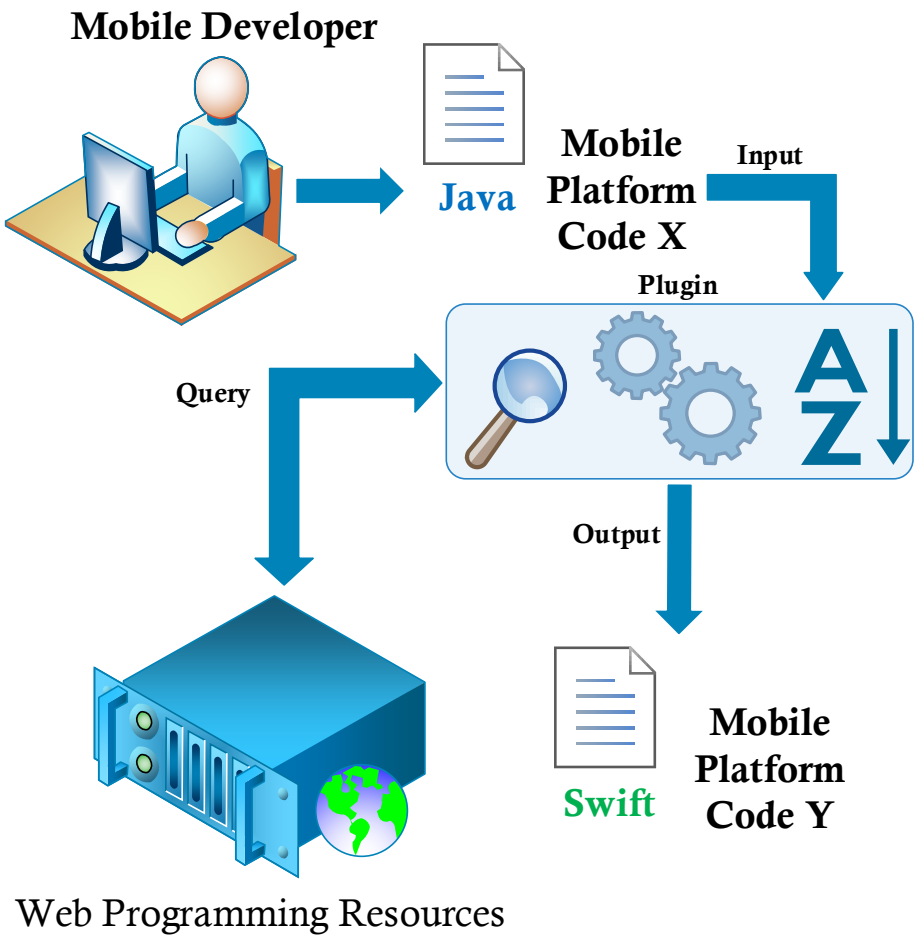

Figure 7.1: A Platform-Agnostic Source Code Recommendation System: Overview

7.1 Implementation

Figure 7.2 shows the program flow of the unidirectional approach where the programmer starts off with a source file written in Java for the Android mobile platform. The programmer selects a certain code block (including associated comments) from the source file for which he intends to get a synthesized Swift code block for his cross-platform iOS implementation that provides the same functionality. The selected code block is lexically tokenized with Java keywords such as `public`, `private`, `static`, and `final` filtered out. The tokenizing algorithm uses the tried and tested bag-of-words model which also allows `camelCase` or `TitleCase` variable and class names to be filtered along with any non-alphanumeric characters.

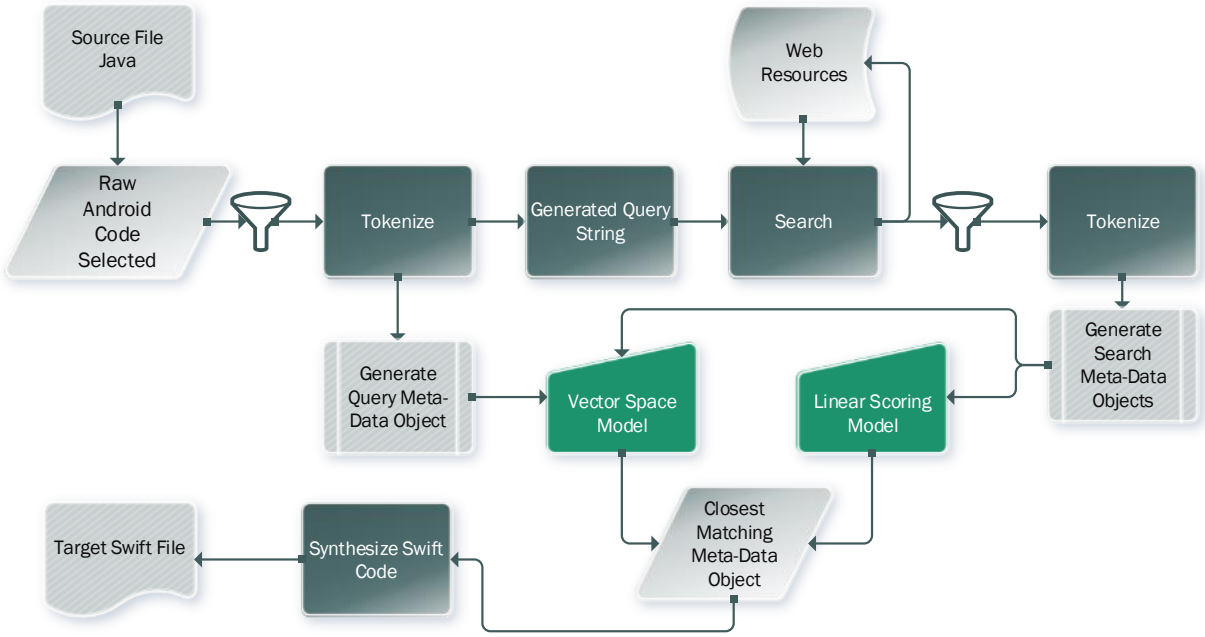

Figure 7.2: Program flow of unidirectional approach

Once these code blocks are tokenized, the resulting tokens are sorted by frequency from highest to lowest occurrence. The top 3 tokens from this sorted list are used as the query string for searching web-based resources such as Google and StackOverflow [97]. These tokens collected from the source document are also used to generate a query meta-data object that is used in a later process for comparison and ranking purposes. Meta-data is data that is used to describe other data. In our implementation, meta-data can be viewed as attributes for a particular result or the input query. When results from the web resources are returned, attributes such as vote count, view count, accepted answer and token frequency are applied and compared with the original source code block's metadata object. The degree of similarity between the query document and the result documents is determined by the ranking

algorithm which uses a combination of a linear scoring model and a vector space model. The vector space model requires the calculation of the cosine value of the angle between a potential target document and the source document. Each document is represented as a vector where the dimension of the vector is calculated using a weighting function known as term-frequency inverse-document-frequency (`tf-idf`). `Tf-idf` accounts for a term's frequency without over fitting by accounting for common terms across the document corpus. In other words, terms that are more unique and meaningful will be assigned a higher weight than terms that are more common and perhaps not as meaningful across the result term corpus. Once these weights for each of the results are calculated, they are sorted and the top two results are returned to the developer. The developer can then select one of the results to be synthesized into a usable Swift file for their iOS application.

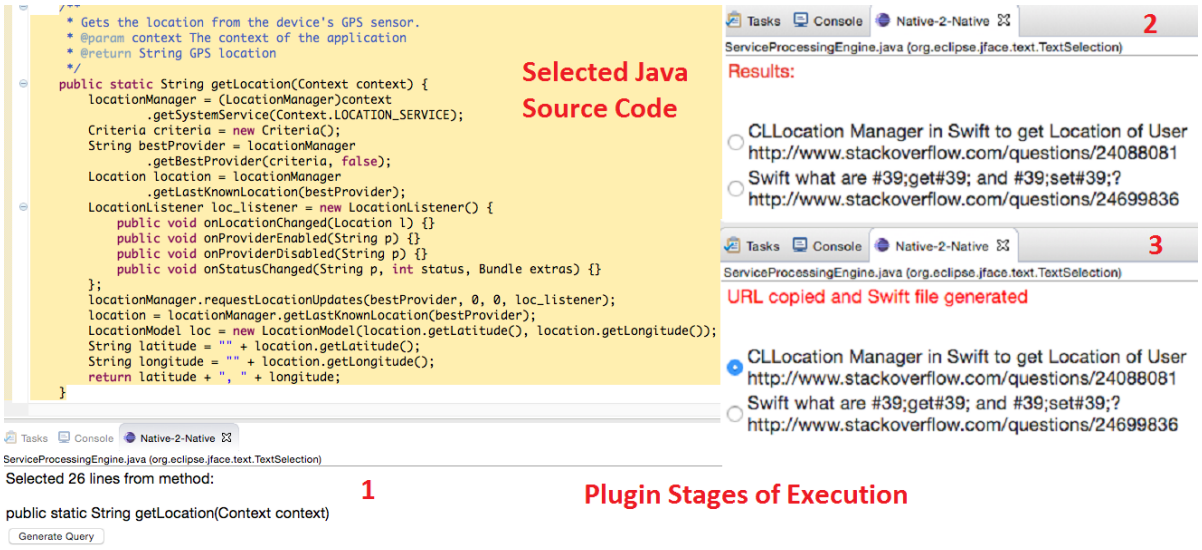

Figure 7.3: Eclipse Plug-in implementation of the code recommendation system

Figure 7.3 shows the approach as an implemented Eclipse IDE plug-in. The section labeled 1 in this figure represents the developer's view of the plug-in where they select a portion of Java source code in the code editor section of the IDE. The plug-in at the bottom portion of the screen automatically displays the method header of the selected method so the developer knows the code block has been verified as a valid code block. Afterwards, the developer will click the `Generate Query` button which will initialize the query generation, searching and ranking algorithms as discussed above. After the query has been processed, the top two results are displayed as radio button options to the developer. Selecting one of the results will copy the URL of that StackOverflow answer to the user's clipboard and will generate a Swift file in the working directory of the source code for the developer to use freely in their iOS version of the application.

7.2 Extension

Using the existing query extraction, searching and ranking algorithms of `Native-2-Native`, I was able to successfully implement a robust converse implementation that takes Swift/iOS source code as input and returns Java/Android code snippets from web resources. Several Swift keywords had to be filtered out and an automated code detection system needed to be implemented for the algorithm to detect what source code language was used as input. Thankfully, both Swift and Java have distinct method headers which allowed for a straightforward detection of whether the intended search query was to find Java snippets or to find Swift snippets from web resources.

Swift is a much more declarative or concise language than Java and that makes it slightly more difficult to generate relevant queries since there isn't as much code and context for frequency analysis to be an accurate query keyword generator. On the other hand, Java is a much more mature language and therefore has a lot more solutions, code snippets and support available on web resources such as StackOverflow. This suggests that despite not having the best possible keywords in the query, there is still a higher chance of encountering a relevant search result and relying on our robust ranking algorithm to display that result in the top 1 or 2 positions for the programmer to use. This proposition is proven and discussed in Chapter 8, where tabular results are shown with analysis on what the numbers signify.

While delving further into the code, some token filtration and minor feature-weighting bugs were identified and solved to give more confidence and validation in the results of the new evaluation that was conducted for both Java $\rightarrow$ Swift and Swift $\rightarrow$ Java. By implementing the bidirectional component, we pave the path for a more generalized code recommendation system which can be used for any combination of languages available.

Chapter 8

Evaluation

8.1 Evaluation Results

In the proof of concept, published at GPCE 2015 [22], we proved the feasibility of the approach by applying the discussed concepts to the problem of replicating native API-related functionality developed for Android using Java on iOS using Swift. Specifically,we applied our approach to various Android/Java native APIs to search for analogous functionality in iOS/Swift, and then examined the synthesized code blocks for their fitness in expressing the functionality at hand. We evaluated various APIs, including sensors (e.g., GPS, accelerometer etc.), network interfaces (e.g., WiFi, Bluetooth Low Energy (BLE), etc.), and canonical library classes/data structures (e.g., `String`, `ArrayList`, `HashMap`, etc.).

We evaluated all the synthesized functionality by hand, which included compiling the code with the Swift compiler and testing its runtime behavior. However, as discussed before, after fixing several bugs and gaining more confidence and trust in our search and ranking algorithm, we ran the evaluation again using a stricter measure of quality of returned results. Table 8.1 shows the new evaluation results of 66 code block experiments written in Java and how 74% of those results in Swift were relevant to the task at hand. For our quality

Table 8.1: Proof of concept - Android/Java → iOS/Swift

API Experiments	Total	Answer Exists	Pre-Rank 1	Pre-Rank 1 or 2	Rank 1	Rank 1 or 2
Services	6	2 (33%)	2 (33%)	2 (33%)	2 (33%)	2 (33%)
String	25	24 (96%)	23 (92%)	24 (96%)	21 (84 %)	23 (92%)
ArrayList	22	17 (77%)	10 (45%)	12 (55%)	13 (59%)	14 (64%)
HashMap	13	13 (100%)	9 (69%)	10 (77%)	8 (62%)	10 (77 %)
Totals	**66**	**56 (85%)**	**44 (67%)**	**48 (73%)**	**44 (67%)**	**49 (74%)**

evaluation, , we used a scale that placed these results into one of 2 categories: (1) Yes, the synthesized code is correct or salvageable for the implementation required or (2) No, the synthesized code is completely irrelevant to the input source code block and unsalvageable. This allowed for a more straightforward evaluation process especially considering that the output results were analyzed for relevancy by hand.

An extension of this proof of concept evaluation involved the reversal of the source and target programming languages. The results below represent the same experiments implemented in Swift and using our code recommendation system to synthesize relevant code blocks in Java. Table 8.2 shows how the evaluation yielded about 91% of Java code being found relevant for the subject applications' Android/Java versions.

Table 8.2: Proof of concept - iOS/Swift → Android/Java

API Experiments	Total	Answer Exists	Pre-Rank 1	Pre-Rank 1 or 2	Rank 1	Rank 1 or 2
Services	6	4 (67%)	2 (33%)	4 (67%)	2 (33%)	4 (67%)
String	25	24 (96%)	20 (80%)	23 (92%)	20 (80 %)	24 (96%)
ArrayList	22	21 (95%)	11 (50%)	15 (68%)	19 (86%)	21 (95%)
Dictionary	13	12 (92%)	7 (54%)	7 (54%)	8 (62%)	11 (85 %)
Totals	**66**	**61 (92%)**	**40 (61%)**	**49 (74%)**	**49 (74%)**	**60 (91%)**

8.2 Discussion

Tables 8.1 and 8.2 show the results of running 66 code block experiments written in both Java and Swift through our code recommendation system. There were 4 categories of API experiments–Services, String, ArrayList and HashMap/Dictionary. These API experiment categories represented the most commonly used mobile and language-based APIs for mobile developers. Our proof-of-concept demo for showcasing heterogeneous device communication contained several bluetooth low energy and GPS based API calls. Along with some service code blocks, various String, ArrayList and Hashmap/Dictionary methods were utilized. These examples can be found in Appendix A through D of this thesis.

The `Answer Exists` column represents the number of experiments that had a relevant answer in their results despite the search platform algorithm ranking (i.e. Google/StackOverflow rank)that they were displayed in. This creates a baseline of the number or percentage of experiments that actually have an answer on the web. This statistic would give us an idea of the accuracy of our query formulation since our ranking algorithm was not applied to the results at that point of time. The results show that there was a higher percentage of answers that existed for the Swift → Java direction since there are more Java results available in web resources.

The `Pre-Rank 1` and `Pre-Rank 1 or 2` columns represent the number of results that were in Rank 1 only or found either in Rank 1 or 2 before our ranking algorithm is applied. This statistic sets a baseline of how accurate the ranking algorithm for our search providers is. As shown in the tables, pre-rank results for both directions were fairly similar, indicating the search provider's ranking algorithm is performing consistently.

The last two columns, `Rank 1` and `Rank 1 or 2`, represent the number of results that were in Rank 1 only or in either Rank 1 or 2 after our ranking algorithm was applied to the same static set of results. This sets a baseline of how our ranking algorithm performs when compared to the state-of-the-art in ranking algorithms. The results indicate that by applying our ranking algorithm, an improvement of 1% when going from Java to Swift and of 17% when going the opposite direction is observed. These improvements are justified because our ranking algorithm takes into account more than just word or token relevance between the source and target documents. The meta-data object for a result contains weights associated with the number of votes or number of views that an answer from a web resource has. It also includes the weighting of whether an answer has been officially accepted or not. These features may carry a stronger influence than just pure token comparison and thus present an improvement in ranking quality.

These discussed results are very interesting because they show how a more mature and imperative (wordy) language such as Java is more likely to produce relevant code snippets from web resources rather than an emerging language such as Swift. In our evaluation for Native-2-Native in the GPCE'15 [22] paper, the percentage of relevant Swift code produced from a Java input was 84%. It is understandable to wonder why there is a 10% decrease in the more recent evaluation numbers when the experimental code blocks have been unchanged. There are several reasons for this decrease:

1. Our previous evaluation relevance percentage included results that were assigned a value of "maybe", which implied that the code snippet was partially relevant and may or may not provide the best solution for the source input and thus increased the relevancy percentage for the previous result.

2. Since the older evaluation, Swift has become more mature and there have been thousands of additional Swift [1] based questions answered. This means that there is a higher probability of extraneous or unrelated answers to be filtered into the results and thus cause a decrease in relevancy.

3. Having a higher confidence in the current, improved and fixed implementation of the searching and ranking algorithms has led us to believe that the previous results gave too much leeway to some answers and the current results are more refined, thus having a lower relevancy percentage.

[1] StackOverflow receives 2.65 new questions per minute and 4.41 new answers per minute on average, reaching close to 6,000 questions a day in peak sessions [98].

Chapter 9

Conclusions and Future Work

This thesis presented a two-pronged approach designed to aid mobile software developers seeking to create cross-platform mobile applications.

The first approach allows developers to overcome the challenges of heterogeneous device-to-device communication by providing them with a programming model and easy-to-integrate runtime support for applications designed for the Android or iOS mobile platform. By providing proof-of-concept demonstrations that utilize our programming model and runtime support, we show how a niche such as cross-platform mobile video game development can benefit from using our runtime to reduce programming effort required for implementing communication and instead focusing efforts on game design and visual quality.

The second approach addresses the challenge of reducing programmer effort for cross-platform mobile development by providing a bidirectional source code recommendation system. This recommendation system takes advantage of the plethora of web-based programming resources to provide developers with a synthesized code block based on the semantics extracted from an input code block. Since the approach is bidirectional, both Android and iOS developers can take advantage of this system to ease their effort in developing a cross platform mobile application. The evaluation showed that 74% of code block results from Java $\rightarrow$ Swift are relevant and that 91% of code block results from Swift $\rightarrow$ Java are relevant. These results prove that a more mature language will yield more relevant results since there are more resources dedicated to support a more popular language.

9.1 Future Work

For our runtime support, we plan to explore several avenues and use cases where resource or service sharing between heterogeneous devices would be useful. For example, if a user is in a foreign country without a valid data plan or service provider, he/she can use our runtime

system to locate a device with a local service provider (regardless of mobile platform) in their vicinity to securely send them a URL which the target device can then process as an HTML request and send the result back to the host for displaying the web page. Another similar scenario is if the mobile device user doesn't have GPS capabilities available on their phone, they can leverage the GPS sensor of a nearby device and use the target device's location as the host's own location. All of this would involve encrypted message transfers through the use of public and private keys and would also require the target devices to grant permission of the type of resources they are willing to share.

The source code recommendation system can be extended to become more generalized to support various combinations of programming languages to search and generate results for. Another potential avenue is for the system to have the ability to locate relevant examples from web resources that can help a mobile developer to remove or fix bottleneck inefficiencies introduced on either platform. The programmer may manually detect and select an inefficiency and use the same search and ranking algorithms to suggest examples that can potentially fix the bottleneck. A prime example can be the conversion of an inefficient synchronous (blocking) callback to a non-blocking asynchronous callback for a simple HTTP request.

Bibliography

[1] N. Alderman. The walk game. http://www.thewalkgame.com/images/5d6f4cc7.iphone-frame.png, 2015 (accessed November 26, 2015).

[2] A. Amiri Sani, K. Boos, M. H. Yun, and L. Zhong. Rio: a system solution for sharing i/o between mobile systems. In *MobiSys:Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 259–272. ACM, 2014.

[3] Android. Ice cream sandwich. http://developer.android.com/about/versions/android-4.0-highlights.html#UserFeatures, 2011 (accessed November 29, 2015).

[4] AppBrain. Number of available android applications. http://www.appbrain.com/stats/number-of-android-apps, 2015 (accessed November 26, 2015).

[5] Apple. Core bluetooth overview. https://developer.apple.com/library/ios/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/Art/CBDevices1_2x.png, 2013 (accessed November 28, 2015).

[6] Apple. Core os layer. https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/CoreOSLayer/CoreOSLayer.html, 2014 (accessed November 26, 2015).

[7] Apple. Core os layer. https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSTechnologies/iPhoneOSTechnologies.html, 2014 (accessed November 26, 2015).

[8] Apple. Platforms state of the union - wwdc session 102. https://developer.apple.com/videos/play/wwdc2015-102/, 2014 (accessed November 26, 2015).

[9] Apple. Background execution. https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/BackgroundExecution/BackgroundExecution.html, 2015 (accessed November 25, 2015).

[10] Apple. Apple app store review guidelines. https://developer.apple.com/app-store/review/guidelines/, 2015 (accessed November 26, 2015).

[11] Apple. Swift 2.0. `https://developer.apple.com/swift/blog/?id=29`, 2015 (accessed November 26, 2015).

[12] Apple. Mfi program. `https://developer.apple.com/programs/mfi/`, 2015 (accessed November 29, 2015).

[13] A. Aucinas and J. Crowcroft. Demo: Phonelets: offloading the phone off your phone for energy, cost and network load optimization. In *Proceedings of the 20th annual international conference on Mobile computing and networking*, pages 263–266. ACM, 2014.

[14] A. Bacchelli, L. Ponzanelli, and M. Lanza. Harnessing stack overflow for the IDE. In *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*, RSSE '12, pages 26–30. IEEE Press, 2012.

[15] S. Bajracharya, J. Ossher, and C. Lopes. Sourcerer: An internet-scale software repository. In *Proceedings of the 2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation*, SUITE '09, pages 1–4. IEEE Computer Society, 2009.

[16] X. Bao, Y. Lin, U. Lee, I. Rimac, and R. R. Choudhury. Dataspotting: Exploiting naturally clustered mobile devices to offload cellular traffic. In *INFOCOM, 2013 Proceedings IEEE*, pages 420–424. IEEE, 2013.

[17] F. Ben Abdesslem and A. Lindgren. Demo: mobile opportunistic system for experience sharing (moses) in indoor exhibitions. In *Proceedings of the 20th annual international conference on Mobile computing and networking*, pages 267–270. ACM, 2014.

[18] C. Berkhoff, S. F. Ochoa, J. A. Pino, J. Favela, J. Oliveira, and L. A. Guerrero. Clairvoyance: A framework to integrate shared displays and mobile computing devices. *Future Generation Computer Systems*, 34:190–200, 2014.

[19] A. D. Birrell and B. J. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, 1984.

[20] M. Bisignano, G. Di Modica, and O. Tomarchio. Jmobipeer: a middleware for mobile peer-to-peer computing in manets. In *Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on*, pages 785–791. IEEE, 2005.

[21] I. Bluetooth SIG. Bluetooth low energy. `http://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics/low-energy`, 2015 (accessed November 29, 2015).

[22] A. Byalik, S. Chadha, and E. Tilevich. Native-2-native: Automated cross-platform code synthesis from web-based programming resources. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*, GPCE 2015, pages 99–108, New York, NY, USA, 2015. ACM.

[23] M. Caporuscio, P.-G. Raverdy, and V. Issarny. ubisoap: A service-oriented middleware for ubiquitous networking. *Services Computing, IEEE Transactions on*, 5(1):86–98, 2012.

[24] A. Central. The history of android. `http://www.androidcentral.com/android-history`, 2015 (accessed November 26, 2015).

[25] S. Chadha, A. Byalik, and E. Tilevich. Heterogeneous device hopping: Bridging the mobile cross-platform gap via a declarative query language. In *Companion Proceedings of the 2015 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity*, SPLASH Companion 2015, pages 9–10, New York, NY, USA, 2015. ACM.

[26] E. da Silva and L. C. P. Albini. Middleware proposals for mobile ad hoc networks. *Journal of Network and Computer Applications*, 43:103–120, 2014.

[27] M. de Sá, D. A. Shamma, and E. F. Churchill. Live mobile collaboration for video production: design, guidelines, and requirements. *Personal and ubiquitous computing*, 18(3):693–707, 2014.

[28] M. K. Denko, E. Shakshuki, and H. Malik. A mobility-aware and cross-layer based middleware for mobile ad hoc networks. In *Advanced Information Networking and Applications, 2007. AINA'07. 21st International Conference on*, pages 474–481. IEEE, 2007.

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

[30] Developer.com. Submitting your windows phone 7 application to the windows marketplace. `http://solutions.developer.com/mobilephone/images/Figure4-VisualStudioIDE.png`, 2012 (accessed November 26, 2015).

[31] S. Dimatteo, P. Hui, B. Han, and V. O. Li. Cellular traffic offloading through wifi networks. In *MASS, 2011 IEEE 8th International Conference on Mobile Adhoc and Sensor Systems*, pages 192–201. IEEE, 2011.

[32] A. Dou, V. Kalogeraki, D. Gunopulos, T. Mielikainen, and V. H. Tuulos. Misco: a mapreduce framework for mobile systems. In *Proceedings of the 3rd international conference on pervasive technologies related to assistive environments*, page 32. ACM, 2010.

[33] D. J. Dubois, Y. Bando, K. Watanabe, and H. Holtzman. Shair: Extensible middleware for mobile peer-to-peer resource sharing. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 687–690. ACM, 2013.

[34] Eclipse. Mars eclipse. `https://eclipse.org`, 2015 (accessed November 24, 2015).

[35] J. Edstrom and E. Tilevich. Reusable and extensible fault tolerance for restful applications. In *Proceedings of the 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom '12)*, 2012.

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

[37] C.-L. Fok, G.-C. Roman, and G. Hackmann. A lightweight coordination middleware for mobile computing. In *Coordination Models and Languages*, pages 135–151. Springer, 2004.

[38] A. Freeman. The singleton pattern. In *Pro Design Patterns in Swift*, pages 113–136. Apress, 2015.

[39] C. Garling. iphone coding language now worlds third most popular. `http://www.wired.com/2012/07/apple-objective-c/`, 2012 (accessed November 26, 2015).

[40] A. Ghosh, S.-w. Li, C. J. Chiang, R. Chadha, K. Moeltner, S. Ali, Y. Kumar, and R. Bauer. Qos-aware adaptive middleware (qam) for tactical manet applications. In *MILITARY COMMUNICATIONS CONFERENCE, 2010-MILCOM 2010*, pages 178–183. IEEE, 2010.

[41] L. Gong. Jxta: A network programming environment. *Internet Computing, IEEE*, 5(3):88–95, 2001.

[42] Google. Introduction to the aidl interface. `http://developer.android.com/guide/components/aidl.html`, 2014 (accessed December 30, 2014).

[43] Google. Bluetoothlegatt sample. `http://developer.android.com/samples/BluetoothLeGatt/index.html`, 2015 (accessed November 25, 2015).

[44] Google. Android wear. `https://developer.android.com/wear/index.html`, 2015 (accessed November 26, 2015).

[45] Google. Glass. `https://developers.google.com/glass/`, 2015 (accessed November 26, 2015).

[46] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan. Towards wearable cognitive assistance. Technical report, DTIC Document, 2013.

[47] Handy. Samsung galaxy s5. `http://www.handy.bg/web/files/products/201403/1013/3303.png`, 2015 (accessed November 26, 2015).

[48] S. Hatakeyama, Y. Sakata, and H. Shigeno. Cooperative mobile live streaming considering neighbor reception. In *AINA, 2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, pages 65–72. IEEE, 2014.

[49] J. Herrman. What windows phone 7 could have been. `http://gizmodo.com/5480387/what-windows-phone-7-could-have-been`, 2010 (accessed November 26, 2015).

[50] K. Herrmann. Meshmd1-a middleware for self-organization in ad hoc networks. In *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, pages 446–451. IEEE, 2003.

[51] J. Hildenbrand. Gingerbread feature: Near field communication. `http://www.androidcentral.com/gingerbread-feature-near-field-communication`, 2010 (accessed November 29, 2015).

[52] R. Holmes and G. C. Murphy. Using structural context to recommend source code examples. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 117–125. ACM, 2005.

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

[54] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot. Pocket switched networks and human mobility in conference environments. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 244–251. ACM, 2005.

[55] IDC. Smartphone os marketshare. `http://www.idc.com/prodserv/smartphone-os-market-share.jsp`, 2015 (accessed November 26, 2015).

[56] IEEE. The 2015 top ten programming languages. `http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages`, 2015 (accessed November 26, 2015).

[57] J2ObjC.org. J2objc, 2015.

[58] I. Jang, D. Suh, and S. Pack. Minimizing content download time in mobile collaborative community. In *ICC, 2014 IEEE International Conference on Communications*, pages 2490–2495. IEEE, 2014.

[59] S. Jia, C. Xu, J. Guan, H. Zhang, and G. Muntean. A novel cooperative content fetching-based strategy to increase the quality of video delivery to mobile users in wireless networks. *Broadcasting, IEEE Transactions on*, 60(2):370–384, 2014.

[60] P. Jiang, J. Bigham, E. Bodanese, and E. Claudel. Publish/subscribe delay-tolerant message-oriented middleware for resilient communication. *Communications Magazine, IEEE*, 49(9):124–130, 2011.

[61] J. Kahn. Apple brings multipeer connectivity to mac, enables cross-platform nearby networking w/ ios. `http://9to5mac.com/2014/06/05/apple-brings-multipeer-connectivity-to-mac-enables-cross-platform-nearby-networkin` 2014 (accessed November 29, 2015).

[62] L. Keller, A. Le, B. Cici, H. Seferoglu, C. Fragouli, and A. Markopoulou. Microcast: cooperative video streaming on smartphones. In *MobiSys: Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 57–70. ACM, 2012.

[63] J. Kim, S. Lee, S.-w. Hwang, and S. Kim. Towards an intelligent code search engine. In *AAAI*, 2010.

[64] D. Koll, J. Li, and X. Fu. Soup: an online social network by the people, for the people. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 143–144. ACM, 2014.

[65] G. Kortuem. Proem: a middleware platform for mobile peer-to-peer computing. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):62–64, 2002.

[66] G. Kortuem, J. Schneider, D. Preuitt, T. G. C. Thompson, S. Fickas, and Z. Segall. When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad-hoc networks. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 75–91. IEEE, 2001.

[67] N. Kotilainen, M. Weber, M. Vapa, and J. Vuori. Mobile chedar-a peer-to-peer middleware for mobile devices. In *Pervasive Computing and Communications Workshops, 2005. PerCom 2005 Workshops. Third IEEE International Conference on*, pages 86–90. IEEE, 2005.

[68] A. Kunwar. Apple launches iphone 6. `http://3.bp.blogspot.com/-kvm1QSUF2CQ/VBBLrVioyiI/AAAAAAAAAlo/rFRkJYnxjTY/s1600/iphone.png`, 2015 (accessed November 26, 2015).

[69] Y. Lee, C. Min, C. Hwang, J. Lee, I. Hwang, Y. Ju, C. Yoo, M. Moon, U. Lee, and J. Song. Sociophone: Everyday face-to-face interaction monitoring platform using multi-phone sensor fusion. In *MobiSys: Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 375–388. ACM, 2013.

[70] Q. Li, Q. Han, X. Cheng, and L. Sun. Queuesense: Collaborative recognition of queuing on mobile phones. In *Sensing, Communication, and Networking (SECON), 2014 Eleventh Annual IEEE International Conference on*, pages 230–238, June 2014.

[71] C. Luo and M. C. Chan. Socialweaver: collaborative inference of human conversation networks using smartphones. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, page 20. ACM, 2013.

[72] C. Ma and J. Bacon. COBEA: A CORBA-based event architecture. In *Proceedings of the 4th conference on USENIX Conference on Object-Oriented Technologies and Systems-Volume 4*, pages 9–9. USENIX Association, 1998.

[73] M. Mamei, F. Zambonelli, and L. Leonardi. Tuples on the air: A middleware for context-aware computing in dynamic networks. In *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, pages 342–347. IEEE, 2003.

[74] E. E. Marinelli. Hyrax: cloud computing on mobile devices using mapreduce. Technical report, DTIC Document, 2009.

[75] R. Meier and V. Cahill. Steam: Event-based middleware for wireless ad hoc networks. In *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*, pages 639–644. IEEE, 2002.

[76] D. T. Milano. Android: Obtaining beautiful screenshots automatically. `http://3.bp.blogspot.com/-xAfNwhMROKg/VT1rhenkPDI/AAAAAAAAMvs/pKX5GS3ux6Q/s1600/0936964802203bc7-com_android_calculator2-com_android_calculator2_Calculator-2015-04-26T18%3A40%3A28.708173.png`, 2015 (accessed November 26, 2015).

[77] A. L. Murphy, G. P. Picco, and G.-C. Roman. Lime: A coordination model and middleware supporting mobility of hosts and agents. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 15(3):279–328, 2006.

[78] K. Nakao and Y. Nakamoto. Toward remote service invocation in android. In *UIC/ATC, 2012 9th International Conference on Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing*, pages 612–617. IEEE, 2012.

[79] A. Neyem, S. F. Ochoa, J. A. Pino, and R. D. Franco. A reusable structural design for mobile collaborative applications. *Journal of Systems and Software*, 85(3):511–524, 2012.

[80] Object Management Group. The CORBA component model specification. Specification, Object Management Group, 2006.

[81] J. Ott, E. Hyytia, P. Lassila, T. Vaegs, and J. Kangasharju. Floating content: Information sharing in urban areas. In *PerCom, 2011 IEEE International Conference on Pervasive Computing and Communications*, pages 136–146. IEEE, 2011.

[82] M. Papadopouli and H. Schulzrinne. Design and implementation of a peer-to-peer data dissemination and prefetching tool for mobile users. In *Proceedings of the first ny metro area networking workshop*, 2001.

[83] G. Paroux, L. Martin, J. Nowalczyk, and I. Demeure. Transhumance: A power sensitive middleware for data sharing on mobile ad hoc networks. In *7th international Workshop on Applications and Services in Wireless Networks (ASWN), Santander, Spain*, 2007.

[84] P. Plebani, C. Cappiello, M. Comuzzi, B. Pernici, and S. Yadav. Micromais: executing and orchestrating web services on constrained mobile devices. *Software: Practice and Experience*, 42(9):1075–1094, 2012.

[85] Z. Plesac. Android studio vs. eclipse. `https://s3.amazonaws.com/infinum.web.production/repository_items/files/000/000/168/original/android-studio-3.png?1393599622`, 2013 (accessed November 26, 2015).

[86] L. Ponzanelli, A. Bacchelli, and M. Lanza. Seahawk: Stack Overflow in the IDE. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1295–1298. IEEE, 2013.

[87] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza. Mining stackoverflow to turn the ide into a self-confident programming prompter. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 102–111. ACM, 2014.

[88] L. Ponzanelli, G. Bavota, M. D. Penta, R. Oliveto, and M. Lanza. Prompter: A self-confident recommender system. In *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution*, ICSME '14, pages 577–580. IEEE Computer Society, 2014.

[89] M. Powell. Bluetooth sig 2014 annual report. `https://www.bluetooth.org/en-us/Members/Annual-Report/2014-Annual-Report/default.aspx`, 2014 (accessed November 29, 2015).

[90] S. Pushp, C. H. Liu, F. Liu, and J. Song. Multi-player gaming in public transport crowd: Opportunities and challenges. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pages 331–336. IEEE, 2014.

[91] G.-C. Roman, R. Handorean, and R. Sen. Tuple space coordination across space and time. In *Coordination Models and Languages*, pages 266–280. Springer, 2006.

[92] A. Salem and T. Nadeem. Colphone: A smartphone is just a piece of the puzzle. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pages 263–266. ACM, 2014.

[93] H. Sapkota. j2objc-eclipse-plugin. `http://hemantasapkota.github.io/images/20-img-002.png`, 2015 (accessed November 26, 2015).

[94] G. Saucedo-Tejada, S. Mendoza, and D. Decouchant. F2fmi: A toolkit for facilitating face-to-face mobile interaction. *Expert Systems with Applications*, 40(15):6173–6184, 2013.

[95] C. Shi, K. Joshi, R. K. Panta, M. H. Ammar, and E. W. Zegura. Coast: collaborative application-aware scheduling of last-mile cellular traffic. In *MobiSys: Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 245–258. ACM, 2014.

[96] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura. Serendipity: enabling remote computing among intermittently connected mobile devices. In *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*, pages 145–154. ACM, 2012.

[97] StackOverflow.com. Stack overflow. `stackoverflow.com`, 2015.

[98] StackOverflow.com. Usage of /info - stack exchange api, 2015.

[99] Statista. Number of available apple app store applications. `http://www.statista.com/statistics/263795/number-of-available-apps-in-the-apple-app-store/`, 2015 (accessed November 26, 2015).

[100] S. Sur, T. Wei, and X. Zhang. Autodirective audio capturing through a synchronized smartphone array. In *MobiSys: Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 28–41. ACM, 2014.

[101] W. Takuya and H. Masuhara. A spontaneous code recommendation tool based on associative search. In *Proceedings of the 3rd International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation*, SUITE '11, pages 17–20, 2011.

[102] M. Talasila, R. Curtmola, and C. Borcea. Collaborative bluetooth-based location authentication on smart phones. *Pervasive and Mobile Computing*, 2014.

[103] Telegraph.co.uk. Apple ios: A brief history. `http://www.telegraph.co.uk/technology/apple/11068420/Apple-iOS-a-brief-history.html`, 2015 (accessed November 26, 2015).

[104] D. Trends. Digital trends - android screenshot. `http://icdn6.digitaltrends.com/image/android-vs-ios-windows-interface-1-640x1136.jpg`, 2015 (accessed November 26, 2015).

[105] D. Trends. Digital trends - ios screenshot. `http://icdn6.digitaltrends.com/image/android-vs-ios-windows-interface-3-640x1136.jpg`, 2015 (accessed November 26, 2015).

[106] D. Trends. Digital trends - windows phone screenshot. `http://icdn6.digitaltrends.com/image/android-vs-ios-windows-interface-2-640x1136.jpg`, 2015 (accessed November 26, 2015).

[107] M. Van Steen, P. Homburg, and A. S. Tanenbaum. Globe: A wide-area distributed system. *IEEE concurrency*, 7(1):70–78, 1999.

[108] B. Vasilescu, V. Filkov, and A. Serebrenik. Stackoverflow and github: Associations between software development and crowdsourced knowledge. In *Social Computing (SocialCom), 2013 International Conference on*, pages 188–195, Sept 2013.

[109] S. Vinoski. RPC under fire. *IEEE Internet Computing*, pages 93–95, 2005.

[110] S. Vinoski. Convenience over correctness. *IEEE Internet Computing*, pages 89–92, 2008.

[111] A. I. Wang, T. Bjornsgard, and K. Saxlund. Peer2me-rapid application framework for mobile peer-to-peer applications. In *Collaborative Technologies and Systems, 2007. CTS 2007. International Symposium on*, pages 379–388. IEEE, 2007.

[112] Windows. Windows store trends - september 2015. `https://blogs.windows.com/buildingapps/2015/10/12/windows-store-trends-september-2015/`, 2015 (accessed November 26, 2015).

[113] T. Yu, Z. Zhou, D. Zhang, X. Wang, Y. Liu, and S. Lu. Indapson: An incentive data plan sharing system based on self-organizing network. In *INFOCOM, 2014 Proceedings IEEE*, pages 1545–1553, April 2014.

[114] A. Zagalsky, O. Barzilay, and A. Yehudai. Example overflow: Using social media for code recommendation. In *Recommendation Systems for Software Engineering (RSSE), 2012 Third International Workshop on*, pages 38–42, June 2012.

Appendix A

Evaluation: Services

Appendix A through Appendix D show the code blocks used to evaluate our code recommendation system for the results displayed in Chapter 8. This appendix shows the Services subject cases representing the `Swift` input for our recommendation system. These services include examples of Bluetooth Low Energy, GPS and MultiPeer connectivity—representing some platform specific APIs that would be non-trivial to implement in another platform after already implementing them on the current platform. These examples represent actual use cases and code blocks used for the development of the runtime support for heterogeneous device communication.

```
/*
 *  Request a read on a given  bluetooth  LE  characteristic .
 */
func peripheral(peripheral: CBPeripheral, didDiscoverCharacteristicsForService
   service:CBService, error: NSError?)
{
    print("read value for characteristic: " + readCharacteristic)
    peripheral.readValueForCharacteristic(readCharacteristic as CBCharacteristic)
}
```

Figure A.1: Example of using Bluetooth Low Energy for reading data from a characteristic.

```
/*
* Write a value to a given Bluetooth LE characteristic .
*/
func peripheral(peripheral: CBPeripheral, didDiscoverCharacteristicsForService
  service:CBService, error: NSError?)
{
    let data:NSData = NSData(base64EncodedString: "write characteristic data",
      options: NSDataBase64DecodingOptions.IgnoreUnknownCharacters)!
    print("write value \(data) for characteristic: " + writeCharacteristic)
    peripheral.writeValue(data, forCharacteristic: writeCharacteristic,
      type: CBCharacteristicWriteType.WithoutResponse)
}
```

Figure A.2: Example of using Bluetooth Low Energy for writing a data value to a specific characteristic.

```
/*
* Write data greater than 20 bytes to a specified characteristic .
*/
func sendData(peripheral: CBPeripheral, characteristic:CBCharacteristic)
{
    while sendDataIndex < dataToSend.length {
        int amountToSend = dataToSend.length - sendDataIndex > 20 ? 20 :
            dataToSend.length - sendDataIndex
        let currentSend:[NSData] = dataToSend[sendDataIndex..<(amountToSend+sendDataIndex)]
        sendDataIndex += amountToSend
    }
    sendDataIndex = 0
    peripheral.writeValue(currentSend, forCharacteristic: writeCharacteristic, type:
      CBCharacteristicWriteType.WithoutResponse)
}
```

Figure A.3: Example of using Bluetooth Low Energy for writing a data value that is greater than the BLE packet limitation of 20 bytes.

```
/*
* Gets the location from the device's GPS sensor.
*/
func locationManager(manager: CLLocationManager, didUpdateLocations locations: [CLLocation])
{
    locManager.requestWhenInUseAuthorization()

    locManager.startUpdatingLocation()

    let location = locations.last

    let result = NSString(format: "%.5f, %.5f", location!.coordinate.latitude,
      location!.coordinate.longitude)
    self.location = result as String;
}
```

Figure A.4: Example of using the device's GPS sensor for receiving the user's current location.

```
/*
* Initialize multipeer connectivity.
*/
func initializeMultipeerService() {
    startServiceBrowser = MCNearbyServiceBrowser(peer: myPeerId, serviceType: startService)
    super.init()
    startServiceBrowser.delegate = self
    startServiceBrowser.startBrowsingForPeers()
}
```

Figure A.5: Example of initializing Multipeer connectivity to discover nearby iOS devices.

```
/*
* Stop multipeer connectivity.
*/
func stopMultipeerService() {
    stopServiceBrowser = MCNearbyServiceBrowser(peer: myPeerId, serviceType: stopService)
    super.init()
    stopServiceBrowser.delegate = self
    stopServiceBrowser.stopBrowsingForPeers()
}
```

Figure A.6: Example of stopping the Multipeer connectivity service to stop discovering nearby iOS devices.

Appendix B

Evaluation: String

Appendix B shows the String subject cases representing the `Swift` input for our recommendation system. These code blocks are designed as wrappers, where each block implements or uses a commonly available library method for a String object.

```
/*
* Compares this string to the specified object. The result is true if and
* only if the argument is not null and is a String object that represents
* the same sequence of characters as this object.
*/
func equals_Wrapper(A: String) -> Bool {
    print("equals_Wrapper testing")
    let equalsString:String = ""
    return equalsString == A
}
```

Figure B.1: String subject case for `equals_Wrapper`

```
/*
* This object (which is already a string!) is itself returned.
*/
func toString_Wrapper(A: String) -> String {
    print("toString_Wrapper testing")
    let toString:String = A
    return toString;
}
```

Figure B.2: String subject case for `toString_Wrapper`

```
/*
* Compares two strings lexicographically . The comparison is based on the
* Unicode value of each character in the strings . The character sequence
* represented by this String object is compared lexicographically to the
* character sequence represented by the argument string . The result is a
* negative integer if this String object lexicographically precedes the
* argument string . The result is a positive integer if this String object
* lexicographically follows the argument string . The result is zero if the
* strings are equal; compareTo returns 0 exactly when the equals(Object)
* method would return true .
*/
func compareTo_Wrapper(A: String, B: String) -> Int {
    print("compareTo_Wrapper testing")
    let compareToString:String = A
    return compareToString.compare(B).rawValue
}
```

Figure B.3: String subject case for `compareTo_Wrapper`

```
/*
* Returns the index within this string of the first occurrence of the
* specified substring , starting at the specified index .
*/
func indexOf_Wrapper(A: String, B: String, C: Int) -> String.Index! {
    print("indexOfStartingAt_Wrapper testing")
    let indexOfStartingAtString:String = A
    return indexOfStartingAtString.substringToIndex(
      indexOfStartingAtString.startIndex.advancedBy(C))
      .rangeOfString(B)?.startIndex
}
```

Figure B.4: String subject case for `indexOf_Wrapper`

```
/*
* Returns the index within this string of the first occurrence of the
* specified substring .
*/
func indexOf_Wrapper(A: String, B: String) -> Int {
    print("indexOfSubstring_Wrapper testing")
    let indexOfStartingAtString:String = A
    let range = indexOfStartingAtString.rangeOfString(B)
    return indexOfStartingAtString.startIndex.distanceTo(range!.startIndex)
}
```

Figure B.5: String subject case for `indexOf_Wrapper`

```
/*
* Returns the string representation of the String argument.
*/
func valueOf_Wrapper(A: String) -> String {
    print("valueOfString_Wrapper testing")
    let valueOfString:String = A
    return valueOfString
}
```

Figure B.6: String subject case for `valueOf_Wrapper`

```
/*
* Returns the string representation of the Boolean argument.
*/
func valueOf_Wrapper(A: Bool) -> String {
    print("valueOfBoolean_Wrapper testing")
    let valueOfBoolean:Bool = A
    return String(valueOfBoolean)
}
```

Figure B.7: String subject case for `valueOf_Wrapper`

```
/*
* Returns the string representation of the Integer argument.
*/
func valueOf_Wrapper(A: Int) -> String {
    print("valueOfInteger_Wrapper testing")
    let valueOfInteger:Int = A
    return String(valueOfInteger)
}
```

Figure B.8: String subject case for `valueOf_Wrapper`

```
/*
* Returns the string representation of the Double argument.
*/
func valueOf_Wrapper(A: Double) -> String {
    print("valueOfDouble_Wrapper testing")
    let valueOfDouble:Double = A
    return String(valueOfDouble)
}
```

Figure B.9: String subject case for `valueOf_Wrapper`

```
/*
* Returns the length of this string. The length is equal to the number of
* Unicode code units in the string.
*/
func length_Wrapper(A: String) -> Int {
    print("length_Wrapper testing")
    let lengthString:String = A
    return lengthString.characters.count
}
```

Figure B.10: String subject case for `length_Wrapper`

```
/*
* Returns true if, and only if, length() is 0.
*/
func isEmpty_Wrapper(A: String) -> Bool {
    print("isEmpty_Wrapper testing")
    let isEmptyString:String = A
    return isEmptyString.isEmpty
}
```

Figure B.11: String subject case for `isEmpty_Wrapper`

```
/*
* Returns the char value at the specified index. An index ranges from 0 to
* length() - 1. The first char value of the sequence is at index 0, the
* next at index 1, and so on, as for array indexing.
*/
func charAt_Wrapper(A: String, B: Int) -> Character {
    print("charAt_Wrapper testing")
    let charAtString:String = A

    let index = charAtString.startIndex.advancedBy(B)
    return charAtString[index]
}
```

Figure B.12: String subject case for `charAt_Wrapper`

```
/*
* Compares this String to another String, ignoring case considerations. Two
* strings are considered equal ignoring case if they are of the same length
* and corresponding characters in the two strings are equal ignoring case.
*/
func equalsIgnoreCase_Wrapper(A: String, B: String) -> Bool {
    print("equalsIgnoreCase_Wrapper testing")
    let equalsIgnoreCaseString:String = A
    return equalsIgnoreCaseString.lowercaseString == B.lowercaseString
}
```

Figure B.13: String subject case for `equalsIgnoreCase_Wrapper`

```
/*
* Compares two strings lexicographically, ignoring case differences. This
* method returns an integer whose sign is that of calling compareTo with
* normalized versions of the strings where case differences have been
* eliminated by calling
* Character.toLowerCase(Character.toUpperCase(character)) on each
* character.
*/
func compareToIgnoreCase_Wrapper(A: String, B: String) -> Int {
    print("compareToIgnoreCase_Wrapper testing")
    let compareToIgnoreCaseString:String = A
    return compareToIgnoreCaseString.lowercaseString
      .compare(B.lowercaseString).rawValue
}
```

Figure B.14: String subject case for `compareToIgnoreCase_Wrapper`

```
/*
* Tests if the substring of this string beginning at the specified index
* starts with the specified prefix.
*/
func startsWith_Wrapper(A: String, B: String, C: Int) -> Bool {
    print("startsWith_index_Wrapper testing")
    let startsWithIndexString:String = A
    return startsWithIndexString.substringFromIndex(
      startsWithIndexString.startIndex.advancedBy(C))
        .containsString(B)
}
```

Figure B.15: String subject case for `startsWith_Wrapper`

```
/*
* Tests if this string starts with the specified prefix.
*/
func startsWith_Wrapper(A: String, B: String) -> Bool {
    print("startsWith_Wrapper testing")
    let startsWithString:String = A
    return startsWithString.substringToIndex(B.endIndex).containsString(B)
}
```

Figure B.16: String subject case for `startsWith_Wrapper`

```
/*
* Tests if this string ends with the specified suffix.
*/
func endsWith_Wrapper(A: String, B: String) -> Bool {
    print("endsWith_Wrapper testing")
    let endsWithString:String = A
    return endsWithString.substringFromIndex(B.endIndex).containsString(B)
}
```

Figure B.17: String subject case for `endsWith_Wrapper`

```
/*
* Returns a new string that is a substring of this string. The substring
* begins at the specified beginIndex and extends to the character at index
* endIndex - 1. Thus the length of the substring is endIndex-beginIndex.
*/
func substring_Wrapper(A: String, B: Int, C: Int) -> String {
    print("substring_begins_ends_Wrapper testing")
    let substringBeginsEndsString:String = A
    return substringBeginsEndsString.substringFromIndex(
      substringBeginsEndsString.startIndex.advancedBy(B))
      .substringToIndex(substringBeginsEndsString.startIndex.advancedBy(C))
}
```

Figure B.18: String subject case for `substring_Wrapper`

```
/*
* Returns a new string that is a substring of this string. The substring
* begins with the character at the specified index and extends to the end
* of this string.
*/
func substring_Wrapper(A: String, B: Int) -> String {
    print("substring_begins_Wrapper testing")
    let substringBeginsString:String = A
    return substringBeginsString.substringFromIndex(
      substringBeginsString.startIndex.advancedBy(B))
}
```

Figure B.19: String subject case for `substring_Wrapper`

```
/*
* Concatenates the specified string to the end of this string.
*
* If the length of the argument string is 0, then this String object is
* returned. Otherwise, a new String object is created, representing a
* character sequence that is the concatenation of the character sequence
* represented by this String object and the character sequence represented
* by the argument string.
*/
func concat_Wrapper(A: String, B: String) -> String {
    print("concat_Wrapper testing")
    let concatString:String = A
    return concatString.stringByAppendingString(B)
}
```

Figure B.20: String subject case for `concat_Wrapper`

```
/*
* Replaces the first substring of this string that matches the given
* regular expression with the given replacement.
*/
func replaceFirst_Wrapper(A: String, B: String, C: String) -> String {
    print("replaceFirst_Wrapper testing")
    var replaceFirstString:String = A
    replaceFirstString.replaceRange(replaceFirstString.rangeOfString(B)!, with: C)
    return replaceFirstString
}
```

Figure B.21: String subject case for `replaceFirst_Wrapper`

```
/*
* Replaces each substring of this string that matches the given regular
* expression with the given replacement.
*/
func replaceAll_Wrapper(A: String, B: String, C: String) -> String {
    print("replaceAll_Wrapper testing")
    var replaceAllString:String = A
    replaceAllString.replaceRange(replaceAllString.rangeOfString(B)!, with: C)
    return replaceAllString
}
```

Figure B.22: String subject case for `replaceAll_Wrapper`

```
/*
* Converts all of the characters in this String to lower case using the
* rules of the default locale. This is equivalent to calling
* toLowerCase(Locale.getDefault()).
*/
func toLowerCase_Wrapper(A: String) -> String {
    print("toLowerCase_Wrapper testing")
    let toLowerCaseString:String = A
    return toLowerCaseString.lowercaseString;
}
```

Figure B.23: String subject case for `toLowerCase_Wrapper`

```
/*
* Converts all of the characters in this String to upper case using the
* rules of the default locale. This method is equivalent to
* toUpperCase(Locale.getDefault()).
*/
func toUpperCase_Wrapper(A: String) -> String {
    print("toUpperCase_Wrapper testing")
    let toUpperCaseString:String = A
    return toUpperCaseString.uppercaseString;
}
```

Figure B.24: String subject case for `toUpperCase_Wrapper`

```
1 /*
2 * Returns a copy of the string, with leading and trailing whitespace
3 * omitted.
4 */
5 func trim_Wrapper(A: String) -> String {
6     print("trim_Wrapper testing")
7     let trimString:String = A
8     return trimString.stringByTrimmingCharactersInSet(
9       NSCharacterSet.whitespaceAndNewlineCharacterSet())
10 }
```

Figure B.25: String subject case for `trim_Wrapper`

Appendix C

Evaluation: Array

Appendix C shows the Array subject cases representing the `Swift` input for our recommendation system. These code blocks are designed as wrappers, where each block implements or uses a commonly available library method for an Array or List object.

```
/*
* Inserts the specified element at the specified position in this list.
* Shifts the element currently at that position (if any) and any subsequent
* elements to the right (adds one to their indices).
*/
func add_Wrapper(A: [String], B: Int, C: String) -> [String] {
    print("addToIndex_Wrapper testing")
    var addToIndexArrayList:[String] = A
    addToIndexArrayList.insert(C, atIndex: B)
    return addToIndexArrayList;
}
```

Figure C.1: Array subject case for `add_Wrapper`

```
/*
* Appends the specified element to the end of this list.
*/
func add_Wrapper(A: [String], B: String) -> [String] {
    print("addToEnd_Wrapper testing")
    var addToEndArrayList:[String] = A
    addToEndArrayList.append(B)
    return addToEndArrayList;
}
```

Figure C.2: Array subject case for `add_Wrapper`

```
/*
* Removes the element at the specified position in this list. Shifts any
* subsequent elements to the left (subtracts one from their indices).
*/
func remove_Wrapper(A: [String], B: Int) -> [String] {
    print("removeAtIndex_Wrapper testing")
    var removeAtIndexArrayList:[String] = A
    removeAtIndexArrayList.removeAtIndex(B)
    return removeAtIndexArrayList
}
```

Figure C.3: Array subject case for `remove_Wrapper`

```
/*
* Removes the first occurrence of the specified element from this list, if
* it is present.
*/
func remove_Wrapper(A: [String], B: String) -> [String] {
    print("remove_Wrapper testing")
    var removeArrayList:[String] = A
    removeArrayList.removeAtIndex(removeArrayList.indexOf(B)!)
    return removeArrayList;
}
```

Figure C.4: Array subject case for `remove_Wrapper`

```
/*
* Returns the element at the specified position in this list.
*/
func get_Wrapper(A: [String], B: Int) -> String {
    print("get_Wrapper testing")
    var getArrayList:[String] = A
    return getArrayList[B]
}
```

Figure C.5: Array subject case for `get_Wrapper`

```
/*
* Returns a shallow copy of this ArrayList instance. (The elements
* themselves are not copied.)
*/
func clone_Wrapper(A: [String]) -> [String] {
    print("clone_Wrapper testing")
    let cloneArrayList:[String] = A
    let duplicateArrayList = cloneArrayList
    return duplicateArrayList
}
```

Figure C.6: Array subject case for `clone_Wrapper`

```
/*
* Returns the index of the first occurrence of the specified element in
* this list, or -1 if this list does not contain the element.
*/
func indexOf_Wrapper(A: [String], B: String) -> Int {
    print("indexOf_Wrapper testing")
    let indexOfArrayList:[String] = A
    return indexOfArrayList.indexOf(B)!
}
```

Figure C.7: Array subject case for `indexOf_Wrapper`

```
/*
* Removes all of the elements from this list. The list will be empty after
* this call returns.
*/
func clear_Wrapper(A: [String]) -> [String] {
    print("clear_Wrapper testing")
    var clearArrayList:[String] = A
    clearArrayList.removeAll()
    return clearArrayList
}
```

Figure C.8: Array subject case for `clear_Wrapper`

```
/*
* Returns true if this list contains no elements.
*/
func isEmpty_Wrapper(A: [String]) -> Bool {
    print("isEmpty_Wrapper testing")
    let isEmptyArrayList:[String] = A
    return isEmptyArrayList.isEmpty
}
```

Figure C.9: Array subject case for `isEmpty_Wrapper`

```
/*
* Returns the index of the last occurrence of the specified element in this
*  list , or -1 if this list does not contain the element. More formally,
* returns the highest index i such that (o==null ? get(i)==null :
* o.equals(get(i))), or -1 if there is no such index.
*/
func lastIndexOf_Wrapper(A: [String], B: String) -> Int {
    print("lastIndexOf_Wrapper testing")
    let lastIndexOfArrayList:[String] = A
    return lastIndexOfArrayList.indexOf(B)!
}
```

Figure C.10: Array subject case for `lastIndexOf_Wrapper`

```
/*
* Returns true if this list contains the specified element
*/
func contains_Wrapper(A: [String], B: String) -> Bool {
    print("contains_Wrapper testing")
    let containsArrayList:[String] = A
    return containsArrayList.contains(B)
}
```

Figure C.11: Array subject case for `contains_Wrapper`

```
/*
* Returns the number of elements in this list .
*/
func size_Wrapper(A: [String]) -> Int {
    print("size_Wrapper testing")
    let sizeArrayList:[String] = A
    return sizeArrayList.count
}
```

Figure C.12: Array subject case for `size_Wrapper`

```
/*
* Returns a view of the portion of this list between the specified
* fromIndex, inclusive, and toIndex, exclusive. (If fromIndex and toIndex
* are equal, the returned list is empty.) The returned list is backed by
* this list, so non-structural changes in the returned list are reflected
* in this list, and vice-versa. The returned list supports all of the
* optional list operations.
*/
func subList_Wrapper(A: [String], B: Int, C: Int) -> [String] {
    print("subList_Wrapper testing")
    let subListArrayList:[String] = A
    var result:[String] = []
    for (index, item) in subListArrayList.enumerate() {
        if (index >= B && index < C) {
            result.append(item)
        }
    }
    return result
}
```

Figure C.13: Array subject case for `subList_Wrapper`

```
/*
* Appends all of the elements in the specified collection to the end of
* this list, in the order that they are returned by the specified
* collection's Iterator.
*/
func addAll_Wrapper(A: [String], B: [String]) -> Bool {
    print("addAll_Wrapper testing")
    var addAllArrayList:[String] = A
    for item in B {
        addAllArrayList.append(item)
    }
    return true
}
```

Figure C.14: Array subject case for `addAll_Wrapper`

```
/*
* Inserts all of the elements in the specified collection into this list,
* starting at the specified position. Shifts the element currently at that
* position (if any) and any subsequent elements to the right (increases
* their indices). The new elements will appear in the list in the order
* that they are returned by the specified collection's iterator.
*/
func addAll_Wrapper(A: [String], B: Int, C: [String]) -> Bool {
    print("addAllAtIndex_Wrapper testing")
    let addAllAtIndexArrayList:[String] = A
    var result:[String] = []
    for (index1,element1) in addAllAtIndexArrayList.enumerate() {
        if index1 != B {
            result.append(element1)
        } else {
            for element2 in C {
                result.append(element2)
            }
        }
    }
    return true
}
```

Figure C.15: Array subject case for `addAll_Wrapper`

```
/*
* Replaces the element at the specified position in this list with the
*  specified element.
*/
func set_Wrapper(A: [String], B: Int, C: String) -> [String] {
    print("set_Wrapper testing")
    var setArrayList:[String] = A
    setArrayList[B] = C
    return setArrayList
}
```

Figure C.16: Array subject case for `set_Wrapper`

```
/*
* Increases the capacity of this ArrayList instance, if necessary, to
* ensure that it can hold at least the number of elements specified by the
* minimum capacity argument.
*/
func ensureCapacity_Wrapper(A: [String], B: Int) -> [String] {
    print("ensureCapacity_Wrapper testing")
    var ensureCapacityArrayList:[String] = A
    ensureCapacityArrayList.reserveCapacity(B)
    return ensureCapacityArrayList;
}
```

Figure C.17: Array subject case for `ensureCapacity_Wrapper`

```
/*
* Trims the capacity of this ArrayList instance to be the list's current
* size. An application can use this operation to minimize the storage of an
* ArrayList instance.
*/
func trimToSize_Wrapper(A: [String]) -> [String] {
    print("trimToSize_Wrapper testing")
    var trimToSizeArrayList:[String] = A
    trimToSizeArrayList.reserveCapacity(trimToSizeArrayList.count)
    return trimToSizeArrayList;
}
```

Figure C.18: Array subject case for `trimToSize_Wrapper`

```
/*
* Removes from this list all of its elements that are contained in the
* specified collection.
*/
func removeAll_Wrapper(A: [String], B: [String]) -> [String] {
    print("removeAll_Wrapper testing")
    var removeAllArrayList:[String] = A
    for element in B {
        if (removeAllArrayList.contains(element)) {
            removeAllArrayList.removeAtIndex(removeAllArrayList.indexOf(element)!)
        }
    }
    return removeAllArrayList
}
```

Figure C.19: Array subject case for `removeAll_Wrapper`

```
/*
* Retains only the elements in this list that are contained in the
*  specified collection . In other words, removes from this list all of its
* elements that are not contained in the specified collection .
*/
func retainAll_Wrapper(A: [String], B: [String]) -> [String] {
    print("retainAll_Wrapper testing")
    var retainAllArrayList:[String] = A
    for element in B {
        if (!retainAllArrayList.contains(element)) {
            retainAllArrayList.removeAtIndex(retainAllArrayList.indexOf(element)!)
        }
    }
    return retainAllArrayList
}
```

Figure C.20: Array subject case for `retainAll_Wrapper`

```
/*
* Returns a list iterator over the elements in this list (in proper
* sequence), starting at the specified position in the list
*/
func listIterator_Wrapper(A: [String], B: Int) -> [String] {
    print("listIteratorFromIndex_Wrapper testing")
    let listIteratorFromIndexArrayList:[String] = A
    for (index, item) in listIteratorFromIndexArrayList.enumerate() {
        if (index >= B) {
            print(item)
        }
    }
    return listIteratorFromIndexArrayList
}
```

Figure C.21: Array subject case for `listIterator_Wrapper`

```
/*
* Returns a list iterator over the elements in this list (in proper
* sequence).
*/
func listIterator_Wrapper(A: [String]) -> [String] {
    print("listIterator_Wrapper testing")
    let listIteratorArrayList:[String] = A
    for item in listIteratorArrayList {
        print(item)
    }
    return listIteratorArrayList
}
```

Figure C.22: Array subject case for `listIterator_Wrapper`

Appendix D

Evaluation: Dictionary/HashMap

Appendix D shows the String subject cases representing the `Swift` input for our recommendation system. These code blocks are designed as wrappers, where each block implements or uses a commonly available library method for a Dictionary (Swift) or HashMap (Java) object.

```
1 /*
2 * Removes the mapping for the specified key from this map if present.
3 */
4 func remove_Wrapper(A: [String:String], B: String) -> [String:String] {
5     print("remove_Wrapper testing")
6     var removeHashMap:[String:String] = A
7     removeHashMap.removeValueForKey(B)
8     return removeHashMap
9 }
```

Figure D.1: Dictionary/HashMap subject case for `remove_Wrapper`

```
1 /*
2 * Returns the value to which the specified key is mapped, or null if this
3 * map contains no mapping for the key.
4 */
5 func get_Wrapper(A:[String:String], B: String) -> String {
6     print("get_Wrapper testing")
7     let getHashMap:[String:String] = A
8     return getHashMap[B]!
9 }
```

Figure D.2: Dictionary/HashMap subject case for `get_Wrapper`

```
/*
* Associates the specified value with the specified key in this map. If the
* map previously contained a mapping for the key, the old value is
* replaced.
*/
func put_Wrapper(A: [String:String], B: String, C: String) -> [String:String] {
    print("put_Wrapper testing")
    var putHashMap:[String:String] = A
    putHashMap[B] = C
    return putHashMap
}
```

Figure D.3: Dictionary/HashMap subject case for `put_Wrapper`

```
/*
* Returns a Collection view of the values contained in this map. The
* collection is backed by the map, so changes to the map are reflected in
* the collection, and vice-versa. If the map is modified while an iteration
* over the collection is in progress (except through the iterator's own
* remove operation), the results of the iteration are undefined. The
* collection supports element removal, which removes the corresponding
* mapping from the map, via the Iterator.remove, Collection.remove,
* removeAll, retainAll and clear operations. It does not support the add or
* addAll operations
*/
func values_Wrapper(A: [String:String]) -> [String:String] {
    print("values_Wrapper testing")
    let valuesHashMap:[String:String] = A
    valuesHashMap.values
    return valuesHashMap
}
```

Figure D.4: Dictionary/HashMap subject case for `values_Wrapper`

```
/*
* Returns a shallow copy of this HashMap instance: the keys and values
* themselves are not cloned.
*/
func clone_Wrapper(A: [String:String]) -> [String:String] {
    print("clone_Wrapper testing")
    let cloneHashMap:[String:String] = A
    let duplicateHashMap = cloneHashMap
    return duplicateHashMap
}
```

Figure D.5: Dictionary/HashMap subject case for `clone_Wrapper`

```
/*
* Removes all of the mappings from this map. The map will be empty after
* this call returns.
*/
func clear_Wrapper(A: [String:String]) -> [String:String] {
    print("clear_Wrapper testing")
    var clearHashMap:[String:String] = A
    clearHashMap.removeAll()
    return clearHashMap;
}
```

Figure D.6: Dictionary/HashMap subject case for `clear_Wrapper`

```
/*
* Returns true if this map contains no key-value mappings.
*/
func isEmpty_Wrapper(A: [String:String]) -> Bool {
    print("isEmpty_Wrapper testing")
    let isEmptyHashMap:[String:String] = A
    return isEmptyHashMap.isEmpty
}
```

Figure D.7: Dictionary/HashMap subject case for `isEmpty_Wrapper`

```
/*
* Returns the number of key-value mappings in this map.
*/
func size_Wrapper(A: [String:String]) -> [String:String] {
    print("size_Wrapper testing")
    let sizeHashMap:[String:String] = A
    sizeHashMap.count
    return sizeHashMap;
}
```

Figure D.8: Dictionary/HashMap subject case for `size_Wrapper`

```
/*
* Returns a Set view of the mappings contained in this map.
*/
func entrySet_Wrapper(A: [String:String]) -> [String] {
    print("entrySet_Wrapper testing")
    let entrySetHashMap:[String:String] = A
    return Array(entrySetHashMap.values)
}
```

Figure D.9: Dictionary/HashMap subject case for `entrySet_Wrapper`

```
/*
* Copies all of the mappings from the specified map to this map. These
* mappings will replace any mappings that this map had for any of the keys
* currently in the specified map.
*/
func putAll_Wrapper(A: [String:String], B: [String:String]) -> [String:String] {
    print("putAll_Wrapper testing")
    var putAllHashMap:[String:String] = A
    for (key,value) in B {
        putAllHashMap[key] = value
    }
    return putAllHashMap;
}
```

Figure D.10: Dictionary/HashMap subject case for `putAll_Wrapper`

```
/*
* Returns a Set view of the keys contained in this map.
*/
func keySet_Wrapper(A: [String:String]) -> [String] {
    print("keySet_Wrapper testing")
    let keySetHashMap:[String:String] = A
    return Array(keySetHashMap.keys)
}
```

Figure D.11: Dictionary/HashMap subject case for `keySet_Wrapper`

```
/*
* Returns true if this map maps one or more keys to the specified value.
*/
func containsValue_Wrapper(A: [String:String], B: String) -> Bool {
    print("containsValue_Wrapper testing")
    let containsValueHashMap:[String:String] = A
    for value in containsValueHashMap.values {
        if B == value {
            return true
        }
    }
    return false
}
```

Figure D.12: Dictionary/HashMap subject case for `containsValue_Wrapper`

```
1 /*
2 * Returns true if this map contains a mapping for the specified key.
3 */
4 func containsKey_Wrapper(A: [String:String], B: String) -> Bool {
5     print("containsKey_Wrapper testing")
6     let containsKeyHashMap:[String:String] = A
7     for key in containsKeyHashMap.keys {
8         if B == key {
9             return true
10        }
11    }
12    return false
13 }
```

Figure D.13: Dictionary/HashMap subject case for `containsKey_Wrapper`

Appendix E

Evaluation: Detailed Results

E.1 Java → Swift

Table E.1: Java → Swift: Services API Detailed Results

API Experiments	Answer Exists	Pre-Rank 1	Pre-Rank 1 or 2	Rank 1	Rank 1 or 2
Location.get	YES	YES	YES	YES	YES
BLE.read	YES	YES	YES	YES	YES
BLE.write	NO	NO	NO	NO	NO
BLE.write20	NO	NO	NO	NO	NO
WiFi.Initialization	NO	NO	NO	NO	NO
WiFi.Reset	NO	NO	NO	NO	NO
	33%	33%	33%	33%	33%

Table E.2: Java → Swift: String API Detailed Results

API Experiments	Answer Exists	Pre-Rank 1	Pre-Rank 1 or 2	Rank 1	Rank 1 or 2
String.equals	YES	YES	YES	YES	YES
String.toString	NO	NO	NO	NO	NO
String.compareTo	YES	YES	YES	YES	YES
String.indexOf	YES	YES	YES	YES	YES
String.indexOf2	YES	YES	YES	YES	YES
String.valueOf(String)	YES	YES	YES	YES	YES
String.valueOf(Boolean)	YES	YES	YES	NO	YES
String.valueOf(Integer)	YES	YES	YES	YES	YES
String.valueOf(Double)	YES	YES	YES	YES	YES
String.empty	YES	YES	YES	YES	YES
String.length	YES	YES	YES	YES	YES
String.charAt	YES	NO	YES	NO	NO
String.equalsIgnoreCase	YES	YES	YES	YES	YES
String.compareToIgnoreCase	YES	YES	YES	YES	YES
String.startsWith	YES	YES	YES	YES	YES
String.startsWith	YES	YES	YES	YES	YES
String.endsWith	YES	YES	YES	YES	YES
String.substring	YES	YES	YES	YES	YES
String.substring	YES	YES	YES	YES	YES
String.concat	YES	YES	YES	NO	YES
String.replaceFirst	YES	YES	YES	YES	YES
String.replaceAll	YES	YES	YES	YES	YES
String.toLowerCase	YES	YES	YES	YES	YES
String.toUpperCase	YES	YES	YES	YES	YES
String.trim	YES	YES	YES	YES	YES
	96%	92%	96%	84%	92%

Table E.3: Java → Swift: ArrayList API Detailed Results

API Experiments	Answer Exists	Pre-Rank 1	Pre-Rank 1 or 2	Rank 1	Rank 1 or 2
ArrayList.add	YES	YES	YES	YES	YES
ArrayList.add	YES	YES	YES	YES	YES
ArrayList.remove	YES	YES	YES	YES	YES
ArrayList.remove	YES	YES	YES	YES	YES
ArrayList.get	YES	NO	NO	YES	YES
ArrayList.clone	YES	YES	YES	YES	YES
ArrayList.indexOf	YES	YES	YES	YES	YES
ArrayList.clear	YES	NO	YES	NO	NO
ArrayList.isEmpty	NO	NO	NO	NO	NO
ArrayList.lastIndexOf	YES	YES	YES	YES	YES
ArrayList.contains	YES	YES	YES	YES	YES
ArrayList.size	NO	NO	NO	NO	NO
ArrayList.subList	YES	NO	NO	NO	NO
ArrayList.addAll	NO	NO	NO	NO	NO
ArrayList.addAll	NO	NO	NO	NO	NO
ArrayList.set	YES	NO	NO	YES	YES
ArrayList.ensureCapacity	YES	YES	YES	NO	YES
ArrayList.trimToSize	YES	NO	NO	YES	YES
ArrayList.removeAll	YES	NO	NO	NO	NO
ArrayList.retainAll	NO	NO	NO	NO	NO
ArrayList.listIterator	YES	NO	YES	YES	YES
ArrayList.listIterator	YES	YES	YES	YES	YES
	77%	45%	55%	59%	64%

Table E.4: Java → Swift: HashMap API Detailed Results

API Experiments	Answer Exists	Pre-Rank 1	Pre-Rank 1 or 2	Rank 1	Rank 1 or 2
HashMap.remove	YES	YES	YES	NO	YES
HashMap.get	YES	YES	YES	YES	YES
HashMap.put	YES	NO	YES	YES	YES
HashMap.values	YES	NO	NO	NO	NO
HashMap.clone	YES	YES	YES	YES	YES
HashMap.clear	YES	YES	YES	YES	YES
HashMap.isEmpty	YES	YES	YES	YES	YES
HashMap.size	YES	YES	YES	NO	YES
HashMap.entrySet	YES	NO	NO	NO	NO
HashMap.putAll	YES	NO	NO	NO	NO
HashMap.keySet	YES	YES	YES	YES	YES
HashMap.containsValue	YES	YES	YES	YES	YES
HashMap.containsKey	YES	YES	YES	YES	YES
	100%	69%	77%	62%	77%

E.2 Swift → Java

Table E.5: Swift → Java: Services API Detailed Results

API Experiments	Answer Exists	Pre-Rank 1	Pre-Rank 1 or 2	Rank 1	Rank 1 or 2
Location.get	YES	YES	YES	YES	YES
BLE.read	YES	NO	YES	NO	YES
BLE.write	YES	YES	YES	YES	YES
BLE.write20	YES	NO	YES	NO	YES
WiFi.Initialization	NO	NO	NO	NO	NO
WiFi.Reset	NO	NO	NO	NO	NO
	67%	33%	67%	33%	67%

Table E.6: Swift → Java: String API Detailed Results

API Experiments	Answer Exists	Pre-Rank 1	Pre-Rank 1 or 2	Rank 1	Rank 1 or 2
String.equals	YES	YES	YES	YES	YES
String.toString	YES	YES	YES	YES	YES
String.compareTo	YES	YES	YES	YES	YES
String.indexOf	YES	YES	YES	YES	YES
String.indexOf2	YES	YES	YES	YES	YES
String.valueOf(String)	NO	NO	NO	NO	NO
String.valueOf(Boolean)	YES	NO	YES	NO	YES
String.valueOf(Integer)	YES	YES	YES	YES	YES
String.valueOf(Double)	YES	NO	YES	YES	YES
String.empty	YES	YES	YES	YES	YES
String.length	YES	YES	YES	YES	YES
String.charAt	YES	YES	YES	YES	YES
String.equalsIgnoreCase	YES	YES	YES	NO	YES
String.compareToIgnoreCase	YES	NO	NO	NO	YES
String.startsWith	YES	NO	YES	YES	YES
String.startsWith	YES	YES	YES	YES	YES
String.endsWith	YES	YES	YES	NO	YES
String.substring	YES	YES	YES	YES	YES
String.substring	YES	YES	YES	YES	YES
String.concat	YES	YES	YES	YES	YES
String.replaceFirst	YES	YES	YES	YES	YES
String.replaceAll	YES	YES	YES	YES	YES
String.toLowerCase	YES	YES	YES	YES	YES
String.toUpperCase	YES	YES	YES	YES	YES
String.trim	YES	YES	YES	YES	YES
	96%	80%	92%	80%	96%

Table E.7: Swift → Java: ArrayList API Detailed Results

API Experiments	Answer Exists	Pre-Rank 1	Pre-Rank 1 or 2	Rank 1	Rank 1 or 2
ArrayList.add	YES	YES	YES	YES	YES
ArrayList.add	YES	YES	YES	YES	YES
ArrayList.remove	YES	NO	NO	YES	YES
ArrayList.remove	YES	YES	YES	YES	YES
ArrayList.get	YES	NO	NO	YES	YES
ArrayList.clone	YES	NO	YES	YES	YES
ArrayList.indexOf	YES	NO	YES	NO	YES
ArrayList.clear	YES	YES	YES	YES	YES
ArrayList.isEmpty	YES	YES	YES	YES	YES
ArrayList.lastIndexOf	YES	NO	YES	YES	YES
ArrayList.contains	YES	YES	YES	YES	YES
ArrayList.size	YES	YES	YES	YES	YES
ArrayList.subList	NO	NO	NO	NO	NO
ArrayList.addAll	YES	NO	NO	YES	YES
ArrayList.addAll	YES	NO	NO	YES	YES
ArrayList.set	YES	NO	NO	YES	YES
ArrayList.ensureCapacity	YES	YES	YES	YES	YES
ArrayList.trimToSize	YES	YES	YES	YES	YES
ArrayList.removeAll	YES	YES	YES	YES	YES
ArrayList.retainAll	YES	NO	YES	YES	YES
ArrayList.listIterator	YES	NO	NO	NO	YES
ArrayList.listIterator	YES	YES	YES	YES	YES
	95%	50%	68%	86%	95%

Table E.8: Swift → Java: Dictionary API Detailed Results

API Experiments	Answer Exists	Pre-Rank 1	Pre-Rank 1 or 2	Rank 1	Rank 1 or 2
Dictionary.remove	YES	NO	NO	YES	YES
Dictionary.get	YES	YES	YES	NO	YES
Dictionary.put	YES	YES	YES	YES	YES
Dictionary.values	YES	YES	YES	YES	YES
Dictionary.clone	YES	NO	NO	YES	YES
Dictionary.clear	YES	NO	NO	NO	NO
Dictionary.isEmpty	NO	NO	NO	NO	NO
Dictionary.size	YES	YES	YES	YES	YES
Dictionary.entrySet	YES	NO	NO	YES	YES
Dictionary.putAll	YES	YES	YES	YES	YES
Dictionary.keySet	YES	YES	YES	YES	YES
Dictionary.containsValue	YES	NO	NO	NO	YES
Dictionary.containsKey	YES	YES	YES	NO	YES
	92%	54%	54%	62%	85%