AppXimity: A Context-Aware Mobile Application Management Framework

by

Ernest E. Alexander Jr. Aaron
B.Sc., Universiti Tenaga Nasional, 2011

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

AppXimity: A Context-Aware Mobile Application Management Framework

by

Ernest E. Alexander Jr. Aaron
B.Sc., Universiti Tenaga Nasional, 2011

Supervisory Committee

---

Dr. Hausi A. Müller, Supervisor
(Department of Computer Science)

---

Dr. Issa Traoré, Outside Member
(Department of Electrical and Computer Engineering)

**Supervisory Committee**

---

Dr. Hausi A. Müller, Supervisor
(Department of Computer Science)

---

Dr. Issa Traoré, Outside Member
(Department of Electrical and Computer Engineering)

## ABSTRACT

The Internet of Things is an emerging technology where everyday devices with sensing and actuating capabilities are connected to the Internet and seamlessly communicate with other devices over the network. The proliferation of mobile devices enables access to unprecedented levels of rich information sources. Mobile app creators can leverage this information to create personalized mobile applications. The amount of available mobile apps available for download will increase over time, and thus, accessing and managing apps can become cumbersome. This thesis presents AppXimity, a mobile-app-management that provides personalized app suggestions and recommendations by leveraging user preferences and contextual information to provide relevant apps in a given context. Suggested apps represent a subset of the installed apps that match nearby businesses or have been identified by AppXimity as apps of interest to the user, and recommended apps are those apps that are not installed on the user's device, but may be of interest to the user, in that location.

# Contents

# List of Tables

# List of Figures

# ACKNOWLEDGEMENTS

I would like to thank:

**Dr. Hausi Müller,** for his guidance and valuable insights. The time and patience he exercised in scrutinizing the content of this thesis, to ensure it is up to proper standards is outstanding.

**my colleagues, friends and family,** for their motivation and moral support, mostly when I became apprehensive about composing this thesis.

*The Internet will disappear. There will be so many IP addresses, so many devices, sensors, things that you are wearing, things that you are interacting with, that you won't even sense it. It will be part of your presence all the time. Imagine you walk into a room, and the room is dynamic. And with your permission and all of that, you are interacting with the things going on in the room.*

Eric Schmidt

# Chapter 1

# Introduction

In an app-driven society, the number of available apps that can be installed on smartphones inevitably increases over time [2]. As a result, navigating to locate apps requires scrolling back and forth between pages. Also, finding apps in an app store is a complex task, because of the vast range of choices available. A need arises, therefore, for mobile-application management, which is the subject of this thesis. This introductory chapter provides a brief overview of the problem, its motivation, and the solutions proposed herein.

## 1.1 Motivation

Smartphones are being adopted at an extraordinary pace, and are tightly integrated into our daily lives. A smartphone combines cell-phone functions, such as voice calling and texting, with functions of a personal computer, such as Internet browsing and the ability to run various third-party applications (known as apps).

IBM Simon was the first smartphone on the market [3]. IBMs novel product was soon imitated by other firms, leading to unprecedented levels of computing power in our pockets. According to Techradar, the Samsung Galaxy Edge S7 Edge is ranked as the best smartphone in the world for 2016, with the iPhone 7 Plus in third place.[1] Smartphones today are equipped with at least a 2.2-GHz processor, an 8MP camera, a 1920x1080 display, and various sensors such as GPS, accelerometers, a proximity/ambient sensor, and a gyroscope.

[1] http://www.techradar.com/news/phone-and-communications/mobile-phones/20-best-mobile-phones-in-the-world-today-1092343/10

Apples iOS and Google's Android are the two most popular operating systems for smartphones today. Approximately 80% of the one billion smartphones sold in the third quarter of 2015 were running the Andriod operating system [4]. According to Statista, the number of smartphone users is expected to surpass five billion by 2019 [4]. The average time spent by youths accessing online content on smartphones is 3.26 hours per day [4]. This extended use of smartphones can be attributed to the fact that users can extend the functionality of the smartphone by the use of third-party applications, which are available for download through various app-distribution platforms.

The two largest app distribution platforms are Apple's App Store for iOS users and Google Play for Andriod users. As of June 2016, it is estimated that Android and Apple users can choose from among 2 and 2.2 million apps, respectively, in their online stores [5]. A study conducted by Nielsen found that on average, smartphone users have installed 41 apps per device [6]. Furthermore, users spend 10% more time interacting with services through these apps than using mobile web browsers [6], among the various services include banking, shopping and bill payments.

Smartphone sensors provide a wide array of user context. Apps leverage this context to personalize users' experiences by anticipating their needs. For example, Spotify automatically recommends a song playlist based on the music a user has previously played. Research into accurately discovering context, efficiently disseminating contextual information, and making use of this information remains at the forefront of context-aware computing [7]. Context awareness remains a key factor in the development of novel applications in ubiquitous computing [7][8].

## 1.2 Problem Statement

With the tremendous growth in the development of new apps, it is easy to predict that the number of apps available in the app stores, as well as in the number of different apps installed per device, will increase. Two immediate consequences follow from this growth: (1) Since the size of mobile displays is, by definition, small, a smartphone user will have to navigate through various pages to locate an app. An existing method of app-management is to group apps into folders manually for easy navigation. However, the alternative possibility of dynamically serving relevant apps to the user at the time they are needed is worthy of investigation. (2) Given the enormous number of available apps in the app stores, it is a cumbersome task for

users to explore and find apps based on their preferences, and in a given context.

Thus, there is value in presenting apps dynamically according to relevance. Contextual data, such as location, time, and user preferences, are important to dynamic app presentation, because users' contexts change constantly as they navigate from place to place with their smartphones [9] [10]. This thesis aims to answer the following research questions:

1. RQ1: How can we gather relevant context from a user's location to provide dynamic app management?
2. RQ2: Given relevant user context, can we dynamically organize the apps a user has installed according to given contexts?
3. RQ3: Given relevant user context, can we recommend apps that a user may need in a given context?
4. RQ4: What is the role of micro-services as a service-oriented architecture for building IoT applications?

## 1.3 Thesis Contribution

This thesis presents AppXimity, a mobile-app-management that exploits user's contextual data such as location, preferences and app usage, to organize apps dynamically according to relevance in a context-aware app widget.

## 1.4 Thesis Overview

Chapter 2 presents an overview of the need for context-aware applications in ubiquitous computing. It also serves as an extension to the motivation behind this work. Chapter 3 provides the overall design of AppXimity, this includes the architectural and user interface design. The key factors considered when choosing the system design is also discussed. Chapter 4 explores in detail, the implementation of each software component is explored in detail. Chapter 5 discusses the experiments that we carried out to validate user acceptance. Chapter 6 summaries the work done in this thesis, and it also provides an overview of future work and contributions of this work.

# Chapter 2

# Background

## 2.1 Smartphone Revolution

In recent years, smartphone usage has sky-rocketed, smartphones have revolutionized the way in which we communicate and interact with each other. The first mobile phone was produced in 1973 by Martin Cooper, a researcher at Motorola [11]. Its initial prototype boasted a talk time of 30 minutes, and required 10 hours to recharge.

Cooper made the first call to his rival, Joel Engel of Bell labs, to boast that he was successful in creating the first mobile phone. A decade later, Motorola launched its first commercial version of the mobile phone – Motorola DyanTAC 8000X. Since then, the new era of mobile phones and their capabilities have evolved. In 1994, IBM Simon was the first ever mobile phone to feature software applications. Among the applications included were: email, calculator, calendar, and a game called scramble.

Table 2.1: First Mobile Phone Specifications[1]

| | |
|---|---|
| Name | Motorola Dyna-Tac |
| Weight | 2.5 pounds |
| Size | 9 x 5 x 1.75 inches |
| Number of Circuit Boards | 30 |
| Talk time: | 30 minutes |
| Recharge Time | 10 hours |
| Features | Talk, listen, dial |

[1]http://www.knowyourmobile.com/nokia/nokia-3310/19848/history-mobile-phones-1973-2008-handsets-made-it-all-happen

It also featured predictive typing; it would attempt to guess the next characters that are required to form a word.[2]

Such novel innovations led us to the very powerful computing devices that we carry around in our pockets today. In 2007, Apple released the first iPhone that supported third party applications. The iPhone included several sensors to enhance the user's experience, like a proximity sensor that would automatically turn off the screen, an accelerometer that could automatically rotate the screen to match the orientation of the device, and an ambient light sensor that could automatically adjust the brightness based on light in the environment. Today, Apple's A10 Fusion Chip, in their latest release of the iPhone 7, is the most powerful chip ever in a smartphone.[3] Some of the features in the iPhone 7 are summarized in Table 2.2.

Table 2.2: iPhone7 Specification (Courtesy of Apple)[4]

| | |
|---|---|
| Messaging | iMessage, SMS (threaded view), MMS, Email, Push Email |
| Sensors | Fingerprint, gyro, proximity, compass, barometer, accelerometer |
| Features | Siri natural language commands and dictation, iCloud cloud service, MP4/H.264 player |
| Talk time | Up to 14 hours (3G) |
| Music play | Up to 40 hours |
| Java | No |

Smartphones and the app store are facilitating the dissemination of a vast variety of third party applications. These apps have a wide array of purposes, ranging from gaming to medical diagnosis. Apple's HomeKit app lets you dim the lights in your home at night, adjust the fan speed, or even see who is at your front door from within the app.[5] My Medical app for iOS allows patients to store their complete medical history online, track, and predict early signs of anomalies.[6]

[2]http://www.bloomberg.com/news/articles/2012-06-29/before-iphone-and-android-came-simon-the-first-smartphone

[3]http://www.apple.com/ca/iphone-7/

[4]http://www.apple.com/lae/iphone-7/specs/

[5]http://www.apple.com/ca/ios/home/

[6]http://mymedicalapp.com/

## 2.2 Internet of Things (IoT)

The prospect of the Internet of things has sparked various interesting discussions about how it integrates into our daily lives. The Internet has become ubiquitous and the cost of creating powerful computing devices equipped with Wi-Fi capabilities and built in sensors are decreasing. In 2013, Cisco estimated that by 2020, 50 billion smart devices will be interconnected.[7] This estimate creates a yearning interest in the world of IoT.

Generally speaking, IoT is the enablement of everyday objects, with sensors and actuators to be inter-networked, allowing them to send and receive data. The IoT allows for objects to be sensed and actuated upon remotely. The term IoT was coined by Ashton in 1999, at Auto-ID centers [12]. Initially, the term was used to refer to objects enabled with radio frequency identification that were tracked over the network without the need for human intervention.


Figure 2.1: Internet Of Things (Courtesy of Huffington Post) [8]

Today, a myriad of definitions for IoT has emerged, but they all describe scenarios in which computing devices and everyday objects are integrated with sensors and employ communication services offered by Internet protocols to exchange data [13].

Cisco's CEO has pegged the Internet of things market value at $19 trillion.[9] The various application domains that can leverage the benefits of the IoT are endless. The Industrial Internet Consortium has identified industries of interest that can directly

[7] http://www.rcrwireless.com/20160628/opinion/reality-check-50b-iot-devices-connected-2020-beyond-hype-reality-tag10

[8] http://www.huffingtonpost.com/news/internet-of-things/

[9] http://www.bloomberg.com/news/articles/2014-01-08/cisco-ceo-pegs-internet-of-things-as-19-trillion-market

leverage these benefits.[10] These industries include: Healthcare, Energy, Transportation, Smart Cities and Manufacturing.

Employing the Industrial Internet in healthcare will result in fewer medical errors. Medical equipment can be monitored, modeled, and remotely controlled, thus providing quality care for patients who are unable to leave home. Ultrasound and infrared can be used to monitor activity in a patients home, detect falls, and trigger automatic ambulatory services [14].

In manufacturing, the Industrial Internet will significantly improve production and efficiency in the supply chain. Intelligent machines and devices will govern processes, and take corrective measures to obviate avoidable breakdowns of machinery. New steering instruments will be interlinked, if any changes are detected in any part of the chain, it will automatically trigger appropriate adjustments on the floor factory [15].

In smart cities, citizens can use their mobile devices to report hazards such as potholes. Smart buildings can automatically adjust temperature and lighting accordingly, thus reducing the carbon footprint. Fire fighters movements can be tracked through the use of wearables, this will improve safety [16].

Computing systems' complexity are approaching the limits of human capabilities [17]. It will be almost impossible to manage and administrator such systems manually — too labour intensive, thus resulting in a costly endeavour and being prone to errors. The success of overcoming such challenges depends greatly on innovating new technologies. IBM proposed autonomic computing—systems are created to manage themselves based on high level goals defined by system administrators [17].

For these systems to be truly self-managing, an Autonomic Manager as depicted in Figure 2.2, is created to monitor applications. The Autonomic Manager (AM) is a component or a system that orchestrates the behaviour of other Autonomic Components in the system [18]. More specifically, the AM consists of a feedback loop known as the MAPE-K loop, which has five main functions: monitoring, analysing, planning, executing and a knowledge bank. Each output in the loop is used to assess the system's state. This assessment determines if the system needs to be modified to fulfill its policy.

In reference to Figure 2.2, the sensors provide a mechanism to collect data about the state of the managed element. The managed element can be any component in the autonomous system, such as database, cluster of servers, or even a complete software application. The sensors and effectors are responsible for controlling the

[10]http://www.iiconsortium.org/members.htm

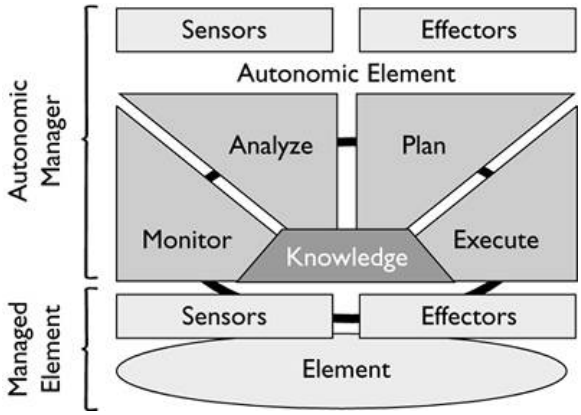


Figure 2.2: Autonomic Manager [1]

managed element. The autonomic manager collects, filters, and provides reports of the data collected from the managed element through its sensors. It then analyses the data and can predict future situations. At the planning stage , the pre-defined policies are used to verify what actions need to be taken for the system to achieve its desired goals. Then finally, execution, this is when the autonomic manager provides a set of commands actuate the managed element. The effectors are the mechanism that alters the configuration of the managed element.

The architecture provides various self-* characteristics that systems in the autonomic computing paradigm can leverage: self-configuring, self-healing, self-protecting, and self-optimizing [19]. Self-configuration – autonomic elements should be able to configure themselves based on the environment. These environments could change from time to time. Self-healing – failing of an autonomic element is inevitable however, the system should be able to quickly and robustly deal with such failures. Self-protecting – the system should be able to protect itself from undesirable attacks and system behaviours. Self-optimization – systems should be able to analyse and make decisions using the most optimal solutions, given the various environmental factors.

## 2.3 Context-Awareness

### 2.3.1 Defining Context

Over the years, various definitions for context have spawned. This subsection reviews a few of these definitions that researchers have coined. The initial definition conceived by Schilit and Theimer [20] refers to context as location, discovery of nearby people

and objects, and changes to those objects over time. Brown *et al.* [21] provided a similar definition that describes context as location, identities of people around, the user, time of day, season, temperature, etc. Dey [22] characterized context as the emotional state of a user, location, orientation, data and time, objects, focus of attention, and people in the user's environment. Ryan *et al.* [23] defined context as the user's location, environment, identity and the time of day. Ward *et al.* [24] refer to context as the condition of an applications environment, and Rodden *et al.* [25] described it as the settings of an application. Villegas defined context as any information useful to characterize the state of individual entities and the relationships among them. An entity is any subject which can affect the behavior of the system and/or its interaction with the user [26].

A mobile device context constantly changes. As users traverse from place to place, factors of their new environment have to be taken into account. The definition of context by Schilit *et al.* are more practical for this research, because they account for the constant change in context. They claim that the most important aspects to context are: the available resources nearby, a user's identify and the identify of others around [27]. Additionally, to account for the constant change in the execution environment, they presented three environmental variables:

> user environment, such as location, collection of nearby people, and social context;
>
> physical environment, such as temperature, lighting and noise levels;
>
> computing environment, such as available processors, devices accessible for the user input and output, network capacity, connectivity and cost of computing.

### 2.3.2 Categorizing Context

Over the years, researchers have attempted to categorize context. This subsection reviews a few of the categorization schemes.

Schilit and Theimer [20] categorized context by presenting three common questions that can be used to determine the context:

1. **Who you are**: this includes location data, such as GPS coordinates, user preferences.

2. **Who you are with**: who are the people in your proximity.

3. **What resources are nearby**: information about smart objects, landmarks, addresses that are nearby.

Henricksen [28] categorized context into four categories:

1. **Sensed**: data that is directly sensed from the sensors.
2. **Static**: information that seldom changes, such as capabilities of the sensor.
3. **Derived**: information that is derived from sensed data.
4. **Profiled**: information that changes with low frequency.

Instead of categorizing context, Van Bunningen *et al.* [29] classified the context categorization schemes into two broader categories:

1. **Conceptual categorization**: categorize context based on the meaning and conceptual relationships between the context.
2. **Operational categorization**: categorize context based on how they were acquired, modelled, and treated.

Perera *et al.* [30] categorized context into two categories:

1. **Primary context**: information retrieved from sensors.
2. **Secondary context**: information that can be computed using primary context. Secondary context can be computed by using sensor data fusion operations or data retrieval operation such as web service calls.

### 2.3.3 Personalizing Context

Acquisition of context is important as it allows applications to exploit a user's environment and take advantage the contextual information. However, not all the information in the user's current space is relevant. For example, a restaurant recommender should not (unless explicitly requested) provide various non-vegetarian eateries to a vegetarian.

Research about personalizing context, based on users' preferences, has been carried out by many researchers in recommender systems. Byun *et al.* [31], and Lee *et al.* [32] used a decision tree approach to infer a user's preferences based on their context history. This approach personalizes context based on preference rules derived

from learning the user's actions, according to their immediate context. Si *et al.* [33] proposed, *Synapse–* applied the Hidden Markov Model [34] in a Bayesian Network. It learns the user's habits then context is personalized based on the user's habits in passive or active mode. With both approaches above, it is difficult to predicate the preferences for new users, and user's have no control over their context.

Villegas proposed a Personal Context Sphere (PCS) [26] – is a distributed repository that contains context information about web entities, relevant to a user's task in a particular domain. This approach is interesting because consumers share this information with the user. Most importantly, the user administrates their PCS.

This research exploits Villegas' Personal Context Sphere (PCS) and Byun's decision tree to incorporate user preferences. It also extends the PCS to implicitly monitor users frequent locations and app usage, to provide a more personalized app recommendation.

## 2.4 Context-Aware Computing

Exploiting context in a dynamically changing environment and adapting quickly to these changes still remains a challenge in context-aware computing. Schilit [20] defines context-aware computing by categorizing context-aware applications. These categories are:

1. **Automatic contextual reconfiguration**: defines a process of adding and removing components, or modifying the connections between components due to contextual changes.

2. **Proximate selection**: refers to a user interface approach where only nearby objects are highlighted. An application's recommendation for dinner will be heavily influenced by a user's location.

3. **Context-triggered actions**: uses conditional statements such as IF-THEN rules that dictate how context-aware systems should adapt.

4. **Contextual information and commands**: aims to exploit and predict user's actions based on their situation. Contextual information can produce different results according to the context in which they are given.

Pascoe *et al.* [35] reviewed Schilit *et al.'s* [20] set of general context-aware categories, and proposed a set of core generic capabilities that uses a vocabulary to

identify and describe context-awareness independently of application, function, or interface. These generic capabilities include:

1. **Contextual sensing**: refers to the computing device ability to detect various environmental states and presents them to the user in a convenient way, effectively increasing the user's awareness. For example, a Global Positioning System (GPS) can alert the user of the estimated travel time between two points of interest.

2. **Contextual adaptation**: refers to the application's ability to tailor themselves by exploiting contextual knowledge and seamlessly integrating it in a user's environment. For example, adjusting the back-light on a screen based on the amount of external light available.

3. **Contextual resource discovery**: refers to the ability of the computing device to discover other resources within the same context as itself, and exploit these resources while they remain in the same contextual space. For example, as users approach a bus stop, a broadcast can be sent to their mobile phones with up-to-date bus arrivals at that location.

4. **Contextual augmentation**: extends the above capabilities by augmenting the environment with additional information. This is achieved by associating digital data with a particular context that it relates to. A tourist visiting a museum with a context-aware museum application is provided with additional information about the artwork that surrounds them.

In a survey paper, Chen *et al.* [7] highlighted that the previous attempts to define context and categorize context-aware computing were inadequate. They presented a different perspective on how mobile applications can take advantage of context. From this survey emerged a two part definition to context aware computing. These definitions are:

1. **Active context awareness:** an application automatically adapts to discovered context, by changing the application's behaviour.

2. **Passive context awareness**: an application presents the new and updated context to an interested user or makes the context persistent for the user to retrieve later.

Perera *et al.* [30] expounded the perspective presented by Chen *et al.*, and added an additional characteristic to the definition of context-aware computing.

1. Personalization allows users to set their preferences, likes and expectations in the system manually.

From the two definitions presented by Chen *et al.* [7], and the addition of Perera *et al.* [30], we see active context-aware computing and personalization are more interesting. Active context-awareness aids in the development of smart applications eliminate non-essential user cooperation. Personalization allows applications to be better tailored on an individual basis.

## 2.5 Context-Aware Applications

Context awareness in mobile applications refer to the paradigm of mobile computing whereby applications discover and take advantage of a user's contextual information. This contextual information includes user location, user activity, time of day, nearby devices and people. Weiser believes that a user's location is predominately at the forefront of context-aware mobile applications [36]. Below, we review two context aware mobile applications; we highlight their active and passive context will be highlighted.

**Yakkit** [37]

**Active context:** User's location, nearby people, and objects.
**Passive context:** User's location.
**Overview:** Yakkit is location based messaging application used to instantly communicate with people in close proximity without the need to create profiles. It also employs the use of virtual billboards within a location space so messages can be left for others.

**Context-aware Mobile Personal ASSistant (COMPASS)** [38]

**Active context:** User's location, user's preferences, current time.

**Passive context:** None.
**Overview:** An application that provides a tourist with information and services that they may be interested in, within the given context. A tourist, who wishes to find accommodation at night will be presented with a list of hotels and campsites that matches the user's preference. Similarly, a tourist who is interested in history and architecture will be presented with nearby monuments.

## 2.6 Mobile Application Management

Initially, most app recommendation techniques, investigated by previous researchers, focused mainly on recommending apps from the app store, through frequent downloads, explicit user ratings, and statistics collected based on types of app installed.

Recent research to personalize app recommendations has produced some innovative techniques. AppJoy combines collaborative filtering with how users actually uses their apps to personalize recommendations [39]. Woerndl *et al.* [40] implemented a mobile app recommendation system that recommends apps based on apps other users have installed, or used frequently at a given location. AppAware shares installation, updates, and removals of Android applications to people in close proximity so that they can discover new apps [41]. Applause exploits user location to recommend apps without collecting any additional user data [42]. Although there have been strides to personalize app recommendations, little to no research has been done on dynamic management of apps – dynamically serve a user with the relevant apps at the right time. Apple has incorporated a suggested app feature in the new iOS, which places the icon of an installed vendor service app on a user's lock screen when the user is in close proximity. Apple's Spotlight search page also has apps recommended by Siri. These apps are recommended based on the apps the user frequently uses. Siri's suggested app feature and lock screen app recommendation feature can be seen in Figure 2.3a and Figure 2.3b, respectively. The idea of this research was inspired by Apple's suggested app feature however, it will expand app management to both installed and recommended apps. Moreover, our Context-Aware App widget will load context apps dynamically.


(a) Siri Suggested Apps


(b) Siri Suggested Icon

Figure 2.3: Siri Suggested Feature

## 2.7 Service Architectures

Server-side applications support a variety of different clients including desktop browsers, native mobile applications and mobile browsers.[11] These application may provide services that expose an API that other services or application consume. The application may also handle HTTP request and exchange messages with other systems. This section reviews two prominent service architectural styles that are implemented when building such applications.

### 2.7.1 Monolithic Architecture

The monolithic architecture composes a software system in a single code-base that is complied together and produces a single artifact. For example, a Java application is deployed as a single WAR file. The monolithic architecture is tightly-coupled hence all its components and dependencies must be present in order for the code to be compiled and executed. Changes in any component requires the entire application to be rewritten and re-compiled.

#### Benefits of the Monolithic Architecture [43]

The benefits of the monolithic architecture are:

1. Simple to develop. Most development tools provide integrated support for the development of monolithic applications.

2. Simple to deploy. Only a single packaged application has to be copied to a server.

3. Simple to scale horizontally by running multiple instance behind a load balancer.

#### Drawbacks of the Monolithic Architecture [43]

When a monolithic application grows, this approach has various drawbacks. The drawbacks are:

1. Monolithic applications are slow at adapting new technologies. A change in the technology stack will affect the entire application and can incur additional time and cost.

[11] http://microservices.io/patterns/monolithic.html

2. Continuous delivery is difficult. The entire application has to be re-deployed on each update.

3. Difficult to scale both horizontally and vertically. Different modules may have conflicting resources.

### 2.7.2 Mirco-services Architecture

Building software using the micro-services service oriented architecture approach is comparatively new in software architecture. Over the years, software has been built and developed using the monolithic architecture. In order to make changes, the server side application has to be rebuilt and re-deployed with the updated version. Although this approach comes with various benefits such as easy to develop, simple to deploy, and simple to scale horizontally, there are some caveats to building IoT applications using this approach.

An IoT application can be viewed as a collection of functions (services) that rely on each other. For example, one function could collect data from sensors, another function processes this data, another function transforms the data by applying predefined logic, and another function could fetch data from a 3rd party API like Google Maps. Such applications have many different variables to consider, such as remote API calls, and must be fault tolerant in the likelihood that a service fails. Also, as the application grows, the code base may need to change and the application may need to scale both horizontally and vertically.

The mirco-services architecture is an approach to developing an application as a small set of independent services. These services run independently on their own processes. Communication between services is done using some lightweight mechanism, usually HTTP [44]. These micro-services are self-sustaining and can be deployed anywhere in the cloud. Micro-services can expose their interface using standard protocols, such as a RESTful API. The output can be consumed by other services, without direct coupling through shared libraries or language bindings. This architecture enables continuous development of a set of manageable services that are easier to understand, maintain, and deploy. It also enables the infrastructure to scale horizontally and vertically, yielding great benefits in the IoT paradigm.

**Benefits of the Micro-services Architecture [45]**

The benefits of the micro-services architecture are:

1. Enables continuous delivery. A change in a single micro-service does not require the entire application to be re-deployed.

2. Design autonomy. Software teams can employ different technologies and frameworks for each micro-service independently.

3. Fault tolerant. Functionality is spread across various services. This eliminates a single point of failure.

**Drawbacks of the Micro-services Architecture [45]**

While the micro-services architecture provide significant advantages over the monolithic architecture, the de-coupling of components has various drawbacks. These drawbacks are:

1. Deploying a micro-service application is more complex than a monolithic application. The application may consist of a large number of services that require multiple run-time instances.

2. Changes that span multiple service are difficult to implement.

3. Degraded Performance. Micro-services communicate over the network and inherent the challenges of distributed computing.

Considering the dynamic expectations of deployment and scalability that comes with IoT applications, we chose a micro-services architecture to implement our prototype for this thesis.

## 2.8 Summary

This chapter presented the mobile device revolution and its integration into our lives. It also presented the prospect of creating software systems that are self-managing. The benefits and drawbacks of using the micro-services and the monolithic architectures for IoT applications were also discussed. Finally, the importance of personalizing context was also explored.

# Chapter 3

# Design

## 3.1 Architectural Design

Location data is becoming significantly more important with the rise of IoT. According to Lundquist[1] :

"Location is a vital dimension of the IoT concept that encompasses the ability of things to sense and communicate their geographic position. In this context, location acts as an organizing principle for anything connected to the Internet."

App developers are realizing the value of using location data to personalize mobile application experiences [46] [47]. In this work, we leverage location data, time and user activity as the primary source of data input.

A users contextual data, apps installed, and preferences play an integral part in the mobile-app-management ecosystem. If a user is at latitude $x$ and longitude $y$, we can bound our search for nearby vendor services (VSs), by a pre-defined radius $r$. After retrieving the list of nearby VSs in the given $r$, we can filter the list to load the apps that match those VSs in the Context-Aware App widget, a dynamic container installed on the user's home screen to house context-aware apps.

For those VSs in the vicinity whose apps are not currently installed on the user's device, the app recommender returns the most relevant apps. If the user has the coarse-grained recommender option enabled, the list of the nearby VS apps will lack personalization. If the fine-grained recommender is enabled, user-specific preferences from their PCS will be leveraged to personalize the list of recommended apps. Assume that there is a coffee shop and an ice-cream parlour within our proximity circle defined

[1] http://clearbridgemobile.com/location-based-technology-for-mobile-apps-beacons-vs-gps-vs-wifi/

by $r$. If the contextual data retrieved from the users PCS indicate that the user is a coffee lover but dislikes ice cream, then the list of recommended apps will include the nearby coffee shop but not the ice-cream parlour.

Figure 3.1 provides a high-level view of the architectural design of the system. The basic flow in communication is as follows: The client arrives at a given location, the location data and user preferences are sent to the API Gateway for processing, the API Gateway calls various other services to get information about places and services nearby and determines whether any such place or service is associated with an installed app. The Gateway then calls other services responsible for recommending apps that the user may need in his/her given context.

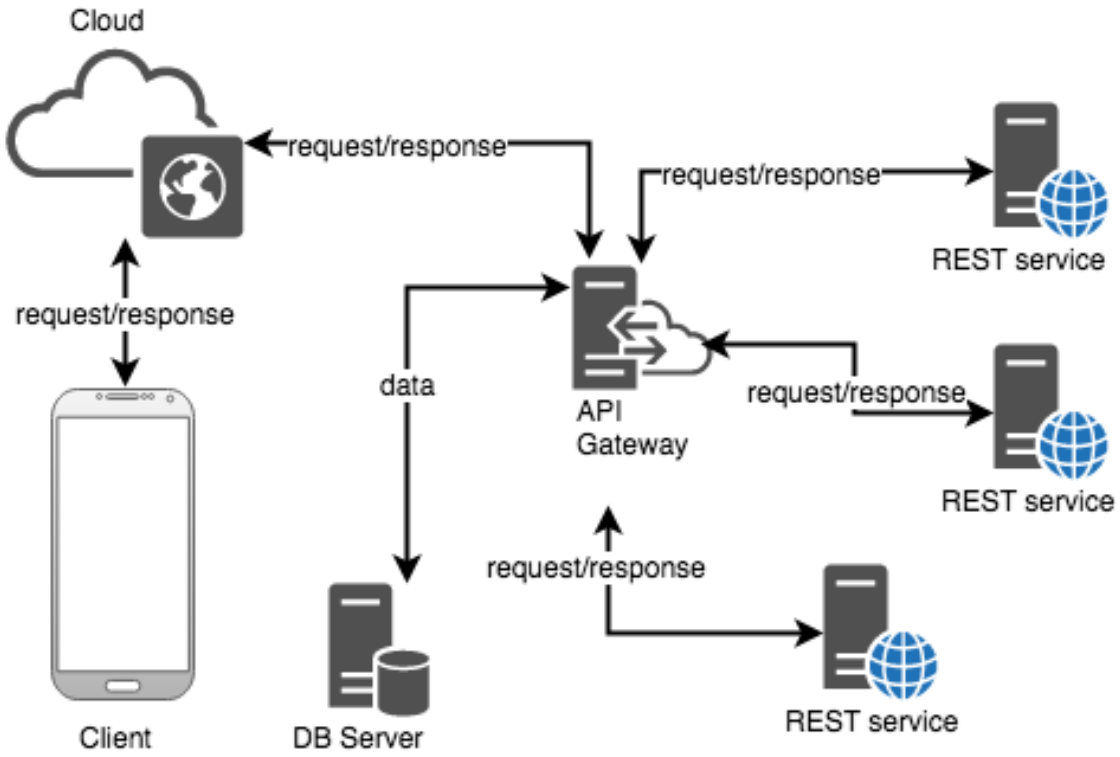


Figure 3.1: Architectural Design

## 3.2 Software Architecture

To realize this framework, we created eleven microservices and used one external web-service API (Google Places API). The eleven microservices components are as follows: A Context-Aware App widget, a Client, an API Gateway, a Personal Context

Sphere, an AppXimity Recommender, String Manipulation, a Repository Manager, Places Nearby, a Keyword Extractor, a GPlay Extractor, and a GPlay Repository, as is depicted in Figure 3.2. The 5 minimum core software components are: A Context-Aware App widget, a Client component, an API Gateway, a Personal Context Sphere and an App-Recommender. The details for each component are discussed in Chapter 4.

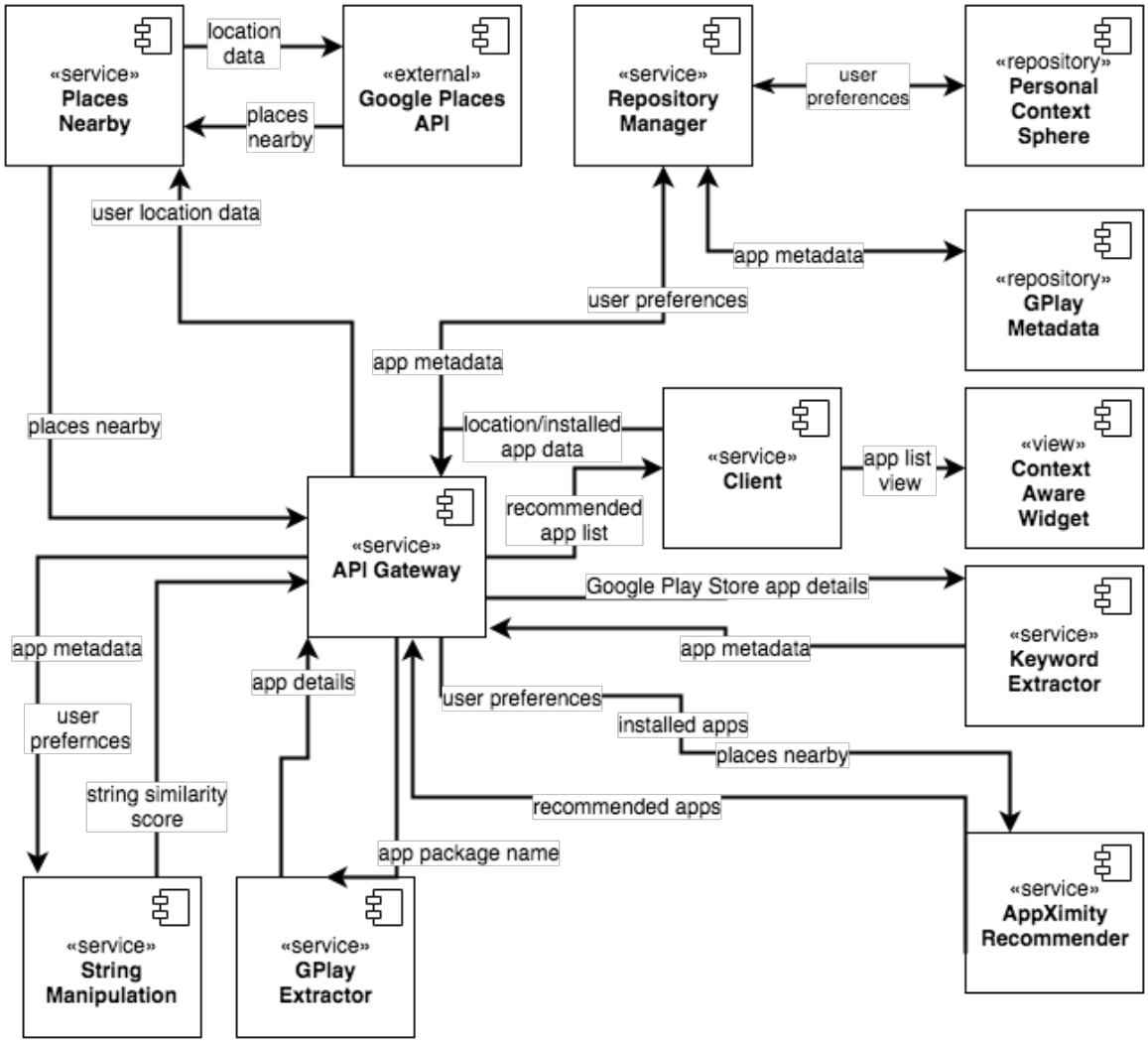


Figure 3.2: Detailed Component Diagram

## 3.3 System Design

### 3.3.1 Software Methodology

All process models have one goal in common: the successful completion of the product. A comprehensive review of the following software-process models was carried out [48]: Waterfall, Incremental, Rapid Application Development (RAD) and Spiral [48]. After careful deliberation, the software-process model chosen to develop our system is RAD. Table 3.1, compares the software process models listed above.

Table 3.1: Software Methodologies Analysis

| **Factors** | **Waterfall** | **Incremental** | **RAD** | **Spiral** |
|---|---|---|---|---|
| Project Size | Large | Moderate | Small | Moderate to large |
| User requirements | Well defined | Partial defined | To gather requirements | To gather requirements |
| Process Flow | Linear | Parallel | Iterative | Iterative |
| Delivery | One shot delivery | Deliver in increments | Many prototypes | Many prototypes |
| Delivery speed | Slow | Moderate | Rapid | Rapid |

According to Vliet and Hans [49], RAD is a software development methodology and the major concern is developing software in much faster time. RAD uses various structural procedures, computer assisted software engineering tools and also prototyping that is used to describe processes to boost the swiftness of software development.

Non-agile processes such as waterfall may take a long time to develop. During

this time, system requirements may evolve, and costs will be incurred to bring the non-agile processes into conformity with the new requirements. As for RAD, small prototypes can be used to gather more requirements, and normally users will be teased when a part of the system is practical; in RAD additions, requirements can be accommodated easily.

Generally the lifecycle of RAD can be categorized into four phases:

1. Analysis and Quick Design. Once a project has been received, an analysis will be carried out to obtain the basis idea or structure of the proposed system, and then a quick design of the system is created.

2. Prototyping. In this phase, models of the system are built and presented to its users. User feedback is collected, and system models are adjusted according to users' needs. Prototyping also provides a means for acquiring more requirements of the system. This process can be repeated as many times as necessary, until user acceptance levels are satisfactory .

3. Testing. The system is tested to ensure that the functionalities stipulated have been achieved.

4. Implementation. The system is deployed and handed over to the respective users.

Martin states [50], RAD should be used when a application can be modularized and delivered in increments. RAD requires that highly skilled developers are apart of the development process and requires user involvement throughout the projects lifecycle.

In AppXimity, the user requirements are not well defined. Using prototypes allows more requirements to be gathered from the users during the developmental process, resulting in enhanced overall acceptance of the final product and significantly reducing development time in comparison to traditional methodologies like Waterfall.

### 3.3.2 Software Development Tools

With the upsurge in varieties of software development languages and frameworks, it is important to identify the right tool to address any given developmental challenge. Some factors to consider when identifying the right tools include:

*Integration*. How well does the tool integrate with other tools currently being used?

*Overhead*. How steep is the learning curve to be able to use of the tool effectively?

*Applicability*. How generic is the tool? Can it be used in different environments? (e.g., a C# windows application cannot be deployed in the web).

We investigated the following tools: Java [51], PHP [52], Node.js [53], MySQL [54], and MongoDb [55]. In the end, we selected Node.js and MongoDb as the tools to implement the prototype.

MongoDb is an open-source database that stores data in JSON-like documents. Documents in a collection can be non-identical, and denormalization of data is common. Related information is stored together for fast query access. As compared to MySQL, MongoDb maps naturally to modern, object-oriented programming languages, and it removes the complex object-relationalmapping layer that MySQL uses to translate objects in code to relational tables. MongoDB also scales better than MySQL.[2] Since AppXimity is accessible to all users in the various app stores, regardless how geographically dispersed, MongoDB provides better scalability than MySQL with no downtime, and without changing the application logic.[3]

Java provides a solid foundation for application development. Its strengths include threading capabilities, debugging features, and a vast array of libraries; however, Java also has various downsides, because it runs on the JVM it is comparatively slower than other programming languages. Also it more memory-consuming that other native programming languages. Node.js, while relatively new to Java, offers various advantages: it is fast, uses nonblocking I/O API, is easy to scale, and is easy to learn.[4] Node.js is not suited for CPU-intensive applications, but the services in AppXimity are not CPU intensive.

AppXimity deploys various micro-services. As the framework grows, the number of services deployed will increase. Node.js provides the infrastructure to build fast, scalable network applications that are capable of handling a huge number of simultaneous connections.

---

[2]https://www.mongodb.com/compare/mongodb-mysql

[3]https://www.mongodb.com/mongodb-architecture

[4]http://www.infoworld.com/article/2975233/javascript/why-node-js-beats-java-net-for-web-mobile-iot-apps.html

## 3.4 RESTful Web Services

A web service is a collection of open standards and protocols used for exchanging data between software applications or systems [56]. Software applications, irrespective of the programming language in which they are written, can use web services to exchange data over the Internet. Web services based on the REpresentational State Transfer (REST) architecture are known as RESTful web services. In the REST architecture, every component is a resource, and each resource is accessed by an interface using HTTP standards [57].

The following HTTP methods are frequently used in RESTful services [57]:

**GET** — provides read-only access to a resource.

**PUT** — updates an existing resource or create a new resource.

**DELETE** — removes a resource.

**POST** — updates an existing resource or create a new resource.

**OPTIONS** — gets the supported operations on a resource.

Each resource is defined by a Uniform Resource Identifier (URI) and various representations. These representations include, but are not limited to, JavaScript Object Notation (JSON) and Extensible Markup Language (XML). XML follows a set of standards for data communication over networks between devices and is parsable [58]. An advantage of XML is its structure: data is added or removed in a predictable format; hence, the parsing will not be affected even if the data changes. One drawback of XML is its size, because it includes the overhead of defining how the document is formatted. JSON is a text-based data-interchange format derived from Javascript [59]. It is formatted using key-value pairs. When compared to XML, it has a lower overhead, due to the fact that it focuses more on content and less on formatting.

Since the client in AppXimity is a mobile device, and may rely on mobile data consumption, the web services will use JSON, as its data set is smaller in size than that of XML. Each resource was built as a micro-service, and these services implement a RESTful architecture. The framework provides an API Gateway that orchestrates the routing of service requests to the appropriate micro-services. The API Gateway handles all requests from the client, invoking multiple micro-services and aggregating

the results. These results are returned to the client using web protocols such as HTTP and a lightweight data-interchange format (JSON). The micro-services architecture is depicted in Figure 3.3.

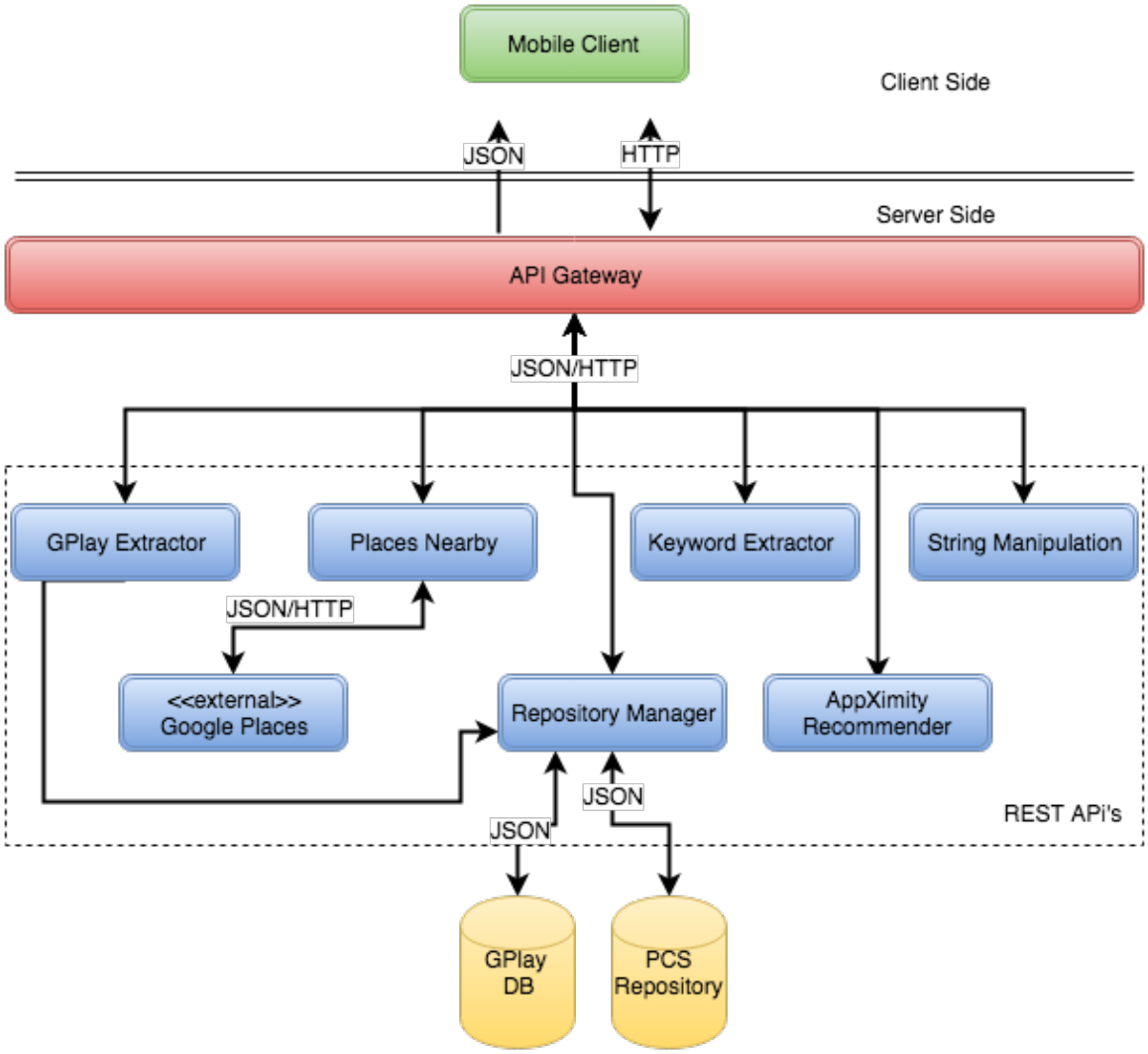


Figure 3.3: AppXimity Micro-services Architecture

## 3.5 User Interface Design

### Building the Client with Android Studio

We developed the Client and the Context-Aware App widget using Android Studio. Android Studio is the official Integrated Development Environment (IDE) for Android

app development and is based on IntelliJ IDEA.[5] Android Studio offers features that enhances productivity when building apps, such as a fast and feature-rich emulator; lint tools to catch performance, usability, version-compatibility, and other problems; and a unified environment for development for all Android devices. Android Studio projects contain one or more modules with source-code files and resource files. Types of modules include Android app modules, Google App Engine modules, and Library modules. Each app module contains the following folders:

A manifest folder. Contains the AndroidManifest.xml file.

A java folder. Contains the Java source code files, including JUnit test code.

A res folder. Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

The graphical user interface for an Android app is built using a hierarchy of View objects and ViewGroup objects. View objects are UI widgets such as text fields and buttons. ViewGroup objects are invisible view containers that define how the user view is laid out, such as in a grid or a vertical list. A screenshot of the Android Studio IDE is depicted in Figure 3.4. The Context Aware Widget source is in Appendix A.1 and A.2.

The client component deploys various services on the user's smartphone. These services include sensing contextual data, which can include location, time, user ID, and apps installed. The client then sends the data to the API Gateway server for processing. Once the server returns the results to the client, it renders the results on the user's smartphone through the Context-Aware App widget.

### AppXimity Widget

The widget is divided in the following sections:

*AppXimity Suggests.* This grid houses the relevant installed apps that the user may need given their context.

*AppXimity Recommends.* This grid allows users to discover apps that they may be useful in their given context but are not currently installed.

Figure 3.5 depicts the AppXimity Widget.

[5]https://developer.android.com/studio/intro/index.html

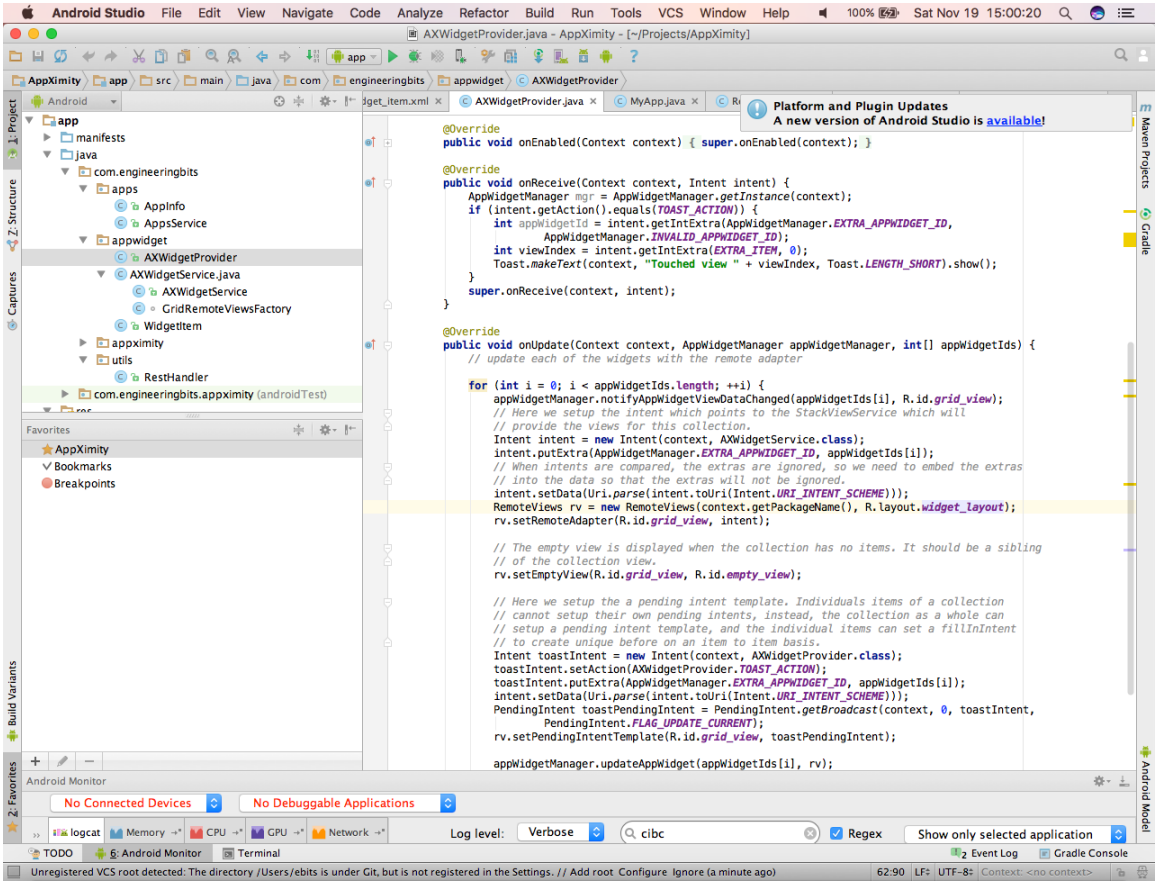


Figure 3.4: Android Studio IDE

### Settings

*Use context.* This allows the user to toggle the use of their context information on and off. If this option is turned on, then the recommender uses the users contextual data to personalize app recommendations.

*Context Sphere*: This allows the user to view and edit the context that is stored in their context sphere.

The settings page is depicted in Figure 3.6.

## 3.6 Managing User Location

One distinctive capability of mobile applications is location awareness. Mobile users take their devices with them everywhere. Thus, adding location awareness to apps can improve the users contextual experience. AppXimity implements the Google Play

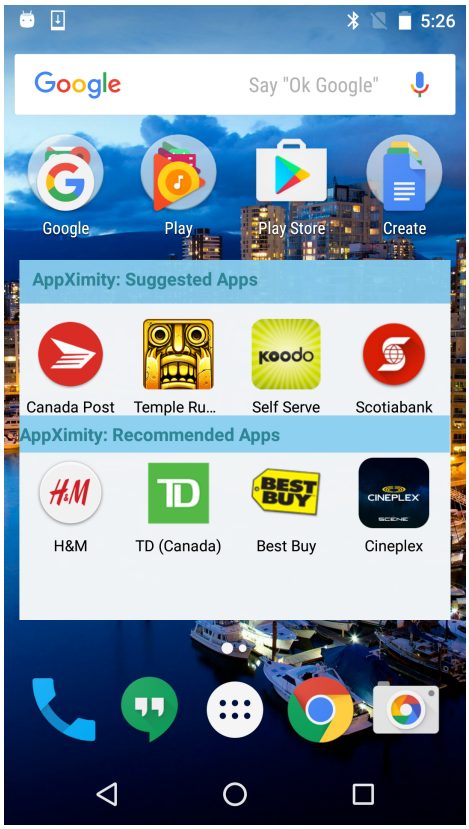


Figure 3.5: AppXimity Widget

service to monitor location, geofencing, and activity recognition.

### Challenges in Determining User Location

A user's smartphone position changes frequently, and often rapidly. Hence, obtaining accurate and precise user-location data is a challenging task. Some of its challenges include the following:

- Variety of location-data sources. Location data can be gathered from GPS, Cell-ID, and Wi-Fi. Choosing which source to use involves trade-offs in reliability, battery-efficiency, speed, and accuracy.
- Movement. The system needs to account for user movement by re-estimating the user location periodically.
- Varying accuracy. Location estimates acquired from various sources may not be identical. Older location data obtained from a more reliable source may be more accurate than newer data from a less reliable source.

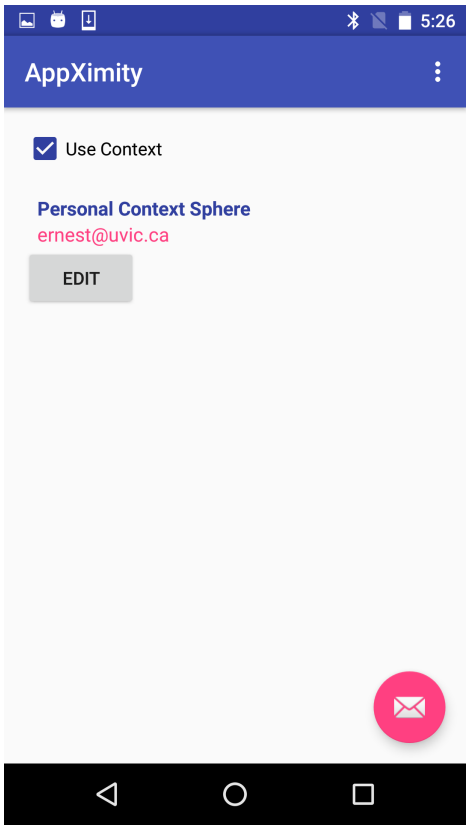


Figure 3.6: AppXimity Settings

### Managing User Location in AppXimity

AppXimity uses Googles Geofencing API to aggregate information about a users current location and the proximity of places of interest, which is then used to monitor and trigger events when the user and a place of interest are both within a specified radius. When user's latitude and longitude are obtained, a radius is set around that location; this is denoted as a virtual fence. The context-aware widget updates to show all the apps relating to places of interest within that virtual fence. Once the user exits that virtual fence, another request is made to update the widget, and a new virtual fence is set. The virtual fence is set to a radius of 50 meters when Wi-Fi is available, and 150 meters otherwise.

Geofencing is useful when a user is walking and constantly needs to have the widget updated; however, location updates are not needed when the user is cycling, driving, or running. Google's Activity Recognition API was implemented to detect the kinds of activity in which a user is engaged. This is accomplished by periodically waking up the device and reading a short burst of sensor data. The use of low-power

sensors for this operation minimizes power usage. If the detected activity involves a speed of movement faster than that of walking, the client does not request that the widget be updated. Figure 4.5 illustrates the location model used when the client requests location updates. The Java code for getDetectedActivity() is as follows:

```
public String getDetectedActivity(int detectedActivityType) {
    Resources resources = this.getResources();
    switch(detectedActivityType) {
        case DetectedActivity.IN_VEHICLE:
            return resources.getString(R.string.in_vehicle);
        case DetectedActivity.ON_BICYCLE:
            return resources.getString(R.string.on_bicycle);
        case DetectedActivity.ON_FOOT:
            return resources.getString(R.string.on_foot);
        case DetectedActivity.RUNNING:
            return resources.getString(R.string.running);
        case DetectedActivity.WALKING:
            return resources.getString(R.string.walking);
        case DetectedActivity.STILL:
            return resources.getString(R.string.still);
        case DetectedActivity.TILTING:
            return resources.getString(R.string.tilting);
        case DetectedActivity.UNKNOWN:
            return resources.getString(R.string.unknown);
        default:
            return resources.getString(R.string.unidentifiable_activity,
                detectedActivityType);
    }
}
```

## 3.7 Summary

This chapter presented the overall design of AppXimity. The design includes both user interface and system design. This chapter also discussed the key factors considered when conceptualizing the architecture of AppXimity. The next chapter discusses the

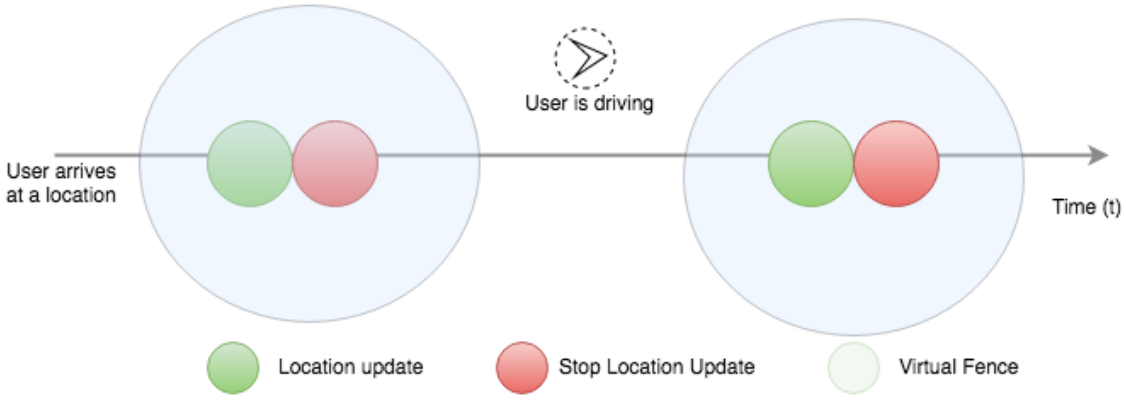


Figure 3.7: Managing User Location

AppXimity micro-services in detail.

# Chapter 4

# AppXimity Services

This chapter presents the AppXimity micro-services implementation. In this chapter, the various services that bind the AppXimity architecture are explored in detail. Each component of the application is implemented as a micro-service that can be accessed using HTTP methods such as GET and POST. For simplicity, the complete source code for selected services are found in the Appendix A, access to the entire repository is found at the AppXimity-Repo[1].

## 4.1 AppXimity Micro-services Overview

In Section 2.7, we argued that the micro-service architecture yields great benefits for IoT applications: in particular, it enables the continuous development of a set of manageable services that are easier to understand, maintain, deploy, and scale. The AppXimity application comprises a set of micro-services that run independently in their own Docker container. These micro-services expose their interface using a RESTful API and communicate using HTTP as previously depicted in Figure 3.3.

For the purpose of app recommendation, AppXimity provides an API Gateway that orchestrates the routing of service requests to the appropriate micro-services. The API Gateway handles all requests from the client, invokes multiple micro-services, and aggregates the results. These results are returned to the client using web protocols, such as HTTP, and the lightweight data-interchange format JSON.

[1] `http://dev.engineeringbits.com:9444/ernest/appXimity-services.git`

### 4.1.1 Building Micro-Services with Docker

When small sets of micro-services are created, they will spread across multiple hosts; it can be tedious to keep track of which hosts are running which services. As the micro-service architecture scales, the number of hosts will grow as well. Moreover, if services are implemented in different programming languages, the deployment of each service will require a different set of frameworks and libraries, which increases the complexity of deployment. Micro-services can be deployed using Docker[2] containers to alleviate such challenges.

Docker containers wrap a piece of software in a complete file system that contains everything it needs to run. Docker allows one to package an application with all its libraries and dependencies into a standard unit, known as a container [60]. It automates the deployment of Linux applications within these software containers. These containers will run on any Linux machine, irrespective of customized settings on that machine. The Docker architecture is depicted in Figure 4.1.

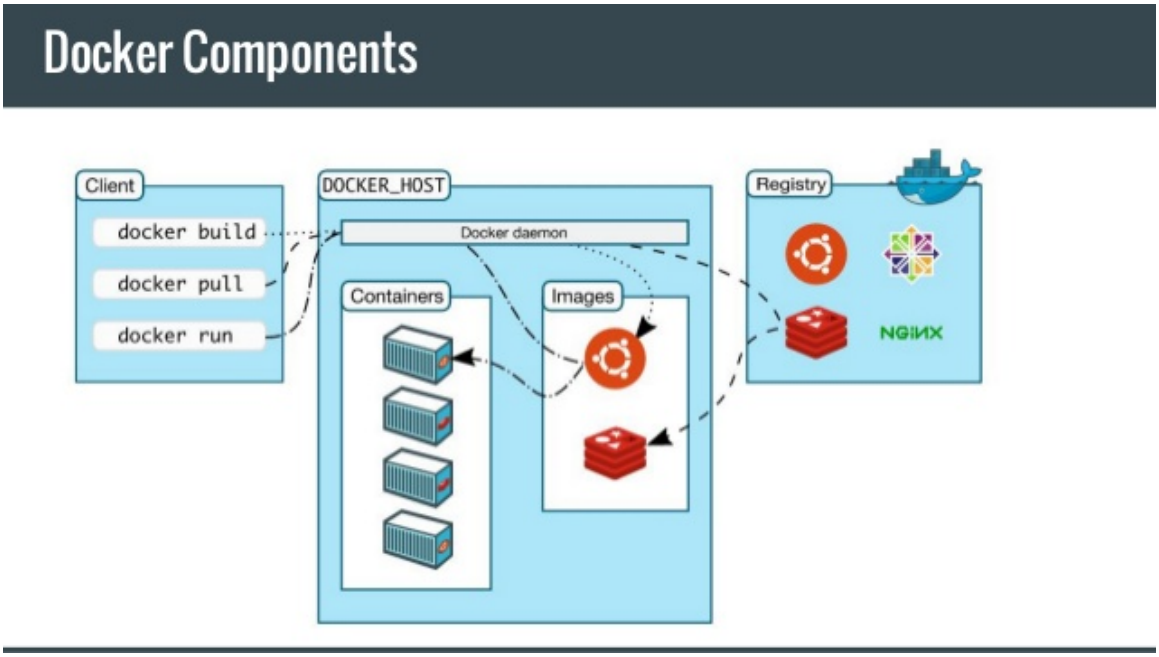


Figure 4.1: Docker Architecture (Courtesy Docker)[3]

[2]https://www.docker.com/what-docker
[3]https://docs.docker.com/engine/understanding-docker/

### 4.1.2 Deploying AppXimity Micro-services in Docker Containers

We first create a Docker Image from the micro-service. In order to allow the image to be deployed anywhere that supports Docker, a Dockerfile is created; a Dockerfile is a recipe that instructs the Docker engine on how to build the image. The following code snippet shows the contents of a *Dockerfile*:

```
# Use Node v4 as the base image.
FROM node:4

# Add everything in the current directory to our image, in the 'app'
    folder.
ADD . /app

# Install dependencies
RUN cd /app; \
    npm install --production

# Expose our server port.
EXPOSE 9444

# Run our app.
CMD ["node", "/app/index.js"]
```

After creating the *Dockerfile* called *repository-manager*, the following commands are executed:

```
docker build -t repository-manager . # Builds a new image
docker run -p 49160:9444 -d ebits/repository-manager
```

- *FROM node:4*, tells the docker engine to use version 4 of Node.js to create the image;
- *docker build*, tells the docker engine we want to create a new image;
- *-t repository-manager*, tag this image with the tag repository-manager. The

image is referred to by this name;

- *docker run*, creates and starts a container in one operation;
- *-p 49160:9444*, map the host port 9444 to the container port 49160;
- *-d ebits/repository-manager*, runs the container in the background mode.

## 4.2 AppXimity Services Explained

### 4.2.1 Personal Context Sphere Service (PCS)

The Personal Context Sphere (PCS) stores the personal interests of each user. These interests are represented by keywords and can be retrieved through various media (e.g. JSON or XML)[26]. To achieve a greater degree of fine-grained personalization, users' app usage and frequent locations are updated in their PCSs. Having this data, we can truly personalize app management. When AppXimity deduces a pattern based on usage of an app in a given location, this pattern is added to the users PCS. Learning in AppXimity enables the recommender to recommend apps of similar categories (e.g. Apple Music, Netflix or Hulu).

In AppXimity, the Repository Manager handle the PCS using two methods. PCSReader() reads the users preferences from their PCS, and PCSUpdate() updates the users preferences. Additional keywords relevant to app recommendation are added to the users PCS; these include apps that they frequently use and locations that they frequent. Below is a code snippet of a user's PCS in xml format [26].

```
<?xml version="1.0" encoding="UTF-8"?>
<pcs>
<!-- From the Context Ontology by Villegas, 2013-->
<pwc:user>Aaron</pwc:user>
<gc:geoLocation type="country">Canada</gc:geoLocation>
<gc:geoLocation type="city">Victoria</gc:geoLocation>
<!-- From the Context Ontology for AppXimity, 2016-->
<!-- Personal Information-->
<pi-lan language1="English" language2="French">English</pi-lan>
pi-gender>M</pi-gender>
<pi-age>Adult</pi-age>
```

```
<music>Reggae</music>
<music>Soca</music>
<sports>Tennis</sports>
<sports>Body Building</sports>
<technology>iOS</technology>
<technology>Android</technology>
<!-- New additions to the PCS <app> and <placeInterest>
<app>com.microsoft.office.outlook</app>
<app>com.netflix.mediaclient</app>
<app>com.imangi.templerun</app>
<placeInterest>33.630481:-117.856796</placeInterest>
<placeInterest>48.428421:-123.365644</placeInterest>
</pcs>
```

### 4.2.2 AppXimity Recommender Service (ARS)

The app recommender infers which apps a user may need given their current contextual data. There are two types of recommendations:

1. The coarse-grained app recommender. This recommender does not use the user's PCS for app recommendations. It only uses location data to suggests nearby vendor services with available apps. Hence, the course-grained recommender does not provide a truly personalized experience.

2. The fine-grained app recommender. This recommender uses the contextual data from the users PCS to provide a more personalized recommendation. All app recommendation requests received by the AGS are forwarded to this service. If the personalization option is turned on, this service will call the PCS to retrieve the users preferences, which are then added to the list of keywords to be compared by the SMS. The JSON response of this service returns a filtered list of the relevant installed apps and a list of recommended apps (i.e., apps that are not installed but may be needed by the user).

A sample JSON response is listed below:

```
            {
  "suggests": [
```

```
        { "packagename": "com.wholefoods.wholefoodsmarket", "icon":
            "//lh3.googleusercon-DPjt" },
        { "packagename": "com.hm.goe", "icon": "//lh3.googleusercon-DPjt"
            }
    ]}

"recommends": [
        { "packagename": "ca.walmart.ecommerceapp", "icon":
            "//lh3.googleusercon-DPjt" },
        { "packagename": "com.scotiabank.mobile", "icon":
            "//lh3.googleusercon-DPjt" }
    ]}
```

### 4.2.3 Google Play Extractor Service (GPES)

Google Play is the official app store for Android smartphones and tablets, where applications developed by Google and by third-party developers are available for download and installation. Some apps are free while others must be purchased. Applications can be tailored to target specific users based on the features of their hardware, such as a front-facing camera for video calling or a motion sensor for playing motion-dependent games. A sample app listing of the Google Play store is depicted in Figure 4.2.

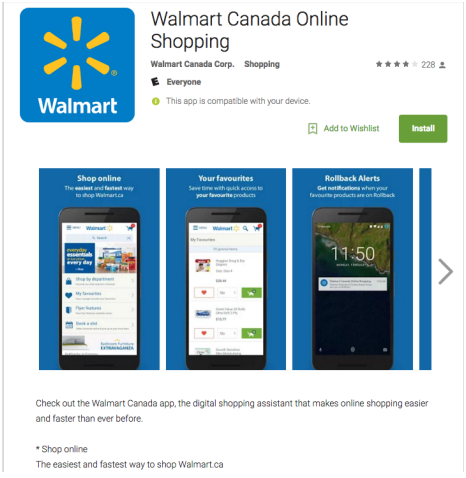


Figure 4.2: Walmart Canada (Courtesy Google Play)[4]

[4]https://play.google.com/store/apps/details?id=ca.walmart.ecommerceapp&hl=en

Google does not provide an official API for requesting app data from the Google Play store. The limited amount of metadata about an app that is stored on a smartphone is not sufficient to accurately associate the app to the respective businesses nearby or to make app recommendations to the user based on their personal context.

This RESTful web service uses web scraping, which is a technique of extracting data from websites using HTTP. In implementation, it uses a Node.js library, by Facundo Olano, to scrap application data from the Google Play store.[5] All responses from this API are in JSON format. The following are some of the methods that are exposed by this API:

*app*: Retrieves the full details of an application.

*list*: Retrieves a list of applications from one of the collections at Google Play store.

*search*: Retrieves a list of apps that results of searching by the given term.

*similar*: Returns a list of similar apps to the one specified.

When a package name is passed to the *app* endpoint below, the full app data will be returned in JSON format. This JSON string includes, but is not limited to, title, summary, price, free/pay, score, reviews, developer, develop email, developer website, genre, genre id, and description.

```
GET http://localhost:3000/gplay/app/ca.walmart.ecommerceapp
```

The following JSON string is a sample successful *200* HTTP response. For all other unsuccessful requests, a *500* Internal Server Error will be returned.

```
{
    "title": "H&M",
    "summary": "The H&M app Your Fashion Destination!",
    "icon": "//lh3.googleusercontent.com/hlWOyB9-DUZFymfyDcxX37WhZPjt
    -6zrFxXi1LlMAH3gAA1VbJpNiucHv7pYtdBIz6GN=w300",
    "price": "0",
    "free": true,
```

[5]https://github.com/facundoolano/google-play-scraper

```
    "genre": "Lifestyle",
    "genreId": "LIFESTYLE",
    "description": "The H&M app Your Fashion Destination!Browse,
    shop and stay on top of the latest trends, anytime and anywhere with
       our easy-to-use app. With the H&M application: Browse the latest
       fashion.
    "url":
       "https://play.google.com/store/apps/details?id=com.hm&hl=en&gl=us",
    "appId": "com.hm"...
}
```


### 4.2.4 Keyword Extractor Service (KES)

Keyword extraction is the provision of descriptive metadata in the form of keywords that best conveys the meaning of a document. As noted in Section 4.2.3, the description returned by the Google Play Extractor Service provides a valid description of an application. The essence of this description can be summarized in a few keywords. These keywords can then be compared to the keywords in users' PCSs, as well as to metadata about businesses in the area. For example, the description that is returned from the Walmart app includes keywords such as *walmart*, *shopping*, and *shop*. These extracted keywords can then be used to discover relations with keywords stored in users' PCSs and nearby businesses.

Various natural keyword extractors are available, both commercial and open source. We reviewed the following keyword extractors was carried out: Alchemy [61], [62], TextRank [63], and RAKE [64]. We selected RAKE for the following reasons: it is more computationally efficient than the others, achieves higher precision and comparable recall scores than the others, and does not require a training set [64].

**RAKE Algorithm [64]**

Rapid Automatic Keyword Extraction (RAKE) algorithm, extracts keywords from text, by identifying non-stopwords and scores these phrases. The input parameters are a list of stop words, a set of phrase delimiters, and a set of word delimiters. RAKE uses stop words and phrase delimiters to separate text into candidate keywords, which are sequences of content words as they occur in the text. After candidate keywords have been identified, and the co-occurrences of words is established, a score is calculated

for each candidate keyword and defined as the sum of its member word scores. RAKE then looks for pairs of keywords that adjoin on another at least twice in the same same document and in the same order. A new candidate word is then created and as a combination of those keywords and new keyword score is the sum of its member keyword scores. Finally, the top scoring candidates are selected as the keywords for the document and the keywords are returned to the caller. Our implementation of the RAKE algorithm is in Appendix A.7.

### 4.2.5 Keyword Extraction in AppXimity

The description returned by the Google Play Extractor Service is passed to this service for keyword extraction. AppXimity then passes the extracted keywords from the description to RAKE. RAKE is a powerful instrument for finding multiword phrases containing frequent words; however, because the description returned by the app store is generally concise, the RAKE algorithm was modified to extend the number of characters can form a word, the number of words that can form a phrase, and the number of keywords that can appear in a text.

The following endpoint implements HTTP *POST*, in its body a JSON object is sent, which contains the various parameters used to customize the keyword extraction.

```
POST http://localhost:3000/keywords/
```

The parameters to this service request are:

*text*: The text to extract keywords from.

*phrasemin*: Amount of words each phrase should contain.

*keyappearsmin*: The amount of times the keyword should appear in the text.

*wordmin*: The minimum amount of characters that a word should have.

*keywords*: The top percent of keywords to return.

The code snippet below depicts the request body.

```
{
    "document":{
      "text":"Check out the Walmart Canada app, the digital shopping
         assistant...",
      "phrasemin":"2",
      "keyappearsmin":"2",
      "wordmin":"5",
      "keywords":"3"
}
```

A successful *200* HTTP response, will return a JSON object with the top 3% of the keywords that best describes the given document. In the Walmart example, the response can be seen in the code snippet below. For unsuccessful requests, a *500* Internal Server Error will be returned.


```
{"keywords":[{"keyword":"makes online shopping easier","score":"13.0"},
{"keyword":"digital shopping assistant","score":"8.66666666667"},
{"keyword":"putting money back","score":"8.5"},
{"keyword":"walmart canada app","score":"8.0"},
{"keyword":"easily access walmart ","score":"8.0"},
{"keyword":"quickly scan items","score":"7.66666666667"},
{"keyword":"weekly flyer items","score":"7.5"},
{"keyword":"walmart app","score":" 5.0"},
{"keyword":"weekly flyer","score":" 4.83333333333"},
{"keyword":"favourite items ","score":"4.66666666667"},
{"keyword":"shop walmart ","score":" 4.5"},
{"keyword":"quick access","score":" 4.5"},
{"keyword":"time back","score":"4.5 "},
{"keyword":"* flyer features ","score":" 4.33333333333"},
{"keyword":"* shop online ","score":" 4.33333333333"}]}
```

### 4.2.6 Places Nearby Service (PNS)

The PNS service implements the external Google Nearby Search Request web service. A Nearby Search request, searches for places within a specific area. The search request can be refined by appending various keywords or specifying the type of place. Google's Nearby Search request endpoint is as follows:

```
https://maps.googleapis.com/maps/api/place/nearbysearch/output?parameters
```

The required parameters for the PNS service are: the application's API key, location, and radius. The output can either be in *xml* or *json* format.

Although the Google Nearby Search API, can be directly accessible from the client application, a service was created for it. This enables a scalable service that can also be accessed from other backend services, without the need of having the request initiated by the client.

This service can be reached via the following endpoint:

```
POST http://localhost:3000/placesNearby
```

When this API is initiated, the latitude, longitude, API key, and radius are sent to this service. A call is then made to the Google Nearby Search API. The code snippet below depicts the request body sent to this service.

```
{
"location":{"lat":"48.455173","radius":"500",
"long":"-123.374385","apikey":"XXXXXX"},}
```

The function below configures the URL to fetch nearbyPlaces for a given location, from Google's Nearby Search API.

```
//Configure the URL to fetch nearbyPlaces from Google's Nearby Places API
function retrieveNearbyPlacesURL(userLocation){
    return AppConfig.GOOGLE_PLACES_API_URL+"location=
    "+userLocation.lat+","+userLocation.long+"&radius=
    "+AppConfig.GOOGLE_PLACES_API_RADIUS+"&key="
    +AppConfig.GOOGLE_PLACES_API_KEY;
}
```

The Javascript function below, accepts the configured URL from the retrieveNearbyPlacesURL() function and fetches the nearby places, in a user's proximity. The full source code is in Appendix A.5

```
//HTTP request to fetch the nearbyPlaces in proximty of the user
var nearbyPlaces=function retrieveNearbyPlaces(nearbyPlacesURL){
    return new Promise (function (resolve,reject){
        var request = require('request');

        request(nearbyPlacesURL, function (error, response, body) {
            if (!error && response.statusCode == 200 &&
            parseToJSONOBJ(body).status=="OK" ) {
                resolve(body);
            }
            else{
                reject(parseToJSONOBJ(body).status);}
```

A successful *200* HTTP response, will return the minimum JSON response that contains: the status determines if the query executed successfully, and results contains an array of places, with information about each. For all other unsuccessful requests, a *500* Internal Server Error will be returned. The following code snippet, shows the response of a successful request to this service:

```
    {
       "geometry" : {
          "location" : {
             "lat" : 48.4555012,
             "lng" : -123.3742274
          },

       "icon" : "...",
       "id" : "..",
       "name" : "Walmart",
       "opening_hours" : {
          "open_now" : true,
          "weekday_text" : []
       },
       "types" : [ "department_store", "store", "point_of_interest",
```

```
        "establishment" ],
      "vicinity" : "3460 Saanich Road, Victoria"
   },
```

### 4.2.7 String Manipulation Service (SMS)

SMS finds the degree of similarity between strings using Dice's Coefficient. The JSON response of the PNS described in Section 4.2.6, includes the name of the business and also the keywords associated with that business. These fields are sent to this service for comparison with the app's name, app's package name or keywords in a user's PCS, to determine if the app should be returned as an app of interest, to the user. The source code for String Manipulation Service is in Appendix A.4.

When using Dice's Coefficient as a string similarity measure, the coefficient can be calculated for two strings, x and y using bigrams as follows [65]:

$$s = \frac{2n_t}{n_x + n_y} \tag{4.1}$$

where $n_x$ is the number of bigrams in string x, $n_y$ is the number of bigrams in string y and $n_t$ is the number of character bigrams found in both strings. For example, Table 4.1, finds the similarity between the name of businesses nearby and the name of the installed app. To identify an acceptable percentage criterion, we manually calculated the similarity score for all businesses at a given location, and 70% was the average. Further research and fine-tuning of other variables could be explored to provide a more sound percentage criterion. If there is a 70% or greater match, that app is returned to be displayed in the widget. The source code for Dice's Coefficient is in Appendix A.6

Table 4.1: App Name and Place Nearby Similarity

| **App Name** | **Nearby Place** | **Similarity** |
|---|---|---|
| Walmart | Walmart, | 1 |
| Scotiabank | Scotiabank | 1 |
| Best Buy | Best Buy | 1 |
| Banking | CIBC Branch & ATM | 0.125 |
| RBC Mobile | RBC Royal Bank & ATM | 0.25 |

The first three records in Table 4.1, show the ideal case, where the app name and nearby business name are identical, however the remaining two similarity scores are poor. For example, the app name for Canadian Imperial Bank of Commerce is *Banking*, and the name that is returned by Google Nearby Places search is *CIBC Branch & ATM*.

If the initial similarity score is less than 70% the app name is measured, various other attributes, such as package name, Google Play data will be extracted to determine a possible match. The steps below outlines the algorithm for determining an app association with a nearby business.

```
GET App Name
GET Places Nearby
COMPUTE Similarity Score for elements in places nearby array with App Name
IF Similarity Score >= 70 THEN
    RETURN success
ELSE IF Similarity Score < 70 THEN
    INIT Package Name Array
    GET app Package Name
    SPILT Package Name as Pack Name using period separator delimiter
    SET Pack Name in Package Name Array
    COMPUTE Similarity Score for each element in Package Name array with
       Places Nearby
    IF
       Similarity Score is >= 70 THEN
       RETURN success
    ELSE IF Similarity Score <70 THEN
       IF Pack Name is substring of elements in Places Nearby THEN
          RETURN success
       ELSE IF Pack Name is not a substring of elements in Places Nearby
          THEN
          GET Pack Description keywords as Package Name description
             keywords.
          GET App Name
          IF App Name OR Pack Description keywords is a substring in
             Package Name array THEN
             RETURN success
          ELSE
```

```
            RETURN failure
         END IF
      END IF
   END IF
END IF
```

Applying the algorithm above, lets revisit the Banking app. Table 4.1 shows the Places Nearby array includes CIBC Branch & ATM, when the app name *Banking* was used as a similarity measure, it scored 0.25. However, the package name for *Banking* is *com.cibc.android.mobi*, creating an array from this we get {com,cibc,android,mobi}. The algorithm was fine tuned to ignore common words such as app, com, mobile,mobi, android. CIBC is a substring of CIBC Branch & ATM, therefore this app is returned as an app of interest.

### 4.2.8 API Gateway Service (AGS)

AGS is the single point of entry into the back-end services and built as a Facade Pattern. The API Gateway handles all requests from the client, invokes multiple micro-services, and aggregates the results. Results are returned to the Client using JSON format.

The API Gateway encapsulates the internal structure of the application. Instead of invoking each service that is required to complete the app recommendation, the client sends its request to the gateway. This reduces the number of round trips between the client and the various micro-services, as well as simplifying the client code. When a user-location event is triggered in the client, the relevant app list needs to be updated to reflect the user's new environment. The AGS service can reached via the following endpoint:

```
POST http://localhost:3001/appximity
```

The request parameters are sent to this service using HTTP POST. These parameters can include user location data, apps installed, device id, date and time and pcs id. When this service receives a request, it then calls the appropriate services to handle various parts of the request. For example, the Places Nearby service is called, and the latitude and longitude of the user's current location is passed, it then returns all the places of interest nearby. The API Gateway aggregates all the results from the

various services and returns the appropriate response, such as the relavant app list, to the Client. The following is a snippet of the data that is sent in the request body to this service:

```
{
    "location":{"lat":"45.311217","long":"-75.907109"},
    "appsInstalled":[
    {"appName":"H&R Block", "appPackageName":"com.aexonic.hnrblock"},
    {"appName":"CIBC", "appPackageName":"com.cibc.android.mobi"}
]
   "datetime":"1479678653723",
   "deviceid":"8734faae3344",
   "pcsid":"ernest@uvic.ca",
   "recommender":"fine"
}
```

### 4.2.9 Repository Manager Service (RMS)

The repository manager is responsible for performing Create, Read, Update, and Delete (CRUD) operations on data that is persisted to the repository. This service handles the reading and updating the user's PCS via *PCSReader()* and *PCSUpdate()* respectively.

This service updates the GPlay repository. The GPlay repository is a local data store to house the metadata for apps in the Google Play Store. The GPlay repository is evaluated for app metadata before making an external call to the Google Play web scrapper. Each record in the GPlay repository is updated with the latest data from the Google Play web scrapper every seven days. This is done to ensure that data in the Google Play Store and the local repository remain consistent.

The following endpoint is used to update the GPlay Repository for a given app:

```
PUT http://localhost:3005/repositorymanager
```

When this API is initiated, the request body includes the repository to update and the data to be updated. The code snippet below depicts the request body sent to this service, to update the GPlay data about an app.

```
{
```

```
  "repo":"gplay",
  "keywords":"walmart....",
  "icon":"http://icon....."
  "title":"Walmart"
  "appname":"Walmart"
  "packagename":"ca.walmart.walmart.ecommerceapp"

}
```

If the CRUD operation was successful, an HTTP 200 success response is returned. Otherwise an HTTP 500, internal server error is returned.

## 4.3 Putting AppXimity Services together

The Sequence Diagram in Figure 4.3 shows the interaction between services and in what order. The order of the actions in the sequence diagram are as follows:

0 send location and installed apps data to API Gateway over HTTP.

1.1 send location data to Places Nearby over HTTP.

1.1.1 send location data to Google Places API over HTTP.

1.1.2 return places nearby a given location to Places Nearby in JSON format.

1.2 return places nearby a given location to API Gateway in JSON format.

2.1 send installed apps data to GPlay Extractor over HTTP.

2.2 return app details to API Gateway in JSON format.

3.1 send user's PCS id to Personal Context Sphere over HTTP .

3.2 return user preferences to API Gateway in JSON format.

4.1 send the JSON data that was returned from places nearby, app details and user preferences to Keyword Extractor.

4.2 return the extracted keywords for each object in 4.1 in JSON format.

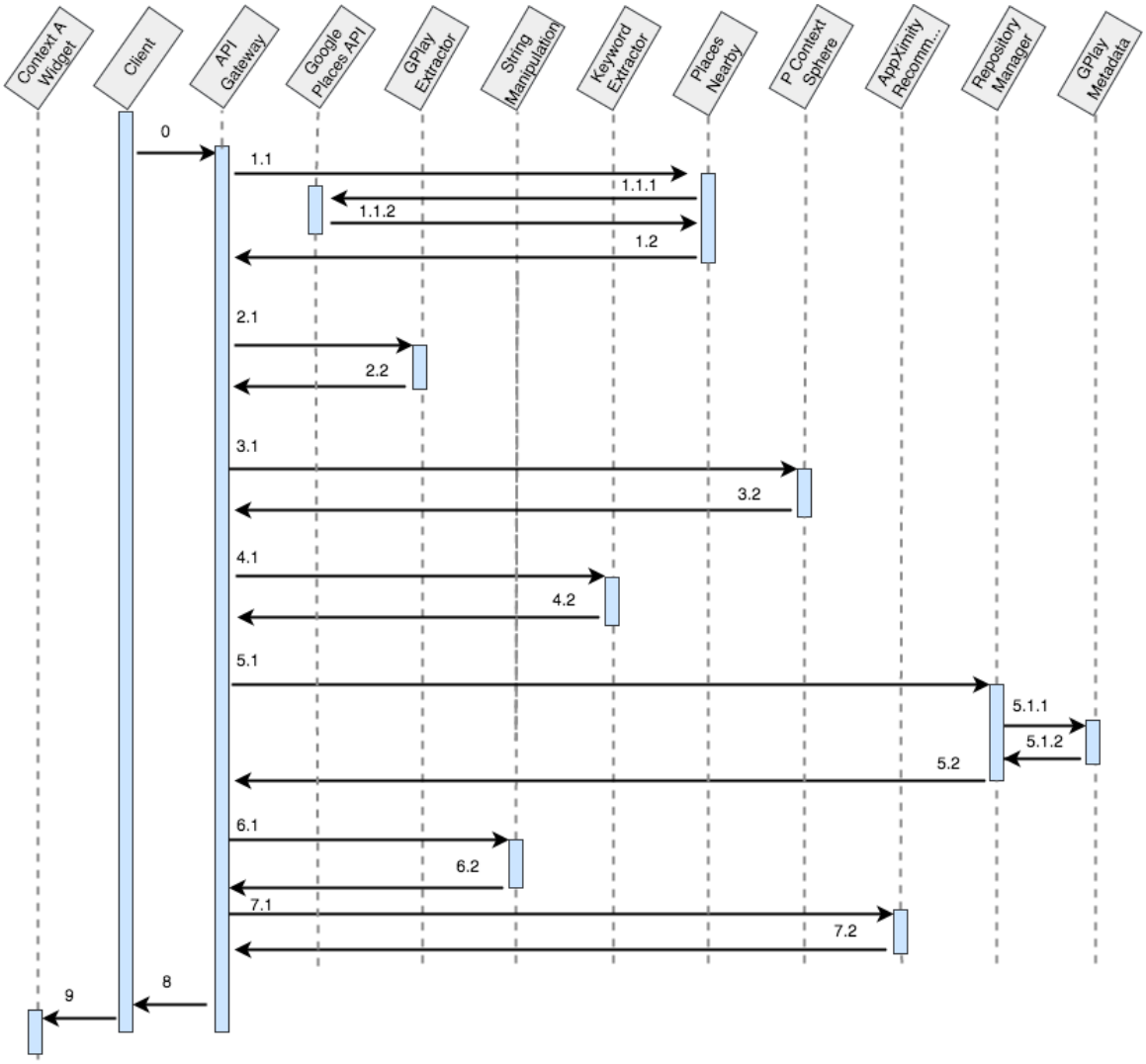


Figure 4.3: AppXimity Sequence Diagram

5.1 send the extracted keywords for the app details to the Repository Manager in JSON format.

5.1.1 store the app meta-data in GPlay Repository.

5.1.2 return the keywords that represent each app to the Repository Manager in JSON format.

5.2 return the keywords that represent each app to the API Gateway in JSON format.

6.1 send the keywords for places nearby, app details, and user preferences to

the String Manipulation.

6.2 return the similarity score for keywords to the API Gateway.

7.1 send the similarity score, user preferences and places nearby keywords to the App Recommender.

7.2 return list of suggested and recommended apps to the API Gateway.

8 return the app list to the Client.

9 list the suggested and recommended apps.

## 4.4 Summary

This chapter explored the various services that bind the AppXimity architecture in detail. Each component of the application is implemented as a micro-service that can be accessed using HTTP methods such as GET and POST. Each service is accessed by its unique endpoint and uses JSON as its data interchange format. AppXimity implements a single entry point, known as the API Gateway. This gateway is responsible for handling all requests from the client and calling the appropriate micro-service to execute a task.

# Chapter 5

# Evaluation

This chapter presents two experiments to illustrate the usefulness and acceptance of AppXimity. We considered several quality attributes of AppXimity from software engineering and end-user perspectives, including functional correctness, accuracy, robustness, usefulness and usability. However, we did not perform formal user studies.

The experiments discussed here have been anonymized to protect the privacy of the participants. Experiment 1 is used to illustrate a first-time user to AppXimity. In this experiment, we assume that AppXimity has not learnt the users frequent locations and app usage. In Experiment 2, the user continuously uses AppXimity and, thus, AppXimity has learnt the users frequent locations and app usage.

## 5.1 Experiment 1

### Persona

Sarah, 22, is a Physics major at the University of Victoria and works as a part time waitress at Cafe Bistro. Studying and working part time is tedious, so she depends on coffee daily to stay alert. She is also a fashionista and enjoys browsing and shopping for the latest styles. While she enjoys eating, her choices are limited, she is a vegetarian.

### Scenario

It is a sunny Saturday in Victoria BC. Sarah has the day off from work and school and decides to take a stroll to Uptown Victoria at 12:00 pm. There she plans to enjoy a cup of coffee with friends and browse various sale items at the nearby boutiques.

**Personal Context Sphere**

Sarah's PCS has keywords such as: *coffee*, *shopping*, *vegetarian*, *sales*, *fashion*, and *android*, *technology* and *tennis*.

**Sensed Context**

*Time:* 12:00 p.m.
*Phone:* Google Nexus 5
*Weather:* 21 Celsius, Sunny day
*Location:* XXX Saanich Rd, Victoria, BC V8Z XXX

**Installed Apps**

*Walmart, H&M, BestBuy, TD Bank, Starbucks, Canada Post, McDonalds...*

### 5.1.1 Analysis

Sarah's location data returned a list of nearby businesses, including Canada Post, Walmart, H&M, Urban Planet, Forever 21, Whole Foods, Scotiabank, RBC, Starbucks, Best Buy, Sizzling Tandoor, Brown Social, and Noodle Box. Sarah had enabled the fine-grain recommender, which uses data from her PCS to offer suggestions and recommendations.

From the installed apps, AppXimity suggested Walmart, BestBuy, TD Bank, Starbucks, and H&M. AppXimity also recommended two uninstalled apps: Whole Foods and Forever 21. Sarah commented that such recommendations aided in business discovery. Whole Foods was a new addition to the shopping complex, and she had been unaware of its existence. Urban Planet, Sizzling Tandoor, Brown Social, and Noodle Box did not leverage the benefits of AppXimity, because they currently have no apps available in the app store.

Although RBC and ScotiaBank have apps available, they did not match any of the keywords in Sarah's PCS. McDonald's has an app, and given that it was 12:00 p.m. this should have been listed; it was not, however, because the keyword vegetarian was not associated with the McDonald's app. Sarah suggested that it would be helpful if AppXimity recommended apps that other users with similar PCS's used in that given location.

## 5.2 Experiment 2

**Persona**

Mark, 21, is an Kinesiology major at University of Victoria. He enjoys writing articles on exercise, health and nutrition. He also enjoys working out and is currently a personal trainer at Goodlife Fitness. For leisure, Mark enjoys watching movies on Netflix and listening to Soul music on Spotify.

**Scenario**

It is a cloudy winters eve, in Victoria BC. Mark is at home, and hopes to watch a great documentary about Body Building while drinking a cup of hot chocolate.

**Personal Context Sphere**

Mark's PCS has keywords such as: *coffee*, *shopping*, *vegetarian*, *XXX Manchester road*, and *ios*, *technology* and *gym*.

**Sensed Context**

*Time:* 11:00 p.m.
*Phone:* Google Nexus 5
*Weather:* 0 Celsius, Cloudy
*Location:* XXX Manchester Rd, Victoria, BC V8T XXX

**Installed Apps**

*NetFlix, Spotify, Hulu, Subway, Tim Hortons, Starbucks, Good Life...*

### 5.2.1 Analysis

Marks location data returned various nearby businesses, including ICBC Claim Centre, Tim Hortons, Victoria Hyundai, Canadian Tire, Good Life, and Subway. Mark had also enabled the fine-grain recommender, which used the data from his PCS to offer suggestions and recommendations.

AppXimity determined that Mark had started frequenting 565 Manchester Road and updated his PCS to reflect that change. AppXimity also added Netflix and Spotify to his PCS as apps used at that location. It was 9:00 p.m., and Mark was

at XXX Manchester Road. From the installed apps, AppXimity suggested Netflix, Spotify, and Hulu. It did not suggest the apps for StarBucks, Tim Hortons, or Goodlife Fitness, because those businesses were closed at that time. AppXimity also recommended uninstalled apps for Crave TV, Shazam, and SoundCloud.

In this experiment, we see that AppXimity also makes recommendations for location-agnostic apps. Its server keeps track of every request made by the client. When a new request is made, it verifies the time the last request was made. If that request was over four hours in length, it stores that as a location of interest. Also each time a user accesses an app through the AppXimity widget, this data is logged. In this way, AppXimity is able to learn and update the users app-usage data in a given locational context. If a user turns off their location data and/or their mobile data usage, this negatively affects the PCS updates. The next time the user makes a request, the last known location will be referenced as a location of interest. However, the reference may be irrelevant, as explained above.

## 5.3 Quality Criteria

We considered several quality attributes of AppXimity from software engineering and end-user perspectives, including functional correctness, accuracy, robustness, usefulness and usability. However, we did not perform formal user studies.

### 5.3.1 Functional Correctness

Software is said to have functional correctness when it conforms to functional requirements and produces correct results. In AppXimity, the required input includes user location data, including latitude and longitude, and installed apps. Given this input, the server should return a list of apps that the user may need in a given location. This list includes a selection from the installed apps as well as recommendations for uninstalled apps. The functional correctness of AppXimity was verified in this experiment, because as the user moved from location to location, the client sent various requests to the server and a list of relevant apps was returned. The software was not evaluated for performance or data usage. In evaluation of the software AppXimity performed as expected with minimal latency. However, before AppXimity is deployed into production, various performance issues could be addressed, including battery usage and data usage.

### 5.3.2 Accuracy

Accuracy refers to how closely the software output approximates reality. A manual check of the installed apps and the nearby businesses was conducted to verify that AppXimity had returned a correct list of relevant apps. In both experiments, the list of apps that were returned by AppXimity agreed with the manually compiled list. One issue that arose is that, if a user does not have a given keyword in their PCS, an app of interest may be overlooked. Conversely, if the user employs highly generic terms to represent an interest, an overly wide variety of apps may be returned. For example, a user employs the keyword shopping to represent an interest in fashion; then groceries stores as well as clothing stores will be returned. Further investigation into procedures for categorizing apps and nearby businesses should be conducted in order to improve the utility of suggestions and recommendations.

### 5.3.3 Usefulness

Usefulness refers to the degree of utility and applicability of a software application. Users indicated that they found the AppXimity Widget very easy to use. The widget was placed on the home screen. Users were pleased by the way in which the widget dynamically reflected the apps that were relevant in their current location. One user remarked on the convenience of having only to unlock the smartphone and click the app of interest.

### 5.3.4 Robustness

The robustness of a software application refers to its ability to cope with various problems that arise during operation. Each time a user leaves the perimeter defined by a specified radius (the virtual fence), AppXimity refreshes the list of relevant apps. In an attempt to evaluate the software's robustness, mobile data was turned off abruptly when the user exited the virtual fence. When the data connection was restored, the widget did not update and returned the same apps as were listed in the last successful request. This was a weakness, as the app list was no longer relevant in the new location. Currently, AppXimity sends a new request to the server when a change in network is detected. A better approach would be to have the client keep a record of the last successful location update; when the client detected a change in the network, it would compare the current location with the last successful update.

This would prevent the duplication of requests to the server.

## 5.4 Summary

This chapter presented two experiments to illustrate the usefulness and acceptance of AppXimity from two different perspectives. We also considered software engineering quality attributes, such as functional correctness, accuracy, robustness and usability, but did not perform a formal evaluation. Our preliminary experiments indicate that AppXimity can augment the user's experience when searching for apps in a given context.

# Chapter 6

# Conclusions

## 6.1 Executive Summary

This thesis explored using users contextual data such as installed apps and user preferences to provide relevant apps dynamically in a given location. AppXimity provides a list of suggested and recommended apps. The suggested apps are a subset of the installed apps that match nearby businesses or have been marked by AppXimity as apps of interest to the user, in that location. AppXimity uses PCS keywords to personalize app recommendations; such recommendations allow for app discovery. On the server-side, AppXimity provides a set of micro-services that expose their interfaces using RESTful API's.

Recommendations in AppXimity are two-tiered: coarse-grained and fine-grained. The coarse-grained tier lacks personalization, it does not leverage users' PCS, whereas the fine-grain does. From the experiments, we saw that combining user preferences with app usage and contextual data delivered a dynamic personalized mobile-app-management framework.

Chapter 2 presented the mobile device revolution and its integration in our lives. It also presented the prospect of creating software systems that are self-managing. The benefits of using the micro-services architecture for IoT applications were also discussed. Finally, the importance of personalizing context was also explored.

Chapter 3 provided the architectural design of AppXimity. The core components of the system were also presented. The key factors considered in the system design were also discussed. The user interface was also explained in this chapter.

Chapter 4 demonstrated the implementation of the micro-services that are de-

ployed in AppXimity. These micro-services expose their interface using RESTful API's and communicate via HTTP. This micro-services oriented architecture, allows external clients, to utilize specific functions in AppXimity, independent of app recommendation.

Chapter 5 presented two experiments to validate user acceptance of AppXimity. We also discussed other attributes such functional correctness, accuracy, robustness, and usability. Our results indicate that AppXimity augments users experience when searching for apps in a given context.

## 6.2 Contributions

The following summaries the contributions of this thesis:

- We identified that micro-services is the preferred service architecture when compared to the monolithic architecture for building IOT applications . Given the dynamic expectations of deployment, and scalability that comes with IoT applications, micro-services offer on-demand scalability, continuous delivery and a set of resilient independent small services that are fault tolerant.

- We explored and extended the PCS to include location data of the places a user frequents as well as the apps they use in that location. By extending the PCS, we are able to make better recommendation to the user. Such personalization increases user experience.

- One limitation of the Google Play Store is that there is no API to retrieve app details. We implemented a web scrapper to retrieve app details and created a local data store to house this data. This service promotes the re-usability of this framework, as this data can be accessed independently of app recommendation.

- We designed and realized a mobile application framework by dynamically exploiting a users context. AppXimity, gathers users location context, personal context, environmental context and in real time suggest apps that they may need in that given context.

- We provided a context-aware widget — that houses the recommended apps, that a user may need in their given context. Ideally nested on the users home screen, for ease of access.

- We implemented a two-tier app recommender: coarse-grain and fine-grain. The course-grain recommender does not leverage personal preferences to make recommendations. The fine-grain recommender leverage personal context to determine which apps to recommend. Fine-grain recommendations enhances user experience.

- Using Dices co-efficient; we created a service that can be consumed independently of AppXimity, to return a similarity score on various keywords.

## 6.3 Future Work

App developers are leveraging location data to personalize mobile application experiences. This thesis presented an alternative way to manually managing mobile applications. This work is a realization of dynamic mobile application presentation. To improve the accuracy, scalability and robustness of AppXimity, the following items represent additional work that could be carried out:

- Retailers are creating beacon-enabled apps to deliver content such as promotions and personalized advertisement to consumers who are in close proximity of their stores. [37] This means, that each store that wants to reach their consumers must create a beacon-enabled app. AppXimity could be extended to model indoor location and store specific promotions are delivered through one interface.

- The AppXimity client was implemented on the Android platform. An implementation in iOS would provided the 90 million iOS users with this mobile-app-management functionality.

- AppXimity, currently only considers users specific preferences. User acceptance testing revealed that users were interested in getting recommendations of apps other users with similar preferences used in that given context. AppXimity could implement some collaborative filtering to improve app discovery.

- AppXimity implements a virtual fence to avoid repeated calls to update the list of relevant apps in the context-aware widget, however if a user returns to the same location more than once, AppXimity sends a new request to the server.

Various caching techniques could be explored to minimize the repetitive network calls.

# Bibliography

[1] R. Murch, *Autonomic Computing*. IBM Press, 2004.

[2] L. Paget and D. L. Frosch, "What will it take to reduce the app gap?," *Journal of General Internal Medicine*, vol. 31, no. 12, pp. 1408–1409, 2016.

[3] Microsoft Research, "Simon." `http://research.microsoft.com/en-us/um/people/bibuxton/buxtoncollection/detail.aspx?id=40`, 2016. Online; accessed 12 October 2016.

[4] The Statistics Portal, "Number of Smartphone Users Worldwide from 2014 to 2020 (in billions)." `https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/`, 2016. Online; accessed 10 October 2016.

[5] The Statistics Portal, "Number of Apps Available in Leading App Stores as of June 2016." `https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/`, 2016. Online; accessed 10 October 2016.

[6] The Nielsen Company, "So many apps, so much more time for entertainment." `http://www.nielsen.com/us/en/insights/news/2015/so-many-apps-so-much-more-time-for-entertainment.html`, 2016. Online; accessed 10 October 2016.

[7] G. Chen, D. Kotz, *et al.*, "A survey of context-aware mobile computing research," *Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College*, 2000.

[8] A. K. Dey, "Understanding and using context," *Personal and Ubiquitous Computing, Springer*, vol. 5, no. 1, pp. 4–7, 2001.

[9] E. Kaasinen, "User needs for location-aware mobile services," *Personal and Ubiquitous Computing*, vol. 7, no. 1, pp. 70–79, 2003.

[10] H. Verkasalo, "Contextual patterns in mobile service usage," *Personal and Ubiquitous Computing*, vol. 13, no. 5, pp. 331–342, 2009.

[11] M. Cooper, R. W. Dronsuth, A. J. Leitich, J. C. N. Lynk, J. J. Mikulski, J. F. Mitchell, R. A. Richardson, and J. H. Sangster, "Radio telephone system," Sept. 16 1975. US Patent 3,906,166.

[12] S. Sarma, D. L. Brock, and K. Ashton, "The networked physical world," *Auto-ID Center White Paper*, 2000.

[13] T. T. Mulani and S. V. Pingle, "Internet of Things," *International Research Journal of Multidisciplinary Studies*, vol. 2, no. 3, pp. 1–4, 2016.

[14] Industrial Internet Consortium, "Healthcare." `http://www.iiconsortium.org/vertical-markets/healthcare.htm`, 2016. Online; accessed 29 October 2016.

[15] Industrial Internet Consortium, "Manufacturing & Smart Factory." `http://www.iiconsortium.org/vertical-markets/manufacturing.htm`, 2016. Online; accessed 29 October 2016.

[16] Industrial Internet Consortium, "Smart Cities." `http://www.iiconsortium.org/vertical-markets/public-sector.htm`, 2016. Online; accessed 29 October 2016.

[17] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *in IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[18] J. Strassner, N. Agoulmine, and E. Lehtihet, "Focale: A novel autonomic networking architecture," 2006.

[19] S. R. White, J. E. Hanson, I. Whalley, D. M. Chess, and J. O. Kephart, "An architectural approach to autonomic computing.," *in Proceedings of the International Conference on Autonomic Computing (ICAC), IEEE*, pp. 2–9, 2004.

[20] B. N. Schilit and M. M. Theimer, "Disseminating active map information to mobile hosts," *in IEEE Network*, vol. 8, pp. 22–32, Sept. 1994.

[21] P. J. Brown, J. D. Bovey, and X. Chen, "Context-aware applications: From the laboratory to the marketplace," *in IEEE Personal Communications Journal*, vol. 4, no. 5, pp. 58–64, 1997.

[22] A. K. Dey, "Context-aware computing: The cyberdesk project," *in Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI) Spring Symposium on Intelligent Environments*, pp. 51–54, 1998.

[23] N. Ryan, J. Pascoe, and D. Morse, "Enhanced reality fieldwork: the context aware archaeological assistant," *Bar International Series*, vol. 750, pp. 269–274, 1999.

[24] A. Ward, A. Jones, and A. Hopper, "A new location technique for the active office," *in IEEE Personal Communications*, vol. 4, no. 5, pp. 42–47, 1997.

[25] T. Rodden, K. Cheverst, K. Davies, and A. Dix, "Exploiting context in HCI design for mobile systems," in *Workshop on Human Computer Interaction with Mobile Devices*, pp. 21–22, 1998.

[26] N. M. Villegas, *Context Management and Self-Adaptivity for Situation-Aware Smart Software Systems*. PhD thesis, University of Victoria, 2013.

[27] B. Schilit, N. Adams, and R. Want, "Context-aware computing applications," in *First Workshop on Mobile Computing Systems and Applications, IEEE*, pp. 85–90, 1994.

[28] K. Henricksen and J. Indulska, "Modelling and using imperfect context information," *in Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, pp. 33–37, 2004.

[29] A. H. Van Bunningen, L. Feng, and P. M. Apers, "Context for ubiquitous data management," *International Workshop on Ubiquitous Data Management, IEEE*, pp. 17–24, 2005.

[30] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *in IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 414–454, 2014.

[31] H. E. Byun and K. Cheverst, "Utilizing context history to provide dynamic adaptations," *An International Journal on Applied Artificial Intelligence*, vol. 18, no. 6, pp. 533–548, 2010.

[32] W.-P. Lee, "Deploying personalized mobile services in an agent-based environment," *Expert Systems with Applications*, vol. 32, no. 4, pp. 1194–1207, 2007.

[33] H. Si, Y. Kawahara, H. Morikawa, and T. Aoyama, "A stochastic approach for creating context-aware services based on context histories in smart home," *Cognitive Science Research Paper-University of Sussex (CSRP)*, vol. 577, p. 37, 2005.

[34] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *in IEEE Computer*, vol. 77, no. 2, pp. 257–286, 1989.

[35] J. Pascoe, N. Ryan, and D. Morse, "Issues in developing context-aware computing," *International Symposium on Handheld and Ubiquitous Computing, Springer*, pp. 208–221, 1999.

[36] M. Weiser, "The computer for the 21st century," *in IEEE Computer*, vol. 1, pp. 19–25, July 2002.

[37] P. Lach, "Anonymous location based messaging: The yakkit approach," Master's thesis, 2015.

[38] M. Van Setten, S. Pokraev, and J. Koolwaaij, "Context-aware recommendations in the mobile tourist application compass," in *International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems,* Springer, pp. 235–244, 2004.

[39] B. Yan and G. Chen, "Appjoy: personalized mobile application discovery," in *Proceedings of the 9th International Conference on Mobile systems, Applications, and Services, ACM*, pp. 113–126, 2011.

[40] W. Woerndl, C. Schueller, and R. Wojtech, "A hybrid recommender system for context-aware recommendations of mobile applications," *in 23rd International Conference, Data Engineering Workshop, IEEE*, pp. 871–878, 2007.

[41] A. Girardello and F. Michahelles, "Appaware: Which mobile applications are hot," in *Proceedings of the 12th International Conference on Human Computer Interaction with Mobile Devices and Services*, pp. 431–434, ACM, 2010.

[42] C. Davidsson and S. Moritz, "Utilizing implicit feedback and context to recommend mobile applications from first use," in *Proceedings of the 2011 Workshop on Context-awareness in Retrieval and Recommendation, Association for Computing Machinery (ACM)*, pp. 19–22, 2011.

[43] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil, "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud," *in Proceedings of the IEEE, 10th Computing Colombian Conference (CCC)*, pp. 583–590, 2015.

[44] D. Uckelmann, M. Harrison, and F. Michahelles, "An architectural approach towards the future internet of things," in *Architecting the Internet of Things, Springer*, pp. 1–24, 2011.

[45] D. Namiot and M. Sneps-Sneppe, "On micro-services architecture," *International Journal of Open Information Technologies*, vol. 2, no. 9, pp. 24–27, 2014.

[46] T. Yan, D. Chu, D. Ganesan, A. Kansal, and J. Liu, "Fast app launching for mobile devices using predictive user context," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, ACM*, pp. 113–126, ACM, 2012.

[47] M.-H. Park, J.-H. Hong, and S.-B. Cho, "Location-based recommendation system using bayesian users preference model in mobile devices," in *International Conference on Ubiquitous Intelligence and Computing,* Springer, pp. 1130–1139, 2007.

[48] B. W. Boehm, "A spiral model of software development and enhancement," *in IEEE Computer*, vol. 21, no. 5, pp. 61–72, 1988.

[49] Hans and J. Van Vliet, *Software Engineering: Principles and Practice*, vol. 3. Wiley New York, 1993.

[50] J. Martin, *Rapid Application Development*, vol. 8. Macmillan New York, 1991.

[51] J. Gosling, *The Java Language Specification*. Addison-Wesley Professional, 2000.

[52] P. Group *et al.*, "PHP Hypertext Preprocessor," 2008.

[53] S. Tilkov and S. Vinoski, "Node. js: Using javascript to build high-performance network programs," *IEEE Internet Computing*, vol. 14, no. 6, p. 80, 2010.

[54] H. E. Williams and D. Lane, *Web Database Applications with PHP and MySQL*. O'Reilly Media, Inc, 2004.

[55] K. Chodorow, *MongoDB: The Definitive Guide*. O'Reilly Media, Inc., 2013.

[56] L. Richardson and S. Ruby, *RESTful Web Services*. O'Reilly Media, Inc., 2008.

[57] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *Association for Computing Machinery (ACM) Transactions on Internet Technology (TOIT)*, vol. 2, no. 2, pp. 115–150, 2002.

[58] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible markup language (xml)," *World Wide Web Consortium Recommendation REC-xml-19980210. http://www. w3. org/TR/1998/REC-xml-19980210*, vol. 16, p. 16, 1998.

[59] D. Crockford, "The application/json media type for javascript object notation (json)," *Internet Engineering Task Force (IETF)*, 2014.

[60] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[61] R. Batool, A. M. Khattak, J. Maqbool, and S. Lee, "Precise tweet classification and sentiment analysis," *2013 IEEE/ACIS 12th International Conference on Computer and Information Science (ICIS), IEEE*, pp. 461–466, 2013.

[62] J. Ramos, "Using TF-IDF to determine word relevance in document queries," in *Proceedings of the First Instructional Conference on Machine Learning*, 2003.

[63] R. Mihalcea and P. Tarau, "Textrank: Bringing order into texts," pp. 404–411, Association for Computational Linguistics, 2004.

[64] S. Rose, D. Engel, N. Cramer, and W. Cowley, "Automatic keyword extraction from individual documents," *Text Mining*, pp. 1–20, 2010.

[65] G. Kondrak, "N-gram similarity and distance," *International Symposium on String Processing and Information Retrieval*, pp. 115–126, 2005.

# Appendix A

# Appendix

This appendix provides complete source for a few services in AppXimity. The complete source code repository for all services can be accessed at `http://dev.engineeringbits.com:9444/ernest/appXimity-services.git`

## A.1 Context-Aware Widget Provider Source Code

This is part 1 of the source code for the Context Aware Widget. The explanation of this code is in Section 3.5.

```
package com.engineeringbits.appwidget;

import android.app.PendingIntent;
import android.appwidget.AppWidgetManager;
import android.appwidget.AppWidgetProvider;
import android.content.Context;
import android.content.Intent;
import android.net.Uri;
import android.widget.RemoteViews;
import android.widget.Toast;

import com.engineeringbits.appximity.MainActivity;
import com.engineeringbits.appximity.R;

/**
 * Created by ebits on 2016-10-11.
```

```
 */
public class AXWidgetProvider extends AppWidgetProvider {
        public static final String TOAST_ACTION =
            "com.example.android.stackwidget.TOAST_ACTION";
        public static final String EXTRA_ITEM =
            "com.example.android.stackwidget.EXTRA_ITEM";

        @Override
        public void onDeleted(Context context, int[] appWidgetIds) {
            super.onDeleted(context, appWidgetIds);
        }

        @Override
        public void onDisabled(Context context) {
            super.onDisabled(context);
        }

        @Override
        public void onEnabled(Context context) {
            super.onEnabled(context);
        }

        @Override
        public void onReceive(Context context, Intent intent) {
            AppWidgetManager mgr = AppWidgetManager.getInstance(context);
            if (intent.getAction().equals(TOAST_ACTION)) {
                int appWidgetId =
                    intent.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
                        AppWidgetManager.INVALID_APPWIDGET_ID);
                int viewIndex = intent.getIntExtra(EXTRA_ITEM, 0);
                Toast.makeText(context, "Touched view " + viewIndex,
                    Toast.LENGTH_SHORT).show();
            }
            super.onReceive(context, intent);
        }

        @Override
```

```
    public void onUpdate(Context context, AppWidgetManager
        appWidgetManager, int[] appWidgetIds) {
        // update each of the widgets with the remote adapter

       //updateWidget(context, appWidgetManager, appWidgetIds);
        updateWidget2(context, appWidgetManager, appWidgetIds);

    }

private void updateWidget2(Context context, AppWidgetManager
    appWidgetManager, int[] appWidgetIds){
    for (int i = 0; i < appWidgetIds.length; ++i) {
        appWidgetManager.notifyAppWidgetViewDataChanged(appWidgetIds[i],
            R.id.grid_view);
        // Here we setup the intent which points to the
            StackViewService which will
        // provide the views for this collection.
        Intent intent = new Intent(context, AXWidgetService.class);
        intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
            appWidgetIds[i]);
        // When intents are compared, the extras are ignored, so we
            need to embed the extras
        // into the data so that the extras will not be ignored.
        intent.setData(Uri.parse(intent.toUri(Intent.URI_INTENT_SCHEME)));
        RemoteViews rv = new RemoteViews(context.getPackageName(),
            R.layout.widget_layout);
        rv.setRemoteAdapter(R.id.grid_view, intent);


        // The empty view is displayed when the collection has no
            items. It should be a sibling
        // of the collection view.
        rv.setEmptyView(R.id.grid_view, R.id.empty_view);

        Intent toastIntent = new Intent(context,
            AXWidgetProvider.class);
        toastIntent.setAction(AXWidgetProvider.TOAST_ACTION);
```

```
            toastIntent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
                appWidgetIds[i]);
            intent.setData(Uri.parse(intent.toUri(Intent.URI_INTENT_SCHEME)));
            PendingIntent toastPendingIntent =
                PendingIntent.getBroadcast(context, 0, toastIntent,
                    PendingIntent.FLAG_UPDATE_CURRENT);
            rv.setPendingIntentTemplate(R.id.grid_view, toastPendingIntent);

            appWidgetManager.updateAppWidget(appWidgetIds[i], rv);
        }
        super.onUpdate(context, appWidgetManager, appWidgetIds);
    }
}
```

## A.2 Context-Aware Widget Service Source Code

This is part 2 of source code for the Context Aware Widget. The explanation of this code is in Section 3.5.

```
package com.engineeringbits.appwidget;

import android.appwidget.AppWidgetManager;
import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.drawable.BitmapDrawable;
import android.graphics.drawable.Drawable;
import android.net.Uri;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.RemoteViews;
import android.widget.RemoteViewsService;

import java.io.UnsupportedEncodingException;
```

```
import java.util.ArrayList;
import java.util.List;

import com.engineeringbits.apps.AppInfo;
import com.engineeringbits.apps.AppsService;
import com.engineeringbits.appximity.R;
import com.loopj.android.http.*;
import cz.msebera.android.httpclient.Header;
import cz.msebera.android.httpclient.entity.StringEntity;
import cz.msebera.android.httpclient.message.BasicHeader;
import cz.msebera.android.httpclient.protocol.HTTP;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

/**
 * Created by ebits on 2016-10-11.
 */
public class AXWidgetService extends RemoteViewsService {
    @Override
    public RemoteViewsFactory onGetViewFactory(Intent intent) {
        return new GridRemoteViewsFactory(this.getApplicationContext(),
            intent);
    }
}

class GridRemoteViewsFactory implements
    RemoteViewsService.RemoteViewsFactory {
    private static final int mCount = 8;
    private List<WidgetItem> mWidgetItems = new ArrayList<WidgetItem>();
    private List<AppInfo> appList = new ArrayList<AppInfo>();
    private Context mContext;
    private int mAppWidgetId;

    private ArrayList<AppsService> appsService = new
        ArrayList<AppsService>();
```

```
public GridRemoteViewsFactory(Context context, Intent intent) {
    mContext = context;
    mAppWidgetId =
        intent.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
            AppWidgetManager.INVALID_APPWIDGET_ID);
}
public void AsyncHttp(List<AppInfo> appList){

    System.out.print("In async");
    JSONObject location =new JSONObject();
    JSONObject params= new JSONObject();
    try{
        location.put("lat","45.311217");
        location.put("long", "-75.907109");

        params.put("location", location);

        JSONArray jsonArr = new JSONArray() ;

        for(AppInfo item : appList) {
            JSONObject jsonObj =new JSONObject();
            jsonObj.put("appName", item.getAppName());
            jsonObj.put("appPackageName", item.getPackageName());
            jsonArr.put(jsonObj);
        }

        params.put("appsInstalled", jsonArr);

        System.out.print("*****"+params+"****");

    String url = "http://dev.engineeringbits.com:3000/appsNearby";
    AsyncHttpClient client = new AsyncHttpClient();
    //RequestParams paramsJson = new RequestParams();
    StringEntity entity= new StringEntity(params.toString());
    params.put("location", );
    //paramsJson.put("request", paramsJson);
```

```
    entity.setContentType(new BasicHeader(HTTP.CONTENT_TYPE,
        "application/json"));
    client.post(mContext, url, entity, "application/json", new
        JsonHttpResponseHandler() {
        @Override
        public void onSuccess(int statusCode, Header[] headers,
            JSONObject response) {
            // Root JSON in response is an dictionary i.e { "data : [
                ... ] }
            // Handle resulting parsed JSON response here
            System.out.println("sucess in async");
            System.out.println(response);
        }
        @Override
        public void onSuccess(int statusCode, Header[] headers,
            JSONArray response) {
            // Root JSON in response is an dictionary i.e { "data : [
                ... ] }
            // Handle resulting parsed JSON response here
            System.out.println("sucess in async");
            System.out.println(response);
        }
        @Override
        public void onFailure(int statusCode, Header[] headers, String
            res, Throwable t) {
            // called when response HTTP status is "4XX" (eg. 401, 403,
                404)
        }
    });
    }
    catch (JSONException e){
        e.printStackTrace();
    }
    catch (UnsupportedEncodingException e){
        e.printStackTrace();
    }
}
```

```
public void onCreate() {
    // In onCreate() you setup any connections / cursors to your data
        source. Heavy lifting,
    // for example downloading or creating content etc, should be
        deferred to onDataSetChanged()
    // or getViewAt(). Taking more than 20 seconds in this call will
        result in an ANR.
    for (int i = 0; i < mCount; i++) {
        //mWidgetItems.add(new WidgetItem(i + "!"))
    }
    System.out.print("in creates");
    AppsService appsService=new AppsService();
    appList= appsService.AppsHandler();
    AsyncHttp(appList);


}

public void onDestroy() {
    // In onDestroy() you should tear down anything that was setup for
        your data source,
    // eg. cursors, connections, etc.
    mWidgetItems.clear();
}

public int getCount() {
    return mCount;
}

public RemoteViews getViewAt(int position) {
    // position will always range from 0 to getCount() - 1.

    // We construct a remote views item based on our widget item xml
        file, and set the
    // text based on the position.
    RemoteViews rv = new RemoteViews(mContext.getPackageName(),
```

```
            R.layout.widget_item);
        rv.setTextViewText(R.id.widget_item_app_name,
            appList.get(position).getAppName());
        BitmapDrawable icon =
            (BitmapDrawable)appList.get(position).getAppIcon();

        Bitmap bitmap = icon.getBitmap();
        rv.setImageViewBitmap(R.id.widget_item_app_icon, bitmap);
        //rv.setImageViewUri(R.id.widget_item_app_icon, Uri.parse("));
        //rv.setImageViewIcon();

        // Next, we set a fill-intent which will be used to fill-in the
            pending intent template
        // which is set on the collection view in StackWidgetProvider.
        Bundle extras = new Bundle();
        //extras.putInt(AXWidgetService.EXTRA_ITEM, position);
        Intent fillInIntent = new Intent();
        fillInIntent.putExtras(extras);
        rv.setOnClickFillInIntent(R.id.widget_item, fillInIntent);

        // Return the remote views object.
        return rv;
    }

    public RemoteViews getLoadingView() {
        // You can create a custom loading view (for instance when
            getViewAt() is slow.) If you
        // return null here, you will get the default loading view.
        return null;
    }

    public int getViewTypeCount() {
        return 1;
    }

    public long getItemId(int position) {
        return position;
```

```
    }

    public boolean hasStableIds() {
        return true;
    }

    public void onDataSetChanged() {
        AppsService appsService=new AppsService();
        appList= appsService.AppsHandler();
        apps.getUserInstalledApps();
    }
}
```

## A.3 Google Play Extractor Service (GPES) Source Code

This is the source code for the Google Play Extractor Service. The explanation of this code is in Section 4.2.3.

```
var express = require('express');
var router = express.Router();
var gPlay=require('../bin/lib/GPlay');

/* GET users listing. */
router.get('/', function(req, res, next) {
  res.send('respond gplay with a resource');
});

/* GET users listing. */
router.get('/app/:packageName', function(req, res, next) {
   console.log(req.params.packageName);
  var gplay = require('google-play-scraper');
  gplay.app({appId: req.params.packageName})
  .then(function(app){
    res.send(app);
  })
```

```
  .catch(function(e){
    res.status(500).send('Unable to fetch app data!');
  });

});
module.exports = router;

var gplay = require('google-play-scraper');

gplay.app({appId: packageName})
  .then(function(app){

  })
  .catch(function(e){
    console.log('There was an error fetching the application!');
  });

'use strict';

const request = require('./utils/request');
const memoize = require('./utils/memoize');
const cheerio = require('cheerio');
const queryString = require('querystring');
const url = require('url');
const R = require('ramda');

const PLAYSTORE_URL = 'https://play.google.com/store/apps/details';

function app (opts) {
  return new Promise(function (resolve, reject) {
    if (!opts || !opts.appId) {
      throw Error('appId missing');
    }

    opts.lang = opts.lang || 'en';
    opts.country = opts.country || 'us';
```

```
    const qs = queryString.stringify({
      id: opts.appId,
      hl: opts.lang,
      gl: opts.country
    });
    const reqUrl = `${PLAYSTORE_URL}?${qs}`;

    request(reqUrl, opts.throttle)
      .then(cheerio.load)
      .then(parseFields)
      .then(function (app) {
        app.url = reqUrl;
        app.appId = opts.appId;
        resolve(app);
      })
      .catch(reject);
  });
}

function parseFields ($) {
  const detailsInfo = $('.details-info');
  const title = detailsInfo.find('div.document-title').text().trim();
  const developer = detailsInfo.find('span[itemprop="name"]').text();
  const summary = $('meta[name="description"]').attr('content');

  const mainGenre = detailsInfo.find('.category').first();
  const genreText = mainGenre.text().trim();
  const genreId = mainGenre.attr('href').split('/')[4];

  const familyGenre = detailsInfo.find('.category[href*="FAMILY"]');
  let familyGenreText;
  let familyGenreId;
  if (familyGenre.length) {
    familyGenreText = familyGenre.text().trim() || undefined;
    familyGenreId = familyGenre.attr('href').split('/')[4];
  }
```

```
const price = detailsInfo.find('meta[itemprop=price]').attr('content');
const icon = detailsInfo.find('img.cover-image').attr('src');
const offersIAP = !!detailsInfo.find('.inapp-msg').length;
const adSupported =
    !!detailsInfo.find('.ads-supported-label-msg').length;

const additionalInfo = $('.details-section-contents');
const description = additionalInfo.find('div[itemprop=description] div');
const version =
    additionalInfo.find('div.content[itemprop="softwareVersion"]').text().trim();
const updated =
    additionalInfo.find('div.content[itemprop="datePublished"]').text().trim();
const androidVersionText =
    additionalInfo.find('div.content[itemprop="operatingSystems"]').text().trim();
const androidVersion = normalizeAndroidVersion(androidVersionText);
const contentRating =
    additionalInfo.find('div.content[itemprop="contentRating"]').text().trim();
const size =
    additionalInfo.find('div.content[itemprop="fileSize"]').text().trim();

let maxInstalls, minInstalls;
const preregister = !!$('.preregistration-container').length;
if (!preregister) {
  const installs =
      installNumbers(additionalInfo.find('div.content[itemprop="numDownloads"]').text()
  minInstalls = cleanInt(installs[0]);
  maxInstalls = cleanInt(installs[1]);
}

let developerEmail =
    additionalInfo.find('.dev-link[href^="mailto:"]').attr('href');
if (developerEmail) {
  developerEmail = developerEmail.split(':')[1];
}

let developerWebsite =
    additionalInfo.find('.dev-link[href^="http"]').attr('href');
```

```
if (developerWebsite) {
  // extract clean url wrapped in google url
  developerWebsite = url.parse(developerWebsite, true).query.q;
}

const comments = $('.quoted-review').toArray().map((elem) =>
    $(elem).text().trim());
const ratingBox = $('.rating-box');
const reviews = cleanInt(ratingBox.find('span.reviews-num').text());

const ratingHistogram = $('.rating-histogram');
const histogram = {
  5: cleanInt(ratingHistogram.find('.five .bar-number').text()),
  4: cleanInt(ratingHistogram.find('.four .bar-number').text()),
  3: cleanInt(ratingHistogram.find('.three .bar-number').text()),
  2: cleanInt(ratingHistogram.find('.two .bar-number').text()),
  1: cleanInt(ratingHistogram.find('.one .bar-number').text())
};
// for other languages
const score = parseFloat(ratingBox.find('div.score').text().replace(',',
    '.')) || 0;

let video = $('.screenshots
    span.preview-overlay-container[data-video-url]').attr('data-video-url');
if (video) {
  video = video.split('?')[0];
}

const screenshots = $('.thumbnails .screenshot').toArray().map((elem) =>
    $(elem).attr('src'));
const recentChanges = $('.recent-change').toArray().map((elem) =>
    $(elem).text());

const fields = {
  title,
  summary,
  icon,
```

```
    price,
    free: price === '0',
    minInstalls,
    maxInstalls,
    score,
    reviews,
    developer,
    developerEmail,
    developerWebsite,
    updated,
    version,
    genre: genreText,
    genreId,
    familyGenre: familyGenreText,
    familyGenreId,
    size,
    description: descriptionText(description),
    descriptionHTML: description.html(),
    histogram,
    offersIAP,
    adSupported,
    androidVersionText,
    androidVersion,
    contentRating,
    screenshots,
    video,
    comments,
    recentChanges,
    preregister
  };

  // replace blank values with undefined
  return R.map((field) => field === '' ? undefined : field, fields);
}

function descriptionText (description) {
  // preserve the line breaks when converting to text
```

```
  const html = '<div>' + description.html().replace(/<\/p>/g, '\n</p>') +
      '</div>';
  return cheerio.load(html)('div').text();
}

function cleanInt (number) {
  number = number || '0';
  // removes thousands separator
  number = number.replace(/\D/g, '');
  return parseInt(number);
}

function installNumbers (downloads) {
  if (!downloads) {
    return [0, 0];
  }

  // TODO iterate instead of copy paste
  let installs = downloads.split(' - ');
  if (installs.length === 2) return installs;

  installs = downloads.split(' et ');
  if (installs.length === 2) return installs;

  installs = downloads.split('');
  if (installs.length === 2) return installs;

  installs = downloads.split('–');
  if (installs.length === 2) return installs;

  installs = downloads.split('');
  if (installs.length === 2) return installs;

  installs = downloads.split(' a ');
  if (installs.length === 2) return installs;

  throw new Error(`Unable to parse min/max downloads: ${downloads},
```

```
     ${installs}‘);
}

function normalizeAndroidVersion (androidVersionText) {
  let matches = androidVersionText.match(/^([0-9\.]+)[^0-9\.].+/);

  if (!matches || typeof matches[1] === ’undefined’) {
    return ’VARY’;
  }

  return matches[1];
}

module.exports = memoize(app);
```

## A.4 String Manipulation Service (SMS) Source Code

This is the source code for the String Manipulation Service. The explanation of this code is in Section 4.2.7.

```
/**
 * Created by ebits on 2016-10-17.
 */
var stringSimilarity = require(’string-similarity’);

module.exports={

    //return similarity between strings
    strSimilarity:function(str1,str2){
       return stringSimilarity.compareTwoStrings(str1.toUpperCase(),
           str2.toUpperCase());
    },
    //check if str1 is contained in str2
    strContains:function(str1,str2){
       return str2.toUpperCase().includes(str1.toUpperCase());
    },
```

```
    strSplit:function(str1,separator) {
        return str1.split(separator)
    }

}
var _forEach = require('lodash/forEach');
var _map = require('lodash/map');
var _every = require('lodash/every');
var _maxBy = require('lodash/maxBy');
var _flattenDeep = require('lodash/flattenDeep');

exports.compareTwoStrings = compareTwoStrings;
exports.findBestMatch = findBestMatch;

function compareTwoStrings(str1, str2) {
  var pairs1 = wordLetterPairs(str1.toUpperCase());
  var pairs2 = wordLetterPairs(str2.toUpperCase());
  var intersection = 0;
  var union = pairs1.length + pairs2.length;

  _forEach(pairs1, function (pair1) {
    for(var i = 0; i < pairs2.length; i++) {
      var pair2 = pairs2[i];
      if (pair1 === pair2) {
        intersection++;
        pairs2.splice(i, 1);
        break;
      }
    }
  });

  return (2.0 * intersection) / union;

  // private functions ----------------------------
  function letterPairs(str) {
    var numPairs = str.length - 1;
```

```
    var pairs = [];
    for(var i = 0; i < numPairs; i++) {
      pairs[i] = str.substring(i, i + 2);
    }
    return pairs;
  }

  function wordLetterPairs(str) {
    return _flattenDeep(_map(str.split(' '), letterPairs));
  }
}


function findBestMatch(mainString, targetStrings) {
  if (!areArgsValid(mainString, targetStrings)) {
    throw new Error('Bad arguments: First argument should be a string,
        second should be an array of strings');
  }
  var ratings = _map(targetStrings, function (targetString) {
    return {
      target: targetString,
      rating: compareTwoStrings(mainString, targetString)
    };
  });

  return {
    ratings: ratings,
    bestMatch: _maxBy(ratings, 'rating')
  };

  // private functions ---------------------------
  function areArgsValid(mainString, targetStrings) {
    var mainStringIsAString = (typeof mainString === 'string');

    var targetStringsIsAnArrayOfStrings = Array.isArray(targetStrings) &&
      targetStrings.length > 0 &&
      _every(targetStrings, function (targetString) {
```

```
        return (typeof targetString === 'string');
      });

    return mainStringIsAString && targetStringsIsAnArrayOfStrings;
  }
}
```

## A.5 Places Nearby Service (PNS)

This is the code for the Places Nearby Service. The explanation of this code is in Section 4.2.6.

```
var express = require('express');
var request = require('request');
var router = express.Router();
var Places=require('../models/Places');
var App=require('../models/App');
var strMan=require('../bin/lib/StrMan');



/* Post user data such as location and installed apps. */
router.post('/', function(req, res, next) {
        //get location data
        //var placesArr;
        //console.log(strMan.strSimilarity(''));
        console.log(strMan.strSimilarity('Best Buy','bestbuy'));
        var nearbyPlacesURL=retrieveNearbyPlacesURL(req.body.location);
        var placesArr;
        var appsArr=extractAppsInstalledData(req.body.appsInstalled);
        var placesJSON;
        nearbyPlaces(nearbyPlacesURL).then(function (result){
            //console.log(result);
            extractPlacesData(result).then(function(result){
             findNearbyPlaceInstalledApp(appsArr,result);
            },function(error){console.log("extract error")});
```

```
        },function(error){console.log(error)})


         res.send();
});

module.exports = router;

//Configure the URL to fetch nearbyPlaces from Google's Nearby Places API
function retrieveNearbyPlacesURL(userLocation){
    return AppConfig.GOOGLE_PLACES_API_URL+"location="+userLocation.lat+","
    +userLocation.long+"&radius="+AppConfig.GOOGLE_PLACES_API_RADIUS+
    "&key="+AppConfig.GOOGLE_PLACES_API_KEY;
}

//HTTP request to fetch the nearbyPlaces in proximty of the user
var nearbyPlaces=function retrieveNearbyPlaces(nearbyPlacesURL){
    return new Promise (function (resolve,reject){
        var request = require('request');

        request(nearbyPlacesURL, function (error, response, body) {
            if (!error && response.statusCode == 200 &&
                parseToJSONOBJ(body).status=="OK" ) {
                resolve(body);
            }
            else{
                reject(parseToJSONOBJ(body).status);
            }
        })
    });
}

//convert json strings to JSON objects
function parseToJSONOBJ(stringJson){
    return JSON.parse(stringJson);
}
```

```
//extract the needed Places data and create an array of Places objects.
function extractPlacesData(nearbyPlacesString)
{
    var placesObjs=[];
    var parsednearbyPlacesJson=parseToJSONOBJ(nearbyPlacesString);
    return new Promise (function (resolve,reject){
           for(var i=0; i<parsednearbyPlacesJson.results.length; i++)
           {
               placesObjs[i] = new Places();
               placesObjs[i].placesName=parsednearbyPlacesJson.results[i].name;
               placesObjs[i].placesOpen=parsednearbyPlacesJson.results[i].opening_hours;
           }
               resolve(placesObjs);
       })
}

//extract the installed apps data and create an array of Installed Apps
   objects.
function extractAppsInstalledData(appsInstalled)
{
    var appInstalledObj=[];
    for (var i=0; i<appsInstalled.length; i++)
    {
        appInstalledObj[i]=new App();
        appInstalledObj[i].appPackageName=appsInstalled[i].appPackageName;
        appInstalledObj[i].appName=appsInstalled[i].appName;
    }
    return appsInstalled;
}

function findNearbyPlaceInstalledApp(appsArr,placesArr){
       var appsJson=[];
       var simValue=1;
       for(var i=0;i<appsArr.length;i++){
           for (var k=0; k<placesArr.length;k++){
               //check similarity between names
               var
```

```
                simNameScore=strMan.strSimilarity(appsArr[i].appName,placesArr[k].plac
            if(simNameScore>=simValue){
                console.log("ghgh"+appsArr[i].appName+placesArr[k].placesName+simNameS
                appsJson.push(appsArr[i].appPackageName);
                break;
             }else{
                //if can't find a match in name check package name
                //var
                   pNameArray=strMan.strSplit(appsArr[i].appPackageName,'.');
                //var
                   pPlaceArray=strMan.strSplit(placesArr[k].placesName,"
                   ");
                //for(var s=0;i<9;i++){
                   //all android apps start with com, skip that
                //  if(pNameArray[s]!="com"){
                      // var
                          simPackageScore=strMan.strSimilarity(pNameArray[i],placesArr
                //   }
                //}
                //console.log("arat"+pNameArray);
                //console.log("arat"+pPlaceArray);
                //var
                   simPackageScore=strMan.strSimilarity(appsArr[i].appName,placesArr[k
             }
          }

      }
 }
```

## A.6 Dice's Coefficient Methods

```
#!/usr/bin/env node
'use strict';

/* Dependencies. */
```

```
var pack = require('./package.json');
var dice = require('./');

/* Arguments. */
var argv = process.argv.slice(2);

/* Program. */
if (
  argv.indexOf('--help') !== -1 ||
  argv.indexOf('-h') !== -1
) {
  console.log(help());
} else if (
  argv.indexOf('--version') !== -1 ||
  argv.indexOf('-v') !== -1
) {
  console.log(pack.version);
} else if (argv.length) {
  getEditDistance(argv.join(' ').split(/\s+/g));
} else {
  process.stdin.resume();
  process.stdin.setEncoding('utf8');
  process.stdin.on('data', function (data) {
    getEditDistance(data.trim().split(/\s+/g));
  });
}

/**
 * Get the edit distance for a list of words.
 *
 * @param {Array.<string>} values
 */
function getEditDistance(values) {
  if (values.length === 2) {
    console.log(dice(values[0], values[1]) || 0);
  } else {
    process.stderr.write(help());
```

```
    process.exit(1);
  }
}
```

## A.7 Rake Algorithm

```
# Implementation of RAKE - Rapid Automtic Keyword Exraction algorithm
# as described in:
# Rose, S., D. Engel, N. Cramer, and W. Cowley (2010).
# Automatic keyword extraction from indi-vidual documents.
# In M. W. Berry and J. Kogan (Eds.), Text Mining: Applications and
    Theory.unknown: John Wiley and Sons, Ltd.

import re
import operator

debug = False
test = True


def is_number(s):
    try:
        float(s) if '.' in s else int(s)
        return True
    except ValueError:
        return False


def load_stop_words(stop_word_file):
    """
    Utility function to load stop words from a file and return as a list
        of words
    @param stop_word_file Path and file name of a file containing stop
        words.
```

```
    @return list A list of stop words.
    """
    stop_words = []
    for line in open(stop_word_file):
        if line.strip()[0:1] != "#":
            for word in line.split(): # in case more than one per line
                stop_words.append(word)
    return stop_words


def separate_words(text, min_word_return_size):
    """
    Utility function to return a list of all words that are have a length
        greater than a specified number of characters.
    @param text The text that must be split in to words.
    @param min_word_return_size The minimum no of characters a word must
        have to be included.
    """
    splitter = re.compile('[^a-zA-Z0-9_\\+\\-/]')
    words = []
    for single_word in splitter.split(text):
        current_word = single_word.strip().lower()
        #leave numbers in phrase, but don't count as words, since they tend
            to invalidate scores of their phrases
        if len(current_word) > min_word_return_size and current_word != ''
            and not is_number(current_word):
            words.append(current_word)
    return words


def split_sentences(text):
    """
    Utility function to return a list of sentences.
    @param text The text that must be split in to sentences.
    """
    sentence_delimiters =
        re.compile(u'[.!?,;:\t\\\\"\\(\\)\\\'\u2019\u2013]|\\s\\-\\s')
```

```
    sentences = sentence_delimiters.split(text)
    return sentences


def build_stop_word_regex(stop_word_file_path):
    stop_word_list = load_stop_words(stop_word_file_path)
    stop_word_regex_list = []
    for word in stop_word_list:
        word_regex = r'\b' + word + r'(?![\w-])' # added look ahead for
            hyphen
        stop_word_regex_list.append(word_regex)
    stop_word_pattern = re.compile('|'.join(stop_word_regex_list),
        re.IGNORECASE)
    return stop_word_pattern


def generate_candidate_keywords(sentence_list, stopword_pattern):
    phrase_list = []
    for s in sentence_list:
        tmp = re.sub(stopword_pattern, '|', s.strip())
        phrases = tmp.split("|")
        for phrase in phrases:
            phrase = phrase.strip().lower()
            if phrase != "":
                phrase_list.append(phrase)
    return phrase_list


def calculate_word_scores(phraseList):
    word_frequency = {}
    word_degree = {}
    for phrase in phraseList:
        word_list = separate_words(phrase, 0)
        word_list_length = len(word_list)
        word_list_degree = word_list_length - 1
        #if word_list_degree > 3: word_list_degree = 3 #exp.
        for word in word_list:
```

```
            word_frequency.setdefault(word, 0)
            word_frequency[word] += 1
            word_degree.setdefault(word, 0)
            word_degree[word] += word_list_degree #orig.
            #word_degree[word] += 1/(word_list_length*1.0) #exp.
    for item in word_frequency:
        word_degree[item] = word_degree[item] + word_frequency[item]

    # Calculate Word scores = deg(w)/frew(w)
    word_score = {}
    for item in word_frequency:
        word_score.setdefault(item, 0)
        word_score[item] = word_degree[item] / (word_frequency[item] * 1.0)
            #orig.
    #word_score[item] = word_frequency[item]/(word_degree[item] * 1.0)
        #exp.
    return word_score


def generate_candidate_keyword_scores(phrase_list, word_score):
    keyword_candidates = {}
    for phrase in phrase_list:
        keyword_candidates.setdefault(phrase, 0)
        word_list = separate_words(phrase, 0)
        candidate_score = 0
        for word in word_list:
            candidate_score += word_score[word]
        keyword_candidates[phrase] = candidate_score
    return keyword_candidates


class Rake(object):
    def __init__(self, stop_words_path):
        self.stop_words_path = stop_words_path
        self.__stop_words_pattern = build_stop_word_regex(stop_words_path)

    def run(self, text):
```

```
        sentence_list = split_sentences(text)

        phrase_list = generate_candidate_keywords(sentence_list,
            self.__stop_words_pattern)

        word_scores = calculate_word_scores(phrase_list)

        keyword_candidates = generate_candidate_keyword_scores(phrase_list,
            word_scores)

        sorted_keywords = sorted(keyword_candidates.iteritems(),
            key=operator.itemgetter(1), reverse=True)
        return sorted_keywords


if test:
    text = "Compatibility of systems of linear constraints over the set of
        natural numbers. Criteria of compatibility of a system of linear
        Diophantine equations, strict inequations, and nonstrict
        inequations are considered. Upper bounds for components of a
        minimal set of solutions and algorithms of construction of minimal
        generating sets of solutions for all types of systems are given.
        These criteria and the corresponding algorithms for constructing a
        minimal supporting set of solutions can be used in solving all the
        considered types of systems and systems of mixed types."

    # Split text into sentences
    sentenceList = split_sentences(text)
    #stoppath = "FoxStoplist.txt" #Fox stoplist contains "numbers", so it
        will not find "natural numbers" like in Table 1.1
    stoppath = "SmartStoplist.txt" #SMART stoplist misses some of the
        lower-scoring keywords in Figure 1.5, which means that the top 1/3
        cuts off one of the 4.0 score words in Table 1.1
    stopwordpattern = build_stop_word_regex(stoppath)

    # generate candidate keywords
    phraseList = generate_candidate_keywords(sentenceList, stopwordpattern)
```

```
# calculate individual word scores
wordscores = calculate_word_scores(phraseList)

# generate candidate keyword scores
keywordcandidates = generate_candidate_keyword_scores(phraseList,
    wordscores)
if debug: print keywordcandidates

sortedKeywords = sorted(keywordcandidates.iteritems(),
    key=operator.itemgetter(1), reverse=True)
if debug: print sortedKeywords

totalKeywords = len(sortedKeywords)
if debug: print totalKeywords
print sortedKeywords[0:(totalKeywords / 3)]

rake = Rake("SmartStoplist.txt")
keywords = rake.run(text)
print keywords
```