GoCity: A Context-Aware Adaptive Android Application

by

Qian Yang
B.Sc., University of Electronic Science and Technology of China (UESTC), 2006

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

MASTER OF SCIENCE

in the Department of Computer Science

Supervisory Committee

GoCity: A Context-Aware Adaptive Android Application

by

Qian Yang
B.Sc., of Electronic Science and Technology of China (UESTC), 2006

Supervisory Committee

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

Dr. Alex Thomo, (Department of Computer Science)
Departmental Member

Abstract

Supervisory Committee
Dr. Hausi A. Müller, (Department of Computer Science)
Supervisor
Dr. Alex Thomo, (Department of Computer Science)
Departmental Member

GoCity is designed to provide city visitors with up-to-date and context-aware information while they are exploring a city using Android mobile phones. This thesis not only introduces the design and analysis of GoCity, but also discusses four problems in leveraging three concepts—context-awareness, self-adaptation, and usability—in current mobile application design. First, few contexts other than location and time have been used in actual mobile applications. Second, there is no clear classification of context information for mobile application design. Third, mobile application designers lack systematic mechanisms to address sensing and monitoring requirements under changing context situations. This is crucial for effective self-adaptation. Fourth, most mobile applications have low usability due to poor user interface (UI) design. The model proposed in this thesis addresses these issues by (i) supporting diverse context dimensions, (ii) monitoring context changes continuously and tailoring the application behavior according to these changes, and (iii) improving UI design using selected usability methods. In addition, this thesis proposes two classifications of context information for mobile applications: source-based classification—personal context, mobile device context, and environmental context; and property-based classification—static context and dynamic context. The combination of these two classifications helps

determine the observed context and its polling rate—the rate at which the context is collected—effectively.

A distinctive feature of GoCity is that it supports two interaction modes—static mode and dynamic mode. In static mode, the application generates results only after the user sends the request to it. In other words, it does not actively generate results for users. In contrast, in the dynamic mode, the application continuously updates results even if the user does not send any request to it. The notion of an autonomic element (AE) is used for the dynamic mode to make GoCity self-adaptive. The polling rates on different contexts are also handled differently in the dynamic mode because of the differences among context properties. In addition, GoCity is composed of, but not limited to, four sub-applications. Each sub-application employs a variety of context information and can be implemented as an independent mobile application. Regarding usability, GoCity focuses on providing a simple and clear user interface as well as supporting user expectations for personalization.

An experiment which involves a person visiting the city of Victoria was conducted to evaluate GoCity. In this evaluation, three determining factors of usability were employed to qualitatively and quantitatively assess GoCity. In addition, the static mode and dynamic mode were evaluated separately.

Table of Contents

List of Tables

List of Figures

Acknowledgments

This work would have been impossible without immense assistance from my supervisor Dr. Hausi A. Müller. He has provided the most valuable advice and moral support.

I also want to express my gratitude to Dr. Alex Thomo for providing feedback for my thesis.

Also, I would like to thank all the members of the Rigi research group at the University of Victoria for their suggestions and corrections.

In addition, I am endlessly grateful to Qin Zhu, Yan Zhuang and Ming Lu for sharing their ideas and advice at all times. Finally, I want to thank my husband, Yangyang Liu, who is always supporting and encouraging me.

Chapter 1 Introduction

New mobile computing environments, such as smartphones and tablets, are relaxing the constraints imposed by stationary desktop computing systems [1]. This trend is accelerated by the huge shipments of smartphones as well as the popularity and versatility of mobile applications in recent years, which motivates this underlying research. As a mobile computing paradigm, context-aware computing should be widely utilized in mobile application design [2]. And as a product, the mobile application's key factor of success is the extent of its friendliness to the user. The structure of this chapter is as follows. The problem statement section illustrates issues existing in designing previous and current user-friendly and context-aware mobile applications. The approach section demonstrates the proposed solutions to these issues. Finally, this chapter outlines the contributions of this research and the organization of this thesis.

1.1 Motivation

Northrop et al. indicated that current systems are evolving from software intensive systems to socio-technical ecosystems, where dynamic groups of users, stakeholders, and businesses, as well as software and hardware infrastructures have to cooperate in complex and changing environments [3]. As a result, smart interactions and smart services proposed by Chignell et al. are key components of socio-technical ecosystems [4]. One critical challenge in this emerging research field is the ability to enhance the behaviour of an application by taking into account the context of its use [5]. Thus,

building a context-aware application is highly desirable in today's complex computing environments.

In addition to academic research, industry provides a large market for building context-aware applications. First, the smartphone market is growing dramatically, and thus thousands of mobile applications are created. Smartphones have become one of the most popular commodities in people's daily lives. More and more people are relying on mobile applications. According to the worldwide smartphone market data published by Canalys as depicted in Figure 1, around 158.3 million smartphones were sold in Q2 2012, which represents a year-on-year growth rate of 46.9% over 2011 [7]. Second, with easy access to infrastructures such as GPS satellites, Bluetooth services, Wi-Fi networks and 3G networks, mobile devices are surrounded by an enormous amount of information. Some pieces of information, such as location, weather, date, and time, have been utilized in modern mobile applications to facilitate context-awareness [2, 13, 23]. Popular mobile applications, such as YELP, demonstrate that as long as the mobile application is able to actively take advantage of and react to the context information collected by smartphones, it will create an excellent user experience. Third, powerful smartphone platforms (e.g., Android, iOS, Windows Phone, Symbian OS, BlackBerry OS) provided by current smartphone leaders (e.g., Google, Apple, Microsoft, Nokia and RIM) already attracted a large number of companies and developers to design and develop great mobile applications. Moreover, the dramatic competitions among those smartphone leaders also revealed that mobile markets provide enormous opportunities as well as challenges for not only devices but also applications. Fourth, as a computing platform, smartphones are particularly suitable for building user-centric context-aware adaptive applications. Since

they always follow the user, they provide valuable information about a user's current situation. If they can record user activities in different situations, they are even able to predict user preferences or follow activities. Moreover, they can easily access various data sources thanks to their powerful hardware, multiple input methods, advanced connectivity and multiple sensors. Finally, modern web services that are usually exposed by an Application Programming Interface (API) simplify the method of leveraging a variety of data sources in mobile application design.

To sum up, designing context-aware adaptive mobile applications is an important trend in current computing. Both academic research and industry display tremendous interest in this field. This ultimately attracts an increasing number of companies and developers dedicated to creating mobile applications. In addition, context-aware adaptive applications provide more intelligent interactions, creating a distinct user experience.

Global smart phone market
Shipments into the channel, split by platform, Q2 2012, Q2 2011

Platform	Q2 2012 shipments (million)	% share	Q2 2011 shipments (million)	% share	Growth Q2'12/Q2'11
Total	158.3	100.0%	107.7	100.0%	46.9%
Android	107.8	68.1%	51.2	47.6%	110.4%
iOS	26.0	16.4%	20.3	18.9%	28.0%
BlackBerry	8.5	5.4%	12.5	11.6%	-32.1%
Symbian	6.4	4.1%	18.1	16.8%	-64.6%
Windows Phone	5.1	3.2%	1.3	1.2%	277.3%
bada	3.3	2.1%	3.1	2.9%	5.1%
Others	1.2	0.8%	1.1	1.0%	15.2%

Source: Canalys estimates, © Canalys 2012

Figure 1. Canalys smartphone analysis, quarterly shipment data [7]

1.2 Problem Statement

Modern mobile applications should be designed to be context-aware, self-adaptive and user-friendly. However, like any other emerging field, challenges and issues are inevitably associated with its growth. This thesis targets four problems as listed below and provides appropriate solutions:

- Few contexts other than location and time have been used in actual mobile applications.
- There is no clear classification of context information for mobile application design.
- Mobile application designers lack systematic mechanisms to address sensing and monitoring requirements under changing context situations. This is crucial for effective self-adaptation.
- Problems in UI design lead to low usability for many mobile applications.

Context-awareness has been studied for over a decade. The benefit of being context-aware is that the user can get better support and the interface can become more invisible if the device knows more about the user, the task and the environment. Most previous research on context-aware applications or systems exhibits a strong focus on location and time [8, 9, 10, 11]. However, there is much more than location and time to context [5, 12]. Some work has been done to enable mobile devices to exploit other context dimensions [13] and current smartphone platforms (e.g., Android, iOS) provide interfaces to developers letting them easily access context information (e.g., battery life, network

information, phone contacts, screen orientation). However, how to use that information effectively is still a challenging problem for application programmers.

Different context types have different properties. For example, time is the most frequently changing context, and thus the way of sensing and managing is different from other contexts. To manage diverse context information effectively that is specifically applied in mobile applications, it is beneficial to provide a way to characterize and categorize them. Context taxonomies such as the general classification proposed by Villegas [5] have been proposed to help build more concrete taxonomies for various application domains. However, currently we still lack a concrete context classification system for mobile applications. As a result, we argue that we need a new, clear and distinct categorization of context information for mobile applications.

A mobile computing environment is a highly changing environment in which context information changes can occur at any moment. These changes affect the mobile application that takes advantage of context information, requiring the application to react to them. For example, when the battery power becomes lower than a threshold, the application disables some functions such as showing maps on the screen in order to save power. This capability in which an application senses the change in the environment and reacts to the change is called self-adaptation. Self-adaptation enables applications to not only accomplish some specific functions like the previous mentioned example, but also prevent possible failures caused by unexpected changes in the environment. For example, network connectivity is a factor affecting applications that require it. A suddenly broken network connection might cause application to crash if the application is not able to adapt to this change. From what was mentioned above, change is a keyword for self-adaptive

systems. The prerequisite of being self-adaptive for mobile applications is that they can sense the change in the environment. Additionally, they need to have proper mechanisms to respond to the change. In current mobile applications, however, designers did not pay much attention to context change. Even if they integrated the idea of adapting to context change into their design, they only focused on changes of location and time. Nevertheless, there is more to context than location and time as we discussed before, and thus there is more to context change than the location and time change. Therefore, we need a systematic mechanism to address sensing and monitoring requirements under changing context situations.

As a product, the application is successful only if it is widely accepted and used by many users. For this, usability is critical to realize more user-friendly applications. Organizations, such as user-centric.com, usability.gov and the HUSAT Research Institute, are doing research on usability engineering. Some of their research focuses on the mobile application field. Thanks to their research, I came up with two challenges in improving the usability of mobile applications. The first one is how to make the user interface simple and clear. Due to a mobile phone's characteristics, such as limited screen size, navigation restrictions and broad audience experience levels, a simple and clear user interface is particularly vital. Today's mobile application has already noticed the importance of the user interface. However, we can do better. Problematic user interfaces frustrate users taking too much time to understand the components and learn the operations of the application, while a simple and clear interface contributes to an easy and enjoyable user experience. The second one is how to support user expectations for personalization. Several years ago, supporting user expectations for personalization might

not have been taken seriously. However, today, users expect significant personalization. This includes their mobile applications. For example, in the case of an application for reading news, users assume that they can set their interested locations or decide the categories that should appear on the first screen. If those options are not available, they often become frustrated and dissatisfied with the application. When designing an application, it is important to clearly indicate which items can be personalized and how users can personalize them. Unfortunately, presently thousands of mobile applications fail to realize this requirement.

I designed and implemented GoCity as a working prototype. It aims to illustrate how the aforementioned problems can be solved with context-aware adaptive mobile applications.

1.3 Approach

In order to solve the aforementioned design issues of context-aware adaptive mobile applications, this thesis proposes a source-based classification and a property-based classification of context information for mobile applications, and a model that particularly addresses the following features:

1) Context information is collected and governed based on the proposed classifications of context information. A key component called a Context Manager collects a variety of contexts from available sources. The property-based classification helps the context manager to use proper polling rates to gather contexts.
2) An autonomic element (AE) proposed by IBM [14] is employed for self-adaptation. Working as the autonomic manager of an AE, the context adaptation

manager—another key component in this model—triggers the context manager to sense the context change and drives the application to reflect upon the context change. The context monitoring rate is also adjusted by the context adaptation manager.

3) A filter component gathers context information required by the consumer application and refines the generated contents to be shown to the user.
4) A service manager component deals with all the interactions between the application and external web services, which minimizes the risk of faults caused by web service failures.
5) A set of adaptation rules contained in the context adaptation manager determines the polling rate on different contexts. Moreover, this knowledge helps determine the current situation and the results that should be generated for the user.
6) Intelligence is enhanced by adding autonomic behaviors. The model supports two interaction modes: static mode and dynamic mode. In the static mode, the application gives results only after the user sends the request to it; while in the dynamic mode, the application continuously updates results by following some adaptation rules even if the user does not send any request to it.
7) It is easy to extend the functionality of an application built this way. This model is composed of several components. Each component is self-contained and easy to extend.

In order to demonstrate the feasibility of this proposed model, a working prototype named GoCity has been designed and implemented on the Android platform. GoCity is a user-friendly context-aware adaptive mobile application, providing city visitors with up-

to-date and context-aware information while they are exploring a city using Android smartphones. The charming feature of GoCity is that it has two interaction modes: static and dynamic. The static mode does not actively generate results for users; whereas the dynamic mode takes advantage of the full power of the context adaptation manager, periodically monitoring context changes and dynamically tuning the polling rates on different contexts and generating results according to the designer's adaptation rules.

To be specific, the context manager in GoCity accesses available sources to get a variety of contexts. In the dynamic mode, GoCity is controlled by the context adaptation manager that has a closed control loop for monitoring context changes as well as guiding the context manager to use proper polling rates to update contexts and GoCity to generate appropriate contents. The initial polling rate for diverse context is different due to differences among context properties. For example, the location of a moving person might change every second while the weather does not change frequently, which leads to a faster polling rate for location than for weather. As a result, GoCity defines different polling rates for different contexts as suggested by Chen et al. [2]. In addition, GoCity is composed of but not limited to four sub-applications, each of which leverages some contexts and can be implemented as an independent mobile application. To improve usability, GoCity provides a simple and clear user interface and supports user expectations for personalization. For example, users can arbitrarily add or delete business types to adjust their favourites through the simple interfaces as depicted in Figure 2. This function also shows that it allows users to personalize their requests.

Figure 2. Demonstration of GoCity's user interfaces

1.4 Contributions

This thesis addresses four significant problems in context-aware adaptive mobile applications and provides solutions. It classifies context information for mobile applications based on its source and property. It proposes an application model for building context-aware adaptive mobile applications. The implemented prototype called GoCity demonstrates this model's feasibility. GoCity uses simple and clear user interfaces and supports user expectations for personalization in order to improve usability.

1.5 Thesis Outline

Chapter 2 introduces the background knowledge which forms the foundation of the thesis. It includes background on smartphones, mobile applications, context-aware

computing, adaptive systems, autonomic computing, Android, web services, and usability. Chapter 3 proposes a classification of context for mobile applications. Chapter 4 discusses the model that is proposed to solve problems in designing context-aware adaptive mobile applications. Chapter 5 introduces the implemented prototype GoCity. Chapter 6 talks about content generation. Chapter 7 discusses the adaptive solutions applied in this approach. The evaluation of GoCity is described in Chapter 8. Finally, this thesis ends with conclusions and future work in Chapter 9.

Chapter 2 Background

This chapter introduces the background knowledge which forms the basis of this thesis's work. The concepts and technologies employed in this work include smartphones, mobile operating systems, Google's Android platform, mobile applications, context-awareness, smart interactions, adaptive techniques, autonomic computing, web services, and usability.

2.1 Smartphones

> **"A smartphone is a cellular telephone with built-in applications and Internet access. It provides digital voice service as well as text messaging, e-mail, Web browsing, still and video cameras, MP3 player, video viewing and often video calling. In addition to their built-in functions, it can run myriad applications, turning the once single-minded cellphone into a mobile computer." [15]**

A smartphone unites the functions of a personal digital assistant (PDA) and a mobile phone. It runs a mobile operating system that provides a standardized interface and platform for mobile application developers to create the third-party applications that run on the phone. Present popular mobile operating systems include Apple iOS, Google Android, Microsoft Windows Phone 7, Nokia Symbian, Research In Motion BlackBerry OS, and embedded Linux distributions such as Maemo and MeeGo [16]. Compared to standard phones, in addition to advanced mobile operating systems, smartphones have

outstanding hardware, such as powerful processors and graphics processing units, abundant memory, and high-resolution screens with multi-touch capability.

The first smartphone called Simon was designed by IBM in 1992, and was released and sold by BellSouth in 1993 [16]. With the development of technology, smartphones increasingly gained widespread popularity due to their convenience and PC-like functions. Especially in recent years, the smartphone market grew tremendously, coming up with massive opportunities for mobile engineers. According to the worldwide smartphone sales data released by IDC as depicted in Figure 3, in 2011 491.4 million smartphones were sold worldwide, up from 304.7 million in 2010, for a growth rate of 61.3% [18]. The largest growth came from suppliers of Android-based handsets (particularly Samsung and HTC) as well as Apple. Nokia lost heavily—it grew only a sluggish 22.8%.

Vendor	FY 2011 Shipment Volumes	FY 2011 Market Share	FY 2010 Shipment Volumes	FY 2010 Market Share	Year Over Year Change
Samsung	94.0	19.1%	22.9	7.5%	310.5%
Apple	93.2	19.0%	47.5	15.6%	96.2%
Nokia	77.3	15.7%	100.1	32.9%	-22.8%
Research In Motion	51.1	10.4%	48.8	16.0%	4.7%
HTC	43.5	8.9%	21.7	7.1%	100.5%
Others	132.3	26.9%	63.7	20.9%	107.7%
Total	**491.4**	**100.0%**	**304.7**	**100.0%**	**61.3%**

Figure 3. Worldwide smartphone shipments by vendors [18]

Figure 4 depicts that Android based smartphones occupy 68.1% worldwide smartphone sales in the second quarter of 2012. Thanks to the Android platform's dramatic growth in the industry and my intimate personal ties with Android's products, GoCity was developed for Android smartphones. These smartphones are based on

Android 2.2. Some of the released devices that support Android are Samsung Galaxy S, HTC Thunderbolt 4G, Motorola Atrix 4G, and HTC Inspire 4G.

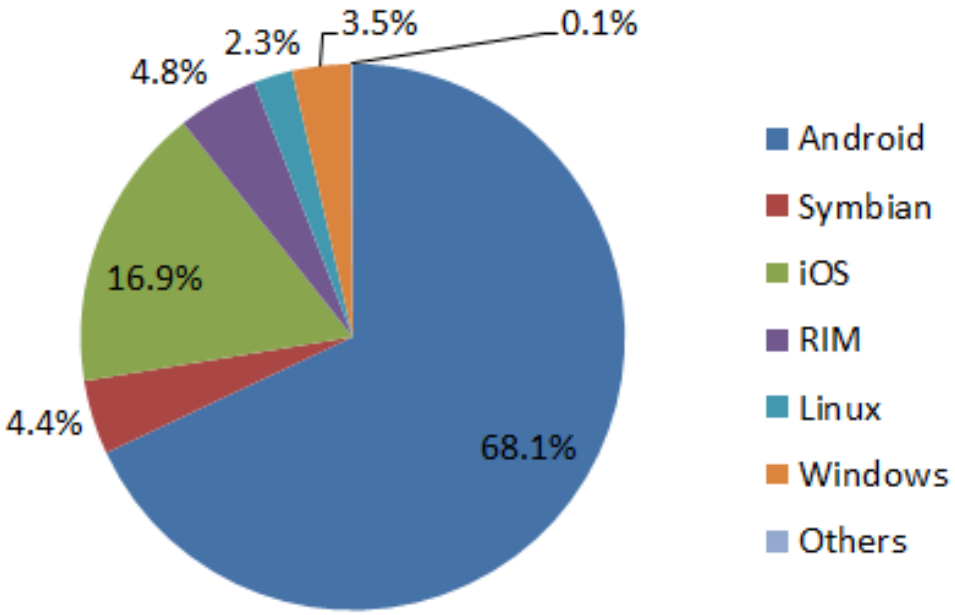


Figure 4. Share of worldwide 2012 Q2 smartphone sales by operating systems, according to IDC [17]

2.1.1 Google's Android Platform

Android is a software stack that not only includes an operating system but also contains middleware and key applications for mobile devices [6]. Android is an OHA (Open Handset Alliance) project and powered by a Linux-based operating system. It enables fast application development in Java. Android was designed to serve the needs of mobile operators, handset manufacturers, and application developers. The members have committed to release significant intellectual property through the open source Apache 2.0 license. It allows developers to easily build third-party mobile applications on it.

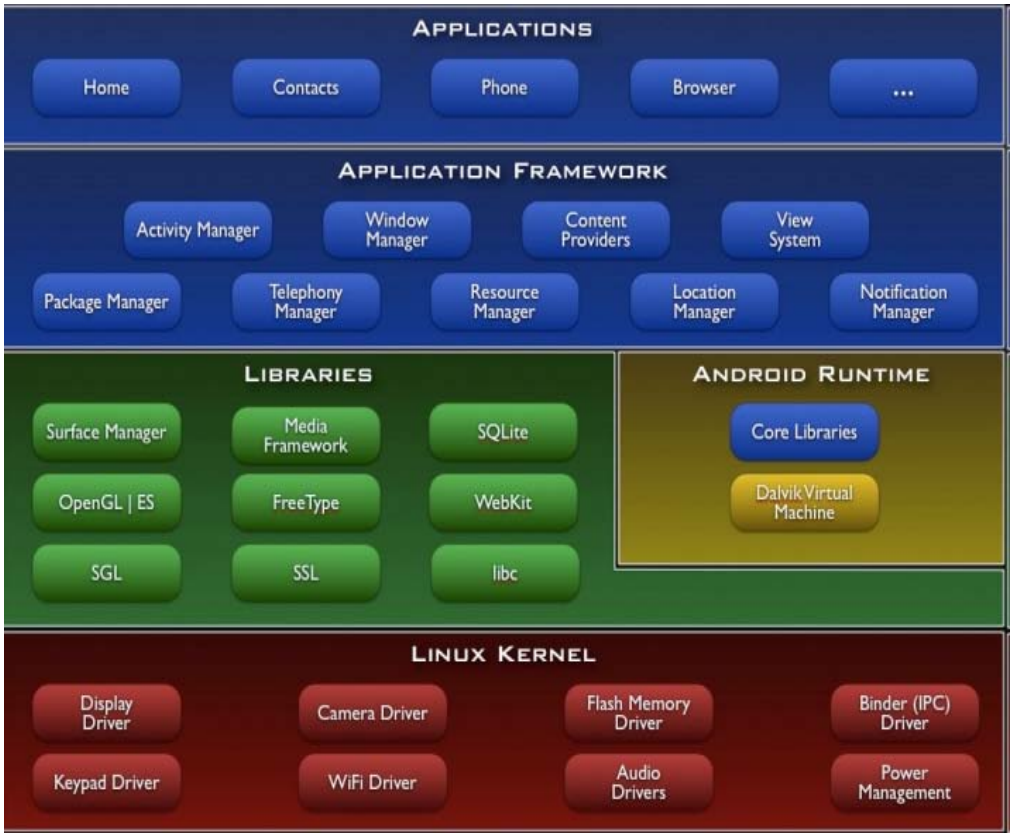


Figure 5. Android platform architecture [6]

As presented in Figure 5, the Android operating system consists of four major components: applications, application framework, libraries and Android runtime, and Linux kernel as discussed below:

- Applications—A set of core applications are already built within Android, such as contacts, browser, an email client, calendar, maps, and SMS program. All of them are written in Java. And each Android application is composed of one or more application components: activities, services, content providers, and broadcast receivers.
- Application framework—Android provides framework APIs that can be fully accessed by developers. These APIs help developers to create, manage and use a

variety of application components to build applications. The key feature of an Android application framework is to enable and simplify the reuse and replacement of application components. Basically, any application's capabilities can be published and then be taken advantage of by other applications. Meanwhile, a set of services and systems can be utilized by all Android applications through this framework. Table 1 shows their functions.

Table 1. Services and systems employed by Android applications

Feature	Role
View System	Used to build an application, including lists, grids, text boxes, buttons, and embedded web browser
Content Provider	Enabling applications to access data from other applications or to share their own data
Resource Manager	Providing access to non-code resources (localized string, graphics, and layout files)
Notification Manager	Enabling all applications to display customer alerts in the status bar
Activity Manager	Managing the lifecycle of the applications and providing a common navigation backstack

- Libraries—These are Android's native libraries written in C/C++. They are used by various components of the Android system to handle different types of data. The Android application framework exposes these capabilities to developers. Table 2 lists some of the important native libraries:

Table 2. Native libraries of Android

Native Library	Function
System C Library	A BSD (Berkeley Software Distribution)-derived implementation of the standard C system library, tuned for embedded Linux-based devices
Media Libraries	Based on PacketVideo's OpenCORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG
Surface Manager	Manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications
LibWebCore	A modern web browser engine which powers both the Android browser and an embeddable web view
SGL	The underlying 2D graphics engine
3D Libraries	An implementation based on OpenGL ES 1.0 APIs; the libraries use either hardware 3D acceleration or the included, highly optimized 3D software rasterizer
FreeType	Bitmap and vector font rendering
SQLite	A powerful and lightweight relational database engine available to all applications

- Android runtime—Consisting of core Java libraries and the Dalvik virtual machine. The core Java libraries are different from the Java SE and Java ME libraries, but come with most functionalities of the Java SE libraries. A virtual machine called Dalvik has been specifically designed for Android and optimized for battery-powered mobile devices with limited memory and CPU. Each Android application runs in its own process with its own instance of the Dalvik

virtual machine. The prominent feature of Dalvik is that it improves a device's efficiency on running multiple VMs by executing files in the Dalvik Executable (.dex) format instead of Java byte code (.class) format. Figure 6 depicts how Java code is finally executed in Dalvik virtual machine. A tool called dx transforms classes compiled by a Java compiler into the .dex format which is optimized for minimal memory footprint.

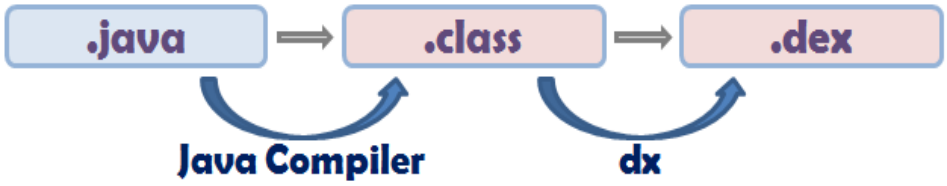


Figure 6. Process of generating .dex file executed by Dalvik VM

- Linux Kernel—Linux version 2.6 provides Android with core system services such as memory management, security, network stack, process management, and driver model. It also acts as an abstraction layer between the hardware and the rest of the software stack.

Regarding developing on the Android platform, a number of features are supported by Android for the development environment. The following six features played an important role in the development of GoCity:

- Storage—SQLite, a lightweight relational database, is available in Android for data storage purposes.
- Connectivity—Connectivity technologies supported by Android include GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC, and WiMAX.
- Multiple Language Support—Android supports multiple human languages.

- Web Browser—A stable web browser is available in Android. It is based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine.
- Java Support—Most Android applications are written in Java, although there is no Java Virtual Machine in the platform and Java byte code is not executed. Java classes are transformed into Dalvik executables and run on Dalvik. J2ME support can be provided via third-party applications.
- Additional hardware support—Android can use video/still cameras, touchscreens, GPS, accelerometers, gyroscopes, magnetometers, dedicated gaming controls, proximity and pressure sensors, thermometers, accelerated 2D bit blits (with hardware orientation, scaling, pixel format conversion) and accelerated 3D graphics.

2.1.2 Development in Android

In Android, developers implement one or more application components that finally compose Android applications. Each component presents a specific behaviour and must be declared in a manifest file. Here are the five core application components:

- Activity—An activity presents a user interface in a single screen. For example, a music player application might have one activity (screen) that shows a list of song and album titles, another activity to play the music, and another activity for showing the lyric. Each activity is independent and might be started by other applications if its original owner allows. For example, a calendar application can start the activity in a weather forecast application that shows weather conditions on a specific day, in order for the user to plan their schedule. An activity is implemented as a subclass of Activity.

- Service—A service runs in the background to perform long-running operations or to offer functionalities for applications, and does not interact with a user. For example, a service can sense a user's location in the background while a user is using another application. Services can be started by another component such as an activity in order to interact with it. A service is implemented as a subclass of Service.
- Content Provider—A content provider is a component for managing application data that can be shared. Android supports a number of methods of data storage, such as the file system, SQLite databases, repositories on the web, or any other persistent storage location accessible by the application. Developers can have one application to query or even modify the stored data of other applications through content providers. A content provider is implemented as a subclass of ContentProvider.
- Broadcast Receiver—A broadcast receiver is able to respond to system-wide broadcast announcements. It does not display a user interface but might generate a notification to alert users when a broadcast event happens. Both the system and applications can initiate broadcasts. For example, a broadcast announcing a low-level battery can be generated by either the system or an application dedicated to monitoring the battery. Developers can use broadcast receiver to catch a specific event and then to initiate another component such as starting a corresponding service. A broadcast receiver is implemented as a subclass of BroadcastReceiver.

- Intent—A component to activate activities, services, and broadcast receivers. It performs as a messenger that requests an action from other components. An intent is created with the Intent object.

Each Android application has a manifest file named AndroidManifest.xml, in which all components that compose the application are declared. This file is located at the root of the application project directory, and is read when the system starts a component of the application. In addition to declaring application components, it also identifies user permissions, hardware or software features that the application requires, API libraries that the application links against, and so on.

A variety of custom tools that help develop Android applications are provided in the Android SDK. Three of the most significant tools used in implementing GoCity are listed in Table 3.

Table 3. Tools provided in the Android SDK used to implement GoCity

Name	Role
Android Emulator	A virtual mobile device that runs on a computer - used to design, debug, and test applications in an actual Android run-time environment. Figure 7 illustrates an emulator for Android 2.2.
Android Development Tools Plugin	It is for the Eclipse IDE. It adds powerful extensions to the Eclipse integrated environment.
Dalvik Debug Monitor Service (DDMS)	It is integrated with Dalvik. This tool supports process management on an emulator and assists in debugging.

To summarize, Android facilitates the ways that software developers implement and test mobile applications.

Figure 7. Emulator for Android 2.2

2.2 Mobile Applications and Usability

Mobile applications are software systems running on mobile devices and performing certain tasks for the user of mobile devices. They occupy an indispensible segment in today's global market and grow fast. According to the analytical data published by International Data Corporation (IDC), in 2010 more than 300,000 applications were downloaded 10.9 billion times [20]. IDC also predicted that global downloads will reach 76.9 billion in 2014 and will be worth US$35 billion. The wide use of mobile applications is due to the many functions they serve. In addition to providing basic services such as messaging and dialling, mobile applications can offer advanced services

such as games and videos. Moreover, modern mobile platforms and hardware expose the power to enable building more versatile and advanced mobile applications in present mobile devices, such as browsers, maps, email clients, and so on. In addition to their functions, the ease of obtaining mobile applications also attracts users to use them. Many mobile phone vendors pre-install applications such as social network clients, browsers and streaming players to attract users to buy their phones. Meanwhile, users can also download their favourite applications over the online mobile application store and then install them themselves. Regardless of how they are discovered by users, mobile applications are a large and continuously growing market served by an increasing number of mobile developers, publishers and providers. Undoubtedly, the market for mobile applications is very competitive. For the mobile applications market, the good news is that customers seem very willing to give new applications a try. However, the bad news is that one in four mobile applications once downloaded does not get a second try by the user according to a study of Localytic [20]. As a result, enhancing the usability of mobile applications is crucial for the success of most current mobile applications.

2.2.1 Usability

International Organization for Standardization (ISO) defines usability as “The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use” [21]. In general, usability refers to how well, easily, and efficiently users can use a product to accomplish their goals, and how satisfied users are with the functionalities and operations of the product. According to what usability is concerned with, a product with high usability is easy to learn and efficient to use, and its functionalities and operations are highly

accepted by most users, and thus its design is successful in the market. In the present mobile application market, however, most applications have low usability. In fact, one in four mobile applications never had a second try once being downloaded and subsequently uninstalled by the user as mentioned in the previous section. Therefore, how to build a mobile application that will become popular among users is still a huge challenge. Developers usually focus on functionality and performance when designing a software application. However, usability is not only determined by these two factors, although it is often associated with them. In fact, there are more factors that need to be taken into account with respect to the usability of a product. Basically, usability measures the quality of a user's experience when using a product and thus is a combination of the following factors [21]:

- **Ease of learning**—How fast can a user who has never used the product before learn to use this product sufficiently well to accomplish basic tasks?
- **Efficiency of use**—Once the user has been familiarized with operating the product, how fast can he or she uses it to accomplish tasks?
- **Memorability**—After a period of not using the product, is it easy for a user to remember how to effectively use it or does the user has to re-learn everything?
- **Errors**—Is that easy to make an error when the user operates the product and how easily can the product recover from the error?
- **Subjective satisfaction**—Is the user satisfied with the process of using the product and how much does the user like using it?

According to these factors, GoCity supports a number of features to provide users with a pleasant experience. First, GoCity provides a simple and clear user interface to ensure users can learn and use it easily and quickly. Second, GoCity supports user expectations for personalization. Third, GoCity is context-aware and thus users achieve desired results with fewer actions. Lastly, GoCity is self-adaptive and can prevent some failures caused by unexpected changes in the environment.

2.3 Context-Aware Computing

Context-aware computing has been studied for over a decade. It provides methodologies for designing applications which are able to discover and take advantage of surrounding contexts. Many researchers and practitioners built context-aware applications. Their work has demonstrated that context-aware applications make the interaction between the user and the application, and the user and the environment, easier [2]. From the developer's perspective, the benefit of making an application context-aware is that the more the context can be utilized by the application, the better the support and the experience for the user is. From the user's perspective, the more the application is context-aware, fewer operations are required by the user to get the desired results.

With the growing popularity of handheld devices, context-aware computing has been intensely leveraged in mobile computing, because mobile devices, such as smartphones, provide ideal platforms for building context-aware applications. The projects described in [1, 24, 25, 26] illustrate this point. First, mobile devices are composed of many advanced sensors which help ascertain the current status of the mobile device and its environment. Second, they always follow users' behaviors and thus users'

preferences are visible to them. Third, they are capable of accessing many data sources such as downloading data through wireless networks. All of these features of mobile devices determine that mobile computing must be linked with context-aware computing to gain the benefits. Due to more and more attention paid to context-awareness in mobile computing, many context-aware mobile applications appear in the market. In Google's Android Developer Challenge conducted in 2008, five out of the top 10 award-winning applications exploit the environment to provide users with a distinctive experience [23]. Accordingly, mobile users also benefit from using context-aware applications. For example, one award-winning application is to detect nearby users so as to establish social connections.

2.3.1 Definition of Context and Context-Awareness

Noticing that building context-aware applications is becoming more prevalent in mobile computing, we first need to know what context and context-awareness are. Without a clear understanding of their definition, application designers can neither effectively choose what context to use nor provide appropriate mechanisms to govern the context data in their applications.

Schilit and Theimer define context as "location, identities of nearby people and objects, and changes to these objects" in their work that first introduced context-aware computing [22]. Since that time, many definitions for context have emerged. However, most of them enumerate examples and use terms such as environment to define context. According to Dey, these definitions are either too abstract or too specific and thus hard to apply in designing or implementing actual context-aware applications [27]. For example, Ryan considered context to be user's locations, time, identity and environment in his

work [33]. The question is whether elements out of the list cannot be considered as context, such as user's preference. Obviously, the answer is no. Therefore, a more accurate concept of context is needed in context-aware computing. In 2000, Dey proposed a definition of context according to his dedicated research on context-aware computing. His definition is widely accepted and cited by other researchers:

"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." [27]

However, this definition neither includes how the context is obtained, processed and maintained for an application, nor considers the dynamic nature of the context. To fill in the blanks, some researchers such as Zimmermann [32] came up with the operational definitions for context. In this thesis, however, I would like to highlight Villegas' operational definition of context proposed in 2010 [5], because her definition includes all aspects that a context-aware system is concerned with.

"Context is 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. This context information must be modeled in such a way that it can be pre-processed after its acquisition from the environment, classified according to the corresponding domain, handled to be provisioned based on the system's requirements, and maintained to support its dynamic evolution."

Regarding context-awareness, Dey also proposed the definition in his dissertation [27] as:

"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task."

According to Villegas's definition of context, context must be modeled first. The process of modeling context is to determine the context that is relevant for the application, and to present this information in a way that the application can understand it. Clearly, even though an application can be affected by many context variables, it is impossible to capture all of them. Therefore, context modeling helps software engineers to focus on context variables that are relevant to the system's requirements. Additionally, relevant context types must be parameterized in such a way that applications can understand them.

After contexts are obtained and modeled, the next step is to handle them. At this point, contexts have been parameterized, and thus are easily processed in the program to accomplish selected goals. For example, after the weather is parameterized to a number, the functionality served by the application can vary according to the change of the weather. When the number is higher or lower than some particular threshold, the behavior of the application can be different. Now, the question is how the application knows the time to capture a specific context parameter (e.g., the weather parameter) and then react to the situation represented by captured contexts. Basically, there are two methods to achieve this purpose. One is to build a model in which each component performs a context management related task so that all contexts can be uniformly

managed. Moreover, this model provides global context-based policies, directing the system to properly react to a specific situation. The other one is a control-oriented approach which utilizes the feedback loop to manage a context unit. It is noteworthy that the feedback loop ensures that the dynamic evolution of the context will never be ignored by the system. In order to offer a global management on contexts as well as their dynamic nature, GoCity, the prototype introduced in this thesis, employs both.

Numerous mobile applications have disclosed that location and date and time are leveraged most. However, there are many more contexts [12]. In fact, thanks to present advanced hardware and service support, modern mobile devices are capable of collecting various contexts from the environment, the user and the device itself.

2.4 Self-Adaptive Systems and Autonomic Computing

Modern software systems consist of dynamic groups of users, stakeholders, businesses, and software and hardware infrastructures. Any change contributed by such a complex environment can possibly affect the normal work of the system or even cause failures. Fortunately, software engineers have noticed this challenge and proposed some methodologies allowing systems to make decisions at runtime according to current situations. One of such methodologies is the engineering of self-adaptive software systems [14, 19, 35].

The “self” prefix indicates that systems are capable of working without or with little human intervention. To be specific, a self-adaptive system has the capability of monitoring its internal state and changes in the external environment, and to adjust its behavior at runtime accordingly. It frees operators from the tedious and time-consuming task of monitoring and managing the system. In some fields, this feature is referred to as

intelligence. However, smart (as in smartphones) might be more appropriate than intelligent. So, how does the system become such intelligent or smart? Typically, a group of policies forming the guidance that the system should comply with are realized in such systems. And these policies represent the high-level objectives of the system, including both functional and non-functional requirements [28].

Since the topic of constructing self-adaptive systems has attracted many researchers, many approaches to developing self-adaptive systems have been proposed from various research areas of software engineering. Some of these areas are requirements engineering [36], software architecture [34], component-based development [31], and middleware-based development [29]. Self-adaptive systems can be categorized into two types—top-down and bottom-up—according to the method of their developments [56]. A top-down self-adaptive system is considered as an individual system. It is often centralized and guided by a central controller or its global policy. Surrounded by an evolving environment, it evaluates its own behaviour according to the functional or non-functional requirements at run time, and adjusts its behaviour when the assessment indicates that its behaviour is not suitable to achieve the global goals in the situation at that time. Such a system often operates with an explicit internal representation of itself and its global goals. It is notable that the behaviour of a top-down self-adaptive system can be composed or deduced by analyzing its components [30]. By contrast, a bottom-up self-adaptive system is designed as a cooperative system. It is typically decentralized and consists of a large number of components that interact locally with each other according to some simple rules to accomplish global goals. Interactions among these components form the global behaviour of the system, and thus it is difficult to deduce the global system's

behaviour by analyzing only local interactions among some components [30]. Unlike a top-down self-adaptive system, a bottom-up self-adaptive system does not use the internal representation of itself and its global goals. Although a self-adaptive system can be built by applying only one of them, in practice engineers prefer to incorporate both.

As discussed above, a self-adaptive system is expected to accomplish requirements at runtime in response to changes. However, the runtime management of a system is usually time-consuming and costly. Thus, an appropriate mechanism to monitor changes of the system and its surrounding environment as well as to adjust the system behaviour by following global rules is expected. Feedback loops, also known as closed control loops, provide such mechanism. Noticing the importance of feedback loops in self-adaptive systems, Müller et al. argued that feedback loops must become first-class citizens in adaptive systems, and should be explicit in the design and analysis and clearly traceable in the implementation [37]. The fundamental question is what are the components of the feedback loop and which properties should the feedback loop possess to accomplish self-adaptation. Salehie and Tahvildari demonstrated that a feedback loop is essentially composed of four processes, as well as sensors and effectors [38]. As Figure 8 depicts, the four adaptation processes are monitoring, detecting, deciding and acting.

At the monitoring process, the data reflecting the current state of the system is collected and correlated from the sensors, and translated into behavioural patterns and symptoms. Usually event correlation or simply threshold checking are leveraged to realize this process. At the detecting process, the symptoms are analyzed to ascertain when and where a change or response is required for the system. At the deciding process, with the help of some policies or rules, what changed in the system and how to change

the system is determined to fulfill the global goals. And as the fourth adaptation process, the acting process is finally responsible for executing planned actions determined by the deciding process with the use of effectors. In addition to the four adaptation processes, sensors and effectors are essential parts of a feedback loop. Sensors monitor the properties of the system and effectors perform actual actions on the system to accomplish adaptation. It is noteworthy that practical systems often involve a number of separate feedback loops to achieve adaptation goals.

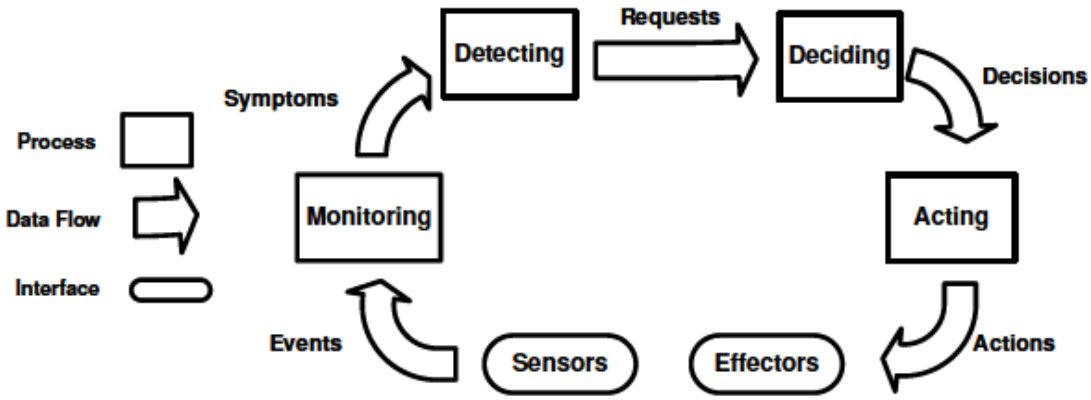


Figure 8. Four adaptation processes in self-adaptive software [38]

2.4.1 Autonomic Computing

In current research, self-adaptive systems are strongly related to autonomic computing. Many researchers did not draw a clear distinction between them and often used these terminologies interchangeably. The primary reason is that both are aimed to provide self-adaptation functionalities to systems for managing complexity. The concept of autonomic computing (AC) was first introduced by IBM in 2001 to describe computing systems that can manage themselves by following high level objectives [14]. Inspired by the autonomic nervous system of human bodies, IBM suggested that

computing systems should also be autonomic systems which have one or more of the following autonomic properties [14]:

- Self-configuration—Capability of reconfiguring automatically and dynamically in response to changes
- Self-healing—Capability of automatically discovering, diagnosing and reacting to disruptions
- Self-optimization—Capability of managing resource allocation and performance in order to satisfy requirements of different users
- Self-protection—Capability of detecting security breaches and recovering from their effects

With these properties, a computing system is able to free system administrators from time-consuming, error-prone system operation and maintenance. Of course, the property that the autonomic system would like to focus on depends on the administrator's goals. For example, GoCity is aimed to employ diverse context types to provide an effective and efficient user experience, as well as to anticipate potential problems and accordingly take proper actions to prevent a failure when working in a complex and changing environment. Therefore, self-optimization and self-healing are emphasized in its design.

In the effort to define a common approach to building an autonomic system, IBM suggested a widely applicable autonomic computing architectural framework called Autonomic Computing Reference Architecture (ACRA) [39]. As depicted in Figure 9, ACRA is a three-layer hierarchy of orchestrating managers, resource managers, and managed resources [39, 40]. All management data can be shared through an enterprise

service interface. ACRA also provides a way to manually control each level through consoles or dashboards.

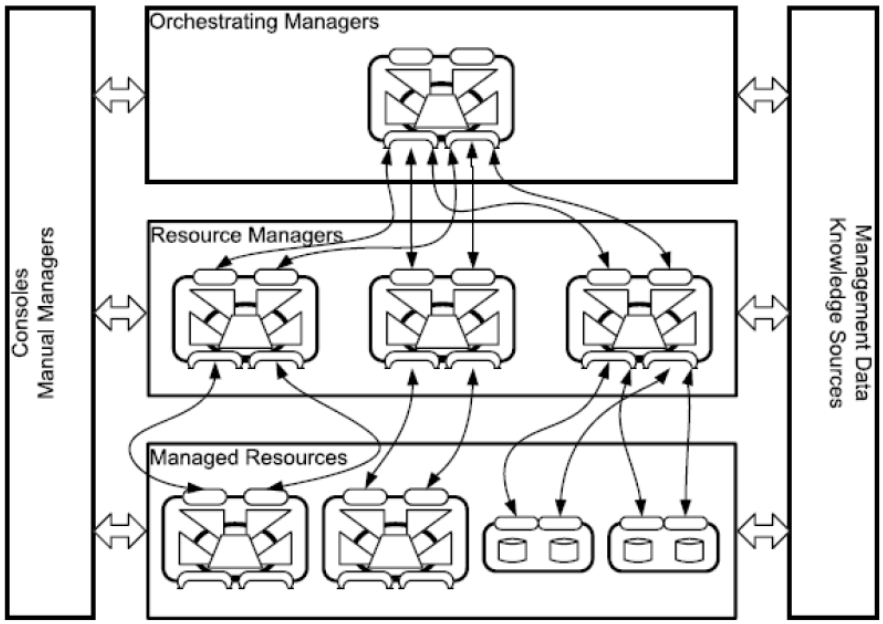


Figure 9. Autonomic Computing Reference Architecture (ACRA) [39]

Feedback control is the heart of a self-adaptive, autonomic or self-managing system, which requires one or more closed control loops for different management purposes [39]. To design an AC system, IBM introduces the notion of the Autonomic Element (AE), the fundamental building block of ACRA [39]. As depicted in Figure 10, an AE is comprised of an Autonomic Manager (AM), a managed element, and two manageability interfaces tied together via a closed control loop called the Monitor-Analyse-Plan-Execute-Knowledge (MAPE-K) loop [39]. Similar to the adaptation processes of a self-adaptive system depicted in Figure 8, the MAPE-K loop also works in four phases over a knowledge base to achieve goals. Table 4 lists the performance of each phase. The knowledge base is maintained by its Autonomic Manager (AM) and exchanged between the four phases.

Table 4. Four phases of MAPE-K loop

Phase	Performance
Monitor	Senses managed elements and their contexts, filters accumulated data, and stores data in the knowledge based for future reference.
Analyse	Compares event data against patterns in the knowledge base to diagnose symptoms and also stores the symptoms for future reference in the knowledge base.
Plan	Interprets the symptoms and devises a plan to execute.
Execute	Executes the change in the managed element through the effectors.

The mobile computing environment is a highly changing environment, in which context information changes might occur at any moment. Such changes definitely affect the mobile application that takes advantage of context information, requiring the application to react to them. As a result, self-adaptive techniques are applied in GoCity.

2.5 Summary

This chapter introduces the background knowledge including smartphones, mobile applications, usability, the Android platform, context-aware computing, and self-adaptive and autonomic computing. All of them form the foundation for this thesis.

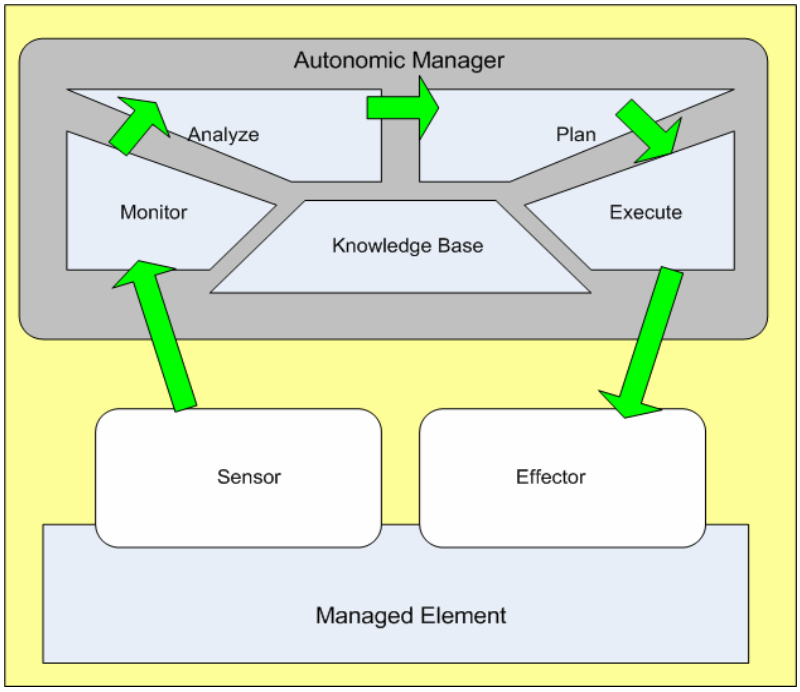


Figure 10. Autonomic Manager with MAPE-K loop [39]

Chapter 3 Context Classification for Mobile Applications

This chapter introduces a context classification for mobile applications. It begins with a brief introduction on related work, followed by the context classification defined in this thesis, based on its source and property.

3.1 Related Work

Dey and Abowd argue that application designers will leverage the categories of context to identify and expose most applicable contextual parameters to the application [41]. Villegas also emphasizes the importance of context classification in order to control and govern context information in a dynamic environment [5]. Therefore, in an effort to define context, several researchers provided the classifications of context.

Schilit et al. proposed the following three important aspects of context: *where you are*, *who you are with*, and *what resources are nearby* [42]. However, the categories only include the location and entity information. In practice, context applications often utilize four kinds of information—*who's*, *where's*, *when's* and *what's* (that is, what is the user doing) of entities [41]—to reason about the current situation and determine the next activity. As a result, the later context classifications are mostly based on the four types. For example, Ryan et al. listed the context as *Location*, *Time*, *Entity*, and *Environment* [33]. However, Dey and Abowd argued that Environment is often used as the synonym for context and thus is too big as one type of context. Thus, he replaced Environment with Activity, which refers to what is happening in the current situation, to generate his own context category [41]. Dey and Abowd also noticed that *Location*, *Time*, *Entity*, and *Activity* not only can answer the most important four questions that context applications

are concerned with, but also can be indicators used to retrieve other context information. For example, given a user's identity, we can retrieve many pieces of related information about the user, such as their email address, phone number, address, preferred food, and more. Dey and Abowd defined *Location*, *Time*, *Entity* and *Activity* as the primary context types [41]. In the previous example, the user's identity is the primary context and the user's email address, address, phone number, and preferred food are the secondary pieces of context.

In 2007, Zimmermann et al. extended Dey and Abowd's classification of context by establishing the dependency between entities [32]. According to Zimmermann, each entity has relationships to other entities. These relations describe a semantic dependency between two entities that emerges from a certain circumstance in which the two entities are involved. The set of relations that an entity has established to other entities builds a structure that is part of the entity's context [32]. Since the number of types of relations in the real world is large, Zimmermann subdivided the relations into *social*, *functional*, and *compositional relations*.

Three years later, Villegas pursued the same method as Zimmermann's to classifying context in order to manage dynamic context. According to her classification, context can be organized along five main categories: *Individual*, *Time*, *Location*, *Activity* and *Relational* as depicted in Figure 11 [5].

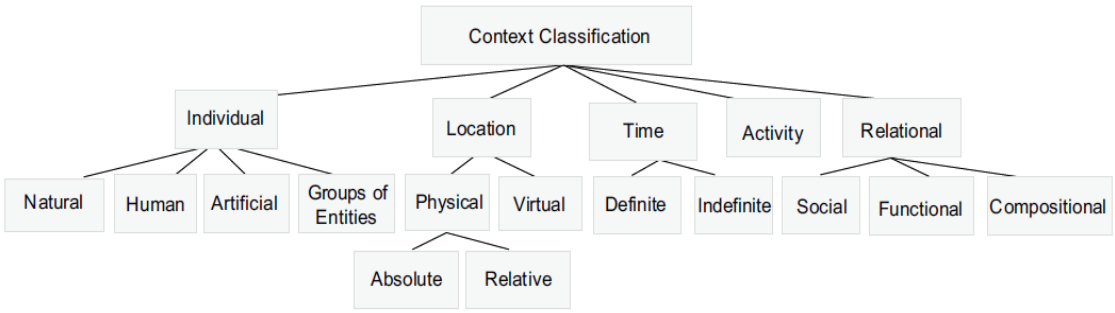


Figure 11. Classification of context by Villegas [5]

According to Villegas, anything that can be observed as an isolated entity is an *individual context*. An entity can either be an independent entity or a group of entities which do not necessarily interact with each other. Individual context is sub-classified into *natural*, *human*, *artificial*, or *group of entities* based on the entity types.

- Natural context—Includes properties of living and non-living entities which are not the result of any human activity. (e.g., weather conditions)
- Human context—Comprises the information related to user's behavior and preferences. (e.g., user's language)
- Artificial context—Refers to any information resulting from human actions or technical processes. (e.g., internet availability)
- Group of entities—A collection of entities which share certain characteristics or can generate certain properties only when grouped together.

The information about the place of an object is classified as *location*, *physical* or *virtual* [5]. A *physical location* represents a geographical location of an object. It can be described as an *absolute location*, meaning the exact location of an object (e.g., the restaurant's address or absolute coordinates), or as a *relative location*, meaning the position of an object with respect to another (e.g., the directions to reach the restaurant

from the mall). In contrast, a *virtual location* is not described using the geographical address. An example of the virtual location is the IP address within a computer network.

As the third type of context, *time* is vital to understand some situations and to obtain the secondary pieces of context [5]. Most statements that describe the situations are related over the temporal dimension. Any time-related activity might be used to suggest to the user the future activity. Besides the straightforward representation of time such as Central European Time (CET), categorical time information such as holidays, working hours, and weekends are often leveraged in context-aware computing [43]. Sometimes, we need to take the duration of an event or activity into account. In terms of duration, time context is sub-classified into *definite* and *indefinite*. Clearly, the *definite time context* represents a time frame with specific start and end points, while the *indefinite time context* implies a recurrent event where duration is impossible to know in advance. Basically, although this recurrent event does not have clear start and end points, it will be triggered or interrupted by other situations. Fortunately, we can actively control the occurrence of some situations to impact this event. In this process, the interval is a very important feature to model and manage contexts which constitute certain situations [5].

Activity is a very important aspect of situations, which helps us understand context and context-awareness better [44]. It expresses information regarding goals, tasks and actions that the entity is currently or will be in the future involved in. It reflects the requirements that the context-aware system should achieve.

The *relational context* describes the semantic dependency between two entities [5]. It is classified into three subcategories: *social*, *functional*, and *compositional*. Basically, *social context* describes the interrelations among individual human and group entities.

Each entity plays a specific role in the relationship. Examples are friends, co-workers, and customers. *Functional context* indicates that one entity makes use of another entity for some purpose. One example is illustrated when a user types on the keyboard to input text. Finally, *compositional context* refers to the relations between a whole and its parts, including aggregation and association subcategories.

3.2 Classification of Context Information for Mobile Applications

Villegas's definition of context suggests that context should be classified according to the corresponding domain [5]. However, currently designers still lack proper context classification for mobile applications. The classification proposed by Villegas categorizes context in a comprehensive way. In order to ensure that this classification is applicable to be extended to all types of context-aware applications or systems, it tries to take all possible contexts applicable in a context-aware application or system into account. Since this general context taxonomy proposed by Villegas can help build more concrete taxonomies for various application domains, the source-based classification proposed in this thesis is its extension for mobile applications.

3.2.1 Context Classification Based on the Source

As introduced in the previous chapter, context is any information that can be used to characterize the situation of an entity. It can be derived from different sources that an application is able to access. Obviously, mobile devices are often small, equipped with multiple sensors, and frequently interact with users. These properties determine that the way to gaining context information on a mobile device is distinct from other devices such as a desktop computer. To most context-aware mobile application designers or

developers, the main concern is which context can be monitored and how. To answer this question, the possible sources that provide context information to mobile devices are investigated in this section and as one dimension to classify context.

With the advancement in hardware, current mobile devices are equipped with a number of advanced sensors, such as location sensors (e.g., GPS sensor), visual sensors (e.g., camera), auditory sensors (e.g., microphone), tilt sensors, etc. These sensors allow us to gain diverse contexts representing the real world situations, and thus should be fully taken advantage of to enhance context-awareness [45]. However, due to the limited size of mobile devices, it is infeasible to equip too many sensors on a device. For example, most mobile devices do not have sensors to obtain weather-related data. What if we need to suggest the user to take an umbrella when it is a rainy day? The web, another data source that exists in everywhere of the computing network, can help obtain such context information.

Notably, the web has been the most significant source of information for over a decade. After its worldwide adoption, countless applications named Web 2.0 applications have been developed to offer their own data and web services [46]. These web services make user-generated contents available to third-party applications (e.g., mobile applications), which in turn are capable of accessing and reusing the data provided by web services to generate their own content for certain purposes. Web service data to some extent represents certain situations of the real world and thus plays a significant role in context-awareness. In other words, web services augment the mobile devices with context-awareness by providing a variety of data when responding to the request from mobile devices. Clearly, as long as the mobile device has access to the internet, a large

amount of data which can constitute certain situations of a subject has been exposed to it, such as the local weather.

A human being who directly interacts with the mobile device is another source from which context information emerges. In order to build user-centric or user-friendly software, current software designers have paid attention to some contexts related to users. For example, a location-based restaurant advisor takes advantage of a user's location and preferences (e.g., food) to suggest to the user restaurants; a game advisor employs a user's social relations (e.g., friends, classmates) and a user's historical gaming records to recommend new games that the user's friends are currently playing, or the user is possibly interested in based on previous preferred game types.

Moreover, the state of some aspects of the mobile device is context information as well. Examples are battery power, GPS sensor availability, and camera availability. Thus, the mobile device itself is a source of context.

So far, several sources from where context information emerges have been listed. According to the discussion above, I classify context into three main types: *personal context*, *environmental context* and *mobile device context*. The second level as depicted in Figure 12 denotes the data source that is possible to offer the context information.

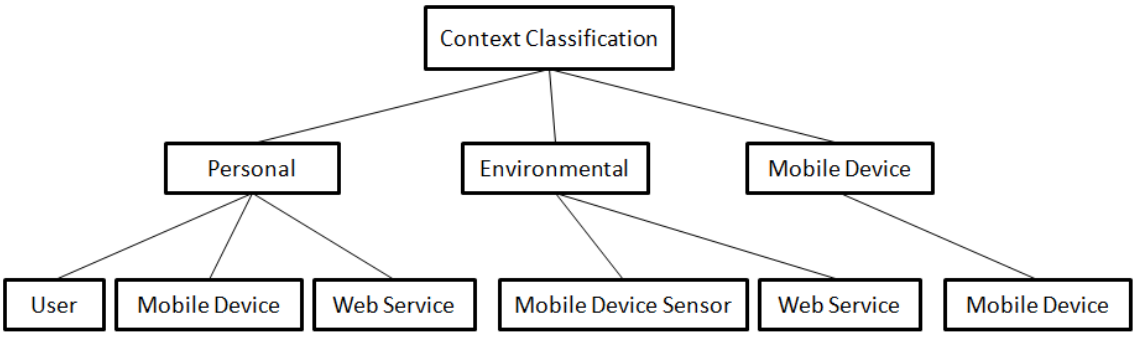


Figure 12. Context classification based on the source

The *personal context* refers to any information related to or derived from the user. It is an extension of the *human context* proposed by Villegas. For instance, a user's current location, interests, social relations, activities, and calendar information. The personal context information can be acquired from multiple sources, including the user who operates the application; the mobile device which stores user-related information such as a user's schedule; and the web service from which a user's social relations can be collected.

The *environmental context* covers any information that describes the outside environmental situations. It is an extension of the *natural context* proposed by Villegas. It does not express any information regarding the user and the mobile device. This context information can be collected through the sensors installed on the mobile device and through web services. Examples are temperature, weather condition, date, time, web service availability, and more.

Finally, the *mobile device context* indicates any information representing the current mobile device's state. It is an extension of the *artificial context* proposed by Villegas. Examples are GPS connectivity, network connectivity, battery status, device orientation, camera availability, and more. Clearly, this context information is directly acquired from the mobile device.

According to this classification, mobile application developers can easily identify and collect the context needed in the development. Moreover, from my point of view, knowing the source of context, developers or designers can provide a better way to govern the context information. For example, a web service might not often update its offered data, and thus we can store the data locally for a certain time after retrieving it

from the web service. During this time, we do not communicate with the web service but retrieve the context information locally. In this way, we not only save the resource (e.g., battery power and network bandwidth), but also enhance efficiency.

3.2.2 Context Classification Based on the Property

It has been accepted that context-aware mobile applications need to be adaptive [23]. Adapting to context changes is a very important aspect that a context-aware adaptive mobile application must accomplish. As a result, monitoring context changes at runtime is necessary. In order to save resources (e.g., prolonging the life time of the battery) and improve efficiency (e.g., relieving the system burdens of retrieving contexts), it is not recommended to monitor all the context changes. In fact, it is not necessary to monitor all the context changes since system requirements do not need to sense some of them. As a result, in order to provide a strategy to monitor context changes, I classify context into two categories: *static context* and *dynamic context*. The *static context* refers to the context whose change is not monitored by the system according to system requirements. In contrast, the *dynamic context* refers to the context whose change is monitored by the system.

Usually, context-aware systems or applications use two methods to sense changes of dynamic contexts: *push* and *poll* [2]. This thesis applies two push methods: (i) Applications subscribe to a middleware or external service that is responsible for delivering context changes to them. Currently, many mobile platforms (e.g., Android) provide such a method, allowing applications to bind to some system-level services that deliver updated context information to the application. (ii) Some UI components of the application provide context information to the application even if the application does not

actively ask for it. For example, when a user interacts with an item (e.g., button) in the user interface, this item triggers an event listener of the application and thus the application can immediately obtain some context information (e.g., user interest) contained in this interaction. Clearly, these dynamic contexts delivered by middleware, external services or application UI components are not actively collected by the application. Compared to push, poll requires the application to use a rate called the polling rate to continuously collect context information. According to G. Cheng and D. Kotz, the polling rate should be handled differently since the different context types have their distinct properties [2]. For example, the location of a moving person may change every second while the weather condition does not change frequently. As a result, according to the properties of dynamic contexts discussed above, I further classify dynamic context for mobile applications into three types: the one obtained by push, the one of high initial polling rate, and the one of low initial polling rate.

It is noteworthy that the dynamic context obtained by push can be handled as the one having initial polling rate. To be specific, if the application is not bound to the context change related component, it can use a polling rate to actively collect this context information from its source. The way to sense context changes is determined by the designer's concern, and thus the category of a dynamic context can be different in an application. More advanced, the application can self-adjust the polling rate of the later two dynamic context types according to the context information's previous changing rates by applying self-adaptive techniques.

3.3 Summary

This chapter described related work on classifying context information to provide a basic understanding of how context can be categorized. In order to simplify the classification of the context information and provide an easy and practical way for mobile software developers to manage context, this thesis classifies context information based on its source and property.

Chapter 4 Application Model

This chapter describes the generic model proposed for building context-aware adaptive mobile applications. The generic model provides two primary functions: one collects and manages diverse contexts; the other supports self-adaptation. A working prototype GoCity was implemented on the Android platform using this application model. An overview of the key features of this application model is described followed by a detailed description of this model.

4.1 Key Features

Currently, there are a few shortcomings involved in the design of context-aware adaptive mobile applications. First, few contexts other than location and time have been used in current mobile applications. Second, there is no dedicated context classification for mobile applications. Third, mobile application designers lack a systematic mechanism to address sensing and monitoring requirements under changing context situations. Fourth, many mobile applications have low usability due to their inefficient UI design.

Chapter 3 provided two ways to classify context information for mobile applications, thereby solving the second issue to a large extent. The application model proposed in this chapter presents a solution to the first and third problems by providing mechanisms to collect and govern various contexts and to achieve the adaptation to context changes. The possible solutions to the fourth concern will be discussed in Chapter 5.

One objective of this thesis is to provide a model for building context-aware adaptive mobile applications. This model is capable of leveraging a variety of context information and adapting to context changes, and thus has the key features described below:

1) Context is the first class citizen in this proposed application model. Context represents situations that the user was, is, or will be involved in. It plays a crucial role in a context-aware application and drives the generation of appropriate content to users.
2) Contexts are collected and governed based on the classification proposed in Chapter 3. The diversity of contexts and their sources magnifies the difficulty in managing them. A key component of the model called *context manager* is utilized to collect various context information from available sources. With the help of this component, applications with disparate sources can be utilized in a uniform manner, thereby obtaining diversified context information.
3) Self-adaptation to context change is supported in this model. The notion of an autonomic element is used to make the model self-adaptive. Working as the autonomic manager of an autonomic element, *context adaptation manager*—another key component of this model—monitors context changes, and accordingly drives the application to react to context changes without user intervention.
4) A *filter* component extracts and formats context information required by the consumer application and refines the generated contents shown to the user.
5) Provision for proactive interactions with a user is supported. We are pursuing a smart application that generates satisfactory results with minimal user operations. The proactive interaction with a user is one of the features that make an application smart. It breaks the traditional method where an application only responds to a user's request. As a matter of fact, it allows an application to initiate

interactions as well as suggest results based on collected contexts without any request from a user.

6) A *web service manager* component deals with all the transactions between the application and outside web services. Web services are not only the sources providing context information but also the data sources of the generated contents. Because many web services are provided without guaranteed safety and accuracy, it is essential to have a component that manages the risk occurring due to web service failures. Thus, a web service manager is leveraged in this model to minimize such risks.
7) A knowledge base is employed in this model. It contains a set of adaptation rules and context history. It is used to assist the context adaptation manager in determining the polling rates for dynamic contexts and the results generated to a user.
8) It is easy to extend the functionality of an application built using this model. Each component of this proposed model is self-contained and provides interfaces to its fellow components in a standard manner. Because of this feature, each component is extensible and thus an application leveraging these components will find it easy to extend its functionality.

Several features discussed above exhibited the key components that constitute the proposed application model. The next section will discuss these key components in detail.

4.2 Model for a Context-aware Adaptive Mobile Application

Usually, software designers create the architectural model at design time to capture significant decisions that meet the requirements. In most practices, they do not represent

the architectural information explicitly at runtime. However, this could be a problem when a user's needs are unexpected and the software requirements need to vary at runtime. Often, a mobile application that closely interacts with users has this design issue. As a solution to this issue, a model must be created that not only represents the dynamic information explicitly but also supports the runtime adaptation for improving the application's usability [47]. We have observed that context plays a significant role in mobile applications. Many design decisions at both design time and runtime are affected by context due to its dynamic nature. As a result, components that manage contexts and feedback loops that provide adaptations to context changes are represented explicitly in our proposed model.

Building a context-aware or adaptive system is not a new topic; a number of projects [31, 48, 49, 50, 51, 52] have demonstrated some techniques and ideas around building context-aware or adaptive software in different fields. Inspired by these projects, this model was created by applying the concept of component-based development (CBD). As a result, this proposed model consists of several components, each of which accomplishes their specific functions for achieving an intended task.

Thanks to the work conducted by Carroll who first introduced the notion of scenario-based design (SBD) [53], before we discusses the proposed model, we first take a look at the main scenario that our proposed model is supposed to work in. As depicted in Figure 13, first, a mobile application built using this model is launched in a user's device. Then, the application activates a set of services which provide contexts. Next, the context information is collected by those services from different sources and stored locally in order to get their historical records. Once the context information is obtained,

the gathered context is analyzed to determine the user's current situation and their new polling rates. Finally, the application sends its requests to web services to generate results for the user, and those collected contexts have new polling rates stored locally. Notice that before the application is shut down, it is able to use new polling rates to update related context information. After a context is updated, the user's situation and a new polling rate of this context are re-determined, and accordingly new contents are generated for the user and the new polling rate is stored.

In this scenario, we can observe two things: first, the application can dynamically generate contents for users without any user's intervention; second, a number of feedback loops are extensively employed in this proposed model to ensure adaptation.

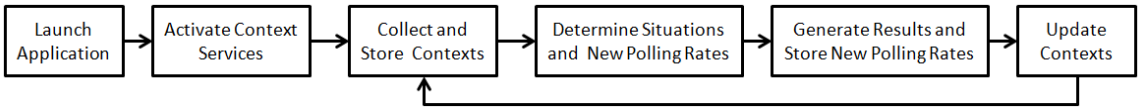


Figure 13. The working flow of our proposed model

4.2.1 Context-Aware Adaptive Mobile Application Model

Discussion in the previous chapters has thrown light on the importance of context. Collecting and managing contexts is the major task that should be accomplished in this model.

The proposed model is depicted in the red rectangle of Figure 14. This model supports two interaction modes: static mode and dynamic mode. In static mode, the application does not dynamically give results to a user. To be specific, after the user sends a request to the application, the context manager gathers required context information and passes them to the filter. The filter leverages the gathered information and sends requests to service mangers according to the application requirement. After the

service manger gets responses from outside web services and delivers results to the filter, the filter refines the results and generates appropriate content for the user. During this process, the context manager does not continuously collect contexts and pass them to the filter to generate new results. As a matter of fact, the application generates contents only after a user sends the request to it. By contrast, in the dynamic mode, the context manager continuously collects context information under the control of the context adaptation manager. The filter is also managed by the context adaptation manger to dynamically generate contents for the user when gathered context information changes. Obviously, the dynamic mode needs to integrate the mechanism of providing self-adaptation.

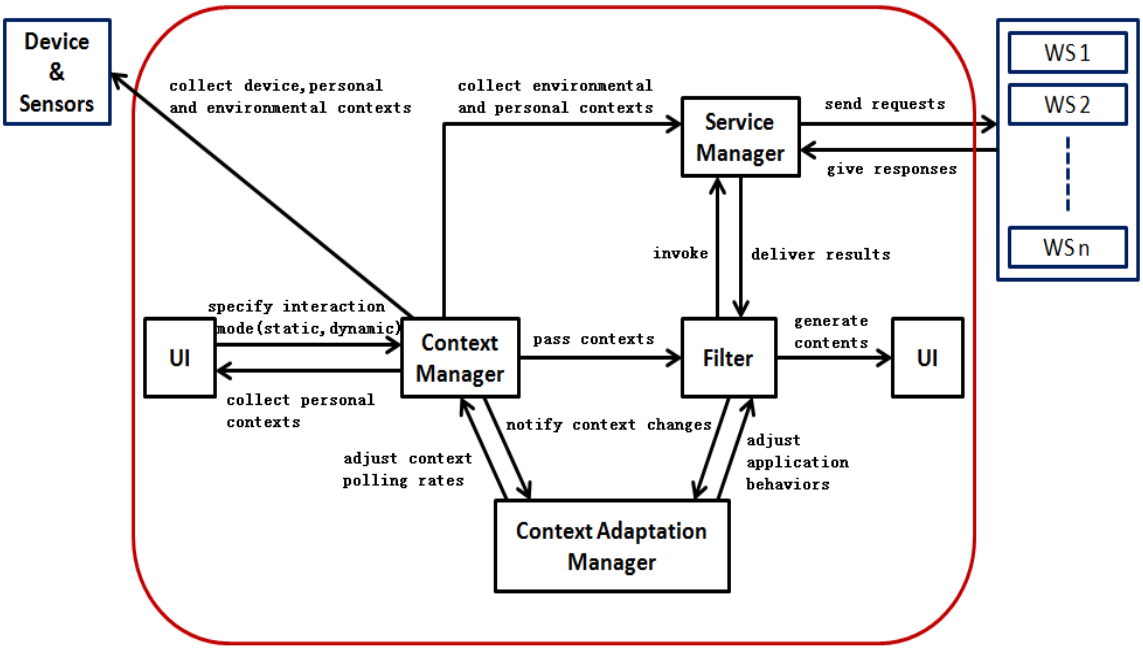


Figure 14. Model of a context-aware adaptive mobile application

A key feature of the proposed model is that context is controlled and governed uniformly. To accomplish this, a key component called *context manager* is employed in this model. The context manager collects various contexts from different sources. As

depicted in Figure 14, the context manager collects personal and mobile device contexts from the user and the mobile device. Since the mobile device does not have many sensors to obtain environmental contexts, some environmental contexts are collected from outside web services. For example, we can get the local weather information by consulting World Weather Online [63] which is a web service offering weather-related data.

The *service manager* component employed in this model deals with all the interactions between the application and external web services. There are several benefits to applying uniform management on interactions with external web services:

- Minimizing the risk of faults caused by web service failures.
- Providing a flexible way to adding more data sources.
- Simplifying the implementation of other components.

The context information collected by the context manager needs to be refined for various applications. Also, results initially generated may need to be further selected so that the content shown to the user is relevant. Therefore, the *filter* component is utilized to accomplish the two purposes stated as follows.

First, it extracts and formats context information provided by the context manager for a given application. For example, after collecting the weather related information, the context manager generates an object that contains the weather condition, temperature, wind speed, humidity, etc. However, only the temperature is needed by the application, and it is required to be an integer. As a result, the filter component only extracts the temperature from the object and formats it for the application. Second, it refines the results shown to the user. For example, a user only wants to find five restaurants nearest

to him/her. However, the application initially generates more than five results. In this case, the filter component helps get rid of unnecessary results to ensure that the user is satisfied with the final generated content.

In the dynamic mode, an adaptive model as depicted in Figure 15 is employed for accomplishing adaptations to context changes. This model is designed based on the AE structure proposed by IBM [39]. Clearly, it is a closed control loop and composed of two major components: the *autonomic manager* which is the context adaptation manager in our proposed model, and the *managed element* which is the composition of the context manager and the filter in my proposed model.

The *context adaptation manager* provides the adaptation functionality to the context manager and the filter. For the context manager, it adjusts the polling rates of dynamic contexts according to the change in context history. For the filter, it analyzes the changed context information passed from the context manager, and modifies application behaviours accordingly. Thus, the context adaptation manager has the following features:

- Determining when and which context is sensed.
- Determining the situation based on the given context information.
- Determining the content generated to the users.

The context adaptation manager has a knowledge base which contains a set of adaptation rules and context records. In the implementation of GoCity, the adaptation rules are simple if-then rules, and the context records are the data about the collection of context information such as context value, current polling rate, adjusted polling rate, etc. The context records can be stored in a local database or file and used by the context adaptation manager to make appropriate decisions.

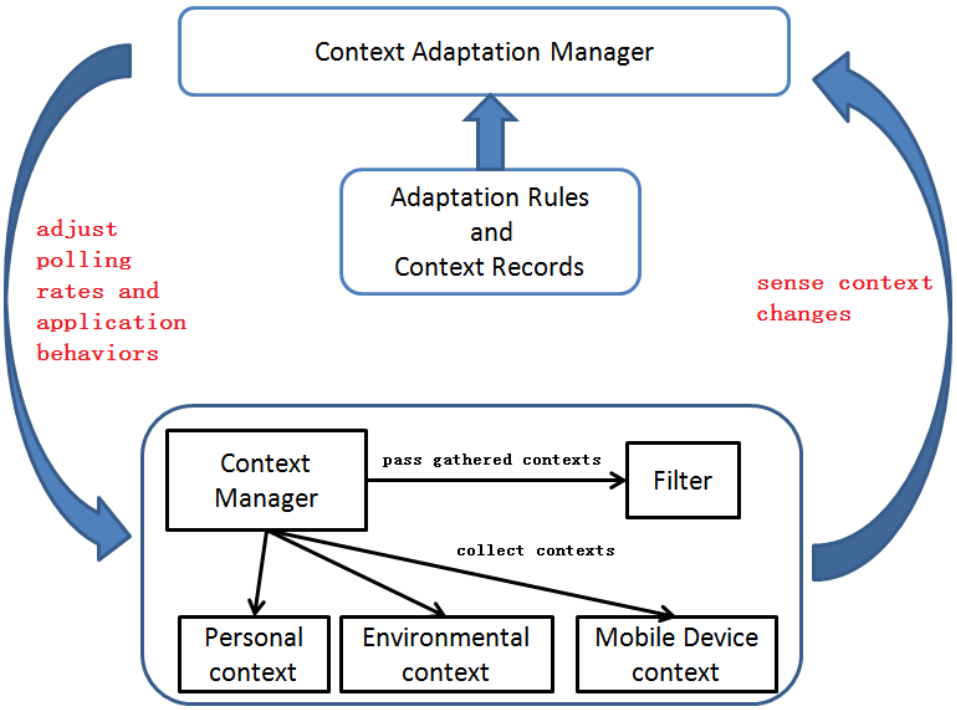


Figure 15. Adaptive model of context-aware adaptive mobile applications

4.2.2 Implemented Model

Currently, there are a number of mobile platforms available to implement this model such as Android, Symbian, iOS, and Windows Phone. However, it is impossible to implement this model on all of them due to time constraints. We chose to implement this model on the Android platform for reasons stated as follows:

First, in accordance with the Android based smartphones, Android mobile applications have covered a huge market and are growing rapidly. With this Android based model, we provide a platform for mobile application developers to implement more and better context-aware adaptive mobile applications.

Second, the Android operating system offers many context-related services and managers that are interfaces or classes having access to global information about the

application environment. These services and managers can be easily leveraged in this model to collect various contexts. For example, the ACCOUNT_SERVICE provides access to online user account registry information and thus can be used to obtain personal contexts. The WIFI_SERVICE allows access to all Wi-Fi related information and can be used to extract environmental contexts such as the state of a network. The Battery Manager helps to obtain battery related information and thus can be applied to gain device context such as the battery power state.

Last but not least, as discussed in Chapter 2, the Android application framework simplifies the reuse and replacement of application components, which is crucial for the application of CBD to build this model. Most parts of the proposed model are not application-specific and support reusability. This key feature of Android undoubtedly satisfies our need to provide reusable components. In addition, the ease of replacing the application components not only permits the application developers to customize their applications flexibly, but also allows scalability in the implemented model.

In the implemented model, the filter is an application-specific component whose implementation is based on the application functions. The filter is composed of a set of Android Activities (UIs) employing the extracted and formatted context information required by the given application to generate content for users. The context manager collects diverse contexts from available sources. Because it is intended to be employed by various applications, it is a reusable component and not application-specific. In its implementation, it takes advantage of many Broadcast Receivers and Services of Android to accomplish the background functionalities of gathering and delivering context information. The context adaptation manager contains two parts: one monitors context

changes and accordingly adjusts the context polling rates used by the context manager; the other monitors the change of the context parameters utilized by a given application and accordingly determines the actions of the application. Clearly, the former part is not application-specific and thus is reused by various applications, while the latter one is application-specific and works according to the adaptation rules of the given application. Similarly, the service manager includes two parts as well: one is employed by the context manager to collect context information from the web and thus is not application-specific; the other is used by the filter to obtain data to generate results for a given application and thus is application-specific.

Today, a number of open source mobile development frameworks are available to enable programmers to build applications for mobile devices, like Open Mobile IS [65] and PhoneGap [66]. Compared to these open source platforms, the implemented model on Android platform has several distinct advantages. First, it provides a component dedicated to gathering diverse contexts from a variety of sources. This makes programmers to easily leverage context information to build context-aware applications. Second, it provides a component and uses feedback loops for supporting the dynamic evolution of context information. This is crucial for self-adaptation. Third, it takes advantage of the full power of Android platform, simplifying the reuse and replacement of the components. As a result, components of this model are easy to be reused for developing various applications and replaced for extending the application's functions.

4.3 Summary

Context plays a crucial role in the context-aware adaptive mobile application model. The proposed model employs a context manager component to uniformly collect and

manage contexts. The service manager component manages all the interactions with outside data sources. The filter component helps extract and format context information and refine the results displayed to users. In order to sense context changes and generate appropriate results for users, the context adaptation manager component is implemented based on the structure of the autonomic manager.

Chapter 5 GoCity and Its User Interfaces

This chapter introduces the application implemented based on the model proposed in Chapter 4 and is known as GoCity. GoCity was developed on the Android platform and can be run on Android version 2.2 and below. Readers are strongly suggested to refer to the background chapter to gain familiarity with the Android platform and its development tools. This chapter provides a quick overview of the Android platform and the Android development tools followed by the main features of GoCity.

5.1 Overview of the Android Platform and Development Tools

Android is a software stack for mobile devices. It includes an operating system, middleware, and key applications. The Android operating system includes four major components: application, application framework, libraries and Android runtime, and a Linux kernel. The key feature of the Android application framework is to enable and simplify the reuse and replacement of application components. Additionally, developers have full access to the same framework APIs that are used by the core applications of Android. The application components—Activity, Service, Content Provider, and Broadcast Receiver—are the essential building blocks of an Android application. Each component is a different point through which the system can access your application. Before using these components, you must extend a super class provided by the Android library. For example, you need to implement an activity component as a subclass of Activity if you want to create one. Activities, services, and broadcast receivers are activated by an asynchronous message termed Intent. Intents bind individual components to each other at runtime, whether the component belongs to your application or another.

All Android applications run on the Dalvik virtual machine (VM). All the components of the application must be declared in a manifest file so that the system knows of their existence.

Three development tools are used when implementing GoCity on Android, which are Android Emulator, Android Development Tools Plugin, and Dalvik Debug Monitor Service (DDMS).

5.2 Features of GoCity

The main purpose of this thesis is to address problems in the design of context-aware adaptive mobile applications. The problems targeted in this thesis are: 1) Few contexts other than location and time have been used in actual mobile applications; 2) There is no clear context classification in mobile application design; 3) Mobile application designers lack a systematic mechanism to address sensing and monitoring requirements under changing context situations; 4) Most mobile applications sacrifice usability due to defective UI design. Thus, the key considerations in the design and implementation of GoCity are context gathering, sensing context changes, adaptation to context changes, and friendly user interfaces.

5.2.1 Main Functions and UIs of GoCity

GoCity is developed to provide city visitors with up-to-date and context-aware information while they explore a city using Android mobile phones. It is composed of four sub-applications as described below. Each sub-application can be implemented as an independent application to provide different functionalities to the user.

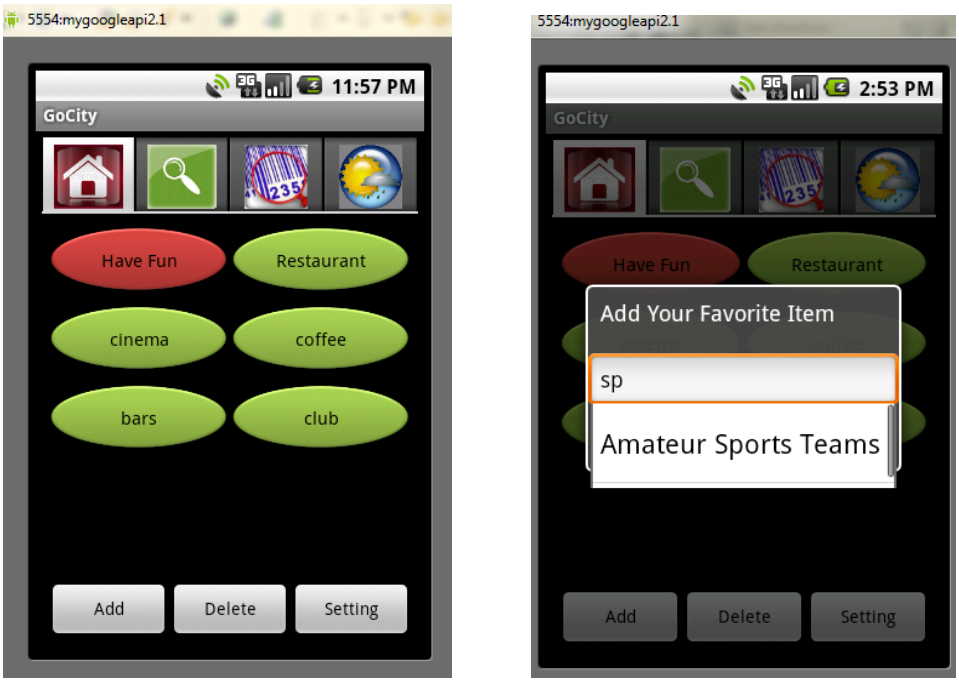
- **NearBy**: Recommendation of businesses near the user

- **Search**: Recommendation of businesses near a specific location
- **Barcode**: Provision of price-related information when the barcode of a product (via camera scan or manually input) is provided
- **Weather**: Provision of current weather information and a 5-day forecast

Nearby and *Search* have similar functionalities. They both leverage diversified context information to recommend nearby businesses based on the user's current situation. They both support static mode and dynamic mode. Given a product barcode by the user, *Barcode* gives price-related information. *Weather* gives current weather information and 5-day forecast based on the location. These four sub-applications respectively utilize their required context information and can be implemented as four independent mobile applications.

In GoCity, two factors that influence usability are taken into account. The first factor is to make the user interface simple and clear without reducing the desired functionalities. The second factor is to support user expectations for personalization. Therefore, UIs of GoCity will be introduced by focusing on these two factors.

The first user interface of *Nearby* is shown in Figure 16(a). Four tabs on the top of the screen respectively represent *Nearby*, *Search*, *Barcode*, and *Weather*. The user only needs to click on one of them to see the corresponding user interfaces.



(a) (b)

Figure 16. User interfaces of Nearby (I)

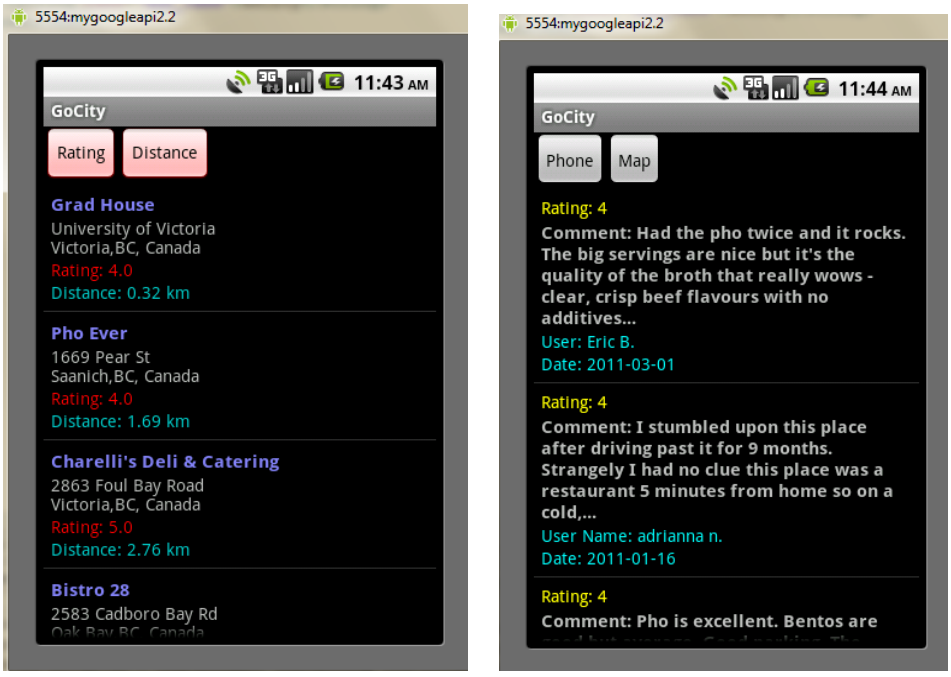
In Figure 16(a), the red button invokes the dynamic mode, and each green button invokes the static mode. Each green button represents a business type that the user is interested in and wants to find. *Nearby* allows the user to create a shortcut for a business type by adding a green button on the screen. It also allows the user to delete the shortcut of a business type by removing the green button associated to that business type. Because of this feature, the UI can be maintained as simple and clear as possible.

The 'Setting' button allows users to personalize the results shown on the screen. For example, if the user prefers the recommended businesses to be listed in order of distance, he/she can personalize this by using the 'Setting' button. *Nearby* provides three ways for the user to personalize the results: (i) Listing the recommended businesses in order of distance; (ii) Listing the recommended businesses in by order of "Star" ratings; (iii) Specifying the number of recommended businesses that are shown on the screen.

As shown in Figure 16(b), when adding a new business type on the screen, *Nearby* provides auto-suggestions based on string match. Auto-suggestions are supplied by the business category library (BCL). The BCL contains all the categories used to filter businesses. It is noteworthy that *Nearby* does not limit the user's input even if it has BCL. As a matter of fact, *Nearby* uses the name to filter businesses if the user does not choose any of the auto-suggestions.

In addition to the business category and name, the search region and the business rating are used to filter the results as well. The search region and the business rating both have default values. The user can reset them according to his/her current preferences. In *Nearby*, the default value of the search range is 5 kilometers, and the default value of the business rating is the one that the user reset to the most times in the past. The default value of the business rating is obtained by consulting the user's usage history which is stored locally. Therefore, *Nearby* recommends 5-star businesses whose distance is within 5 kilometers from the user's current location, if the user does not reset the search range and the business rating to other values (e.g., 4 kilometers, 4-star) and 5-star is the one that the user reset to the most times in the past.

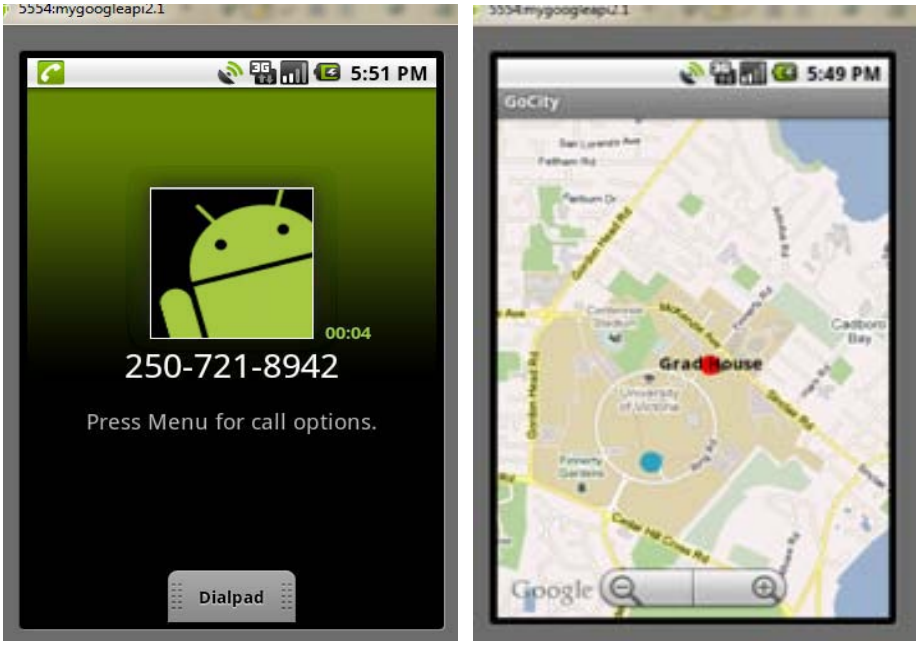
A list of businesses near the user are recommended as depicted in Figure 17(a) after a green button representing the user's current interest is pressed. The list of businesses is ordered by distance initially. Hence, when the user comes to this interface the first time, the business recommended at the top of the screen is the nearest one to the user. Users can personalize the result sets by setting the business rating or the distance as described before. In this case, further ordering is based on the user's interest on the rating or the distance. Figure 17(b) illustrates that the user reviews a business.



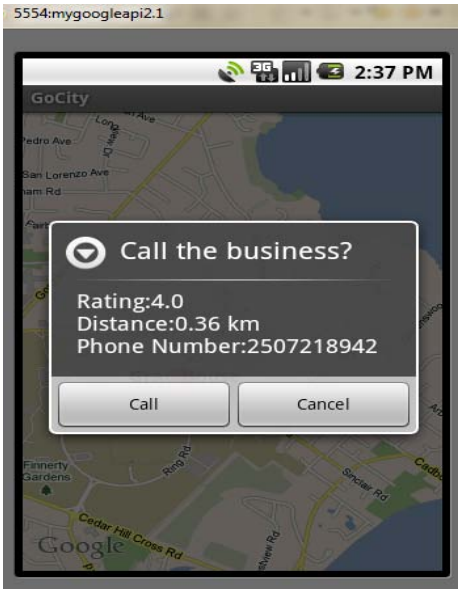
(a) (b)

Figure 17. User interfaces of Nearby (II)

The phone button enables the user to call to the selected business as depicted in Figure 18(a). The map button allows the user to see his/her current location and the business's location as shown in Figure 18(b). The user can also obtain directions to this business and review the business information as shown in Figure 18(c) by clicking the red dot.



(a) (b)



(c)

Figure 18. User interfaces of Nearby (III)

Search has similar functionalities to *Nearby* and also supports both the dynamic and static mode. In addition to the user interface, the difference between *Nearby* and *Search* is that *Search* provides a more flexible and personalized mechanism. The first user interface of *Search* is shown in Figure 19(a). If the user does not manually input any

address information into the second or third textbox, the application automatically generates information regarding the user's current location on the screen. This feature implies that recommendations will be generated based on the user's current location.

Different from *Nearby*, *Search* allows users to change the address information so that they can find businesses near any location. This feature of not limiting user access to information, especially location information, was highlighted in [24, 54]. *Search* also supports auto-suggestion as shown in Figure 19(b). To activate the dynamic mode, the category textbox is left empty.

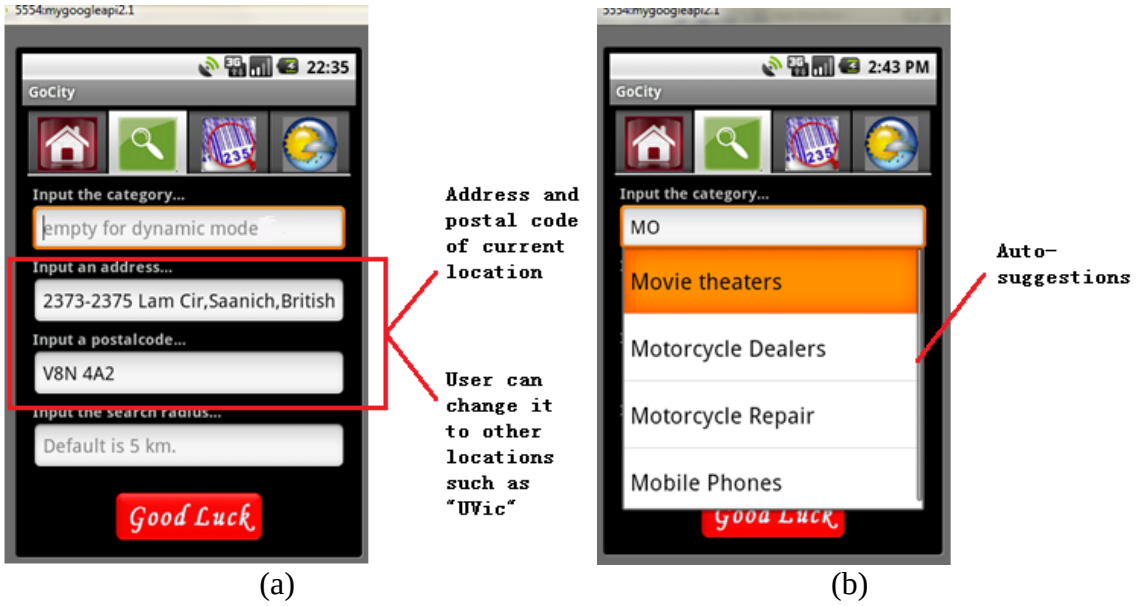
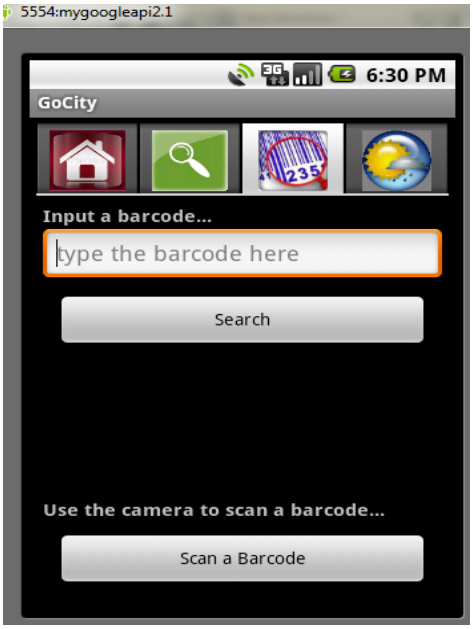


Figure 19. User interfaces of Search

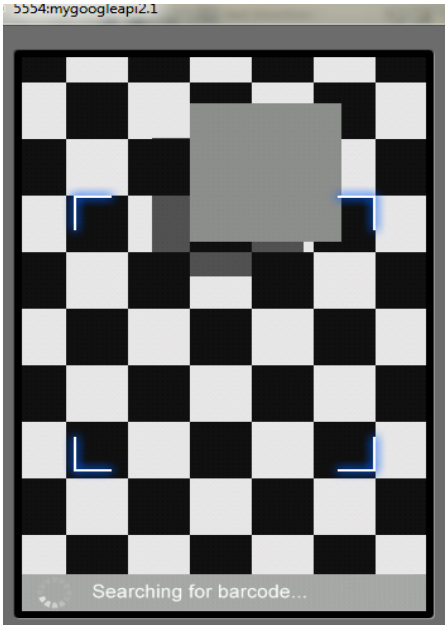
Since functionalities of *Nearby* and *Search* are similar to a popular Android application Yelp, we compared them to see whether *Nearby* and *Search* can provide a good user experience with high usability. The results of the comparisons are: (a) They both have clear user interfaces; (b) *Nearby*'s and *Search*'s user interfaces are simpler; (c) Yelp has better looking user interfaces; (d) They all allow users to find any type of

business near any location; (e) Yelp does not have the dynamic mode; (f) Yelp provides more functionalities in the static mode; (g) They all support personalizing the result sets.

Figure 20 is the user interfaces of *Barcode*. *Barcode* allows the user either to manually input a barcode number in the textbox as depicted in Figure 20(a) or to use a phone camera to scan a barcode image as shown in Figure 20(b). For scanning, the barcode image is transferred to the corresponding barcode number and then sent to a web site. The response data from the web site is displayed in a web page which contains a list of price-related information as shown in Figure 20(c). The user can re-sort the list either by the price or by the condition of the product.



(a)



(b)

(c)

Figure 20. User interfaces of Barcode

Figure 21 is the user interface of *Weather*. The section above provides current weather information and the section below displays a 5-day weather forecast.

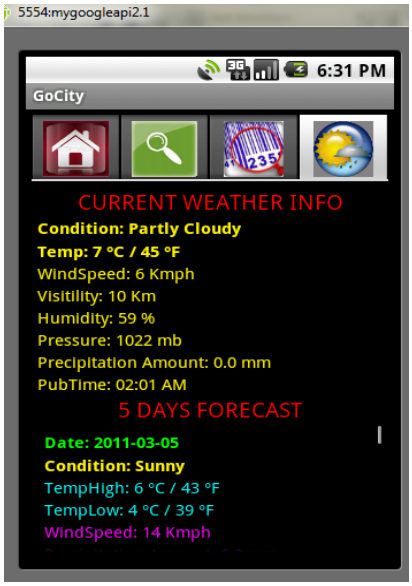


Figure 21. User interface of Weather

5.2.2 Context Collection

The first question that one should ask when designing a context-aware mobile application is what context will be used in the application based on the requirements. In order to apply the most diverse contexts possible, four sub-applications described above were implemented in GoCity to provide different functions. As a result, the following contexts were collected and managed in GoCity:

- Categorized based on source:

- Personal context:
  - user location, user interest, user activity history, event recorded in the user's calendar
- Environmental context:
  - weather condition
  - date, time of day
  - availability of web services
  - network connectivity
- Mobile Device context:
  - battery status, device orientation, imagine taken by camera

- Categorized based on property:

- Static context:
  - user activity history, date, event recorded in the user's calendar
- Dynamic context:
  - Obtained by push: user interest, imagine taken by camera, device orientation

- High initial polling rate: user location, time of day, battery status
- Low initial polling rate: weather condition, network connectivity, availability of web services

5.2.3 Adaptive Functionality

In the design and implementation of *Nearby* and *Search*, an adaptive solution is exploited in the dynamic mode. A feedback loop is employed to ensure that the result is updated once the context change is sensed and an adaptation rule asking for regenerating results is matched. For example, one adaptation rule states that if the battery power is under 20%, the application asks the user whether to stop updating the location on the map. The purpose of this rule is to save battery power, and most importantly, not constrain the user's expectation for seeing an updated map.

Additionally, the polling rates of dynamic contexts are adjusted by the context adaptation manager according to the stated adaptation rules. The adaptation rules will be discussed in detail in Chapter 7.

5.3 Summary

GoCity is composed of four sub-applications, each of which leverages several contexts and can be implemented as an independent context-aware mobile application. GoCity provides a simple and clear user interface and supports user expectations for personalization. However, development activity is underway for improving GoCity's functionality and user interface design. *Nearby* and *Search* both support dynamic mode, meaning that GoCity provides an adaptive solution for managing context changes.

Chapter 6 Content Generation

This chapter introduces the process of content generation. It begins with an overview of the data sources that are selected as information providers for GoCity. Next, it discusses the service manager and the filter components in the implementation of GoCity. It ends with a summary.

6.1 Data Sources

Data sources are very significant for building a context-aware mobile application. First, some context information can only be obtained from outside data sources due to limitation in mobile devices. For example, most current smartphones do not have sensors to obtain local weather information. As a result, applications built on these smartphones have to take advantage of web services to get weather information. In addition, most web-based mobile applications need to explore outside data sources to generate desirable results for users. For example, a tour assistant mobile application wants to display maps on the screen; as a result, it leverages Google's map service to get such data.

Two types of data sources are used in this thesis: web service APIs and web pages. A web service API is typically a defined set of HTTP request messages along with a definition of the structure of response messages that are usually expressed in JSON or XML. Finding a good and free web service API is challenging. First, some web services are vulnerable and thus not stable for developers to use. Second, many web services are not well documented, and thus they are often used incorrectly. After an extensive search,

seven web service APIs as listed below are leveraged in GoCity. They provide stable services and are documented well.

- Yelp API [57]
- Flickr API [58]
- Yahoo Local API [59]
- Yahoo Weather API [60]
- Google Maps API [61]
- Android Location API [62]
- World Weather Online API [63]

These APIs are used by *Nearby*, *Search*, and *Weather*. *Barcode* employs another data source—UPC database [64], a web site that offers price-related data—to get web pages after a barcode number is passed along with an HTTP request.

6.2 Service Manager

The service manager manages interactions between the application and outside web services. It provides interfaces to receive parameters that are passed from the context manager and the filter. Then, the service manager sends HTTP requests that contain parameters to related web services. After receiving responses, the service manager sends them back to the context manager and the filter. Finally, the context manager and the filter use several parsing techniques to extract their demanded information, since the responses are usually in JSON, XML, or other similar format.

The service manager component is composed of a set of classes, where each is dedicated to working with a certain web service as depicted in Figure 22. The class was drawn using UML.

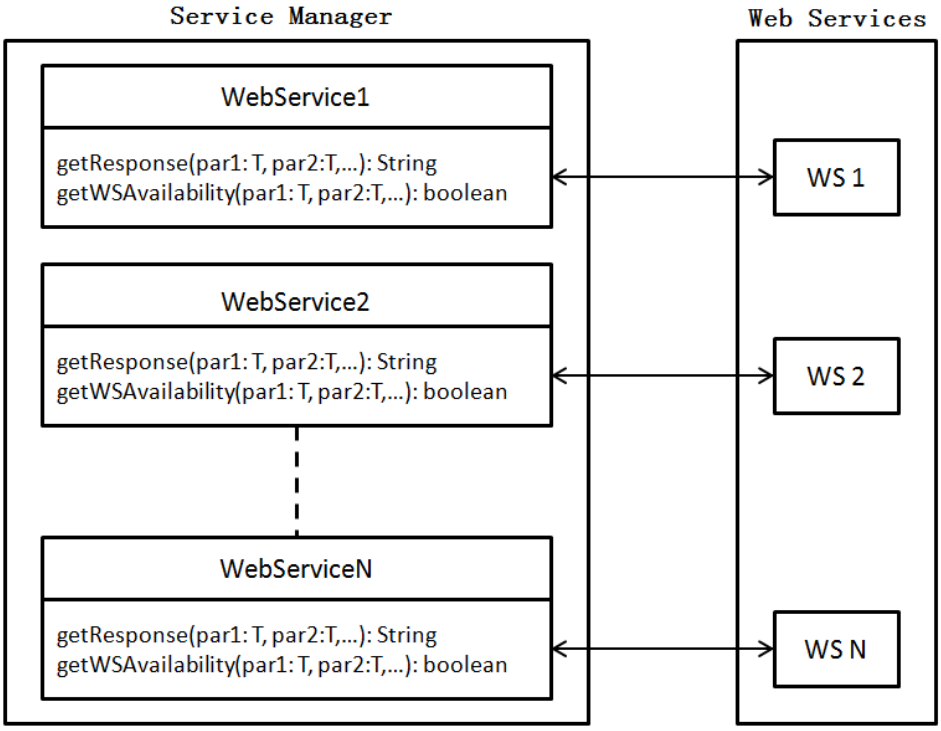


Figure 22. Interaction between the service manager and web services

The service manager offers a way of uniformly managing all the interactions with outside web services. Several advantages can be obtained by using this technique.

First, web service related contexts such as web service availability are easily collected and managed. This context information can be utilized to minimize the risk of faults caused by web service failures. For example, an HTTP status code indicates whether an HTTP request to a server has been successfully completed or not. This code is used by the service manger to generate the environmental context—web service availability. This context is finally obtained by the context manager and analyzed by the

context adaptation manager to determine whether a web service is functioning correctly. If this web service is not working correctly, related actions are given to avoid possible crashes that are caused by web service failure.

Second, it provides a flexible way to add more outside data sources for the application without interrupting other components' functions. The service manager provides a standard way to communicate with outside web services and send responses to its fellow components.

Third, it simplifies the implementation of other components and avoids possible repetitive work. Without the help of the service manager, the context manager and the filter have to communicate with outside web services by themselves. As a result, in each communication, the context manager or the filter needs to initiate an HTTP request and send it to the related web service. Accordingly, the context manager or the filter needs to receive the original response from the web service and further extract the data they want. In this situation, the complexity of the context manager and filter would be significantly higher, and the code that interacts with a specific web service could possibly become repetitive in the context manager and filter components.

6.3 Filter

A context collected and delivered by the context manager can contain a lot of information and thus might not be most efficiently leveraged by a given application. Also, the data obtained by the service manager for generating results needs to be further selected so that the content shown to the users is satisfactory. Therefore, the filter component was implemented to accomplish these two purposes.

In GoCity, the filter component provides two functions. First, it extracts and formats the context information delivered by the context manager. For example, the context manager obtains the weather related information and generates an object called Weather. The Weather object has the following information: weather condition, temperature, wind speed, humidity, etc. Currently, GoCity only requires the weather condition and the temperature and thus the filter component extracts and formats these two pieces of information from the Weather object based on the requirements. The reason that we format the context information is that the original context information provided by the context manager might not have the right format that is required by the given application. Thus, formatting the context information is accomplished by the filter as well. Second, it helps to refine the generated results that are shown to the user. The response data is initially processed by the service manager. However, the data that is used to generate the results might not be precise. The filter checks the returned data and further refines it to avoid improper content shown to the user. For example, a user only wants to find five restaurants near to him/her, but unfortunately the application initially generated more than five results. At this moment, the filter helps to eliminate the redundant results to ensure that the user is satisfied with the final generated content.

The filter component is composed of a number of Android Activities in the implementation of GoCity. It not only receives and extracts context information required by the given application, but also shows the refined content in the UIs. In the dynamic mode, the filter sends the formatted context information to the context adaptation manager. Then, the context adaptation manager analyzes the updated information and accordingly adjusts the application behavior. According to its features, the filter is an

application-specific component and hence each application must have its own filter component.

6.4 Summary

Data sources play a critical role in a context-aware mobile application. They not only provide the context information but also are the source for generating results. Seven web service APIs and one web site are employed in GoCity. The service manager component manages interactions between the application and outside web services. The filter component not only extracts and formats the context information required by the mobile application but also refines the results obtained by the service manager to show appropriate content to users.

Chapter 7 Self-Adaptation

This chapter describes the mechanism employed by GoCity to accomplish self-adaptation. It begins with an overview of how the self-adaptive behaviors are embedded into GoCity. Then, it describes the adaptation rules that govern the operations of GoCity. It ends with a summary.

7.1 Self-adaptive System

It has been noticed that mobile applications exist in a highly changing environment. These changes need to be attached importance to, because a user's concern can be affected by the changes and some unexpected changes can possibly cause failure of the application. GoCity, the implemented context-aware adaptive mobile application of this thesis, accomplishes its self-adaptation in two aspects: First, it monitors the context change, analyzes the lifetime of a context, and tunes the context manager to use the proper polling rate to collect diverse contexts; second, it also monitors the context change, but analyzes whether the value of a context parameter extracted by the filter component has changed or not, and updates the content shown to the user. Clearly, all the adaptive behaviors built in GoCity are based on context change. As a result, a component called Context Adaptation Manager was implemented in GoCity to monitor context change and accordingly modify the application's behavior.

Figure 15 in Chapter 4 has shown the adaptive model proposed in this thesis. The proposal of this model is inspired by the notion of AE introduced by IBM in 2006 [40]. An autonomic manager, a managed element, and two manageability interfaces compose

the AE. The autonomic manager collects the details it needs from the system. The details might be events, sensory data or any information that can be obtained by the autonomic manager. Then, it analyzes the details to determine whether to change something in the system. With the analysis results, it creates a plan which finally directs it to perform a sequence of actions on the managed element. The interaction between the autonomic manager and the managed element can be supported through the two manageability interfaces—sensors and effectors. In our proposed adaptive model, the context adaptation manager is the autonomic manager, and the context manager and filter are the managed elements.

In Figure 15, the context manager and the filter components are depicted as one managed element whose adaptive features are governed by the context adaptation manager. As a matter of fact, in GoCity, the context adaptation manager is divided into two parts to allow self-adaptive behaviors in the context manager and the filter respectively. The reason we do this is to distinguish the reusable part and the un-reusable part of the context adaptation manger.

In the implementation of GoCity, the context adaptation manager is divided into two parts: One analyzes the lifetime of an updated context and accordingly adjusts the polling rate of this context; the other analyzes the change of a context parameter that is utilized by a given application and accordingly determines and executes a sequence of actions to adjust the application behavior. Clearly, the former part is not application-specific and thus can be reused by various applications; while the latter is application-specific and utilizes the adaptation rules of the given application and thus is hard to reuse.

As we described before, the feedback loop (a closed control loop) that automates some management functions is the hard-core of a self-adaptive system. In the autonomic manager, it is represented as a MAPE-K loop which operates in four phases to accomplish four functions. The reader who is not familiar with the MAPE-K loop and each phase's functions can refer to section 2.4.1 for more details. Figures 23 and 24 both illustrate that the implemented parts that support self-adaptation in GoCity leverage the structure of the AE to a great extent.

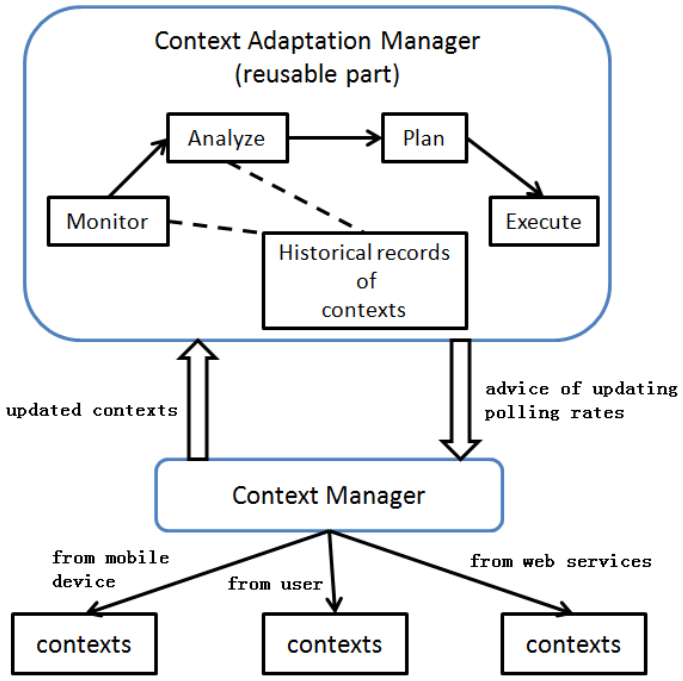


Figure 23. Self-adaptive system employed for governing the self-adaptive behaviors of the context manager

In GoCity, the polling rates of dynamic contexts are initialized at start-up. These initial polling rates are stored in the local database (SQLite database in Android) and become one part of the knowledge base that is used in the self-adaptive system. After querying the polling rates, the context manager updates the context at a specific time. Then, it notifies the context adaptation manager of the updated context. As depicted in

Figure 23, the context adaptation manager is similar to the autonomic manager with a MAPE-K loop, and the context manager is obviously the managed element. The monitoring phase examines which context has been updated and stores it in the historical records of contexts. Then, the analysis phase checks if the updated context differs from the last one and it generates some polling rate related data (e.g., property Changed, property Adjusted Polling Rate as shown in Table 5) according to the analysis result. The planning phase and execution phase are integrated together. In other words, the execution phase is an extension of the planning phase and thus not self-contained. The planning phase formulates the policies to determine the newest polling rate and estimate the lifetime of the context. The execution phase carries out the actual actions which update the polling rate related data in the database and trigger the context manager to use the newest polling rate to collect the context.

The context manager component can collect a variety of contexts. However, a single application might only use some of them. As a result, the filter helps extract the contexts that are only used by the given application. The filter extracts the required context information and then passes it to the context adaptation manager as depicted in Figure 24. The monitoring phase checks which context has been updated and it stores them in the context records. The analysis phase investigates whether any updated context parameter differs from the previous one. If so, it assesses if the recommendation needs to change because of the new context parameter. The context records and the policies are applied in this process. Similar to the reusable part, the realised un-reusable part of the context adaptation manager integrates the planning phase and the execution phase together as well. The planning phase formulates the policies which are a set of if-then rules, and then

the execution phase triggers the filter to generate new results according to the recommendation rules. The dotted lines between the filter and the service manager as depicted in Figure 24 illustrates that the application might not be a web-based application. In other words, if the application does not need the data from the web to generate the results, the filter does not need the support of the service manager.

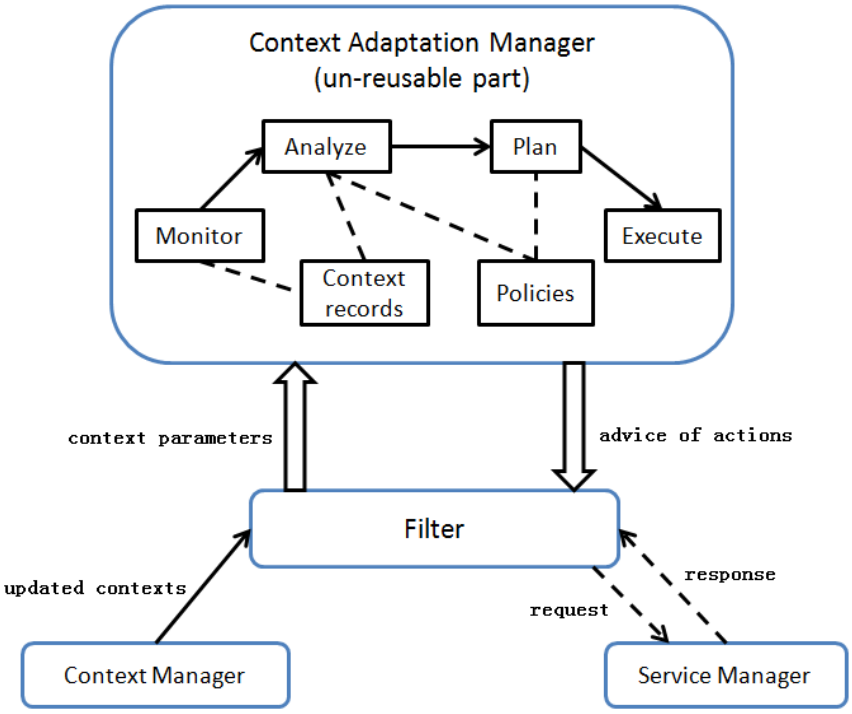


Figure 24. Self-adaptive system employed for governing the self-adaptive behaviors of the filter

7.2 Policies

In a self-adaptive system, policies are basically a set of if-then rules used to maintain the managed system for having desired states or behaviors. Two sets of policies are employed in GoCity: One suggests the proper polling rates that are used to update

contexts; the other helps generate appropriate contents for the user. They will be discussed in the following two sections.

7.2.1 Policies on Polling Rates

Contexts have been classified into static and dynamic categories based on their properties in this thesis. As described in Chapter 3, we only need to monitor the change of dynamic contexts that are collected by poll. Therefore, the policies that are designed in GoCity only adjust the polling rates of dynamic contexts gathered by poll. Since dynamic contexts that are gathered by poll are further classified into the one that has a low initial polling rate and the other that has a high initial polling rate, I set the low initial polling rate to be 1 hour and the high initial polling rate to be 10 minutes in realised GoCity. As a result, GoCity has the following dynamic contexts:

- High initial polling rate (10 minutes) context: user location, time of day, battery status.
- Low initial polling rate (1 hour) context: weather condition, network connectivity, availability of web services.

In order to optimize the polling rates, we use the policies which are represented by using the pseudo code in Java as shown below. The actual polling rate is the real time interval after which the context is re-obtained. The suggested polling rate is the lifetime of the context that we deduce. The adjusted polling rate is either a positive value or a negative value that is used to adjust the suggested polling rate, and it is preset at start-up. The adjusted polling rate of the first collected context is the initial polling rate. The adjusted polling rate of the unchanged context is a certain positive value (e.g., 0.5 hour in

GoCity), while the adjusted polling rate of the changed context is a certain negative value (e.g., -0.5 hour in GoCity).

```
List<Context_T> contexts = new ArrayList<Context_T>();
int size = contexts.size();
if (size == 0)
{
   Context_T context = new Context_T();
   context.actual_polling_rate = 0;
   context.adjusted_polling_rate = INITIAL_POLLING_RATE;
   context.suggested_polling_rate = INITIAL_POLLING_RATE;
   triggerContextManagerCollectContextUsingActualPollingRate();
   Thread.currentThread().sleep(context.actual_polling_rate+60000);
   context.vaule = getCollectedContextValue();
   context.changed = true;
   contexts.add(context);
}
else
{
   Context_T context = new Context_T();
   Context_T pre = contexts.get(size-1);
   if (pre.changed == true)
     {
        context.actual_polling_rate = pre.suggested_polling_rate;
     }
   else
     {
        context.actual_polling_rate = pre.adjusted_polling_rate;
     }
   triggerContextManagerCollectContextUsingActualPollingRate();
   Thread.currentThread().sleep(context.actual_polling_rate+30000);
   context.vaule = getCollectedContextValue();
   if (context.value == pre.value)
      {
         context.changed = false;
         context.adjusted_polling_rate = POSITIVE_VALUE;
      }
   else
     {
        context.changed = true;
        context.adjusted_polling_rate = NEGATIVE_VALUE;
      }
   if ((pre.suggested_polling_rate+context.adjusted_polling_rate)
                                              < INITIAL_POLLING_RATE)
      {
```

```
            context.suggested_polling_rate = INITIAL_POLLING_RATE;
        }
    else
        {
            context.suggested_polling_rate =
                    pre.suggested_polling_rate+context.adjusted_polling_rate;
        }
    contexts.add(context);
}
```

Table 5 is an example showing how the policies work to adjust the polling rate of a context based on its historical records. When the context manager collects the context at the first time it does not wait for a period, and thus the *actual_polling_rate* is 0. The *adjusted_polling_rate* and the *suggested_polling_rate* of the first collected context are both preset to 1 hour. Except the first collected context, the context's *actual_polling_rate* is assigned to the *suggested_polling_rate* of its previous context if its previous context has changed (the property *changed* of its previous context is true); otherwise, the context's *actual_polling_rate* is assigned to its previous context's *adjusted_polling_rate*. After the context is obtained, if it differs from its previous context, the *adjusted_polling_rate* is assigned to a negative value (e.g., -0.5) and the property *changed* is assigned to true; otherwise, the *adjusted_polling_rate* is assigned to a positive value (e.g., 0.5) and the property *changed* is assigned to false. The context's *suggested_polling_rate* is assigned to the sum of its previous context's *suggested_polling_rate* and its *adjusted_polling_rate*. If the context's *suggested_polling_rate* is less than its initial polling rate, it is reset to the initial polling rate.

Table 5. An example of adjusting polling rates using policies

Index	Context value (temperature)	Actual polling rate	Adjusted polling rate	Suggested polling rate	Changed
1	25	0	1 hour (initial)	1 hour	true
2	25	1 hour	0.5 hour	1.5 hours	false
3	25	0.5 hour	0.5 hour	2 hours	false
4	24	0.5 hours	-0.5 hour	1.5 hours	true
5	23	1.5 hour	- 0.5 hour	1 hour	true
6	22	1 hour	-0.5 hour	1 hour	true
7	22	1 hours	0.5 hour	1.5 hours	false
8	21	0.5 hour	-0.5 hour	1 hour	true

7.2.2 Policies on Generating Satisfactory Contents

As discussed in chapter 5, GoCity is composed of four sub-applications: Search, Nearby, Barcode, and Weather, each of which leverages several contexts to generate the results to the user. Compared to Search and Nearby, Barcode and Weather are simpler because they do not have built-in self-adaptive behaviors. This means that they do not sense the context change and thus do not dynamically generate results. In other words, only Search and Nearby use policies for dynamically generating results.

Search and Nearby have very similar functions, they are both designed to recommend various businesses (e.g., shops, restaurants, bars, cinemas) to the user based on collected contexts (e.g., location, user interest, time of day, weather). The selected data sources that offer the business listings require several attributes (e.g., business type, location, and distance to the specified location). Unfortunately, these attributes are very

limited and hence sophisticated policies are impossible to implement in the current prototype. For example, a sophisticated policy involving sales information, working hours, and parking availability cannot be achieved at the current moment. As a result, all implemented policies of Search and Nearby are currently only based on location, time, weather, user interest, user activity history, network availability, data source availability, and battery power status. Table 6 contains the policies implemented in Search and Nearby.

Table 6. Advice offered based on selected contexts

Contexts	Advice
Interest: Specified by the user (e.g., food)	Business related to the user's selection or input (e.g., restaurant, bistro); Switch to static mode
Interest: Extracted from events in the calendar (e.g., birthday)	Business related to the event. (e.g., boutique, restaurant)
Interest: Any Time of day: 6:00 am – 10:00 am	Cafe and restaurants offering breakfast
Interest: Any, Time of day: 10:00 am – 12:00 pm	Cafe offering coffee and tea
Interest: Any Time of day: 12:00 pm – 14:00 pm, 17:00 pm – 20:00 pm	Restaurants offering lunch and dinner
Interest: Any Time of day: 14:00 pm – 17:00 pm	Business on shopping (e.g., malls, stores, gallery)

Contexts	Advice
Interest: Any Time of day: 20:00 pm – 24:00 pm	Business on night life (e.g., cinema, bar, club)
Interest: Any Time of day: 0:00 am – 6:00 am	Business selected at random
Preferred distance: Specified by the user (e.g., 5 km)	Business up to 5 km from the location
Preferred distance: Not specified Weather: Good (e.g., sunny, cloudy)	Business up to 5 km from the location
Preferred distance: Not specified Weather: Bad (e.g., rain shower, snowy)	Business up to 2 km from the location
Location: Specified by the user (e.g., University of Victoria)	Business nearby the specified location (e.g., business nearby University of Victoria)
Location: Not specified	Business nearby the user's current location
Battery power: < 20%	Notification showing battery power is low; Choices provided to the user to disable the map showing.
User preference: Rating stars specified by the user (e.g., five stars)	Business of rating stars upper or equal to the specified rating (e.g., business of five rating stars)

Contexts	Advice
User preference: Rating stars not specified User activity history on selecting rating stars	Business of rating stars upper or equal to the rating that the user selected the most frequently in the past
Network availability: No	Notification showing network is currently unavailable; Choices provided to the user to shut down the application
Data source availability: The data source that is currently used is not available (e.g., World Weather Online web service is unavailable), but the alternative one is available.	An alternative data source that is currently available substitutes it. (e.g., Yahoo weather web service replaces WWO web service.)
Data source availability: Data sources that all provide a specific context (e.g., weather condition) are unavailable.	Notification showing this issue; Choices provided to the user: either shut down the application or use old data.

Below is a piece of code showing some simple if-then rules.

```
//set search range and category based on the time and
weather condition.
     public void setRC()
     {
          //the weather is good
          if (weatherStatus == true)
          {
               range = "5";
          }
          //the weather is bad
          else
          {
               range = "2";
```

```
        }
        //6:00AM TO 10:00AM
        if(6<=hour&&hour<=9){
             category = "Breakfast_brunch";
        }
        //10:00AM TO 12:00PM
        else if(10<=hour&&hour<=11){
             category = "Coffee";
        }
        //12:00PM TO 14:00PM
        else if(12<=hour&&hour<=13){
             category = "Restaurants";
        }
        //14:00PM TO 17:00PM
        else if(14<=hour&&hour<=16){
             category = "Shopping";
        }
        //17:00PM TO 20:00PM
        else if(17<=hour&&hour<=19){
             category = "Restaurants";
        }
        //20:00PM TO 24:00PM
        else if(20<=hour&&hour<=23){
             category = "Night Life";
        }
        else{
             category = "";
        }

    }
```

Sometimes two or more policies might apply at the same time and thus the advice offered for both policies possibly conflict with each other. For example, the user selects “library” as his/her current interest and the application is notified with a birthday event from the calendar at the same time. In this case, one policy takes the library as the user interest and thus suggests nearby libraries, while another policy takes the birthday as the user interest and hence suggests boutiques and restaurants. To avoid this conflict, we prioritize the policies with the help of several extra rules. In the fore-mentioned example,

the policy involving a calendar's event is prior to the one involving a user's selection. The reason is that this event might only be sensed once and thus it is better not to miss it.

7.3 Summary

The AE proposed by IBM is employed to a great extent to equip GoCity with self-adaptation. The context adaptation manager, which works as the autonomic manager of an AE, is implemented with a MAPE-K loop. The context manager and the filter are the managed elements in the realised self-adaptive system. The context adaptation manager is divided into a reusable part and an un-reusable part. The reusable part monitors the context change and accordingly adjusts the polling rates used by the context manager. The un-reusable part monitors the contexts leveraged by the given application and dynamically generates content for the user. Two sets of policies are applied to govern the self-adaptive behaviors of the context manager and the filter.

Chapter 8 Evaluation

This chapter evaluates GoCity by means of an experiment. In this experiment we assessed the static mode and the dynamic mode separately by discussing the efficiency, effectiveness, and user's satisfaction. This chapter begins with an overview of the experiment. Then, we separately evaluate the static mode and dynamic mode. It ends with a summary.

8.1 Overview of Experiment

GoCity is designed to provide city visitors with up-to-date and context-aware information while they are exploring a city using Android mobile phones. It is composed of four sub-applications—Search, Nearby, Barcode, and Weather—all of which employ a number of contexts to generate results. Except Barcode and Weather, Search and Nearby both have two operational modes—static mode and dynamic mode—which means they both have self-adaptive features to dynamically produce results. Search and Nearby both enable users to explore the businesses they are interested in. Barcode is to find price related information of a product and Weather provides weather information. The experiment set up to evaluate GoCity assumes an exchange student who is new to the city of Victoria wants to find out interesting places, buy books with good deals for his upcoming semester, and have the weather information to plan his travel with the help of GoCity.

In the experiment, we directly installed GoCity to a Samsung Galaxy S Android smartphone before starting the static mode evaluation. When estimating the dynamic mode, it is impossible to make some real changes in the environment (e.g., changing the

weather, shutting down a web service). Hence, we hard coded some fake changes in GoCity and then installed it in Galaxy S to complete the evaluation of the dynamic mode. For example, below is the function checking the web service availability:

```
public boolean getWSAvailability(String latitude,String
longitude) throws URISyntaxException,
ClientProtocolException, IOException
     {
          String Uri =
"http://www.worldweatheronline.com/feed/weather.ashx?q="+q+
"&format="+FORMAT+"&num_of_days="+NUM_OF_DAYS+"&key="+APIKE
Y;
          HttpClient client = new DefaultHttpClient();
          HttpGet request = new HttpGet();
          request.setURI(new URI(Uri));
          HttpResponse response = client.execute(request);
          int statusCode =
                response.getStatusLine().getStatusCode();
          if(statusCode == 200) return true;
          else return false;
     }
```

Instead of calling getWSAvailability(latitude, longitude) to check the web service availability as show below:

```
ws_availability = ws1.getWSAvailability(latitude,longitude);
```

We dynamically change the web service availability in the code to simulate that this web service is unavailable at a certain time period.

```
    private Handler mHandler = new Handler();
    public static final TIME_INTERVAL = 600000;
    private Runnable periodicTask = new Runnable() {
        public void run() {
           if(20<=hour&&hour<=21){
                 ws_availability = false;
              }
           else ws_availability =
                    ws1.getWSAvailability(latitude,
longitude);
           mHandler.postDelayed(periodicTask,
                                TIME_INTERVAL);
        }};
```

8.2 Evaluation of GoCity

Since usability determines how successful a mobile application can be, the measurable factors that contribute to usability are applied to assess GoCity. According to ISO, these measurable factors include efficiency (time or effort used to achieve the intended task), effectiveness (the degree to which the objectives are achieved, and the extent to which the targeted problems are solved), and user satisfaction (a measure of how the application meets or surpasses user's expectations). In general, efficiency can be quantitatively determined. For example, how much time is spent to load a web page, whereas effectiveness and user satisfaction are both qualitative measures.

In the assessment of GoCity, efficiency is determined by two quantitative parameters: time spent and the number of steps that a user operates to achieve the intended task. The effectiveness is established by the accuracy of generated results. We assess effectiveness using three scales: low, moderate, and high. User satisfaction is an emotional measure and usually has the following scales: very satisfied, somewhat satisfied, neither satisfied nor dissatisfied, somewhat dissatisfied, and very dissatisfied. [55]

8.2.1 Evaluation of the Static Mode

For testing the static mode of GoCity, we proposed the following scenario to generate several requests:

An exchange student Michael has been in Victoria for three days. Since he has already settled well, he decides to explore some interesting things near his location. Before he determines where to go, first he wants to obtain local weather information.

Then, he wants to find a nearby bookstore to browse interesting books. Unfortunately, the book he wants to buy is expensive in the bookstore he visits. Hence, he finally decides to buy the book from an online store. All of his requests can be achieved by using GoCity, and thus he launches GoCity installed on Galaxy S Android Smartphone, which only takes two seconds. Note that, in the following assessment, the time and steps needed to launch the application are not taken into account. Moreover, since Nearby and Search both provide similar functions, we only assess Nearby in this chapter.

Task one:

Get current weather information

Application:

GoCity - Weather

Context:

- Date and time: 11:30, Sunday, May 6, 2012
- Location: 2375 Lam Circle Rd, Victoria, BC (48.41259,-123.36489)
- Data connectivity: 3G
- Available web service: World Weather Online, Yahoo
- Battery status: 86%
- User interest: Current weather
- Mode: Static

Results:

Condition: Sunny

Temperature: 9°C/ 48.2 °F

Wind Speed: 4 mph

Humidity: 78%

Pressure: 1033 mb

Precipitation: 0.0 mm

Observation Time: 11:00

Evaluation:

Efficiency: 1 step (Click the tab ‘Weather’), 4 seconds

Effectiveness: Moderate. Michael got the weather information, but it was observed half an hour ago by World Weather Online or Yahoo.

User satisfaction: Very satisfied. Michael got the information very quickly and with little effort. Although the provided weather information was observed half an hour ago, he felt that the current weather had not changed a lot.

Task two:

Find a nearby bookstore.

Application:

GoCity - Nearby

Context:

- Date and time: 11:45, Sunday, May 6, 2012
- Location: 2375 Lam Circle Rd, Victoria, BC (48.41259,-123.36489)
- Weather: Sunny
- Search range: Within 5 kilometers (determined by the weather condition)
- Data connectivity: 3G
- Available web service: Yelp, Yahoo, Flickr, Google, World weather online
- Battery status: 83%

- User interest: Bookstores, rated as 4 stars and above (determined by the user's previous selections)
- Mode: Static

Results:

- Cadboro Bay Book Co, 3840 Cadboro Bay Rd, 5.0 stars, 1.2 km
- Curious Comics, 1581 Hillside Ave, 4.0 stars, 4.2 km
- Bolen Books, 1644 Hillside Ave, 5.0 stars, 4.9 km

Evaluation:

Efficiency:

- 2 steps (if it is the first time for the user to search bookstores), 9 seconds

  Step 1: Add the button 'Bookstores' (3 Clicks and type 'Bookstores'), 5 seconds

  Step 2: Click button 'Bookstores' (1 click), 4 seconds
- 1 step (if the user has searched bookstores before), 4 seconds

  Step 1: Click button 'Bookstores' (1 click), 4 seconds

Effectiveness: High. All the results were nearby bookstores of 4 stars and above within 5 kilometers.

User satisfaction: Very satisfied. First, Michael did not need to type the entire word 'Bookstores' thanks to the auto-suggestions, which saved time. Second, the search range (5 kilometers) determined by the current weather (sunny) was acceptable. Third, in the previous search history, Michael preferred to search businesses of 4.0 stars and above, thus the results selected by the application were 4.0 stars and above. In summary, Michael felt that he used little effort to get the satisfactory results very quickly. Most

importantly, Michael also has the ability to change the search range and the preferred rating star to get different results.

Task three:

Find online stores who offer the book Michael wants.

Application:

GoCity - Barcode

Context:

- Data connectivity: 3G
- Available web service: UPC Database
- Battery status: 80%
- Book barcode: 9781430226598
- User interest: Online stores who offer that book
- Mode: Static

Results:

A web page containing the following price related information:

Merchant	Condition	Price	Shipping	Total	Purchase
College Book Renter	Rental	$0.00	$2.99	$2.99	Buy Now!
AbeBooks.com	Used	$1.25	$2.95	$4.20	Buy Now!
Half.com	Used	$1.25	$3.49	$4.74	Buy Now!
Textbooks.com Marketplace	Used	$1.25	$3.99	$5.24	Buy Now!
Amazon Marketplace	Used	$1.25	$3.99	$5.24	Buy Now!
Half.com	New	$4.08	$3.49	$7.57	Buy Now!
Amazon Marketplace	International	$3.72	$3.99	$7.71	Buy Now!
Biblio.com	New	$4.08	$3.99	$8.07	Buy Now!
AbeBooks.com	New	$4.08	$3.99	$8.07	Buy Now!
Amazon Marketplace	New	$4.08	$3.99	$8.07	Buy Now!
Amazon.com	New	$4.30	$3.99	$8.29	Buy Now!
Biblio.com	Used	$5.00	$3.50	$8.50	Buy Now!
Half.com	International	$5.12	$3.49	$8.61	Buy Now!

Merchant	Condition	Price	Shipping	Total	Purchase
eCampus Marketplace	Used	$10.10	$3.97	$14.07	Buy Now!
Barnes & Noble eBooks	E-Book	$22.79	$0.00	$22.79	Buy Now!
TextBookX.com	Used	$18.95	$4.96	$23.91	Buy Now!
BarnesAndNoble.com	New	$27.16	$0.00	$27.16	Buy Now!
CampusBookRentals	Rental	$28.68	$0.00	$28.68	Buy Now!
eCampus Marketplace	New	$30.00	$3.97	$33.97	Buy Now!
TextBookX.com	New	$36.13	$4.96	$41.09	Buy Now!
BookByte.com	New	$37.50	$3.65	$41.15	Buy Now!
eCampus.com	New	$37.32	$3.97	$41.29	Buy Now!
Chegg	Rental	$41.99	$4.99	$46.98	Buy Now!

Evaluation:

Efficiency:

There are two ways to achieve this task:

- Enter a barcode:

  2 steps (type the barcode number and 1 click), 37 seconds

- Scan a barcode:

  1 step (1 click), 32 seconds

Effectiveness: High. All the results were the online stores who rent or sell the book that Michael wanted.

User satisfaction: Somewhat satisfied. Michael felt that the results were very useful because he could get that book from one of those online stores with an acceptable price. However, the time to load that web page was longer than expected.

8.2.2 Evaluation of the Dynamic Mode

There are two ways to trigger the dynamic mode: one is to click the button ‘Have Fun’ in Nearby; the other is to leave the textbox of category empty and click the “Good

Luck" button in Search. Three context updates were captured in this evaluation: time, weather condition, and web service availability.

Task

Explore nearby businesses.

Initial Context:

- Date and time: 14:30, Sunday, May 6, 2012
- Location: 3624 Shelbourne St, Victoria, BC (48.45954, -123.33312)
- Weather: Rainy
- Data connectivity: 3G
- Search range: Within 2 kilometers (determined by the weather condition)
- Available web service: Yelp, Yahoo, Flickr, Google, World Weather Online
- Battery status: 71%
- Search business: Shopping (determined by the time), rated as 4 stars and above (determined by user's previous selections)
- Mode: Dynamic

Results:

- Bolen Books, 1644 Hillside Ave, 5 stars, 1.6 km
- Curious Comics, 1581 Hillside Ave, 4 stars, 1.8 km
- The Root Cellar, 1286 McKenzie Ave, 4.5 stars, 1.97 km

Evaluation:

Efficiency: 1 step (click button 'Have Fun' or click button 'Good Luck'):

- Click button 'Have Fun' in Nearby, 8 seconds
- Click button 'Good Luck' in Search, 5 seconds

Effectiveness: Moderate. Since the time had been 14:30 PM, GoCity assumed that the user's interest was shopping at that time. Hence, all the generated results were nearby stores of 4 stars and above within 2 kilometers.

User satisfaction: Neither satisfied nor dissatisfied. Michael did not want to go shopping due to the weather, so he did not think any of the results were suitable for him.

Context Update I:

- Date and time: 17:00, Sunday, May 6, 2012
- Location: 3840 Cadboro Bay Rd, Victoria, BC (48.46149, -123.29676)
- Battery status: 47%
- Search business: Restaurants (determined by the time), rated as 4 stars and above (determined by user's previous selections)

Results

- Olive Olio's, 3840 Cadboro Bay Rd, 5.0 stars, 0.03 km
- Grad House, University of Victoria, 4.5 stars, 1.17 km

Evaluation:

Efficiency: 0 step (The results were generated without any user operation), N/A.

Effectiveness: Moderate. Since the time was 17:00, GoCity assumed that the user was interested in restaurants at that time. Hence, all the results were nearby restaurants of 4 stars and above within 2 kilometers.

User satisfaction: Somewhat satisfied. 17:00 was dinner time, thus Michael felt that the generated results were reasonable. However, he preferred to find a Chinese restaurant and none of the results was suitable.

Context Update II:

- Date and time: 17:45, Sunday, May 6, 2012
- Weather: Sunny
- Searching range: Within 5 kilometers (determined by the weather condition)
- Battery status: 38%

Results

- Olive Olio’s, 3840 Cadboro Bay Rd, 5.0 stars, 0.03 km
- Grad House, University of Victoria, 4.5 stars, 1.17 km
- Paprika Bistro, 2524 Estevan Ave, 4.5 stars, 2.77 km
- The Village Restaurant, 2518 Estevan Ave, 4.0 stars, 2.78 km
- Charellis Cheese Shop Delicatessen, 2851 Foul Bay Road, 4.5 km, 2.93 km
- Mutsuki-An, 2075 Cadboro Bay Rd, 5.0 stars, 3.68 km
- Eugenes Greek Restaurant, 1990 Fort St, 4.0 stars, 4.0 stars, 3.78 km
- Ottavio Italian Bakery & Delicatessen, 2272 Oak Bay Ave, 4.5 stars, 4.08 km
- Oak Bay Bistro, 2250 Oak Bay Ave, 4.5 stars, 4.1 km
- Penny Farthing Public House, 2228 Oak Bay Ave, 4.0 stars, 4.11 km
- Oak bay Marina Restaurant, 1327 Beach Dr, 4 stars, 4.24 km
- Nar Cafe Bistro, 2540 Windsor Rd, 4.5 stars, 4.32 km
- Blighty's Bistro, 2006 Oak Bay Ave, 4 stars, 4.33 km
- Mee Wah Restaurant, 1950 Oak Bay Ave, 4.0 stars, 4.38 km
- CosMos 2 For 1 Pizza & Pasta, 1557 Pandora Ave, 5.0 stars, 5.0 km

Evaluation:

Efficiency: 0 step (The results were generated without any user operation), N/A

Effectiveness: Moderate. Since the weather was sunny, the search range had been updated to 5 kilometers. Hence, all the new generated results were nearby restaurants of 4 stars and above within 5 kilometers.

User satisfaction: Somewhat dissatisfied. Michael wanted to find a Chinese restaurant; however, none of the results were suitable.

Context Update III:

- Date and time: 20:00, Sunday, May 6, 2012
- Location: 605 Yates Street, Victoria, BC (48.42646, -123.36708)
- Available Web service: Yelp, Yahoo, Flickr, Google
- Battery status: 23%
- Searching business: Night Life (determined by the time), rated as 4 stars and above (determined by user's previous selections)

Results:

- Garricks Head Pub, 1140 Government St, 4 stars, 0.12 km
- Ferris Oyster Bar, 536A Yates St, 4.5 stars, 0.15 km
- Darcys Pub, 1127 Wharf St, 4.0 stars, 0.21 km
- The Mint, 1414 Douglas St, 4.0 stars, 0.22 km
- The Local Bar & Grill, 1205 Wharf St, 4.0 stars, 0.22 km
- Solstice Cafe, 529 Pandora Ave, 4.0 stars, 0.26 km
- Swans Hotel, 506 Pandora Ave, 4.5 stars, 0.29 km
- Big Bad Johns Strathcona Hotel, 919 Douglas St, 4.0 stars, 0.36 km
- Bartholomews Bar & Rockefeller Grille, 777 Douglas St, 4.5 stars, 0.49 km
- Clives Classic Lounge, 740 Burdett Avenue, 5.0 stars, 0.53 km

- Blue Crab Bar & Grill, 146 Kingston St, 4.0 stars, 1.05 km
- Stage, 1307 Gladstone Ave, 4.5 stars, 1.7 km

Evaluation:

Efficiency: 0 step (The results were generated without any user operation), N/A

Effectiveness: Moderate. Since the time was 20:00, GoCity assumed that the user was interested in night life at that time. Hence, all the results were nearby businesses offering night life of 4 stars and above within 5 kilometers.

User satisfaction: Neither satisfied nor dissatisfied. Michael did not think any of the results were suitable for him. But, one good thing was that the unavailability of World Weather Online did not affect the usage of GoCity.

8.3 Summary

GoCity was assessed by an experiment involving a case study, in which an exchange student used his Android phone to explore the city of Victoria. Three determining factors of usability—efficiency, effectiveness, and user satisfaction—were introduced in this evaluation to qualitatively and quantitatively assess GoCity. The results of this evaluation exhibited both positive and negative sides of GoCity. The dynamic mode worked more efficiently than the static mode, but the user is more satisfied with the generated results of the static mode. This is because the static mode has more interactions with the user than the dynamic mode and thus it gains a clearer picture of the user's concerns and can accordingly generate more reasonable and suitable results. The user accomplished his tasks in both modes effectively.

Chapter 9 Conclusion and Future Work

This chapter concludes my work and outlines potential future work.

9.1 Summary

The mobile application market has become an indispensible segment in today's global IT market. Its fast growth has attracted a large number of companies and research institutions. Context-awareness has gained much attention in mobile application design. However, the current approaches have significant challenges according to some researchers and my personal observations. This thesis targeted four of them and developed ideas on how to address these challenges:

- Few contexts other than location and time have been used in actual mobile applications.
- Context is not clearly classified in the field of mobile application design.
- Mobile application designers lack systematic mechanisms to address sensing and monitoring requirements under changing context situations. This is crucial for effective self-adaptation.
- Problems in UI design lead to low usability for many mobile applications.

This thesis first classified the contexts used by mobile applications using two dimensions: (1) Based on its source, context was classified into personal, environmental, and mobile device context; (2) based on its property, context was classified into static and dynamic context.

After the classification of context, this thesis proposed a model for designing context-aware adaptive mobile applications. This model is composed of four key

components: context manager, context adaptation manager, filter and service manager. All of these components support two interaction modes: static and dynamic. Since Android has become popular and provides many powerful features to developers, the model was built on the Android platform. The context manager collects diverse context information relevant to users' needs to help generate appropriate results. The service manager deals with all transactions between the application and the outside web services. The filter extracts the exact context information employed by the given application and also refines the results shown to the user. Finally, the context adaptation manager, implemented using the autonomic manager introduced by IBM, provides adaptation functions to the context manager and the filter. In the context adaptation manager, a set of adaptation rules are applied to determine the polling rate on dynamic contexts and the results that will be shown to the user.

In order to verify the feasibility of the proposed model, I implemented a prototype called GoCity on the Android platform. GoCity provides city visitors with up-to-date information and helps a user explore a city. It is composed of four sub-applications: Nearby, Search, Barcode, and Weather, each of which leverages several contexts to provide diverse functions. As for the design of the user interface, GoCity focuses on providing a simple and clear user interface and supporting user expectations for personalization.

We conducted one experiment to evaluate three factors of GoCity: efficiency, effectiveness, and user satisfaction. The results of this qualitative and quantitative assessment of GoCity brought to light positive and negative aspects of GoCity. The assessment showed that the dynamic mode works more efficiently than the static mode

but the user is more satisfied with the results generated by the static mode, and both static mode and dynamic mode accomplish their tasks effectively.

9.2 Contributions

This thesis discussed and investigated background information and related work regarding the design of context-aware adaptive mobile applications. According to our study, we came up with four significant issues and then proposed a model and implemented the prototype GoCity. The main contributions of my work are as follows:

- Identified four significant issues for context-aware mobile applications;
- Conducted background research and related work regarding the design of context-aware adaptive mobile applications;
- Proposed the classification of context in the field of mobile applications;
- Proposed and implemented a model for context-aware adaptive mobile applications by leveraging the Android platform, CBD, SBD, and AC technologies;
- Implemented a prototype to verify the feasibility of the proposed model;
- Implemented a simple and clear user interface to support user expectations for personalization to enhance usability.

9.3 Future work

There are a number of aspects of my work that could be investigated:

- Separate the context manager, the context adaptation manager and the service manager from the filter to provide a lightweight application. Each application

built upon the current model contains all of the four components. However, only the filter component and some part of the context adaptation manager component are application-specific. As a result, if we could separate those components that are not application-specific from each application, the application could be more lightweight.

- Use more user history as additional contexts. In GoCity, user history of rating stars is applied as one context to generate results. Obviously, there is more user history that can be used as additional contexts, such as user history regarding business interests.
- Further improve the user interface. For the user interface we only focused on providing simple and clear user interface and supporting user expectations for personalization, therefore, the user interface is not aesthetically pleasing.
- Add more data sources. It is not only the way to offer more contexts, but also provide additional alternative contexts to avoid failures when acquiring contexts. Additionally, the results generated for the user can be further enriched and improved.
- Improve the adaptation rules and enhance the recommendation mechanism. According to the results of the evaluation of GoCity, the user sometimes was not satisfied with the suggested results.
- Enhance the mechanism to adjust the polling rate of the context to get the synchronized context change. The polling rate is currently determined by a simple algorithm, and thus it is impossible to achieve synchronized context change.

Bibliography

[1] G.D. Abowd, C.G. Atkeson, J. Hong, S. Long, R. Kooper and M. Pinkerton, "Cyberguide: A mobile context-aware tour guide," Wireless Networks 3(5):421-433, 1997.

[2] G. Chen and D. Kotz, "A survey of context-aware mobile computing research," Dartmouth Computer Science Technical Report TR2000-381, 2000.

[3] L. Northrop, P. Feiler, R.P. Gabriel, J. Goodenough, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan and K. Wallnau, "Ultra-large-scale systems: the software challenge of the future," Technical report, Carnegie Mellon University Software Engineering Institute, 2006.

[4] M. Chignell; J. Cordy; J. Ng; Y. Yesha (eds.), *The smart internet*, LNCS 6400, Springer, Nov 2010.

[5] N.M. Villegas, H.A. Müller, "Managing dynamic context to optimize smart interactions and services," in *The Smart Internet*, M. Chignell; J. Cordy; J. Ng; Y. Yesha (eds.), LNCS 6400, Springer, pp. 289-318, Nov 2010.

[6] Android Application Framework official website: http://developer.android.com/about/versions/index.html.

[7] Canalys, "Canalys Q2: 68% of all smartphones shipped were Android; China's the biggest market by a wide margin," http://techcrunch.com/2012/08/02/canalys-q2-68-of-all-smartphones-shipped-were-android-chinas-the-biggest-market-by-a-wide-margin/, 2012.

[8] P. Bahl and V.N. Padmanabhan, "Radar: An in-building RF-based user location and tracking system," In Proceedings of IEEE INFOCOM 2000, pp. 775-784, 2000.

[9] P. Beadle, B. Harper, G.Q. Maguire and J. Judge, "Location aware mobile computing," In Proceedings of IEEE/IEE International Conference on Telecommunications (ICT 1997), 1997. Also available at http://web.it.kth.se/~maguire/LocationAware/ICT97/ict97.htm.

[10] A. Ward, A. Jones and A. Hopper, "A new location technique for the active office," IEEE Personal Communications 4(5):42-47, 1997.

[11] Micheal Beigl, "MemoClip: A location-based remembrance appliance," Personal Technologies 4(4):230-233, September 2000.

[12] A. Schmidt, M. Beigl and H. Gellersen, "There is more to context than location," Computers & Graphics Journal, Elsevier 23(6):893-902, 1999.

[13] P. Korpipaa, J. Mantyjarvi, J. Kela, H. Keranen and E.J. Malm, "Managing context information in mobile devices," Pervasive Computing, IEEE 2(3):42-51, 2003.

[14] J.O. Kephart and D.M. Chess, "The vision of autonomic computing," Computer *36*(1):41-50, 2003.

[15] Smartphone:http://www.pcmag.com/encyclopedia_term/0,2542,t=Smartphone&i=51537,00.asp.

[16] Smartphone: http://en.wikipedia.org/wiki/Smartphone

[17] Mark Brownlow, "Smartphone statistics and market share," http://www.email-marketing-reports.com/wireless-mobile/smartphone-statistics.htm, October 2012.

[18] IDC, "Smartphone market hits all-time quarterly high due to seasonal strength and wider variety of offerings, according to IDC," http://www.idc.com/getdoc.jsp?containerId=prUS23299912, February 2012.

[19] Dagstuhl Seminar 08031 on Software Engineering for Self-Adaptive Systems, http://www.dagstuhl.de/08031, January 13-18, 2008.

[20] “Global mobile statistics 2012 Part E: Mobile apps, appstores, pricing and failure rates,” http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/e, June 2012.

[21] Usability: http://en.wikipedia.org/wiki/Usability.

[22] B. Schilit and M. Theimer, “Disseminating active map information to mobile hosts,” IEEE Network 8(5):22-32, 1994.

[23] M. Sama, S. Elbaum, F. Raimondi, D. Rosenblum and Z. Wang, “Context-aware adaptive applications: Fault patterns and their automated identification,” IEEE Transaction on Software Engineering 36(5):644-661, Sept.-Oct. 2010.

[24] K. Cheverst, N. Davies, K. Mitchell and P. Smith, “Providing tailored (context-aware) information to city visitors,” In Proceedings of the International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, pp. 73-85, 2000.

[25] L. Capra, W. Emmerich and C. Mascolo, “CARISMA: Context-aware reflective middleware system for mobile applications,” IEEE Transaction on Software Engineering 29(10):929-945, October 2003.

[26] M. Raento, A. Oulasvirta, R. Petit and H. Toivonen, “ContextPhone: A prototyping platform for context-aware mobile applications,” Pervasive Computing, IEEE 4(2):51-59, Jan.-Mar. 2005.

[27] A.K. Dey, “Providing architectural support for building context-aware applications,” PhD. Thesis Dissertation, College of Computing, Georgia Tech., 2000.

[28] D. Dawson, R. Desmarais, H.M. Kienle and H.A. Müller, “Monitoring in adaptive systems using reflection,” In Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems, pp. 81-88, 2008.

[29] H. Liu and M. Parashar, “Accord: A programming framework for autonomic applications,” IEEE Transactions on Systems, Man, and Cybernetics 36(3):341-352, 2006.

[30] Y. Brun, G. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H.A. Müller, M. Pezzè and M. Shaw, “Engineering self-adaptive systems through feedback loops,” In: Software Engineering for Self-Adaptive Systems, pp. 48-70, Springer, Heidelberg, 2009.

[31] C. Peper and D. Schneider, “Component engineering for adaptive ad-hoc systems,” ACM 2008 International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2008), pp. 49–56, 2008.

[32] A. Zimmermann, A. Lorenz and R. Oppermann, “An operational definition of context,” In Proceedings of the 6th International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT 2007), pp. 558-571, 2007.

[33] N. Ryan, J. Pascoe and D. Morse, “Enhanced reality fieldwork: The context-aware archaeological assistant,” Computer Applications in Archaeology, 1997.

[34] D. Garlan, S. W. Cheng and B. Schmerl, “Increasing system dependability through architecture-based self-repair,” In: Architecting Dependable Systems, pp. 61-89, Springer, Heidelberg, 2003.

[35] J. Kramer and J. Magee, “Self-managed systems: An architectural challenge,” *FOSE 2007:* 2007 Future of Software Engineering, 29th ACM/IEEE International Conference on Software Engineering (ICSE 2007), pp. 259-268, May 2007.

[36] G. Brown, B. H. Cheng, H. Goldsby and J. Zhang, “Goal-oriented specification of adaptation requirements engineering in adaptive systems,” ACM 2006 International Workshop on Self-Adaptation and Self-Managing Systems (SEAMS 2006), pp. 23-29, 2006.

[37] H.A. Müller, M. Pezzè, and M. Shaw, “Visibility of control in adaptive systems,” In Proceedings of the 2nd International Workshop on Ultra-Large-Scale Software-intensive Systems (ULSSIS 2008), pp. 23-26, 2008.

[38] M. Salehie and L. Tahvildari, “Self-adaptive software: Landscape and research challenges,” ACM Transactions on Autonomous and Adaptive Systems (TAAS) 4(2):1-42, May 2009.

[39] H. A. Müller, H. M. Kienle and U. Stege, “Autonomic computing: now you see it, now you don’t,” In: Software Engineering, pp. 32-54, Springer, Heidelberg, 2009.

[40] IBM, “An architectural blueprint for autonomic computing,” IBM White Paper, 2006.

[41] A.K. Dey and G.D. Abowd, “Towards a better understanding of context and context-awareness,” In Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing (HUC '99), pp. 304-307, 1999.

[42] B. Schilit, N. Adams and R. Want, “Context-aware computing applications,” In Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications (WMCSA '94), pp. 85-90, 1994.

[43] T. Gross and M. Specht, “Awareness in context-aware information systems,” In Proceedings of Mensch & Computer, pp. 173-181, 2001.

[44] J. Cassens and A. Kofod-Petersen, “Using activity theory to model context awareness,” In: Lecture Notes in Computer Science, Vol. 3946, pp. 1-17, 2006.

[45] H.W. Gellersen, A. Schmidt and M. Beigl, “Multi-sensor context-awareness in mobile devices and smart artifacts,” Mobile Networks and Applications 7(5):341-351, Oct. 2002.

[46] O. Filho and M. Ferreira, “Semantic web services: a RESTful approach,” IADIS International Conference WWWInternet 2009, pp. 169-180, 2009.

[47] F. Irmert, T. Fischer and K. Meyer-Wegener, “Runtime adaptation in a service-oriented model,” In Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems (SEAMS 2008), pp. 97-104, 2008.

[48] I. Amendola , F. Cena , L. Console , A. Crevola , A. Goy , S. Modeo , M. Perrero, I. Torre and A. Toso, “UbiquiTO: a multi-device adaptive guide,” In Proceedings of Mobile Human-Computer Interaction (MobileHCI 2004), pp. 409-414, 2004.

[49] C. Becker and C. Bizer, “Dbpedia mobile: a location-aware semantic web client,” In Proceedings of the Semantic Web Challenge, 2008.

[50] N. Weissenberg, R. Gartmann and A. Voisard, “An ontology-based approach to personalized situation-aware mobile service supply,” Geoinformatica 10(1):55-90, Mar. 2006.

[51] J. Froehlich, M. Y. Chen, S. Consolvo, B. Harrison and J. A. Landay, “MyExperience: A system for in situ tracing and capturing of user feedback on mobile phones,” In Proceedings of the 5th International Conference on Mobile Systems, Applications and Services (MobiSys 2007), pp. 57-70, 2007.

[52] H. Rahnama, P. Kramaric, A. Sadeghian and A. Shepard, “Self-adaptive middleware for the design of context-aware software applications in public transit systems,” In Proceedings of the 13th International Conference on Ubiquitous Computing (UbiComp 2011), pp. 491-492, 2011.

[53] J. M. Carrol, “Scenario-based design,” In: International Encyclopedia of Ergonomics and Human Factors, Second Edition, Vol. 1, pp. 198-204, 2006.

[54] K. Cheverst, N. A Davies, K. C. Mitchell, A. Friday and C. Efstratiou, “Developing a context-aware electronic tourist guide: some issues and experiences,” In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 17-24, 2000.

[55] M. Conklin, “Measuring and tracking customer satisfaction,” White paper, retrieved from: http://www.zoomerang.com/whitepapers/customersat.pdf, 2006.

[56] B.H.C Cheng, et al., “Software engineering for self-adaptive systems: A research roadmap,” in *Software Engineering for Self-Adaptive Systems*, LNCS Hot Topics, pp. 1-26, 2009.

[57] Yelp API, retrieved from: http://www.yelp.ca/developers/getting_started/api_access.

[58] Flickr API, retrieved from: http://www.flickr.com/services/api/.

[59] Yahoo Local API, retrieved from: http://developer.yahoo.com/local/.

[60] Yahoo weather API, retrieved from: http://developer.yahoo.com/weather/.

[61] Google Map API, retrieved from: https://developers.google.com/android/maps-api-signup.

[62] Android Location API, http://developer.android.com/guide/topics/location/index.html.

[63] World Weather Online API, retrieved from: http://www.worldweatheronline.com/free-weather.aspx.

[64] UPC database, retrieved from: http://www.upcdatabase.com/itemform.asp.

[65] Open Mobile IS official website: http://www.openmobileis.org.

[66] PhoneGap official website: http://www.phonegap.com/.

Appendix A Glossary

3G Third Generation is a family of standards for mobile telecommunications, which includes UMTS, WCDMA, and several others. 3G allows simultaneous use of speech and data services at a high speed.

4G The fourth generation of cell phone mobile communications standards. It is a successor of 3G. A 4G system provides mobile ultra-broadband Internet access to other mobile devices such as laptops with USB wireless modems and smartphones.

AC Autonomic Computing

ACRA Autonomic Computing Reference Architecture

AE Autonomic Element

Android A Linux-based operating system primarily designed for mobile devices. Android became the world's leading smartphone platform at the end of 2010.

API Application Programming Interface

BCL Business Category Library

CBD Component Based Design

CPU Computer Processing Unit

Dalvik Dalvik is the process virtual machine in the Google Android operating system. It improves a device's efficiency when running multiple VMs by executing files in the Dalvik Executable (.dex) format instead of Java byte code (.class) format

DDMS Dalvik Debug Monitor Service. A tool that manages processes on an Android emulator and assists in debugging

dx	dx is a tool for converting java class file into the Dalvik Executable (.dex) format
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
J2ME	Java to Micro Edition
JSON	JavaScript Object Notation
MAPE-K	Monitor – Analyse – Plan – Execute – Knowledge
OS	Operating System
PC	Personal Computer
PDA	Personal Digital Assistant
SBD	Scenario Based Design
SDK	Software Development Kit
SMS	Short Messaging Service
SQLite	A rational database management system contained in a small C programming library
UI	User Interface
UML	Unified Modeling Language
VM	Virtual Machine
Wi-Fi	Any wireless local area network (WLAN) products that are based on the Institute of Electrical and Electronics Engineers (IEEE) 802.11 standards
XML	Extensible Markup Language