ABSTRACT

Elpida Kyriakou

University of Cyprus, 2012

Context is an important aspect to computing, especially in mobile computing. The context can be used to adapt the user interaction with an application, or provide general services and information to the user. Computers cannot offer this kind of information regarding the conditions, in which the communication act occurs, which means they cannot think in context. For this reason, context-aware application becomes important. The introduction of powerful mobile devices has raised the potential of building novel context-aware applications. Such applications let the users enjoy a better experience by sensing their context and automating tasks that would otherwise require significant user attention.

This thesis presents a context-aware application, built on top of a presented middleware system. By describing the development steps, we reveal how development of a context-aware application using middleware system becomes easier for developers; relieving them from having to develop and mesh such code in their apps. It is shown that this approach reduces the required development and maintenance effort and thus lowers the associated cost.

BUILDING CONTEXT-AWARE APPLICATIONS WITH REUSABLE COMPONENTS FOR THE ANDROID PLATFORM

Elpida Kyriakou

University of Cyprus, 2012

A Thesis

Submitted in Partial Fulfilment of the

Requirements for the Degree of

Master of Science

at the

University of Cyprus

Recommended for Acceptance

By the Department of Computer Science

May, 2012

APPROVAL PAGE

Master of Science Thesis

BUILDING CONTEXT-AWARE APPLICATIONS WITH REUSABLE COMPONENTS FOR THE ANDROID PLATFORM

Presented by

Elpida Kyriakou

Research Supervisor	
	George A. Papadopoulos
Committee Member	
	Nearchos Paspallis

University of Cyprus

May 2012

ACKNOWLEDGEMENTS

First, I would like to thank my supervisor Professor George A. Papadopoulos, who provided his support throughout the progress of my dissertation. Also, many thank to my advisor Nearchos Paspallis, who also provided his support and guidance throughout the progress of my thesis. My advisor Nearchos Paspallis, provided me with a lot of advice, and help through the process of my project especially in the implementation of context-aware middleware system.

CREDITS

Table of Contents

Chapter 1: Introduction 1

1.1 Motivation 1-2

1.1.1 The role of mobile computing 2-4

1.1.2 The role of context aware computing 4-6

1.1.3 The role of middleware 7-8

1.2 Thesis statement 9

1.3 Approach 10

1.4 Thesis structure 10-11

Chapter 2: Related Work 12

2.1 Categories of middleware systems 12-14

2.2 Requirements of context-aware middleware systems 14-15

2.3 Existing context-aware middleware systems 16-19

2.4 IST-MUSIC 19-20

2.5 Summary and Conclusions 21

Chapter 3: Android model 22

3.1 Android Architecture 22-24

3.2 Components of an Android application 25-30

3.3 AIDL (Android Interface Definition Language) 31

3.4 Context-Awareness in Android 31-35

3.4.1 Android Sensors 33-34

3.4.2 Android Sensor Framework 34-35

3.5 Conclusion...35

Chapter 4: An implementation of Context-Aware Application using the presented middleware system running on Android 36

4.1 Introduction.. 36-37

4.2 The context-aware middleware system framework 37-38

4.3 Implementation of Context-aware Middleware system framework........... 39-44

4.3.1 Developing context-aware applications 40-41

4.3.2 Context plug-ins 41-44

4.4 Context-aware media player application (CaMPlayer) 44-52

4.4.1 Main Functional aspects of CaMPlayer 45-46

4.4.2 Register CaMPlayer application 46-52

4.5 Conclusions...52

Chapter 5: Evaluation 53

5.1 Component based and Monolithic based application.............................. 53-54

5.2 Practical Evaluation.. 54-55

5.3 Requirement Driven Evaluation.. 55-59

Chapter 6: Conclusions and Future Work 61

6.1 Conclusions and Future Work 61-63

References 64-66

Chapter 1

Introduction

When the widespread adoption of mobile and ubiquitous computing devices is considered, it is nowadays becoming evident; that there is significant interest for developing applications featuring context-aware and self-adaptive behaviour. During this period many new opportunities arise for creating more intelligent application, which can sense the environment and understand the user actions in a more profound way, in other words applications that can sense and react to context. This thesis studies the role of mobile computing, context-awareness, and middleware and presents an extensive case study of using a middleware-based architecture for building context-aware applications. This architecture is evaluated and shown to bring important benefits for mobile application developers.

1.1 Motivation

Currently, the common paradigm for developing a context-aware mobile application is building it from scratch. When a developer needs access to simpler context data (e.g., battery level or GPS location) then the underlying platform (such as the Android Middleware) offers some well-formed APIs (e.g., BatteryManager and LocationManager respectively). However, when it comes to enabling generic, as well as more sophisticated context-aware behaviour (say identify when the user is driving or sleeping), then the corresponding code is developed either from scratch or copy-pasted from a relevant source (e.g., from an open source implementation, or some other form of documentation).

Naturally, this approach suffers from two important drawbacks. First, individual developers are required to spend time repeating the same tasks in different situations, often reinventing a wheel that others have already created (but did not share with the rest). Second, with a wide variety of isolated solutions used interchangeably in different applications, the context-aware behaviour appears to be fragmented and inconsistent to the end users.

Today, there are many smart phones that contain the basic building blocks for context-awareness like GPS, accelerometers, light sensors, and physical sensors, which let the developers create their own applications. This is not enough, as users expect the application to combine information about their physical location with data from other applications. This thesis studies such middleware solution implemented for the Android platform, where the developers of context-aware applications use ready-made components for realizing the context sensing part of their applications.

1.1.1 The role of mobile computing

Mobile computing is introduced as an important technology underlying context-aware computing, due to the fact that mobility causes frequent and interesting changes in application context, which may be used to proactively influence application behaviour.

There are many definitions for mobile computing; Most notably: “Mobile computing is a variety of wireless devices that has the ability to allow people to connect to the Internet providing wireless transmission to access data and information from where ever location they may be" [1]. Alternatively: "A technology that allows transmission of data, via a computer, without having to be connected to a fixed physical link" [2].

Additionally, in a consideration based, towards context-aware computing as a subset of pervasive computing, mobility causes frequent changes to the context in which an application executes. In marked contrast to stationary systems, mobile systems may experience rapid changes in location, administrative domain, bandwidth availability and economy, temperature, speed, proximity to other devices, and a host of other environmental parameters. Related to this consideration is the fact that awareness of the dynamic execution context by an application on a mobile device allows the application to initiate specific activity, for instance, reallocation of resources. As a result, mobile computing environments exhibit a range of characteristics that both challenge the developer of applications for such environments, as well as provide a source of input to applications that may be used to control behaviour. Some of these characteristics are listed below.

- Portability: Ability to move a device within different environments with ease.
- Social Interactivity: Allows for data sharing and collaboration between users.
- Context Sensitivity: Ability to gather and respond to real or simulated data unique to a current location, environment, or time.
- Connectivity: Ability to be digitally connected for the purpose of communication of data in any environment.

On the other hand mobile computing provides some limitations. One of these is the limited bandwidth: mobile Internet access is generally slower than direct cable connections. Security standard is another limitation of mobile computing; since when you are working on mobile you are dependent on public networks and your data may be eavesdropped. Also, mobile devices rely entirely on battery power, thus another limitation is power consumption and the fact that mobile devices require expensive batteries so they can provide the necessary battery life. While mobile computing is a form of human–computer interaction by which a computer is expected to be transported during normal usage, it is important to refer two limitations that refer to the human. The first limitation refers to the potential health hazards that is; people have more accidents because they use the mobile during driving as a result to destruct them [1]. The second limitation is called as the human interface with device while screens and keyboards tend to be small, which may make them hard to use [1]. Alternate input methods such as speech or handwriting recognition require training.

Our proposed solution aims, primarily at mobile computing environments and reviewing the limitations and advantages of mobile computing, guide us to the implementation of our system.

1.1.2 The role of context-aware computing

Context-awareness was first discussed as “as software that adapts according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time” by Schlitz and Theimer [3,4]. Many definitions were proposed throughout the years while new research evolves context-aware computing.

In order to better understand the role of context-aware applications we first need to declare the definition of context and how it can be used, that will enable application developers to choose what context to use in their applications and to determine what context-aware behaviours to support in their applications. 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 application themselves" [3,4]. Context-aware application is defined as follows: "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" [3, 4].

For a long time, context-aware applications were hard to develop because there weren't devices able to support it. Nowadays, almost every smart phone is equipped with sensors and communication systems which can provide a lot of information about the environment, sufficient to act as input for such applications. Despite the complexity of writing this kind of applications, especially when the information comes from many different types of sensors, each with their own unique programming interface; context-aware applications have increased in the last few years and can also make assumptions about the user's current location or activity. There are some of characteristics that wearable computers should meet to be appropriate for hosting context-aware applications such as: [4]

- Portable while operational: a wearable computer is capable of being used while the user is mobile. When the user is mobile, his context is much more dynamic. He is moving through new location and does other activities so the services and information she requires will change based on these new entities.
- Sensors: a wearable computer should use sensor to collect information about the user's surrounding environment.
- Proactive: a wearable computer should be acting on its user's behalf even when the user is not explicitly using it. This is the essence of context-aware computing:

the computer analyzes the user's context and makes task relevant information and services available to the user, interrupting the user when appropriate.

- Always on: this is important for context-aware computing because the wearable computer should be continuously monitoring the user's situation or context so that it can adapt and respond appropriately. It is able to provide useful services to the user anytime.

However there are some challenges when building context-aware applications. One challenge of mobile distributed computing is to exploit the changing environment with a new class of applications that are aware of the context in which they are run. Such context-aware software adapts according to the location of use, the collection of nearby people, hosts, and accessible devices, as well as to changes to such things over time. A system with these capabilities can examine the computing environment and react to changes to the environment. Three important aspects of context are: where you are, whom you are with, and what resources are nearby. Context encompasses more than just the user's location, because other things of interest are also mobile and changing. Context includes lighting, noise level, network connectivity, communication costs, communication bandwidth, and even the social situation. Furthermore another challenge is the lack of reusable context-aware mechanisms; that is every context-aware system requires mechanism in order to support context sensing for gathering information and context reasoning dealing with the interpretation of the gathering information. Finally, the last challenge of context-aware computing is the privacy. People are worried about how computer systems use and share their personal information. Context-aware application has this limitation and this raises the concern for user privacy.

Despite the existence of application that provide context information there is still a need of a direct, uniform way to access the information.

1.1.3 The role of middleware

In a distributed computing system, “middleware is the software layer that lies between the operating system and the applications on each site of the system” [6]. There are many uses of middleware such as the reuse of legacy code. By this, many enterprise-wide information systems need to integrate legacy code with newly implemented components, in a result to avoid cost when they want to re-implement them. Another use is for mediation systems. Many complex systems of multiple devices interconnected via a network. Management involves tasks such as monitoring performance, logging alarms, executing remote maintenance function etc. A third use of middleware is for component-based architecture that are based on separation-of-concerns and on well-defined standards interfaces. For an example if you have 3-tier architecture and provide common services through a layer, it is possible to further facilitate the development.

Formally, middleware is defined as “reusable software that resides between the applications and the underlying operating systems, network protocol stacks, and hardware” [7]. Based on this definition, the role of middleware is to provide common programming abstractions, to hide the heterogeneity and the distribution of the underlying hardware and operating systems and finally to hide low-level programming details.

Make use of middleware has some benefits and drawbacks. Some of the benefits are; the reuse of code, the independence of language and platform, the ease application development and maintenance and finally and more important the lower development cost and time. On the other hand, there are two main drawbacks. The first one, is the possible performance penalty, this can be caused of the messages that are often required to pass through multiple layers. And the second one is the re-training efforts for developers.

Although there are some challenges in middleware design. First of all is the performance penalties that can arise from the interception and indirection mechanisms that middleware rely on. A second challenge is the large-scale system; modern applications are overly complex and they involve a large number of distributed objects, users and devices and this results to a problem with respect to the capability of communication and the complexity of administration. A third challenge is that enables the vision of ubiquitous computing that requires significant leaps in the front of mobility and dynamic reconfiguration. And finally, there is a managements challenge, that is handling such issues such as security and resource management for large, heterogeneous application poses new challenges.

One example of middleware architecture is MUSIC [8]. MUSIC dynamically monitor context and adapt the applications in order to optimize the quality perceived by the user, we will refer later in detail in chapter 2. This thesis presents an implementation of a middleware solution, where the developers of context-aware applications use ready-made components for realizing sensing part of their application.

1.2 Thesis statement

This thesis consists of two parts: First, it presents a middleware solution, where the developers of context-aware applications use ready-made components for realizing the context sensing part of their applications. Instead of embedding such code in their own apps, developers will be able to utilize the proposed middleware and have access to the equivalent functionality. The developed middleware will offer a centralized and uniform way of accessing a dynamically changing set of context types. By adding new context sensor plug-ins, the middleware will be enriched with additional context sensing capabilities, relieving individual developers from having to develop and mesh such code in their apps. It is argued that this approach offers ease of development for sophisticated context-aware applications, via a market of context-aware plug-ins, fostering the development of and promoting excellence in context sensing techniques and easier maintenance of the resulting applications as the context sensing code is separated from the context-aware application (and, consequently, even their corresponding lifecycles are separated).

Second, we evaluate the resulting development approach by designing and implementing a context-aware application as component framework. The constituent components realize the roles of context providers (referred to as context plug-ins) and context consumers (i.e., context-aware, self-adaptive applications), and can be reused across multiple platforms and/or shared by concurrently deployed applications. The approach is then compared to the straight-forward (brute force) implementation of the context-aware application.

1.3 Approach

The thesis is based both on theoretical and practical aspects. The theoretical aspect referred to a development methodology that is based on the examination of related work and the examination of the Android platform. The practical aspect consists of building a context-aware application and evaluating the middleware architecture. This middleware provides support for the deployment of context-aware applications—which are constructed using the proposed methodology—and facilitates context sensing, management, and distribution. The steps we followed to manage our target are:

1. We develop a quick prototype of the architecture, along with a few simple context sensors and a simple pilot app,
2. We formalize the middleware architecture and the plug-in specification
    a. Examine related work,
    b. Study and assess the relevance of the APIs available in Android (i.e., main context services, content providers, event mechanism)
3. We revise the implementation, and develop additional (real) sensors
4. Implement a real context-aware application showcasing the architecture
5. Evaluate the impact of the proposed solution

1.4 Thesis structure

The thesis is organized as follows:

Chapter 1 presented the motivation of the thesis and a literature review of mobile computing, context-awareness, and middleware that this thesis is dealing with. We gave a description about their advantages, drawbacks, and challenges for each field. Following that, a description about the approach we have followed to build a context-aware application using middleware-based architecture was described.

Chapter 2 presents literature review with respect of existing middleware systems and context-aware applications. In this literature, requirements and challenges were identified that we took into consideration while building our context-aware application.

In chapter 3, a presentation of the Android platform was given. This includes the android architecture, the components of android: activities, services, broadcast receivers and intents and content provides. Also it refers to the android interface definition language (aidl) and to the sensors on android.

Based on the analysis of the related work of chapter 2, chapter 4 presents the implementation of our context-aware application. It describes the steps we have followed to develop this context-aware application using the presented middleware. Furthermore, it describes a way that developers can register the presented middleware to their own context-aware application.

Following that, in chapter 5 an evaluation is described of the context-aware application using the presented middleware. The evaluation consists of two phases. The first phase gives a comparison of a component context-aware application and a monolithic context-aware application. The latter revisited the requirements derived from the literature and listed in chapter 2 evaluated the proposed solution against them.

Finally, the thesis ends with chapter 6 that refers to some conclusions, as well as some lists of directions for future work.

Chapter 2

Related Work

In a distributed computing system, “middleware is the software layer that lies between the operating systems and the applications on each site of the system” [6]. In traditional way, the role of middleware is to provide common programming abstractions, to hide the heterogeneity and the distribution of the underlying hardware and operating systems and finally to hide low-level programming details.

This chapter provides a survey of context-aware middleware systems. First, the categories for context-aware middleware systems are studied, followed by an extensive list of requirements as they are documented in the literature. These requirements are further discussed and used in the rest of this thesis as they provide a well-formed benchmark for the design of the context-aware application using the presented middleware. Finally, a number of representative context-aware middleware systems are presented, that provide some method of adapting to changes in the context, and methods for collecting context.

2.1 Categories of middleware systems

This section provides a list of the categories of context-aware middleware systems, that the evaluation of such software systems should be made [9, 10].

- **Environment:** A middleware system makes explicit or implicit assumptions about the environment it is to be used in. We have two types of environment the infrastructure and self-contained. The first refers to middleware systems that assume the existence of an infrastructure, which offers services needed by the middleware and applications. The second refers to systems that assume that devices have some method of communication and does not rely on external services.
- **Storage:** Some systems provide a context-aware data store, which order data based on context information, allowing it to be retrieved based on certain context-parameters. For example, some systems provide file systems where data is ordered according to the current context. Other systems provide centralized storage facilities for context information, allowing applications to retrieve it.
- **Quality:** Quality is a measure of how well a service can be performed or how good data is. In the case of context-aware middleware quality is mostly concerned with quality of service, how well a resource can be provided. However, some systems provide quality measures of the offered context.
- **Composition:** Some middleware do component composition based on contextual events. For example, entities might be composed with all entities in their vicinity, or composition might be changed if some context event occurs.
- **Migration:** Some systems provide migration of entities. Some of the systems merely provide mechanisms for migrating running code when the application decides, possibly based on context, while other systems migrate entities automatically based on context.
- **Adaption:** When context-information is available, systems can adapt to changes in the context. Different parts might all use contexts in different ways, but most

middleware systems do not use context-information on all parts of the system. Adaption to changes might happen in middleware or in the applications. If adaption takes place at the middleware level, there are three sub-categories:

a. **Transparent**: The middleware reacts to changes in context without the application being aware of it.

b. **Profile:** The middleware receives a profile from the application detailing what kind of service is interested in. It is then the responsibility of the middleware to adapt so that it can provide a service as close to the requested as possible.

c. **Rules:** Rules are typically of the form if a then b. les are provided by either applications or by users, and indicate what action the middleware should or must take when a happens. When adaption is the responsibility of the application, the programmer is free to use context in any way imaginable. However, some middleware systems provide methods for invoking certain actions in the application based on context changes in the form of rules, typically with a callback to the application.

2.2 Requirements of context-aware middleware systems

This section provides the requirements for context-aware middleware systems as we have identified through many publications. The middleware we are presenting must address many of the requirements of traditional distributed systems, such as hetero, scalability, and tolerance for component failures, ease of development and configuration, adaptivity etc. [9,11].

- **Scalability:** context processing components and communication protocol must perform adequately in very changing domains.
- **Support for privacy:** flows of context information between the distributed components of a context-aware system must be controlled according to user's privacy needs and expectations.
- **Tolerance for component failures:** sensors are likely to fail in the ordinary operation of a context-aware system; disconnection may also occur.
- **Ease of deployment and configuration:** it must be easily deployed and configured to meet user and environmental requirements.
- **Dynamic reconfiguration:** detecting changes in available resources and reallocating them or notify the application to change its behavior.
- **Adaptivity:** the ability of a system to recognize unmet needs within its execution context and to adapt itself to meet those needs.
- **Asynchronous paradigm:** decoupling the client and server components and delivering multicast messages.

2.3 Existing context-aware middleware systems

While the previous section focused on the requirements identified in the literature that should be taken into consideration during the analysis of context systems, this section provides existing context-aware middleware systems that support those requirements.

ACAM [13]: is a middleware specially designed to facilitate the development of context-aware applications. It provides a direct, uniform way to access the information.

ACAM sits between the operating system and the context-aware applications. The ACAM consists of three components that are organized on several layers. The first layer deals with the data acquisition that is responsible for gathering information from the surrounding environment. Based on the current capabilities of mobile devices make use different context providers: sensors, data links, clock and user data. Sensors are used to capture information such as current temperature, altitude, location, the movement of the user (acceleration or direction), or the light intensity in the environment. ACAM also includes support for data links (through WiFi or Bluetooth technologies test the presence of nearby devices, access and exchange information), clock (leads to discovery of current user activity), and user data (outside the inference mechanism, the user is able to control the system response to context changes).

The second layer deals with abstraction. Information from context sources is gathered by the context manager and organized based on concepts from predefined model. This actually represents an abstraction layer, which is used by application to access context information. The domain described by the model acts as a contract between the middleware and the applications.

The last layer deals with the applications. Application can include the context in response to stimulus (interior or exterior request) so to better serve the user's needs. The application can react to context changes and take actions depending on some predefined rules. For this, data is retrieved and conditions are evaluated periodically. ACAM includes a rule engine capable of interpreting context-based rules. It provides the base for developing context-aware application that can change their appearance, and provide more dynamic interaction.

All these components are interconnected. The context acquisition is performed by specialized Monitoring Modules. A module can collect the data for a specific sensor or some specific operating platform. The Context Manager is responsible with the management of these modules. It provides context information to the applications in a structured form. The context depends on the information provided by the context modules. The user can download from a remote repository the context modules his current application requires. Also the Context Manager maintains a directory with information currently provided by the loaded monitoring modules. When a request for a specific context parameter is received, the manager mediated the request towards the corresponding context monitoring modules. A common scenario for this middleware system, in context-aware computing is when changes in the context trigger a specific action: for example when the clock shows 8 a.m the alarm goes off and announce the user that a new day has started. Another important function of the context-aware middleware is to serve such necessities again in the most flexible manner as possible. So, they designed a built-in rule engine that evaluates business rules from an input XML file and decides which actions should be started based on the current context parameters.

CARISMA [14,15]: deals with adaption of middleware depending on the needs of the applications. Profiles for each application are kept as meta-data of the middleware and consists of *passive* and *active* parts. The passive parts define actions the middleware should take when specific context events occurs, such as shutting down if battery is low. The active information defines relations between services used by the application and the policies that should be applied to deliver those services. The active part is thus only used when the application requests a service. Different environmental conditions may be specified, which determine how a service should be delivered. At any time, the application can use reflection to alter the profile kept by the middleware through an XML representation. To deal with conflicts between profiles, CARISMA adopts a micro-economic approach, where a computing system is modeled as an economy where consumers makes a collective choice over a limited set of goods. In this case, the goods are the policies used to provide services, not the resources providing them. The middleware plays auctioneer in an action protocol, where each application submits a single, sealed bit on each alternative profile. The auctioneer then selects the alternative, which maximizes the sum of bids. To determine the bid each of the applications are willing to pay, functions, which translate from, profile requirements to values are defined. Like profiles, these functions may be changed at any time through reflection. This type of protocol makes sense because CARISMA delivers the same service to all participants.

CARMEN [16] is intended for handling resources in wireless setting assuming temporary disconnects. It uses proxies, mobile agents residing in the same CARMEN environment as the user. If a user moves to another environment the proxy will migrate using wired connections. Each mobile user has a single proxy, which provides access to resources needed by the user. When migrating, the proxy makes sure that resources are also available in the new environment. This can happen by: moving the resources with the agent, copying the resources, using remote references, or re-binding to new resources which provide similar services. The method is determined by inspecting the profile of the device.

Each entity in CARMEN is described by a profile. User profiles contain information about preferences, security settings, subscribed services etc. Device profiles define the hardware and software of devices. Service component profiles define the interface of services and Site profiles group together the profiles which all belong to a single location. Thus, context information in CARMEN describes the entities, which make up the system.

2.4 IST-MUSIC

MUSIC (Mobile Users In Ubiquitous Computing Environments) is a focused initiative aiming to alleviate the development of context-aware mobile applications that dynamically and seamlessly adapt to changes in the user and execution context MUSIC can be deployed on Windows, Linux, Windows Mobile, and Android devices [30].

The motivation of the creation of MUSIC was mostly the high use of handheld devices [31]. When users moving around in ubiquitous computing environment many situations derived that affect different services such as the network connections come and go and QoS carry handheld devices varies. Therefore services available for use come and go, service quality varies, user tasks vary and are interleaved with tasks related to movement, and social interaction computing resources and power are limited.

In such environments applications and users will benefit a lot from context awareness and self-adaptiveness. The demand for applications exhibiting such properties is accelerating mobile computing, ubiquitous computing, service oriented computing. So, developing such applications with existing methods and technology is difficult, time-consuming, and costly.

The role of MUSIC project is to simplify the development of adaptive applications, for this reasons it offers [31]: a model-driven development approach that facilitates the development of self-adaptive applications and the reuse of adaptive components and services; and a sophisticated middleware that enables the dynamic adaptation of component-based applications. Some of the benefits that MUSIC offers and can be observed in many areas are [31]:

- Architecture Complexity: The MUSIC approach features a structured (layered) architecture that is easier to understand and easier to maintain.

- Code reusability: In addition MUSIC provides an extensible set of standardized reusable context sensors and reasoners (context plug-ins), facilitating the monitoring of common context elements.
- Resource management: Only the components, which are actually activated, are loaded (better memory management). Context sensor plug-ins are activated only when the corresponding context data is needed (managed by the middleware).
- Multiple applications: Context sensing is externalized and encapsulated in with context plug-ins. These plug-ins are deployed directly in the middleware and thus are easily shared and managed. Adaptation reasoning is also handled by the middleware, which is a centralized authority capable of reasoning on the complete adaptation domain, which might include multiple applications (possibly deployed on multiple devices).

To conclude, MUSIC provides an open platform for developing and deploying easily innovative applications suited for mobile users in ubiquitous and service oriented computing environments, which are context aware and self-adapting.

It should be noted that the MUSIC context system and its pluggable architecture were the inspiration for the core of this work, as it provides developers a methodology for developing context-aware applications and a middleware platform for deploying them. Our aim was to realize the equivalent functionality, while exploiting the facilities of the ANDROID platform.

2.5 Summary and Conclusions

This chapter presented a representative subset from the perspective of designing and building context-aware middleware systems. The examined approaches were analyzed in three aspects: categories of middleware, requirements, and existing systems. The author of this thesis, while designing the system, adopted the requirements. Furthermore, the current state-of-the-art of context-aware middleware explores quite different approaches to support ubiquitous and mobile computing based on context information. All of the middleware systems provided some method of adapting to changes in the context, and methods for collecting context, but otherwise use different entities and have different focus.

Chapter 3

Android Model

Android designed for mobile operating systems offering low power consumption. As smart phones and tablets become more popular, the operating systems such as Apple's iPhone, Android and Windows mobile for those devices become more important. The main difference between Android and the other operating systems is that, they are built on proprietary operating systems that often prioritize native applications over those created by third parties and restrict communication among applications and native phone data. In spite that these operating systems provide richer, simplified development environment for mobile applications, Android, from the other hand is an open development environment built on an open source Linux kernel that offers new possibilities for mobile applications. This chapter discusses the android model, which include the android architecture, android components, and android interface component language (aidl). Furthermore, it refers to the context-awareness in Android including context sensor support.

3.1 Android architecture

The first step to understand Android is to see how the general application model looks like. Android it is not only an operating system, it is also a software stack divided into five layers: Linux kernel, libraries, the android runtime the application framework and the applications. Figure 3.1 contains an architectural overview of the Android as it is published in the Android Developers Guide [17].

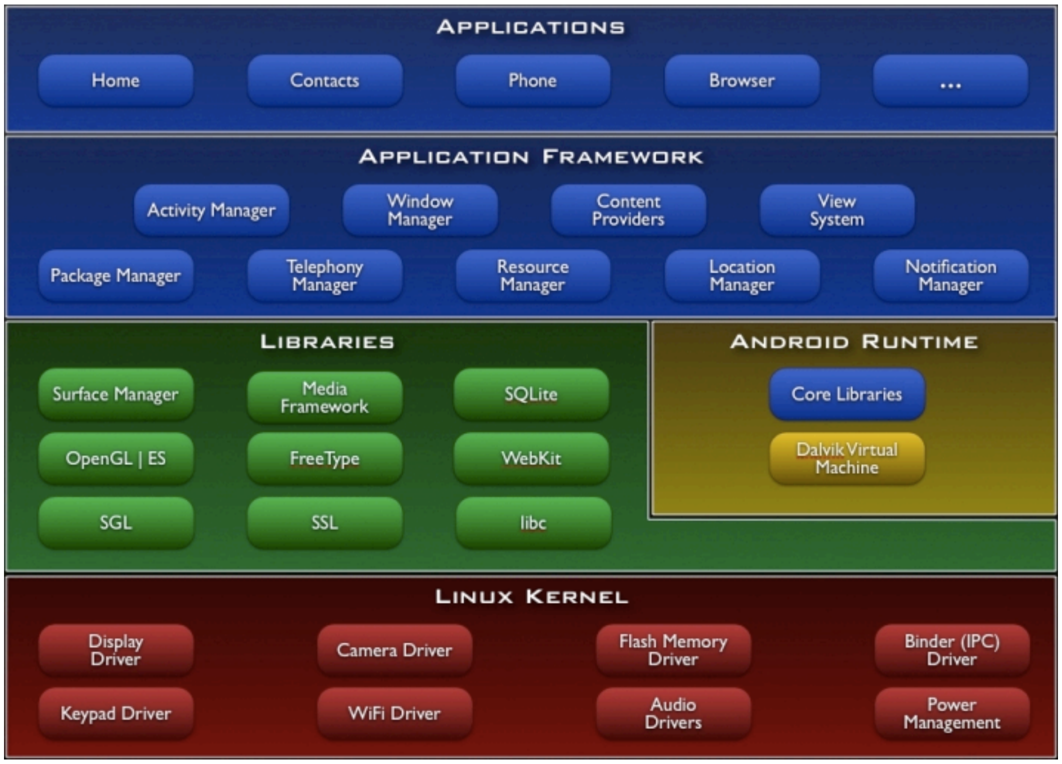


Figure 3.1: Android System Architecture

A brief description is given below for the five layers is of the Android architecture starting from the last level of the system [17]:

- Linux kernel: Android makes use of the standard Linux 2.6 kernel which is responsible for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware, which is responsible for supplying low-level functionalities like threading, and low-level memory management; and the rest of the software stack.

- Libraries: includes a set of C/C++ libraries used by various components in the Android system. Most of these libraries exist open source libraries, tuned for execution on embedded Linux-based systems.
- Android Runtime: unlike other operating systems that are written in C/C++ libraries, Android applications are written in Java and run in virtual machine. There are a set of core libraries using the functionality of core Java libraries. Android, features the Dalvik virtual machine, which execute its own byte code and every Android application runs with its own instance of Dalvik machine. Dalvik it is used to run multiple virtual machines efficiently and it is optimized for mobile devices. It relies on Linux kernel for underlying functionality such as threading and low-level memory management.
- Application framework: Developers have full access to the same framework APIs used by the core applications. The application architecture is designed to simplify the reuse of components; any application can publish its capabilities and any other application may then make use of those capabilities. This same mechanism allows components to be replaced by the user.
- Applications: Android will ship with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language and run on top of the application framework.

Android's architecture offers you the option to reuse its components, allowing you to share and publish activities, services, and data with other applications without the need to develop the code, just to install it using some security restrictions from your application.

3.2 Components of an Android application

Mostly, Android applications written in java and are built up using four essential [18]: activities, services, broadcast receivers, and content providers. Each component has its own lifecycle that defines you how the component is created and destroyed.

1. Activities: Is the fundamental concept that allows interaction with the user. At any point in time, a single activity is visible on the screen. A simple application might have only a single activity, while more complex applications consist of several activities that, together, form a user interaction model. Normally, an activity is started at the moment that it needs to interact with the user and it is stopped as soon as it is not visible on the screen anymore. Activities in the system are managed as an activity stack. When a new activity is started, it is placed on the top of the stack and becomes the running activity — the previous activity always remains below it in the stack, and will not come to the foreground again until the new activity exits. Figure 2.2, shows an Activity diagram with its methods [19]:
    - onCreate: Called when your activity is first created. This is the place you normally create your views, open any persistent data files your activity needs to use, and in general initialize your activity. When calling onCreate, the Android framework is passed a Bundle object that contains any activity state saved from when the activity ran before.
    - onStart: Called just before your activity becomes visible on the screen. Once onStart completes, if your activity comes to the foreground control will transfer to onResume. If the activity cannot become the foreground activity for some reason, control transfers to the onStop method.

- onResume: Called right after onStart if your activity is the foreground activity on the screen. At this point your activity is running and interacting with the user. You are receiving keyboard and touch inputs, and the screen is displaying your user interface. onResume is also called if your activity loses the foreground to another activity, and that activity eventually exits, popping your activity back to the foreground. This is where your activity would start (or resume) doing things that are needed to update the user interface (receiving location updates or running an animation, for example).
- onPause: Called when Android is just about to resume a different activity, giving that activity the foreground. At this point your activity will no longer have access to the screen, so you should stop doing things that consume battery and CPU cycles unnecessarily. If you are running an animation, no one is going to be able to see it, so you might as well suspend it until you get the screen back. Your activity needs to take advantage of this method to store any state that you will need in case your activity gains the foreground again—and it is not guaranteed that your activity will resume. Followed either by onResume() if the activity returns back to the front, or by onStop() if it becomes invisible to the user. Once you exit this method, Android may kill your activity at any time without returning control to you.
- onStop: Called when your activity is no longer visible, either because another activity has taken the foreground or because your activity is being destroyed. Followed either by onRestart() if the activity is coming back to interact with the user, or by onDestroy() if this activity is going away.
- onDestroy: The last chance for your activity to do any processing before it

is destroyed. Normally you'd get to this point because the activity is done and the framework called its finish method. But as mentioned earlier, the method might be called because Android has decided it needs the resources your activity is consuming. If an activity is paused or stopped, the system can drop the activity from memory by either asking it to finish, or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state.

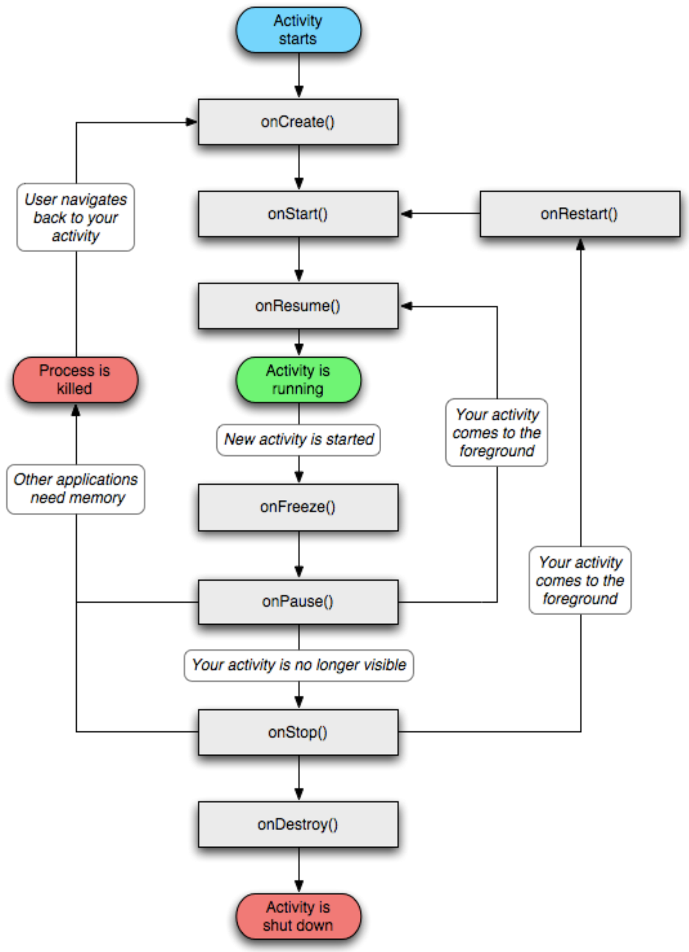

Activity starts
onCreate()
User navigates back to your activity
onStart()
onRestart()
onResume()
Process is killed
Activity is running
New activity is started
Your activity comes to the foreground
Other applications need memory
onFreeze()
Your activity comes to the foreground
onPause()
Your activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Figure 3.2.1: Activity Life cycle [19]

2. Services: are application components that run in the background and are not visible to the user, i.e. that don't have a user interface. Services can run in the background if a user switches to another application. Moreover, components can bind to a service that is already running in order to communicate with them. There are two types of services [20].

   - The first one is the *Started Service* that starts by calling startService(). This type of service runs in the background indefinitetly. Usually it only performs a single operation and does not return a result to the caller and when the operation is finished, the service should stop itself.
   - The second type of service it the *Bound Service* that binds by calling bindService(). It offers a client-server interface that allows components to interact with the service, send requests and get results. This service does not run indefinitetly, it destroyed when components unbind to it.

   Despite of these two types, you can create services that can use both types. Just like activities services must be declared in the manifest file. Figure 3.2.2 show the lifecycle for a service that is similar to an activity but simpler [20]:

   - onCreate and onStart differences: Services can be started when a client calls the Context.startService(Intent) method. If the service isn't already running, Android starts it and calls its onCreate method followed by the onStart method. If the service is already running, its onStart method is invoked again with the new intent.
   - onResume, onPause, and onStop are not needed: Recall that a service generally has no user interface, so there isn't any need for the onPause, onResume, or onStop methods. Whenever a service is running, it is always in the background.

- onBind: If a client needs a persistent connection to a service, it can call the Context.bindService method. This creates the service if it is not running, and calls onCreate but not onStart. Instead, the onBind method is called with the client's intent, and it returns an IBind object that the client can use to make further calls to the service. It's quite normal for a service to have clients starting it and clients bound to it at the same time.
- onDestroy: As with an activity, the onDestroy method is called when the service is about to be terminated. Android will terminate a service when there are no more clients starting or bound to it. As with activities, Android may also terminate a service when memory is getting low. If that happens, Android will attempt to restart the service when the memory pressure passes, so if your service needs to store persistent information for that restart, it's best to do so in the onStart method [20]:

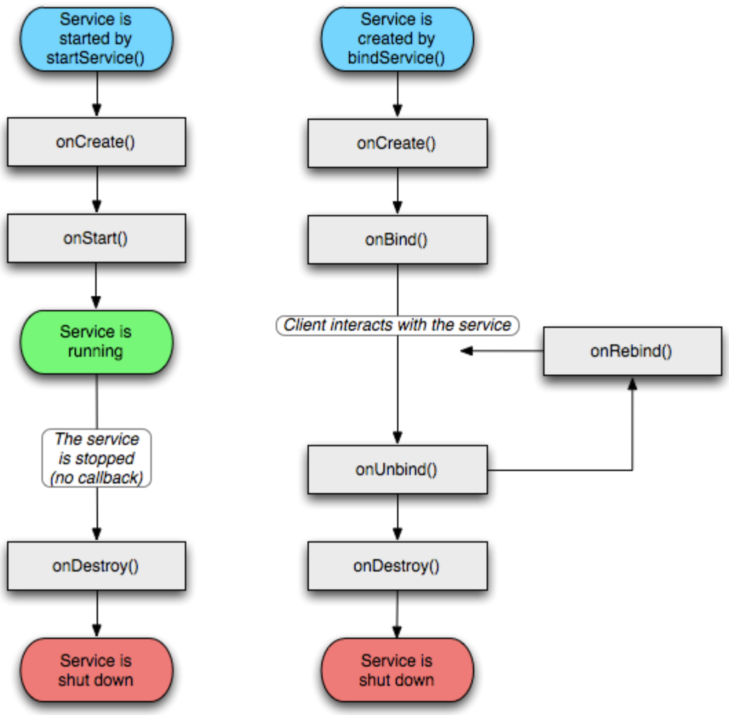

Service is started by startService()
onCreate()
onStart()
Service is running
The service is stopped (no callback)
onDestroy()
Service is shut down
Service is created by bindService()
onCreate()
onBind()
Client interacts with the service
onRebind()
onUnbind()
onDestroy()
Service is shut down

Figure 3.2.2: Service life cycle [20]

3. Broadcast receivers and intents: These respond to requests for service from another application. A Broadcast Receiver responds to a system-wide announcement of an event. These announcements can come from Android itself (e.g., battery low) or from any program running on the system. An Activity or Service provides other applications with access to its functionality by executing an Intent Receiver, a small piece of executable code that responds to requests for data or services from other activities. The requesting (client) activity issues Intent, leaving it up to the Android framework to figure out which application should receive and act on it.

   Intents are one of the key architectural elements in Android that facilitate the creation of new applications from existing applications. Intents are useful in your application to interact with other applications and services that provide information needed by your application.

4. Content providers: provide an interface to applications to access data provided by the content provider application. Often, a content provider uses the file system or an SQLite database to store the data, but any method that's appropriate for the type of data is acceptable.

Our implementation makes use of bound service instead of started service. That is because we want to send requests and receive responses. A brief description about AIDL (Android Interface Definition language) is given below as we used with our service to manage the implementation of context manager that we will refer in the Chapter 4.

3.3 AIDL (Android Interface Definition Language)

Clients interact with the service through a programming interface called IBinder. This is required when you create service that provides binding. There are three ways to define the programming interface and one of these is the AIDL. We choose AIDL for our implementation, as we want our service to handle multiple requests simultaneously, and allow clients from different applications to access our service for IPC and to handle multithreading in our service. The role of the AIDL is to decompose objects into primitives that the operating system can understand and marshals them across processes to perform IPC. Finally, to use AIDL directly, you must create an .aidl file that defines the programming interface. The Android SDK tools use this file to generate an abstract class that implements the interface and handles IPC, which you can then extend within your service.

3.4 Context Awareness in Android

The increasingly popularity of smart phones, become a major subject. People, prefer smart phones than usual mobiles for the functionality they can offer. Smart phones are ideal for context-aware applications. Before we decided in which platform to implement our system, we have been through a comparison about other platforms such as iPhone and Windows. Today, most popular platforms are iPhone and Android because they offer high usability, powerful CPUs and available sensors. [25]. After this research we have decided to implement our system in Android. As we have referred earlier, in spite that these operating systems provide richer, simplified development environment for mobile applications, Android, from the other hand is an open development environment built on an open source Linux kernel that offers new possibilities for mobile applications.

Further more, it provides more access to more OS functionalities, and the Android SDK is available on multiple platforms. Android provides access to a wide range of useful libraries and tools that can be used to build rich applications. For example, Android enables developers to obtain the location of the device, the user activity etc.

The context support in Android application framework consists of two main parts [25]:

- Raw context data sources: contains several packages and classes such as for the camera, Bluetooth scanning of nearby devices, sensor manager for controlling interaction with physical sensors on the Android device, geographical location, time, and sound recording. The sensor manager enables Android applications to access a wide range of sensors: accelerometer, light, magnetic field, orientation, pressure, proximity, and temperature.
- Context processing: contains functionality for processing raw context data into more useful contextual data and includes face recognition, speech recognition, text-to-speech, location proximity, and a Google Maps API.
- Another important thing is that separation between context acquisition and usage is very important for context-aware system architectures [27, 28]. Such separation of concern is well supported in the Android application framework through the broker architecture that provides an intent-based communication between components. The Android application framework uses a middleware infrastructure for context acquisition providing interfaces for various sensors in such a way that no data is accessed directly from the hardware. Further, access to remote context servers are supported in Android through various network APIs as well as specific APIs such as for Google Maps.

Below we have referred to some of the available sensors in android and also about the android sensor model that we can use to access these sensors and acquire raw context data sources.

3.4.1 Android sensors

Android sensors are an integral part of the Android powered devices. Android offers support for various sensors that are used to measure motion, orientation, and environmental conditions [21]. A brief description is given below of the three sensor categories:

- Motion Sensor [22]: this category includes accelerometers, gravity sensors, gyroscopes, and rotation sensors. Using this type of sensor you can monitor the device movement such as shake, rotation etc. These sensors are not used to monitor the device position. In order to achieve this functionality, other sensors can use them to determine the device position. As a result, returns a multi-dimensional array of sensors values for each SensorEvent.
- Environment Sensors [23]: this category includes barometers, photometers, and thermometers. Using this category of sensors you can monitor the monitor relative ambient humidity, luminance, ambient pressure, and ambient temperature near an Android-powered device. As a result, returns a single sensor value for each data event.
- Position sensors [24]: this category includes orientation sensors and magnetometers. Using this type of sensor you can monitor the physical position

of a device. As a result, returns a multi-dimensional array of sensors values for each SensorEvent.

3.4.2 Android Sensor Framework

In order to access these sensors and acquire raw sensor data you can use the Android sensor framework. Using the android sensors framework we can determine which sensors are available on device, determine an individual sensor's capabilities, acquire raw sensor data and register and unregister sensor event listeners that monitor sensor changes

The sensor framework is a part of the android.hardware package and includes the following classes and interfaces: [29]

- android.hardware.SensorManager: a class that permits access to the sensors available within the Android platform. Not ever Android-equipped device will support all of the sensors in the SensorManager, though its exciting to see about the possibilities.
- android.hardware.SensorListener: an interface implemented by a class that wants to receive updates to sensor values as they change in real time. An application implements the interface to monitor one ore more sensors available in the hardware.
- android.hardware.Event: a class that creates an instance of a specific sensor. It lets you determine various sensor's capabilities.

- android.hardware.SensorEvent: this class provides information about sensor event usch as the raw sensor data, the type of sensor that generated the event, the accuracy of the data, and the timestamp for the event.

To interact with a sensor, an application must register to listen for activity related to one or more sensors.

3.5 Conclusion

This chapter has introduced the Android platform, which is based on the Linux kernel. Linux kernel provides huge power and power for Android. We chose Android as the primary mobile platform target for our middleware, because it supports Java programming and provides many powerful APIs and libraries for location-awareness, GUI development, and access to Google Maps. Also, using an open source foundation unleashes the capabilities of numerous talented individuals and components to move the platform forward. Also, Android gives developers a special opportunity to write mobile applications that change the way people use their phones rather than writing small-screen versions of software that can be run on low-power devices. This is very important in the world of mobile devices where the products change so quickly.

Chapter 4

An implementation of Context-Aware Application using the presented middleware system running on Android

This chapter presents the implementation of context-aware application using the presented middleware system running on Android. As we have mentioned in previous chapter, the middleware system sits between the operating system and the application, and it offers context information in a simple way. The presented framework is a solution for developers of context-aware applications to use ready-made components for realizing the context sensing part of their applications. Instead of embedding such code in their own apps, developers will be able to utilize the proposed middleware and have access to the equivalent functionality. Furthermore by separating the role of context providers and context consumers; this middleware system facilitates code reuse with the notion of context plug-ins.

4.1 Introduction

As we have mentioned before, the presented system aims for mobile computing. Despite the existence of such smart phones that are endowed with various hardware devices that can be used as context sources; there is still a need of a direct, uniform way to access the information. The main scope of this system is to help both developers at design-time and users at run-time.

A number of requirements specified in chapter 2, that guide us to the implementation phase as well as for the evaluation as you will see in chapter 5. The main requirement of the system is to provide application specific access to context information. More requirements were analyzed that also guide us to the implementation of the system such as interoperability, adoption etc.

4.2 The context-aware middleware system framework

Our platform consists of two levels, which represent the main components of our system; that is a Context-aware Application (Client), and the ContextMiddleware. Figure 4.1 presents these two levels and shows the general interaction between the middleware system and a context-aware application.

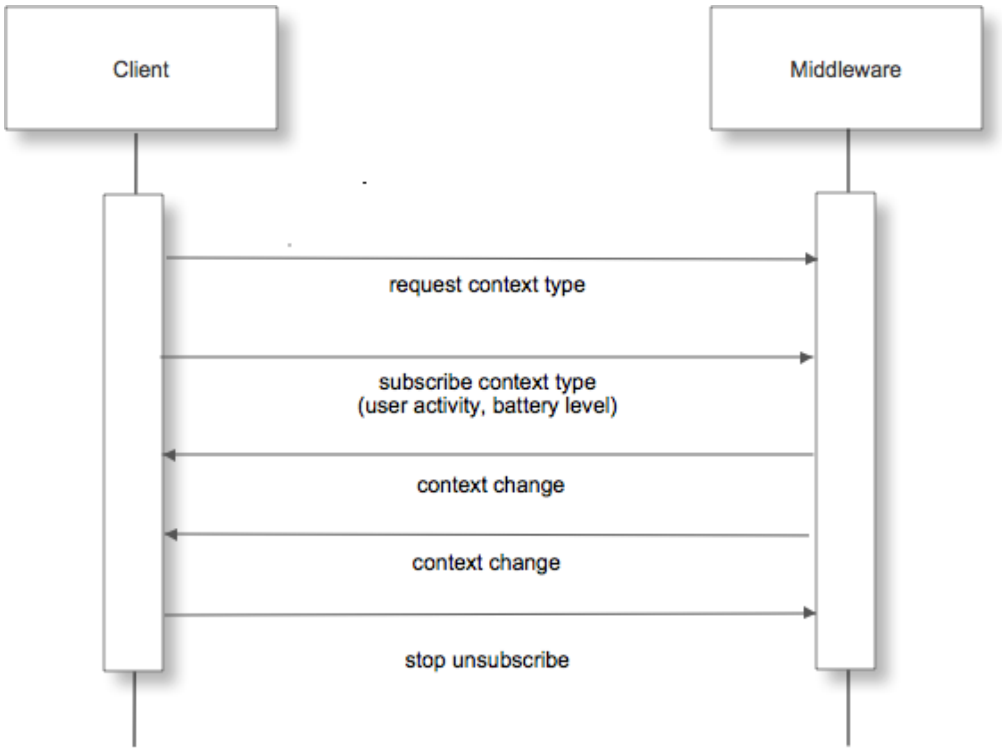


Figure 4.2: The contextualization platform

1. Context-aware application: the application as showed in Figure 4.2 is our client. The role of the application is to request for a context type that is for example; user activity or battery and to be notified about the state of each context type (e.g., stopped, started, battery low,) of each context type has change. When the request is received, it is responsible to subscribe or unsubscribe the context types that the client requests for. Context types are responsible for gathering data, which will help to define the context.
2. After gathering data from context type then the context manager that is our middleware takes place. Its role it to gather the information of context types. The developed middleware offer a centralized and uniform way for accessing a dynamically changing set of context types. For example, by adding new context-sensor plug-ins, the middleware will be enriched with additional context sensing capabilities, relieving individual developers from having to develop and mesh such code in the apps.
3. When the context manager gathers the information it needs, it will contact again the context-aware application and it will notify it about the state of context type and its behaviour.

In the end of this thesis we present details about an implemented context-aware application called CaMPlayer that we have designed to evaluate the Context-aware Middleware.

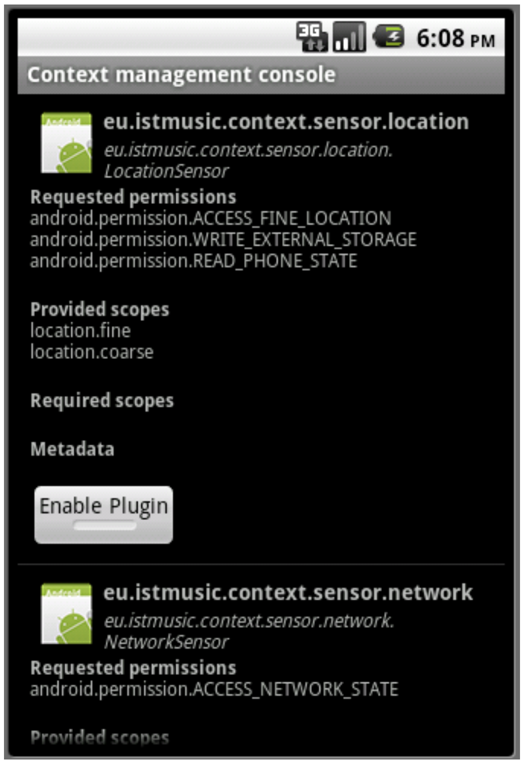
4.3 Implementation of Context-aware Middleware system framework

This implementation is closely coupled to the notion of context plug-ins and the context manager that is the middleware that collects the generated data from the plug-ins and processes, stores and provides it to context listeners.

Our implementation consists of the following projects:

1. Library: It is not installed as a separate application. It is included in the applications as a library project.
2. Middleware: is the main project and has no User Interface. It is a service and thus runs in the background acting as we have referred in the previous section.
3. ControlPanel: is an optional application, which allows visualization of the individual context plug-ins, which are dynamically installed or removed and deployed to provide the necessary context types (e.g., a location plug-in which uses an underlying GPS sensor, a Wi-Fi plug-in that uses the internet connection). It provides a simple User Interface to view the plug-ins. It connects with the middleware at runtime, and when connected it displays the installed plug-ins that lets us to enable and disable them accordingly. Figure 4.3 shows the Control panel with the plug-ins.
4. battery_plugin: It is a simple context sensor with battery values. It illustrates how to build context sensors using the battery broadcast receiver.
5. Location_plugin: It is a simple context sensor with location values. It illustrates how to build context sensors using the location service.
6. User-activity_plugin: It is a simple context sensor with user-activity values. It illustrates how to build context values giving the activity of the user.

7. Network_plugin: It is a simple context sensor with Internet connection values. It illustrates how to build context sensors using the network service.



eu.istmusic.context.sensor.location
eu.istmusic.context.sensor.location.LocationSensor
Requested permissions
android.permission.ACCESS_FINE_LOCATION
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.READ_PHONE_STATE

Provided scopes
location.fine
location.coarse

Required scopes

Metadata

eu.istmusic.context.sensor.network
eu.istmusic.context.sensor.network.NetworkSensor
Requested permissions
android.permission.ACCESS_NETWORK_STATE

Provided scopes

Figure 4.3: Installed plug-ins

4.3.1 Developing context-aware applications using the presented middleware

This section provides a method for designing and implementing context-aware applications using the presented middleware system.

- The first step is to identify the relevant context types. Referring to our system, we have four context types (battery, location, user-activity and network connectivity).
- Secondly, a context pug-in should be defined (such as battery, location, user-activity and Wi-Fi plug-ins accordingly) for each context type.

- Then, implement your functional requirements for your application according to the relevant context types.
- And finally, register your application to the relevant context types interacting with the middleware and adapt the code accordingly. These includes:
    - A service that will refer to the IContextAccess
    - Implement the IContextListener interface and define the code that will handle the asynchronous context notifications.

4.3.2 Context plug-ins

Context plug-ins are reusable pieces of code responsible for generating context data as needed. They are the main components defined in our system. They are classified into context sensors and context reasoners. Context sensors are pure providers of context information, typically used as wrappers of physical hardware sensors (e.g Bluetooth or GPS adapters) and context reasoners are more elaborate processors that take as input elementary context data and produce higher level context information (e.g. user activity, WiFi signal). Both of them correspond to pluggable code components implementing the IContextPlugin interface. This interface specifies methods for activating and deactivating the individual plug-in components, as well as for accessing its associated metadata. The context plug-in reflects information on the context types provided and possibly required by the corresponding plug-in.

Plug-ins that are used in context-aware applications are not always active. The plug-ins are activated only when required. This led to an intelligent activation mechanism, which autonomously decides when each plug-in is activated based on its dependencies and requirements. The context information is provided by context plug-ins and consumed by context consumers (context-aware applications). To handle the dynamic availability of context plug-ins and context consumers, the context system monitors both and reacts on events involving changes to their availability. For instance, when a new context plug-in is installed or an existing one removed, the context system must react accordingly. Similarly, when new context consumers— *i.e.,* context-aware applications— are started or existing ones are stopped, the context system must also react to ensure that the appropriate context plug-ins are activated and deactivated accordingly. More accurately, the context system attempts to reconfigure the set of active plug-ins only when the needed context types change.

Context plug-ins are designed as independent components and, optionally, along with context-aware, self-adaptive applications. This is particularly useful because it can be packaged once and be reused multiple times on different devices. In our implementation we have created 4-context plug-ins. The system supports only context sensors plug-ins:

- Battery plug-in: Its role is to notify the user when the battery is low.
- Location plug-in: Its role is to obtain periodic updates of the device's geographical location and notify the user.
- User activity plug-in: Its role it to gather information for user activity. For example, the user may be sleeping, or walking etc.
- Wi-Fi plug-in: Its role is to notify the user if it has an Internet connection or not.

A brief description is given about the steps we have followed in order to developed our context plug-ins.

1. XML Declaration file: As context plug-ins are designed as independent components, each of them has its own XML file. Table 4.3.2 shows the

declaration of battery plug-in in the XML file. Observing the table 4.3.2, each plug-in must specify exactly one category property, in order to allow the middleware to pick the right one. The categories for each one of the plug-in we have created are unique and match the service ID.

```
   <application android:label="@string/app_name">
           <service android:name=".BatterySensor"
android:exported="true">
               <intent-filter>
                   <!-- These are the interfaces supported by the
service, which you can bind to. -->
                   <action
android:name="eu.istmusic.middleware.context.SELECT_CONTEXT_PLUGI
N" />

                   <category
android:name="eu.istmusic.context.sensor.battery.BatterySensor"
/>
               </intent-filter>
               <!-- The metadata can be used in later versions to
allow for cleverly selecting the most appropriate plugin -->
               <meta-data android:name="provided.1"
android:value="battery.scale"/>
               <meta-data android:name="provided.2"
android:value="battery.level"/>
               <meta-data android:name="provided.3"
android:value="battery.voltage"/>
               <meta-data android:name="provided.4"
android:value="battery.temp"/>
           </service>
       </application>
```

Table 4.3.1: XML file for Battery Sensor

2. Extend a Sensor Service or Reasoner Service.
3. Implement activate and deactivate methods to start and stop the generation of context events. We have created a mechanism to enable and disable the context-plug is that the system will make use of.

4. And finally and more important, is the implementation of ContextChanged() method. This method is used to receive events and analyzes to infer whether the user activity (or absense) should trigger an event.

4.4 Context-aware media player application (CaMPlayer) showcasing the presented context-aware Middleware architecture

CaMPlayer is a simple context-aware application, which we have implemented in order to test the presented middleware system. The use of CaMPlayer is to play songs from sd card or streaming according to the internet connection while exhibiting the following context-aware behaviour:

- When the battery of the user device is low then the music stops and notifies the user that “Battery level low: Stopping music player”.
- When the user, changes location then the player changes from sd card to streaming or otherwise and notifies the user that “Location changed: Playing from sd card now” or “Location changed: Playing from stream now”.
- When the device has Internet connectivity, the player will play songs from stream and notifies the user that “Network changed: Playing from stream now” and when the connection lost it plays from sd card and notifies the “Network changed: Playing from sd card now”.
- When the user is eating, or sleeping then the music stops and notify the user “User-status changed: Stopping playing music”, or when the user is walking, running then the music starts and notify the user that “User-status changed: Starting playing music”.

User can see notification in notification bar instead viewing that from inside the application.

4.4.1 Main Functional aspects of CaMPlayer

Figure 4.4.1, provides the main functional aspects of CaMPlayer. It consists of next and previous buttons, play and pause, seek bar that shows the duration of the selected audio and a button that we can change manually to play either from sd card or streaming. When the button is green then the player plays songs from sd card and the title of the selected song is shown in the screen. Otherwise, when the button is not selected it plays from streaming showing the title of the song. Songs titles are showed on the top of the application.

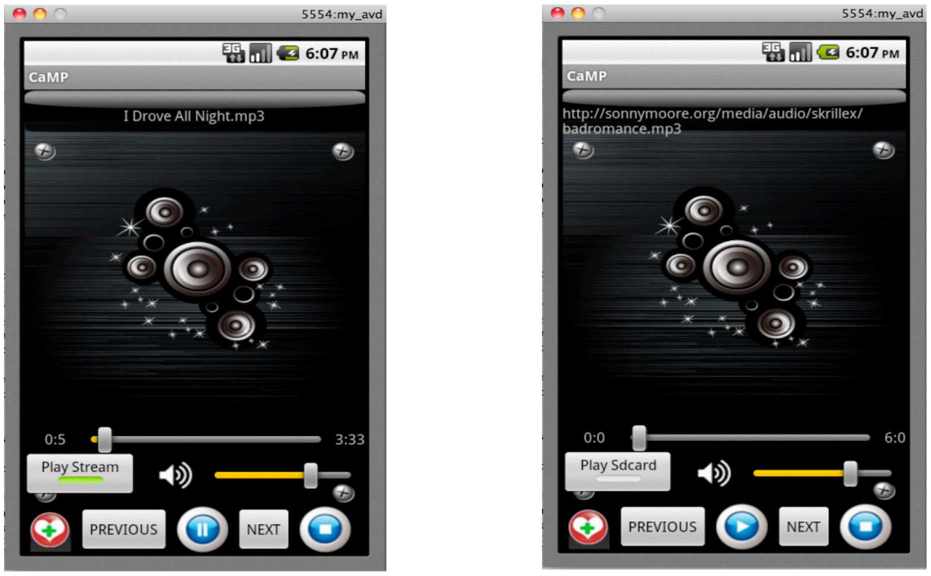
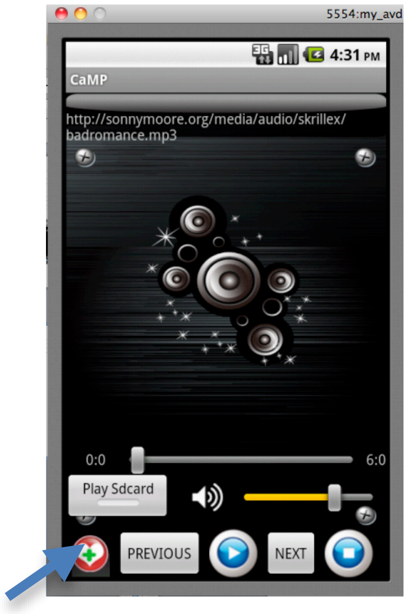


Figure 4.4.1.1: Functional aspects of the application

Figure 4.4.1.2 shows the option that our application provides. It is an option that allows you to add favourite songs. When you click the favourite button while a song is playing it adds it automatically in the favourite song lists. Then when you click on your menu button of your device, you have the option to start playing from your favourite list, sd card, streaming or to exit the application.



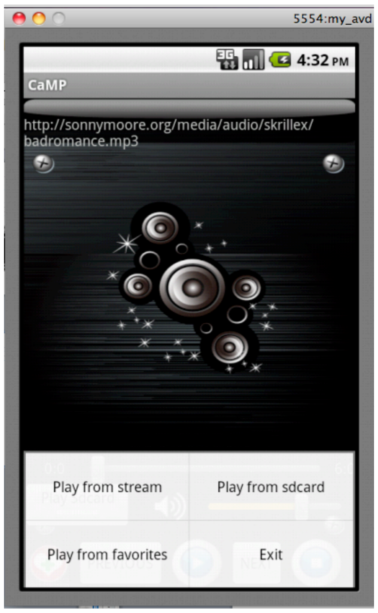


Figure 4.4.1.2: Using the favourite button functionality

4.4.2 Register CaMPLayer application to the relevant context types and implements the code that adapts the application accordingly.

Registering an application to the relevant context types need to make the application interact with the presented middleware system. To do this, we have defined a service descriptor with a reference to the IContextAccess service and implement appropriate binding methods. Table 4.4.2.1 shows the declaration of IContextAccess in the XML manifest file.

```
    <service
android:name="eu.istmusic.middleware.context.ContextService">
                <intent-filter>
                <!-- These are the interfaces supported by the
service, which you can bind to. -->
                <action
android:name="eu.istmusic.middleware.IContextAccess" />
                <action
android:name="eu.istmusic.middleware.IContextManagement" />
            </intent-filter>
        </service>
```

Table 4.4.2.1: IContextAccess Declaration

The last step is to develop the “IContextListener” interface and define the code that will handle the asynchronous context notifications. The most interesting part here is the implementation of the *contextChanged* method, defined in the *IContextListener* interface. Table 4.4.2.2 shows the implementation of contextChanged() method in CaMPlayer.

```
public void onContextValueChanged(ContextValue contextValue) {
           // TODO Auto-generated method stub

if(contextValue.getScope().equalsIgnoreCase(BatterySensor.SCOPE_B
ATTERY_LEVEL)){
        int l = (Integer) contextValue.getValue();
             if(l<60){

     if(!notificationHelper.isExistNotification(NOTIFYFORBATTERY)
){

     notificationHelper.createNotificationForDifferentBehavior(NO
TIFYFORBATTERY, "Battery level low");
                }
                notificationHelper.updateNotification("Battery
level low: Stopping music player", NOTIFYFORBATTERY);
                player.stopMusic();
             }else{
                message.setText(contextValue.toString());
             }
        }else
if(contextValue.getScope().equalsIgnoreCase(LocationSensor.SCOPE_
LOCATION_COARSE)||contextValue.getScope().equalsIgnoreCase(Locati
```

```
onSensor.SCOPE_LOCATION_FINE)){
           Coordinates coord = (Coordinates)
contextValue.getValue();

     if(!notificationHelper.isExistNotification(NOTIFYFORLOCATION
)){

     notificationHelper.createNotificationForDifferentBehavior(NO
TIFYFORLOCATION, "Location changed");
           }
;
           if(coord.getLatitude()>0 && coord.getLongitude()>0){
                if(category!=1){
                     playStream();
                notificationHelper.updateNotification("Location
changed: playing from stream now", NOTIFYFORLOCATION);
                }
           }else{
                if(category!=2){
                     playSdcard();

     notificationHelper.updateNotification("Location changed:
playing from sdcard now", NOTIFYFORLOCATION);
                }
           }
        }else
if(contextValue.getScope().equalsIgnoreCase(NetworkSensor.SCOPE_N
ETWORK_STATE)){
           Boolean networkConnection = (Boolean)
contextValue.getValue();

     if(!notificationHelper.isExistNotification(NOTIFYFORNETWORK)
){

     notificationHelper.createNotificationForDifferentBehavior(NO
TIFYFORNETWORK, "Network changed");
           }
           if(!networkConnection){
                if(category!=2){
                     playSdcard();

     notificationHelper.updateNotification("Network changed:
playing from sdcard now", NOTIFYFORNETWORK);
                }
           }else{
                if(category!=1){
                     playStream();

     notificationHelper.updateNotification("Network changed:
playing from stream now", NOTIFYFORNETWORK);
                }
           }
        }else
if(contextValue.getScope().equalsIgnoreCase(UserActivitySensor.SC
OPE_USER_ACTIVITY_STATE)){
           String userStatus = (String) contextValue.getValue();

     if(!notificationHelper.isExistNotification(NOTIFYFORUSERACTI
VIY)){

     notificationHelper.createNotificationForDifferentBehavior(NO
TIFYFORUSERACTIVIY, "Userstatus changed");
           }

     if(userStatus.equalsIgnoreCase("sleeping")||userStatus.equal
sIgnoreCase("sleeping")||userStatus.equalsIgnoreCase("sleeping"))
{
                player.stopMusic();
                notificationHelper.updateNotification("Userstatus
changed: Stopping music player", NOTIFYFORUSERACTIVIY);
           }else{
                if(!player.isPlaying()){
                     start();

     notificationHelper.updateNotification("Userstatus changed:
Starting music player", NOTIFYFORUSERACTIVIY);
                }
           }
        }

     }
```

Table 4.4.2.2: Implementation of contextChanged() method

The contextChanged() method implements the context-aware logic. Revisiting the context-aware our application reacts with the following behavior. Figure 4.4.2.3 shows the context dependencies on CaMPlayer.

- When the battery of the user device is low then the music stops and notifies the user that “Battery level low: Stopping music player”.
- When the user, changes location then the player changes from sd card to streaming or otherwise and notifies the user that “Location changed: Playing from sd card now” or “Location changed: Playing from stream now”.
- When the device has Internet connectivity, the player will play songs from stream and notifies the user that “Network changed: Playing from stream now” and when the connection lost it plays from sd card and notifies the “Network changed: Playing from sd card now”.
- When the user is eating, or sleeping then the music stops and notify the user “User-status changed: Stopping playing music”, or when the user is walking, running then the music starts and notify the user that “User-status changed: Starting playing music”.

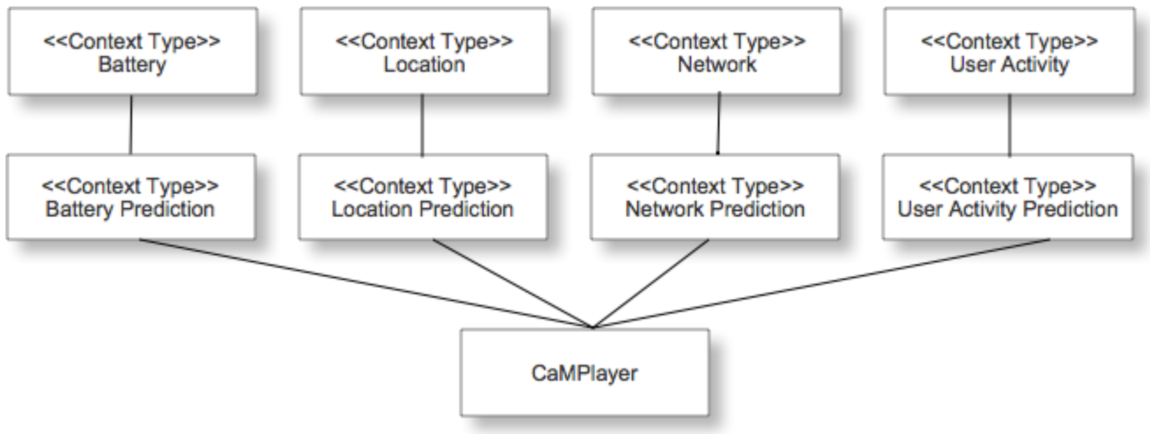
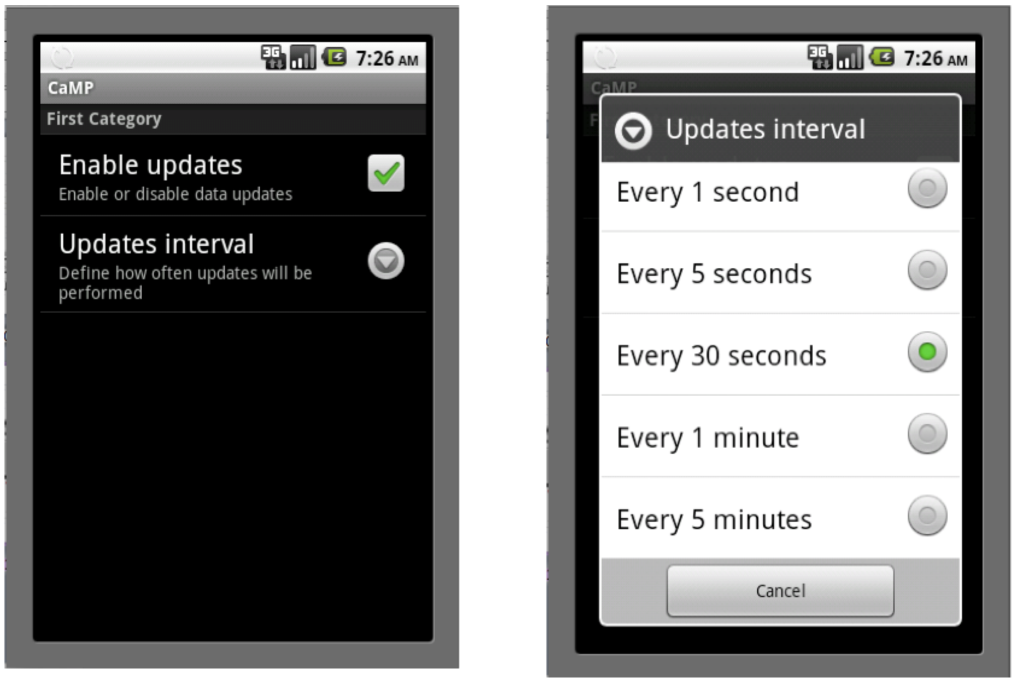

<<Context Type>> Battery
<<Context Type>> Location
<<Context Type>> Network
<<Context Type>> User Activity
<<Context Type>> Battery Prediction
<<Context Type>> Location Prediction
<<Context Type>> Network Prediction
<<Context Type>> User Activity Prediction
CaMPlayer

Figure 4.4.2.3: Context dependencies of CaMPLayer

Furthermore we have extended our context-aware application by adding an activity inside the CaMPlayer. This activity is responsible for user-activity plug-in. That means, it handles the notification for user-activity by selecting a specific time. Selecting the menu option from your device, you can select “Settings” button. You will move to a new window that it guide you to click on the preference button. In there you can see a check boxes and a drop down list. We have designed check boxes that will handle notifications for user activity automatically or manually. Manually option, uses the drop down menu that we can select how often updates of user state will be performed. Automatically option it gives us notification for user activity every time the state of the user changes. Figure 4.4.2.4 shows the menu for user activity plug in. When you click on “Update Interval” the window with the time appears and selects the time you want to get update for the user activity.


7:26 AM

CaMP

First Category

Enable updates ✓
Enable or disable data updates

Updates interval
Define how often updates will be performed

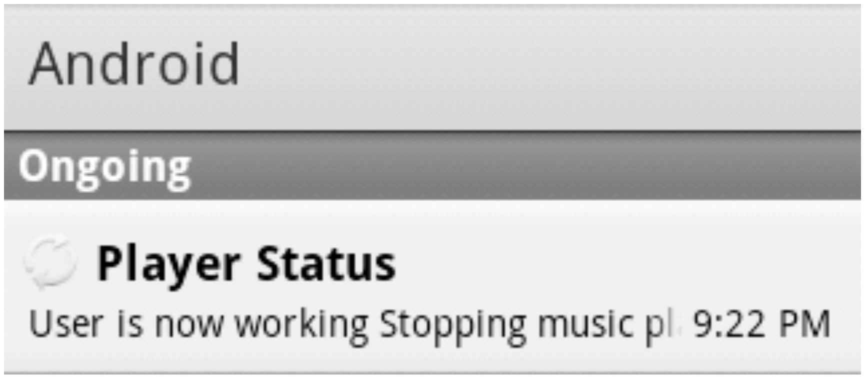
7:26 AM

Updates interval

- Every 1 second
- Every 5 seconds
- [✓ Every 30 seconds]
- Every 1 minute
- Every 5 minutes

Figure 4.4.2.4: Control of User activity plug-in

Figure 4.4.2.5 shows the application with the notification message in the notification bar system. This message refers to the user activity context plug-in and notifies the user with the following message. “User-status changed: Stopping playing music”.



Android

Ongoing

Player Status

User is now working Stopping music pl 9:22 PM

Figure 4.4.2.5: Context dependencies of CaMPLayer

4.5 Conclusion

This chapter presented the steps required building a context-aware application and context plugins based on the middleware. A brief description is given, providing the steps that developers can follow to register their application to the presented middleware system. This method it is much easier for them as they save time during design time and implementation time.

Chapter 5

Evaluation

This chapter describes the approach that was followed to evaluate the middleware system. The evaluation consists of two parts: the first part is to analyze through a specific process if the system succeeded its main goal; that is to help both developers at design-time and users at run-time; using ready-made components for realizing the context sensing part of their applications. Instead of embedding such code in their own apps, developers will be able to utilize the presented middleware and have access to the equivalent functionality. The second part is to evaluate our proposed middleware system through the analysis of the requirements identified in chapter 2, in relation to the proposed middleware system.

5.1 Component based and Monolithic based applications

The presented middleware system showcased a context-aware component based application development. Component-based middleware views an application as a composition of components. Szyperski [32] defines a component as: " a unit of composition with contractually specified and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."

In order to show that the middleware system aims to help both developers at design-time and users at run-time; we have implemented our context-aware application as a context-aware monolithic application; means views an application was developed within a single project, without an underlying middleware. That means, the functionality of middleware and context-plug ins are all in one project running in one apk. With this approach a developer would not be able to reuse existing plug-ins or the presented middleware. The only project that is separated it is the library project that is it used by the application as a library.

Having those two separated projects we run them as different programs and get some results. The following section provides the results we have got and analyses them comparing those two frameworks.

5.2 Practical Evaluation

We have used the program Source Monitor to evaluate the two projects. Source monitor "lets you see inside your software source code to find out how much code you have and to identify the relative complexity of your modules" [33]. We chose this program to test the complexity of those two projects. Table 5.2.1 shows the results of each project.

Type of Context-Aware Applications	Lines of Code	Classes	Calls	Avg. Complexity
Component Application	1, 513	43	366	2.83
Monolithic Application	3,275	69	822	3.55

Table 5.2.1: Comparison of context-aware component and context-aware monolithic application

Observing the table above, we conclude that running a context-aware component application provides less complexity than running a context-aware monolithic application. Component application provides less lines of code, fewer classes as well as fewer calls than monolithic application. This comes to imply that developers can save time during run-time and design time when creating component based applications.

Developing a context-aware component based application is much easier for developers and users. Let's say a developer wants to make an application that will notify the user for the restaurants that are close to him. The developer will only have to implement the main functional aspects of the application and make use of the proposed middleware registering the application to the relevant context types and implement the code that adapts the application accordingly. Also, the user will reuse the plug-ins that he needs for the application. From the other hand, if the developer implements this context-aware application as a monolithic block of code, he should implement from scratch the plug-ins and the middleware supporting it.

Generally component-based applications try to move away from applications implemented as a single, inflexible piece of code to more manageable, more flexible applications constructed from many smaller, reusable components. Each component is structured to represent a self- contained piece of functionality that has been designed to be generic enough to be reused by more than one application.

5.3 Requirement-driven evaluation

This section evaluates the requirements identified in section 2.2, by revisiting them and evaluating the presented middleware system against them. The following paragraphs discuss both of functional and extra-functional requirements, arguing to which extend has each requirement been addressed in the current state of the middleware architecture. In some cases, directions for improvement are also proposed.

5.3.1 Privacy of context information

Privacy of context information refers to the requirement of protecting the identity of the user, along with his private context information. As the context distribution system is treated in this thesis as an external component, the main effort is placed on protecting the privacy in the latter scenario. In this case, the main step towards user privacy protection would be the realization of a password-protected, context repository system with data encryption, preventing unauthorized access to the user's data. However, the implementation of such a system was beyond the scope of the work in this thesis.

5.3.2 Ease of building

One of the main goals of this thesis was to allow building context-aware applications in an easy, and efficient manner. For this purpose, it was decided that enabling code reuse and development with separation of concerns would greatly facilitate this goal. While it is difficult to claim that the proposed methodology supported by the middleware architecture fulfill this requirement completely, it is argued that the current indications are quite positive.

5.3.3 Code reuse

Enabling code reuse was one of the main goals of the proposed middleware architecture. Adopting a model, which treats the context providers as independent, pluggable components, greatly facilitates this goal. The developed components are treated as black-boxes, where their internal functionality is hidden and only their context offerings and context requirements are explicitly defined (as metadata). These metadata are also used for publishing the plug-ins in component repositories, further facilitating code reuse.

5.3.4 Scalability

Scalability refers to the ability of architecture to gracefully accommodate an increasing number of components, both locally and remotely. With regard to context distribution, it was already mentioned that the design of an appropriate, possibly scalable, distribution system is beyond the scope of this architecture. With regard to local scalability (i.e., in terms of the number of context provider and context consumer components), it is argued that the middleware architecture offers a bottleneck-free path for deploying context-aware applications. The resolution mechanism is triggered only when a new plug-in is installed or an existing one uninstalled, while the activation algorithm is used only when new applications are started or existing ones are stopped.

5.3.5 Dynamic behavior

Allowing new context plug-ins to be installed and activated at run-time enables dynamic behavior. As the context providers and the context consumers are only loosely coupled, it is possible to have applications replace their context-aware logic at run-time in a seamless manner. This feature is important as it allows mobile context-aware applications to take advantage of richer context information when it becomes available, and rolling back to basic context data use when it becomes unavailable.

5.3.6 Adoption of existing patterns and standards

The proposed development methodology and the middleware architecture build on existing patterns and standards when possible. For instance, the middleware architecture uses the common publish-subscribe pattern to enable asynchronous context access.

5.3.7 Ease of deployment and configuration

The deployment and configuration of context-aware applications can be challenging for non- experts, especially when specialized hardware and software is used. From the developers' perspective, the fact that the middleware and the context-aware applications are developed and packaged as java packages makes it an easier task for the end-users. From the end-user perspective, the ease of deployment and configuration remains a challenge that must be met by the individual developers of the context-aware applications, depending on the actual software and hardware required by them.

5.3.8 Context triggered action

Context triggered actions are necessary for the development of efficient context-aware logic. By adopting the classic publish-subscribe pattern, the developers can register their code for asynchronous notification of relevant context changes without having to explicitly inquire it periodically. In the proposed middleware architecture, this requirement is addressed by the context access service, which allows for asynchronous context queries. For example, the CaMP application uses the context access service to subscribe to changes in the context type corresponding to the user's activity.

5.3.9 Fault tolerance

Fault tolerance is an important feature, aiming to guarantee that the system is able to overcome faults that are limited to specific parts of the software or the hardware. In the case of distributed scenarios, faults often occur at the network level, thus requiring that the underlying mechanisms are capable of overcoming them. On the other hand, at a local level it is possible that the code implementing a context provider plug-in halts either because of a hardware problem in the underlying sensor. The middleware architecture presented in this thesis it deals with the network plug-in that appears an error in the Internet connection. Many scenarios developed to overcome this fault but none of them work properly.

5.4 Conclusion

This chapter evaluates the context-aware application, showing that developed context-aware component application it is much more easier for developers and users. Also, the requirement evaluation shows that the presented middleware as well as the context-aware application meets most of the requirements that we have referred in chapter 2.

Chapter 6

Conclusions

This chapter concludes the thesis by summarising its proposed solution and by providing a discussion of key topics for future work.

6.1 Conclusions and Future work

The primary goal of this thesis was the development of a context-aware application using the presented middleware system, where the developers of context-aware applications use ready-made components for realizing the context sensing part of their applications. With the increase of smart mobile phones, this solution becomes an important aspect for smart phones. Despite the advantages that these smart phones offer, such as portability, Internet connectivity, users now ask the necessary hardware capabilities to sense the environment.

The presented middleware was designed for Android operating systems and allow developers to utilize it according to their needs and have access to the equivalent functionality, so the implementation of context-aware application becomes easier. Context-aware applications can adapt to new context-conditions, can understand more easily the user needs, and communicate with them more efficiently.

A critical analysis of the existing work in this domain has proven that current consolations are too dependent on specific hardware component or too cantered on a specific functionality. Our middleware system offers a centralized and uniform way for accessing a dynamically changing set of context types. Provides an extensibility of adding new context sensor plug-ins and enriched with additional context sensing capabilities, relieving individual developers from having to develop and mesh such code in their apps.

The implementation of the middleware system has proven the great advantages it provides in terms of simplicity and flexibility, and the compatibility with the Android operating systems. Experiments were carried out using an emulator and, for a complete validation of the system, a real Android device was used. Furthermore, an evaluation was carried, comparing a context-aware component application that uses the middleware and context plug-ins as separated projects, and a context-aware monolithic application that includes the plug-ins and the middleware functionality into one project. We have concluded to the result that having a context-aware component application which makes use of already implemented plug-ins is much easier in the implementation of a context–aware application. Furthermore, it is much easier for developers to reuse them and make changes accordingly to their needs instead of developing them from scratch.

In the future, we aim at enabling the implementation of more context sensors plug-ins in our middleware system to be easier for developers to utilize the system instead of implementing them in their own. So more scenarios of CaMPlayer should be imagined and implemented, along with the necessary context sensors plug-ins and context types.

One more improvement is to extend our middleware system to support context reasoners. So far, our system supports only context sensors. Adding context reasoners provides more functionality to the user because, as we referred to in previous chapters they are more elaborate processors that take as input elementary context data and produce higher-level context information.

Although the concept of context-awareness is not necessarily a new one, the recent developments in mobile hardware offer to totally new possibilities for this domain. The proposed platform tries to prove this and offers a simple and general solution to access context information within applications.

References

[1] Wikipedia: Mobile Computing, Available from:

http://en.wikipedia.org/wiki/Mobile_computing#cite_note-0

[2] Koudounas, Vasilis. Iqbal, Omar. "Mobile Computing: Past, Present, and Future"

http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/vk5/report.html

[3] “Ubiquitous Computing Fundamentals”, John Krumm, ed., CRC Press, 2010.

[4] “Context-aware computing applications”, B Schilit, N Adams, R Want - Mobile Computing Systems and Applications, 1994

[5] B. Rhodes, The wearable remembrance agent: A system for augmented memory, in proceedings of the 1st International Symposium on Wearable Computers, October 1997, pp 123-128.

[6] SCHANTZ, R. E., AND SCHMIDT, D. C. Middleware for distributed systems - evolving the common structure for network-centric applications. Encyclopedia of Software Engineering (2001).

[7] Aniruddha Gokhale, Krishnakumar Balasubramanian, Arvind S. Krishna, Jaiganesh Balasubramanian, George Edwards, Gan Deng, Emre Turkay, Jeffrey Parsons, Douglas C. Schmidt “Model Driven Middleware: A New Paradigm for Developing Distributed Real-time and Embedded Systems.” Institute for Software Integrated Systems, Vanderbilt University, Campus Box 1829 Station B, Nashville, TN 37235, USA

[8] Romain Rouvoy, Paolo Barone, Yun Ding, Frank Eliassen, Svein Hallsteinsen, Jorge Lorenzo, Alessandro Mamelli, and Ulrich Scholz. “MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments”

[9] Kristian Ellebæk Kjær, "A Survey of Context-Aware Middleware," Proceeding SE'07 Proceedings of the 25th conference on IASTED International Multi-

Conference: Software Engineering, 2007.Marco Bessi and Leonardo Bruni, A survey about context-aware middleware, Italy, 2009

[10] Marco Bessi and Leonardo Bruni, A survey about context-aware middleware, Italy, June 2009

[11] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam. Middleware for distributed context-aware systems. In International Symposium on Distributed Objects and Applications (DOA, pages 846–863. Springer, 2005).

[12] Marco Bessi and Leonardo Bruni, A survey about context-aware middleware, Italy, June 2009

[13] Flavious-Stefan Manea, Contextualization platform for mobile environments, Academic dissertation, University of Bucharest, 2011

[14] L. Capra. Mobile computing middleware for context- aware applications. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 723–724, New York, NY, USA, 2002. ACM Press.

[15] L. Capra, W. Emmerich, and C. Mascolo. Carisma: context-aware reflective middleware system for mobile applications. *IEEE Transactions on Software Engineering*, 29(10):929 – 45, 2003/10/.

[16] P. Bellavista, A. Corradi, R. Montanari, and C. Stefanelli. Context-aware middleware for resource management in the wireless internet. *IEEE Transactions on Software Engineering*, 29(12):1086–1099, 2003.

[17] Android, Inc. Android Developers Guide. http://developer.android.com/guide/basics/what-is-android.html

[18] Android, Inc. Android Developers Guide. http://developer.android.com/guide/topics/fundamentals.html

[19] Skill Guru, Android Application Life cycle. http://www.skill-guru.com/blog/2011/01/13/android-activity-life-cycle/, 2011

[20] Android, Inc. Android Developers Guide. http://developer.android.com/guide/topics/fundamentals/services.html

[21] Android, Inc. Android Developers Guide. http://developer.android.com/guide/topics/sensors/index.html

[22] Android, Inc. Android Developers Guide. http://developer.android.com/guide/topics/sensors/sensors_motion.html

[23] Android, Inc. Android Developers Guide. http://developer.android.com/guide/topics/sensors/sensors_environment.html

[24] Android, Inc. Android Developers Guide. http://developer.android.com/guide/topics/sensors/sensors_position.html

[25] A. I. Wang, B. Wu, and S. K. Bakken. Camf - context-aware machine learning framework for android. In Proceedings of the International Conference on Software Engineering and Applications (SEA 2010), CA, USA, November 2010.

[26] S. P. Hall and E. Anderson, "Operating systems for mobile computing," *J. Comput. Small Coll.,* vol. 25, pp. 64- 71, 2009.

[27] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *Int. J. Ad Hoc Ubiquitous Comput.,* vol. 2, pp. 263-277, 2007.

[28] M. Miraoui, C. Tadj, and C. b. Amar, "Architectural survey of context-aware systems in pervasive computing environment," *Ubiquitous Computing and Communication Journal,* vol. 3, 2008.

[29] Android, Inc. Android Developers Guide. http://developer.android.com/reference/android/hardware/Sensor.html

[30] IST-MUSIC project, Available from: http://ist-music.berlios.de/site/

[31] Svein Hallsteinsen, Self-adapting applications for mobile users in ubiquitous computing environments, 5 January 2010

[32] Szyperski, Available from: http://en.wikipedia.org/wiki/Component-based_software_engineering

[33] Source Monitor, Available from: http://www.campwoodsw.com/sourcemonitor.html

[34] My Life with Android, Available from: http://mylifewithandroid.blogspot.com/