CodeDroid: A Framework to Develop Context-Aware Applications

Lillian B. R. de Oliveira Antonio A. F. Loureiro
Department of Computer Science
Federal University of Minas Gerais
31270-010 Belo Horizonte, MG, Brazil
{*lillys,loureiro*}*@dcc.ufmg.br*

***Abstract*—Context-aware computing enables the development of systems that adapt themselves to the context of the user, device and surrounding environment. To represent the context information associated with the application, this work proposes the use of a profile. This paper also proposes the CodeDroid framework, based on the Android platform, to help a designer in the development of context-aware mobile applications for different domains. This strategy showed to provide greater modularity and reuse of components. The CodeDroid framework also brings together the most generic services used to collect context, as discussed in the development of Places2Go, a tourism application.**

***Keywords*-context-aware computing; interface; mobile application;**

I. INTRODUCTION

The use of mobile devices is a common aspect of our lives. These devices have more features, more processing power and communication capacity. With the increased use of mobile devices new computing platforms were designed for that environment. Among them we can mention the Android platform that deservers special attention since is becoming the dominant operating system in mobile devices. Due to the success of these platforms, there is an increase in the design of context-aware applications that make use of existing information collected from sensing devices and other user-related information. To facilitate the development of applications and achieve greater productivity, it is recommended to use *frameworks* or other similar computational tools that help designers to optimize their work. For example, a single application for a mobile computing environment may have to consider dozens of portable devices with different characteristics and it can be customized to meet the requirements of different users.

Another incentive for the development of mobile applications is related to the financial aspect. Developers can sell their own applications to users around the world through virtual markets available, for example, by manufacturers. This scenario shows how important the agility of developing mobile applications is, especially, in a competitive market such as mobile computing.

Context-aware computing allows the development of systems that adapt to the context. The context may include information from the user, device and environment. Context-aware applications rely on context to provide services to users, however, not all context information is relevant to the application. In this work, we employ the concept of profile to centralize information about relevant user profile data, as described in the next section.

Currently, the development of applications for the Android platform is not a simple task. The developer must understand the architecture, the execution flow of applications and characteristics of the platform components. For example, the code of an application on Android can be developed into a single component opposing the practice of internal cohesion and low coupling. To facilitate the use of the profile and organizing the architecture of context-aware applications we propose the CodeDroid framework that allows the development of context-aware mobile applications for the Android platform.

In the literature, there are some proposals for framework for context-aware applications developed on the Android platform [2], [3]. But they do not use the concept of centralized profile as information on the context. Other proposals do not address sensing services implementations used by the generic context-aware applications.

This work aims to use the concept of profile in the design of context-aware applications. We have developed the CodeDroid framework and applied it to design an application called *Places2Go* for a tourism case study. The application was evaluated using real and emulated environments with good results demonstrating the feasibility of the proposed strategy.

This work is organized as follows. Section II discusses the concept of profile used in this work. Section III gives a brief overview of the Android platform, which was used to develop our proposed solution. Section IV presents the CodeDroid framework that was designed to help designers in the development of context-aware mobile applications for the Android platform. Section V evaluates the proposed framework in both real and simulated environments. Finally, Section VI presents our conclusion and future work.

II. PROFILE

Dey et al. [1] define context as "any information that can be used to characterize the situation of entities (i.e., a person, place or object) that are considered relevant to the interaction between a user and an application, including

 ISBN: 978-1-61208-167-0

user and the application. Context is typically the location, identity and status of people, groups and computational and physical objects". To aggregate the information relevant to context, applications can use a concept called profile in this work.

Definition (Profile) *Entity responsible for representing the context information of a user, device and environment that is relevant to the interaction with the application.*

The profile is very important for context-aware applications because it helps to decide the flow of the application execution. The application starts interacting with the profile, so the the user and sensing services do not need to be frequently involved in these interactions. The profile centralizes for information and acts as interface between application and other services.

The contents of the profile can be modified at any time and this information is fundamental to offer for more adequate services to users. Such changes can be observed in situations where the actual user moves from one location to another one or when other users and/or resources move into and/or outside the application area of interest. The dynamics of the profile makes its use more complex, however, its constant update allows to better meet the goals of both user and application.

The profile structure is the way used to express the context information and allow its exchange. In the current scenario, each application uses a particular way of structuring and modeling context, which can be completely different from another application, even if they belong to the same domain. In this way, it becomes very complicated to exchange information among those applications. Thus, it is important to standardize the context representation to make easy the communication between user and application and among applications.

The structures to represent a profile can be divided into the following categories: model marking scheme, model of key-value pairs, object-oriented model, logic-based model and ontology. This work uses the model marking scheme that allows to represent information in a robust way, without demanding too much processing to treat the information. Another advantage is that the profile defined in this way allows to reuse it in different domains, what does not happen if we use ontologies, for example.

In the literature, there are several ways to represent context such as XML (*Extensible Markup Language*), RDF (*Resource Description Framework*), RDF-Schema and OWL (*Web Ontology Language*). Due to the choice of the markup model as a way of structuring the profile, we chose XML, which is widely used for information exchange, especially in web services. XML also works as the basis for other forms of representation mentioned above. Therefore, the designer can still use other representation forms provided they are described in XML.

The profile contents incorporates information about the user's context, device and environment. The user information may describe personal characteristics (e.g., name, age, sex and education) and interests. The device context includes the device specification (e.g., model, processor, display size, networking capabilities and existing sensors) and conditions (e.g., battery level, available memory and connected network). Some of the environment characteristics that may be present in the profile are weather conditions, noise levels and traffic conditions.

This paper addresses the design of an entity called "profile" with some predefined attributes. Since the framework goal is to support different domains, we chose the more generic attributes that can be used in various context-aware applications. These attributes were chosen based on the evaluation of various context-aware applications in different scenarios. The attributes considered to be the most generic to comprise the profile are: identifier, name, email address, location and battery level. However, the designer can also define additional attributes for the profile. This situation only occurs when the application needs to define more context data beyond those already listed in the profile of the framework. In this case, the designer is responsible for defining the attribute and collecting context data to assign to the attribute information.

To build the profile we have to collect data about the context. The data collection can happen in different ways: interactively, import from an existing data source or service sensing. The interactive form depends on the user who must provide the necessary data directly. In this case, the application may stay in an idle state waiting for a user response, hindering its operation. The form of collecting by importing data implies that the profile represents data from a file to be imported by the application. This import does not guarantee that the data is updated and requires a standard format for the context representation file.

Data collection through sensing services is highly recommended. It allows to obtain real and updated data. Another feature is that most of the devices already have the necessary sensors to provide services to access the hardware. We evaluated context-aware applications in different scenarios and identified the need to use the following basic services:
– *Location Service*: obtains the user's location. The resulting information is expressed as coordinates (latitude and longitude) or a full address.
– *Weather service*: Provides weather information about a given location.
– *Energy management service*: checks the availability of the battery energy.

These services can work independently from other services. However, some of them depend on other services. For example, the weather service depends on the information provided by the location service. The framework addresses this issue by allowing a dependency hierarchy defined by

 ISBN: 978-1-61208-167-0

the designer.

Another important aspect is the frequency and service execution time. It is possible to configure each service to perform in a given period of time, saving device's resources. It is the developer's responsibility to check the need to use a service and the consumption of battery.

III. Android Platform

We used the Android platform[1] to implement the CodeDroid framework. Among the advantages of this platform we highlight the portability, because it uses its own virtual machine (*Dalvik*). It is based on open code, which allows greater flexibility in the application development. It also has a robust development environment with tools for code debugging and performance evaluation. It also provides facilities to access device features (e.g., GPS, camera, battery, network, among others). To build applications for the Android platform, we use the following components:

Activity: most activities interact with the user. Therefore, the activity is responsible for creating the window in which the graphical user interface will be built. An activity may be presented as a floating window or embedded in another activity.

Service: recommended for functionalities that need to run indefinitely in the background. It has no visual interface.

Broadcast receivers: receives and responds to notifications sent to the system.

Content providers: component responsible for storing and retrieving data and making them accessible to all applications.

In the Android architecture there is no structure and organization that determines the communication pattern between components. All component types can instantiate components of the same type or not. For example, an activity can communicate with a service, a broadcast receiver or a content provider. It is possible to create Android applications using only one activity that connects to pre-existing services in the platform. In this case all code may belong to just one class, making it less cohesive, high coupling, less readable and more complex to maintain. Making an analogy between the Android architecture and the MVC architecture model, we can say that *Activity* performs the role of the vision and control layer. The *ContentProvider* would be the model layer.

IV. CodeDroid Framework

The goal of the CodeDroid framework is to assist the development of mobile context-aware applications for the Android platform. In addition to the contributions of a *framework* (extensibility, control inversion, modularity and reuse), CodeDroid allows a user to add other features through the standardization of the application structure: better organization of the application code; adaptation of the MVC pattern to develop Android applications, greater modularity; definition of an access point to the sensing services based on the *Facade* standard; implementation of notifications as a result of context changes based on the *Observer* design pattern; definition of the "entity" for representing context; profile abstraction allowing its management in a transparent manner; and providing services for the collection of context sensing.

[1] http://www.android.com/

A. Framework Architecture

The framework architecture is depicted in Figure 1. The classes of the CodeDroid framework are detailed below.

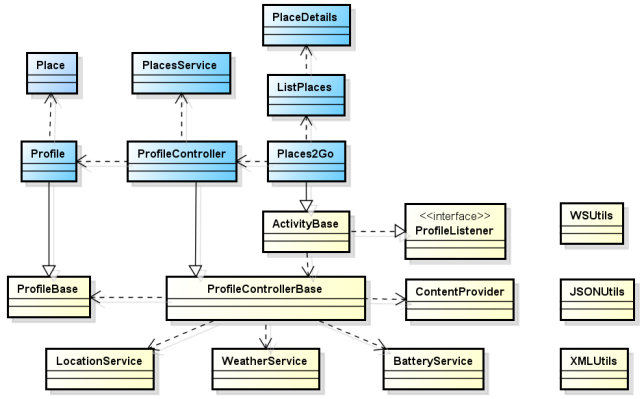

Figure 1. Class diagram of the application *Places2Go* and its relationship with the classes of CodeDroid framework.

ActivityBase: extends the class *Activity* already defined in Android. The goal is to relate all the basic settings of an *Activity* in a single class, which depends on the class *ProfileControllerBase* to use the other framework services. The application developer must define a main *Activity* for the application that extends the *ActivityBase* class. Thus, the developer can use the profile management service without knowing its configuration and take advantage of the framework.

ProfileControllerBase: aims to bring together all profile management services. It acts as the application control that receives the instructions of the *ActivityBase* class, and manages which operations will be performed. This class is also connected to the layer model that represents the profile.

The class *ProfileControlerBase* is the extension of *Service* component defined by Android. As the features already listed about this component, the class *ProfileControlerBase* can run even in the background. This class already implements the most used services in context-aware applications. The available services in the framework are: location service (*locationservices*), battery management service (*BatteryService*) and weather service (*WeatherService*).

When using the framework, the developer must create a subclass of *ProfileControllerBase*, which represents the

application control layer. It is important to notice that, besides the services available at the CodeDroid framework, the developer can add other services. The developer should preferably define each service in a class, ensuring a greater cohesion and a lower coupling. Additional services should be connected to the subclass of *ProfileControllerBase*, following the logic of the other services. This shows the flexibility and extensibility of the framework.

ProfileBase: default class that represents the entity profile. This class lists the essential attributes of the profile, i.e., those that are commonly used in context-aware applications in different scenarios. The class has the attributes: identifier, name, email address, location, and battery level.

ProfileListener: class that aims to manage the change notifications of the profile. The class *ActivityBase* implements this interface. Therefore, the main *Activity* of the application, which extends the class *ActivityBase*, must implement the methods defined in the interface. These methods will be responsible for processing events generated when the profile is modified.

For each event that occurs in pre-service implemented by the framework, the method *onProfileChanged* of the class *ProfileListener* is invoked. The developer should handle the events in the implementation of the method *onProfileChanged* of the main *Activity* application. Without this class, the developer would have to implement the listeners of all services needed by the application.

BatteryService: manages the battery level of the user's device. It captures the real information directly from the user's device. This is a subclass of the *Service* class available at Android, which runs in background. The class *ProfileControllerBase* manages this service and other sensing services as well.

LocationService: service responsible for obtaining location information from the user's GPS device. It is a subclass of the *Service* class.

WeatherService: returns the weather conditions of a particular region. This region may be the user's current location, obtained from the location service, or any other location such as the latitude and longitude coordinators or address. It is a subclass of *Service*.

WSUtils: class responsible for mediating the interaction between the CodeDroid framework and Web services, i.e., this class helps to integrate queries to Web services.

XMLUtils: class responsible for performing the processing of XML data. It interprets data in XML format that comes from the Web services and translates it to a format understood by the CodeDroid framework.

JSONUtils: class responsible for processing data in JSON format (*JavaScript Object Notation*). It interprets data in JSON format that comes from the Web services and translates it to a format understood by the CodeDroid framework.

B. Design Patterns

The framework CodeDroid has been developed based on design patterns. The standards used in architectural modeling of Android were MVC (*Model-View-Controller*), *Facade*, *Singleton* and *Observer*. The MVC architectural pattern was the starting point in the design of the CodeDroid framework. However, we found that the Android platform presents some shortcomings to use this model. The purpose of the CodeDroid framework is to organize the architecture of Android applications to follow the MVC model. The class *ActivityBase*, its subclasses and other activities represent the view layer. Another important component that is included in the view layer is the layout — XML files built from predefined tags that represent the visual components of the Android platform. The offered service by the class *ProfileControllerBase* together with their subclasses is the application control layer. The model layer is represented by the entity profile.

The class *ProfileControllerBase* was defined based on the standard *Facade*. It is responsible for intermediating the communication between clients and other services. That is, this class receives commands from an *Activity* and triggers the actions to the corresponding service.

The service *ProfileControllerBase* was also built based on the standard *Singleton*. We identified the need to define a unique service to perform profile management. The applications call this service whenever they need context information. All context-aware applications developed with the CodeDroid framework access the same service *ProfileControllerBase*. A single service ensures a better performance because it would be very difficult if, for each execution, the application would create different new services. Another advantage is that applications can take advantage of context information collected by other applications, reducing its running time and energy consumption, memory and processing.

V. Framework Evaluation

To evaluate the CodeDroid framework we developed a context aware mobile application for a tourist scenario. In tourist cities that attract many visitors, it is very common to find a tour guide. The role of the guide is to help people to know the city, drive and visite the main sightseeing, recommend interesting places and local restaurants based on the profile, help visitors locate themselves in the city, among other functions.

The application *Places2Go*, a context-aware tour guide, was developed using the CodeDroid framework. The purpose of the application is to list points of interest located near the user. This application requires context information about the environment that is not provided by the CodeDroid framework. For getting information about locations in a given region, we used the Web service "*Search Venues*"

 ISBN: 978-1-61208-167-0

by *foursquare APIv2*[2]. The class diagram of the application Places2Go is depicted in Figure 1. The application classes are described below.

Place: entity responsible for storing information about places near the user. This entity has the attributes name, address, city, telephone number, category, distance, latitude and longitude.

Profile: represents the user's application profile. It is a subclass of the *ProfileBase* class of the CodeDroid framework. As expected, this subclass inherits all the attributes already defined for that profile and adds the attribute *Places*, a list of objects that represents the user's proximity locations.

ProfileController: acts as the control application and sends the commands to the responsible services. It is a subclass of *ProfileControllerBase* of the CodeDroid framework, inheriting the provided services already defined.

Places2Go: main application class that represents the initial screen. It is a subclass of *AcitivityBase* of the CodeDroid framework responsible for initializing the necessary services for the application in addition to those already started by the framework. It displays the current user's location and sets two buttons. The first one is the "*Update location*" to update the location information and the second one is "*Places*", which looks for a list of places near the user.

PlacesService: search service that looks for interest places in the region where the user is located. This service connects the Web service of the API *foursquare* and obtains the data related to the local region. It is a subclass of the Android's *Service* class.

ListPlaces: class responsible for displaying the list of places near the user. Each location is represented as a list item. For each one of them there is an associated icon with a visual representation of the place's category. Since this is a screen, it is a subclass of the Android's *Activity* class.

PlaceDetails: class that displays the details of a site. The fields are name, address, city, telephone number, category and distance. The latitude and longitude were omitted because they are not of interest to the user. Since this is a screen, it is a subclass of the Android's *Activity* class.

The location service is activated and deactivated through the *ProfileController* class, subclass of *ProfileControllerBase*. It was not necessary to define a new location service because the methods to access the service available at the framework CodeDroid were inherited from the *ProfileControllerBase* class. The application was implemented in such a way that whenever there is a change to the user's location to a distance greater than one kilometer, the location information is updated automatically.

The application displays the user's current location, and in case it is outdated, the user can select the "*Update location*". Upon detecting the location change, the application starts representing the new user's address. Then the user can select "*Places*" to know about suggestions for places nearby and, thus, the application returns the results for the query. Due to performance reasons, the user receives at most 50 suggestions. For each result, it is returned the name of the place, the distance between the user's location and the place, and an icon that represents the type of place. In some situations, the site was not properly categorized, so icons can be inconsistent, i.e., the application cannot take the responsibility for this kind of problem. On the screens of our example, illustrated in Figure 2, the user is located at Tiradentes Plaza in Ouro Preto[3], Minas Gerais. This is a landmark of the historic city. Around it, we have the most visited tourist sites. The application *Places2Go* returns important sites such as the Museum of Inconfidência, the Church of Nossa Senhora do Carmo, House of Tales, among others.

The application was successfully evaluated in two environments: the emulator using the Android platform and using the smartphone Samsung Galaxy S that runs Android version 2.1. The Android platform provides tools for evaluating the application. Among them, we use the feature that records the CPU usage. Figure 3 shows the CPU state in four moments that occurred during the application execution. The share of the application is identified by the name of the application package "br.ufmg.dcc.mestrado". We noticed that the CPU consumption in the worst case was equal to 11.12%. The CPU consumption was relatively low, showing that the application developed by the CodeDroid framework is a viable solution to be used on mobile devices.

VI. Conclusion and Future Work

This paper presented the concept of a profile as an entity to represent context, adding the most relevant information for decision-making applications. To demonstrate the use of the profile and help the development of context-aware mobile applications, we developed the CodeDroid framework. The application *Places2Go*, developed by the framework, has the capability of being extensible, modular and allows the reuse of components. The tool eases the development of applications, reducing the need for coding the generic functionalities provided by the CodeDroid framework. The application was evaluated in both emulated and real environments, with good performance results. As future work, we can mention the extension of the CodeDroid framework to support new features (e.g., sensing devives), the evaluation of the framework in different scenarios and its portability to other mobile platforms.

References

[1] Anind K. Dey, Gregory D. Abowd, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a Better Understanding of Context and Context-Awareness. In *HUC*

[2]http://developer.foursquare.com/

[3]Ouro Preto is a world heritage city in Brazil. For further information please refer to http://whc.unesco.org/en/list/124.

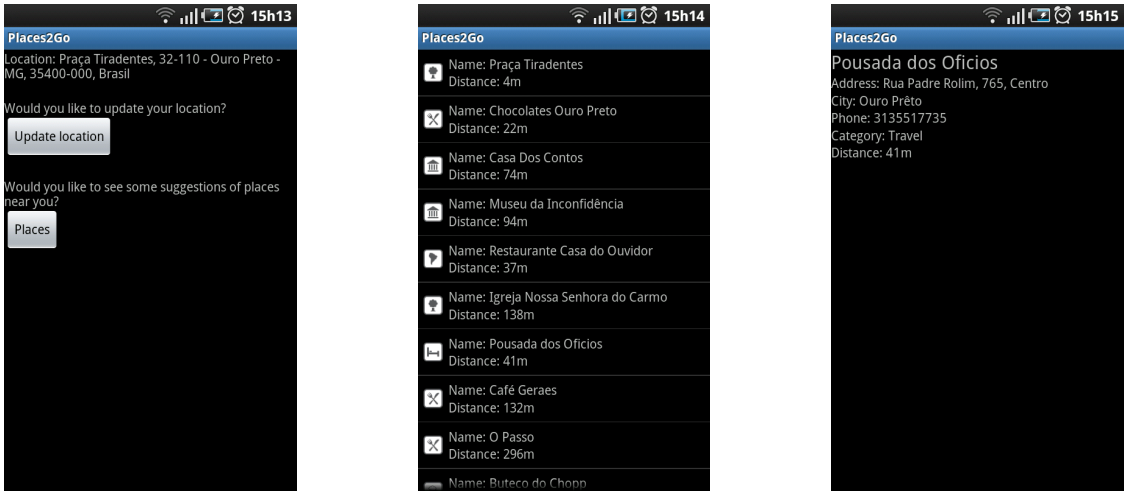

Figure 2. On the left, the application screen after clicking on *Places* (middle). On the right, the screen listing the places resulting from the action.

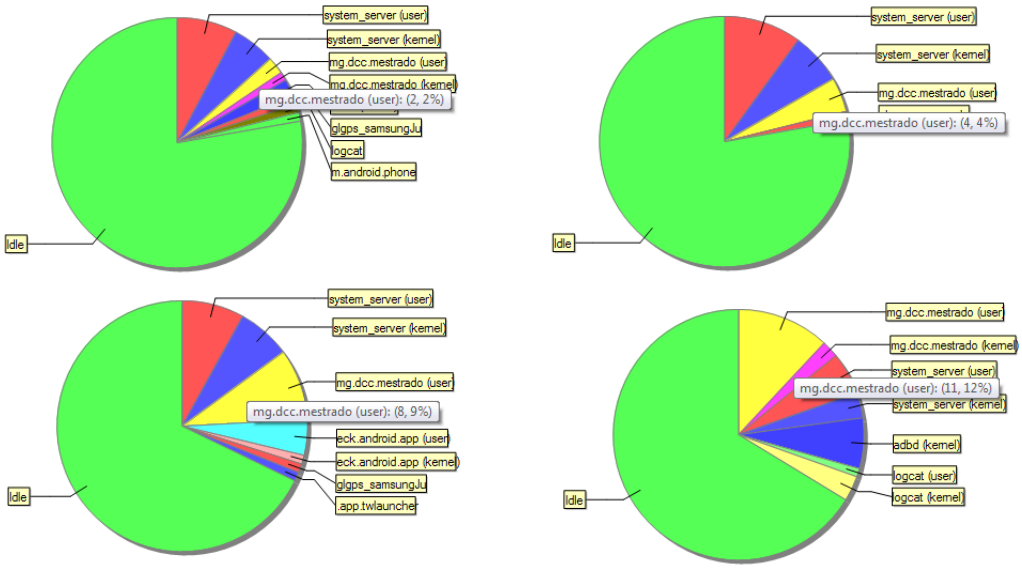

Figure 3. CPU usage of the smartphone running the application *Places2Go*.

'99: Proc. of the 1st Int'l Symp. on Handheld and Ubiquitous Computing, pages 304–307, 1999.

[2] Bart van Wissen, Nicholas Palmer, Roelof Kemp, Thilo Kielmann, and Henri Bal. ContextDroid: An Expression-Based Context Framework for Android. In *PhoneSense*, Zurich, Switzerland, 2010.

[3] Alf Inge Wang and Qadeer Khan Ahmad. CAMF – Context-Aware Machine Learning Framework for Android. In *IASTED International Conf. on Software Engineering and Applications – SEA*, Marina Del Rey, CA, USA, 2010.

 ISBN: 978-1-61208-167-0