Università di Pisa

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea

Design and implementation of an Android library for supporting network-aware applications

Relatori
Prof. Luciano Lenzini
Ing. Alessio Vecchio

Candidato
Gloria Ciavarrini

Anno Accademico 2012–2013

Abstract

In the last years, research about context-aware systems has been particularly intense. Nevertheless, most of the proposed approaches and systems failed to flow from research to the industrial world. We propose ANARC a library that eases the development of network aware applications for smartphones. ANARC does not try to cope with all the possible meanings and variations of context, it instead focuses on a specific restriction of context: the network and associated properties. To make things easier for designers and developers, ANARC adopts a rule and trigger based approach: when the network context matches the one described in a rule, the corresponding notification is sent to the application level. Examples of use of the proposed library are also included.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Over the last decade advances in digital electronics have made computers smaller, cheaper, and faster. This trend, along with other industry advances, has promoted the development and rapid market growth of small computers that can be carried from place to place. It has also created a revolution in the consumer marketplace where computers are now commonly embedded in everything from household appliances to automobiles. Whereas once it was necessary to visit special climate controlled buildings housing computer centres in order to interact with mainframe computer systems, it is now possible to carry computers and smartphones with us and to communicate on the spot with everyone. The personal nature of smartphones and the intimate relationship with their owners fuelled the adoption of these nowadays ubiquitous devices as the de-facto platform for context-aware applications. A popular definition of context is the one provided by Dey and Abowd [1]: any information that can be used to characterize the situation of an entity, where an entity can be a person, place of object that is relevant in the interaction between users and applications. Context-aware applications use it to customize services and provide a better users' experience. In many cases, context-aware systems include reasoning functionalities, to deduce new and/or higher level context information from row data. The reader is forwarded to [2] for a survey about context-aware mobile networking and additional references on such topic, whereas a general background about context modeling and reasoning techniques can be found in [3]. Despite the large amount of research carried out in the last years and the ubiquitous diffusion of smartphones as a possible implementation platform,

development of context-aware application is still not a common activity. In this area, research mostly concentrated on two directions: i) middleware systems to support the development of context-aware applications and ii) the definition of methods and techniques useful to represent and manage context-related information. The challenge of interacting with network-aware computer technology is the motivation for this research. With the new technological advances and strong move towards *Future Internet* and *Internet as a Platform* a new environment is emerging. This environment is generative, social, strongly interactive and collaborative, so users play a fundamental role in it. In this environment, context and context-awareness plays a fundamental role, as context gives meaning and accurately describes the situation of a user. Context-aware computing is a mobile computing paradigm in which applications can discover and take advantage of contextual information. Many researchers have studied this topic and built several context aware applications to demonstrate the usefulness of this technology. However, research in the field suffers from fundamental methodological weaknesses. The approaches are somewhat empirical, giving a sense that the designers already know what systems to build and what problems to overcome. Context-aware applications have never been widely available to everyday users, yet. Elements that can interact with the user and the application are distributed across the environment. With the arrival of mobile devices, context becomes a more considerable influence on the requisite behavior of computer systems. People are using mobile devices in extensively varied environments that are relatively unstable from one moment to the next. Almost all context-aware systems are made up of two components: context provider and context consumer. The first one supplies context information about users and environment while the second one makes use of this information in building context-aware applications.

The main purpose of this thesis is to design and to develop a reusable network-aware Android library called ANARC (*ANARC is a Network Aware library for Reactive Computing*). Instead of facing the multiple meanings of context, it tries to support the development of applications in a single specific domain. This should not be seen as a constraint: we aim at providing a depth analysis of network-aware mobile applications. In addition to that, we try to fill the gap between complex existing approaches and the real necessities of beginner designers and programmers of smartphone

applications because, we believe that, in some cases, these mechanisms can be too far from their necessities and that a library, which can be integrated with limited effort, can useful increase the network awareness of nowadays smartphone apps. ANARC shares with ContextPhone [4] the goal of filling the gap between OS-level functionalities and programmers' needs, with emphasis on network-awareness. Even though the primary goal of ANARC is to support the development of network reactive applications, it is important to notice that in several situations network-context can be used to gain more general information. For instance, if a smartphone is connected to a given access point during the night hours more or less regularly, this network information can be used to infer with reasonable confidence that such access point is located at user's home. Then, subsequently, when the same access point will be visible, we can infer that the user is at home and perform specific actions. ANARC supports the programmer in designing and implementing network-aware applications. In particular, it acts as a runtime component that encapsulates all the intricacies related to the detection of network events and context changes, easing the production of network reactive applications. With ANARC, the programmer specifies the context properties he is interested into in a simple way, and receives notifications about relevant changes.

The advantages of meeting these challenges can be illustrate by some examples:

Management of time-varying resources - Software should be able to tune itself so it doesn't access the network on demand, but rather takes into account the changing bandwidth and monetary costs balanced with the priority of the task at hand. News readers might download and cache many news groups, even if they may not get read, before disconnecting from a high bandwidth network.

Supporting opportunistic interaction - When system components come and go it may be necessary for applications to postpone actions until certain prerequisites are met. It is desirable to let users act as if they are fully connected even when they are not. For example, in the case of a mobile host, a high definition video should be automatically downloaded just when a high speed Internet connection is available.

Contextual customization - Mobile systems enable computer interaction

in non-traditional settings, such as house, meeting rooms, office and airports. Software customization should not only take into account a user and host, but also the broader context of use. For example, the type of meeting a person is attending, and the other people present, should help decide whether e-mail messages or telephone pages need to be immediately delivered.

Configuring dynamic systems - The particular physical devices accessible to a mobile user depends on proximity. To simplify the user's choices a user interface might use knowledge about location and accessibility to assist in the selection and presentation of printers, displays, and other devices. For example, when asking a system for a printer it should be possible to show the closest printer at the top of the selection list.

A key distinction that the reader should take away from this section is the emphasis on the mobile person who interacts with a changing multitude of people, computers, devices, and environments. This is in contrast to more traditional models that focus on individual computers, mobile or otherwise.

The thesis is structured into four more chapters. Chapter 2 gives the state of art of the existing projects on mobile, ubiquitous, location-based, smartphone-based and network-aware computing. Chapter 3 describes the design and the implementation process of the ANARCs' core and modules. It includes, also, a description on how the library works, the component interaction and a high level example of use. Chapter 4 provides an extensive explanation of the implementation of an example application to create a coverage signal map using ANARC library and the previously described modules. Chapter 5, finally, concludes this thesis and gives an overview on some of the possible future developments.

Chapter 2

Related work

The design of a library that allows applications to exploit a rich collection of information about their context of use draws from several areas of research. *Mobile computing* attempts to manage variable communication and hardware characteristics. *Ubiquitous computing* aims to augment a user's computing experience. Since location is a large part of a user's context, *Location-based computing* has to be taken into consideration. When utilizing the various internal sensors and interconnection to external devices, modern smartphones can become on-body hubs for sensor data acquisition, processing, and feedback in *smartphone-based computing*. This chapter describes work in these areas and compares their approaches to the problems of this thesis.

2.1 Context-aware Computing

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.

In order to use context effectively, we must understand both what context is and how it can be used. This step is of fundamental importance to anticipate the design challenges. The *context* is defined as ¡¡*the circum-*

stances that form the setting for an event, statement, or idea, and in terms of which it can be fully understood¿¿ in Oxford dictionary; since this is a general definition, it is not helpful. Some researchers tried to formally define context. The work that first introduces the term *context* was written by Schilit and Theimer [5]. They introduced that concept as location, collection of nearby people and objects and changes on these objects over the time. According to [6], three important aspects of context are: where you are, who 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.

We list a few concrete examples to illustrate:

- To help navigate the computerized world by providing a display of interesting located- objects, both nearby and far away.
- To keep a record of located-objects and people one has encountered, for use by applications such as "activity-based information retrieval" which uses the context at the time the data were stored to assist in retrieval [7].
- To detect location-specific information, for example, electronic messages left for the user or for public reading.
- To keep a look out for nearby devices that can be used opportunistically by applications, such as additional display terminals in a room.
- To detect nearby people, located-objects, or services that are relevant to reminders or actions set to be triggered by their presence.
- Tracking a particular located-object as it moves around a region. Examples include tracking a co-worker you wish to talk to and tracking the office coffee cart in order to be made aware when either is nearby.
- Tracking located-objects with a specified set of attributes in a particular region. An example is tracking all members of a work-group.

People's actions can often be predicted by their situation. There are certain things we do when in the library, kitchen, or office. Contextual information and commands aim to exploit this fact. Similarly, context can parameterize

"contextual commands" for example the print command might, by default, print to the nearest printer.

Context concept attempts to contain all changing circumstances and user needs. Mobile devices are context-sensitive that is they are capable to detecting the user's setting and as a result to offer this information to the application. Additionally, real world ethnographic studies are not considered important and no special effort has been put in delivering solid design methodologies for mobile applications. As a result, various issues remain to be investigated: first of all, effective and efficient positioning and context awareness methods and models. In other words, models to present useful information to the user with respect to the information communicated to him by the environment. Information should be contextualized and personalized according to personal needs, and presented to the user rather than having the user searching endlessly for useful information. This step should be achieved through a deeper understanding of the tasks in certain environments and clarification of cognitive issues related to those tasks.

Many mobile user devices are permanently connected to the Internet. The number of potential context sources (cameras, sensors, microphones, etc.) is increasing significantly. A variety of context information will be available by processing and inferring this raw information,

2.2 Managing context information in mobile devices

Mobile device users want to be able to access and manipulate information and services specific to their location, time, and environment. Context information gathered from sensors, networks, device status, user profiles, and other sources can enhance mobile applications' usability by letting them adapt to conditions that directly affect their operations. To achieve true context awareness, however, mobile systems must produce reliable information in the presence of uncertain, rapidly changing, and partially true data from multiple heterogeneous sources. Mobile devices equipped with low-cost sensing elements can recognize some aspects of context. However, extracting relevant context information by fusing data from several sensors proves challenging because noise, faulty connections, drift, miscalibration, wear and tear, humidity, and other factors degrade data acquisition. Extracted

contexts overlap, change with time, and yield only partially reliable approximations. Furthermore, mobile devices' dynamic environments require that we learn context descriptions from multidimensional data. Learning systems can't easily generalize beyond training data, however. Using even sufficiently reliable derived contexts directly to control mobile applications poses problems because users with different ideas of "context" might find application behavior irritating. To address these challenges, we present a library that provides systematic methods for acquiring and processing useful context information from a user's surroundings and giving it to applications.

2.3 Mobile Computing

Mobile computing is fundamentally about increasing our capability to physically move computing services with us. As a result, the computer becomes a taken-for-granted, ever-present device that expands our capabilities to inscribe, remember, communicate, and reason independently of the device's location. This can happen either by reducing the size of the computing devices and/or providing access to computing capacity over a broadband network through lightweight devices. In principle, this evolution has been marked by the gradual movement of computers from mainframe, personal computers and laptops to truly mobile devices like smartphones. Mobile computing enables the localization and personalization of content, always providing the user with the right information at the right time [8]. According to Lyytinen et al. [9], in mobile computing an important limitation is that computing model does not considerably change while moving. Connectivity, communication characteristics and the configuration of peripherals are examples of things that change much more frequently in mobile than in desktop systems. One goal that has motivated much of the work in mobile computing is the idea that applications run on mobile hosts should be the same as those run on desktop systems. In this view, mobility and its consequences should be made transparent to applications. An example of how systems can support application transparency under mobile conditions is Columbia's mobile internetworking work. Mobile internetworking addresses the problem of providing network access to hosts whose physical location changes over time. It exploits locality of host mobility to efficiently manage tracking and routing information. In addition it employs on-demand acqui-

sition of mobile host location information to aide in scalability. This kind of systems is a particularly good mechanism for hiding system details from applications. In cases it is also possible for services to automatically and transparently reconfigure in order to maintain a high level of service. One example is an algorithm for paging from a mobile computer into the memory of the closest paging servers [10]. One important aspect of this work is that client-server match-ups are made dynamically and vary over time.

Another dimension in making the computer invisible is the idea of pervasive computing. This concept implies the computer has the capability to obtain the information from the environment in which it is embedded and utilize it to dynamically build models of computing. The process is reciprocal: the environment can and should also become *intelligent* in that it also has a capability to detect other computing devices entering it. This mutual dependency and interaction results in a new capacity of computers to act "intelligently" upon and within the environments in which we move. This is the very idea of pervasive computing, an area populated with sensors, pads, badges, and virtual or physical models of the physical and social/cognitive environments. Pervasive computing services can be built either by embedding models of specific environments into dedicated computers or, more generally, by building generic capabilities into computers to inquire, detect, explore, and dynamically build models of their environments. Currently, the main challenge of pervasive computing is the limited scope and large effort involved to teach a computer about its environment. This makes the availability and usefulness of such services limited and highly localized because of the large effort required to design and maintain such services, thus preventing users from effectively exploiting the computing resources of their environments.

2.4 Ubiquitous Computing

Ubiquitous computing is the idea that invisible computation everywhere can enhance life in the real world. Ubiquitous computing aims to address end-user needs pertaining to various areas such as assisted living, home automation or energy management. Because it requires expertise in many fields (e.g., networking, multimedia, and systems), programming ubiquitous computing systems is very challenging. Even more challenging is the fact that

this programming must be made accessible to end-users because ubiquitous computing applications are intimately involved in our everyday life. Also, the spectrum of potential application areas requires the development process to be open-ended, enabling new entities, whether devices or components, to be integrated [11].

The proliferation of computing into the physical world promises more than the ubiquitous availability of computing infrastructure; it suggest new paradigms of interaction inspired by constant access to information and computational capabilities. The idea of ubiquitous computing first arose from contemplating the place of desktop computer in activities of everyday life. In particular, anthropological studies of work life [12] taught us that people primarily work in a world of shared situations and unexamined technological skills. However the computer was isolated and isolating from the overall situation, and fails to get out of the way of the work. The challenge was to create a new kind of relationship of people to computers, one in which the computer would have to take the lead in becoming vastly better at getting out of the way so people could just go about their lives.

It was not an easy task. This was not a graphical user interface (GUI) problem, but is a property of the whole context of usage of the machine and the affordances[1] of its physical properties: the keyboard, the weight and desktop position of screens, and so on. The problem is not one of "interface". For the same reason of context, this was not a multimedia problem, resulting from any particular deficiency in the ability to display certain kinds of real-time data or integrate them into applications. The challenge is to create a new kind of relationship of people to computers [13].

2.5 Location-based Computing

Early work on location-based applications was undertaken by Olivetti Research Lab (ORL) [14]. This research focused for the most part on the hardware design and implementation of infrared beaconing badges (called active badges) worn by individuals, and networks of infrared receivers. Unique badge identifiers sent to the stationary receivers provide location information to a software system. The main software application is an "aid for a

[1] An *affordance* is a property of an object, or an environment, which allows an individual to perform an action. For example, a knob affords twisting, and perhaps pushing, while a cord affords pulling.

telephone receptionist" showing a table of names alongside a dynamically updating location and telephone extension. The system also provides a limited set of commands such as showing which badge wearers are in the same room. Staff wearing badges can have telephone calls directed to their current location. The original ORL system did not take context into account. Badge wearers expressed a desire to control call forwarding using context information: who they are with, where they are, and the time of day. There has been some more general work on location-based systems. The system developed by Harter [15] uses a subscription based location service. Each badge transmits an infrared message periodically to some base station. The transmission interval is dynamically adapted according to light intensity. However, the previous research and the Olivetti one didn't support multiple location sources and unstable connection between devices.

2.6 Smartphone-based Computing

Mobile phones are becoming the convergent platform for personal sensing, computing, and communication. Smartphones are envisioned to provide applications and services. They integrate such diverse functionality as voice communication, audio and video playback, web browsing, short-message and email communication, media downloads, gaming and more. Mobile smartphones are personal in nature, i.e. they stay and travel together with one person most of the time, and thus enter various social contexts of that person. Therefore, nodes' movements in these networks are usually repetitive to a certain extent, and bear the social network properties of their owners. The main challenge that impacts the sensing stage is to accurately recognize the required context with a minimum number of sensors and sensing frequency. Through preprocessing, the phone's context ambiguity is resolved via a calibration process prior further processing steps. The constraints of computation and memory resources also limit the implementations of preprocessing and classification techniques to less computational intensive methods.

In view of these considerations, mobile devices can execute a wide range of application fields such as health care, entertainment, networking, etc. It's impossible to provide an exhaustive analysis of all mobile applications so, according to this thesis' goals, we will focus on some context-aware mobile applications. *TagSense* [16] identifies the individuals in a picture; it gathers

sensor readings in order to identify activities and contextual information; it optimizes the energy budget for sensing, communicating, and computing. *VEDE* [17] is a prototyping project implementing a vehicle-to-driver communication and a vehicle-to-environment communication based on a smartphone core and a wireless Bluetooth medium. The system is targeted to increase the safety level of a motorcycle. A smartphone application translates the notification from the gateway into a stream of audio data (voice synthesis). It also turns the speaker data from the helmet into a command to the gateway (speech recognition). The smartphone also manages the communication to and from the web server (remote point) according to the HTTP protocol (natively embedded in the mobile). *ContextPhone* [4] is a prototyping platform for context-aware mobile applications. The design goal and philosophy of ContextPhone is to provide context as a resource and to enable rapid application development. To fill the gap between operating system's functionalities and the needs of application developers, ContextPhone provides modules that abstract sensors, system services, and communication. ContextPhone was available for Symbian-based phones. *WhozThat* [18] is a system that ties together social networking and context-awareness with smartphones. In this case, the interaction between the physical and the virtual world is bidirectional: when a user meets other people, the devices cooperate to discover the social network IDs of the involved people, then information about users, downloaded from Facebook, LinkedIn, etc., is used to discover possible common interests. These information are also used to customize the physical context that surrounds them (e.g. to play music that is enjoyable for all of them). *Cyberguide* [19] project, a series of prototypes of a mobile, hand-held context-aware tour guide. Initially, It uses a part of the user's context, specifically location and orientation. Knowledge of the user's current location, as well as a history of past locations, is used to provide more of the kind of services that we come to expect from a real tour guide.

Mobile phones are widely used for tracing human mobility since mobile phones have almost 100% penetration, are closely tied to daily life and are capable of locating themselves using various approaches. The GPS and Wireless Positioning System (WPS) using cell tower and Wi-Fi access points (AP) are common technologies that provide a user's raw coordinates (i.e., latitude and longitude) [20]. Ambient fingerprints are often constructed

to recognize semantic places with room-level accuracy using radio beacons (e.g., cell towers, Wi-Fi APs, and Bluetooth) and surrounding factors (e.g., light, color, texture, and sound patterns) [21, 22].

As investigated by Carrol et al. [23], mobile devices derive the energy required for their operation from batteries. The battery capacity is severely restricted due to constraints on size and weight of the device. This implies that energy efficiency of these devices is very important to their usability. Hence, optimal management of power consumption of these devices is critical. At the same time, device functionality is increasing rapidly. Modern high-end mobile phones combine the functionality of a pocket-sized communication device with PC-like capabilities. The rich functionality increases the pressure on battery lifetime, and deepens the need for effective energy management. A simple choice for monitoring mobility is to periodically sense a user's location context. Such a scheme, however, significantly reduces the battery's lifetime in mobile devices. To optimize energy consumption for continuous sensing, various approaches have been proposed. These include sensor selection by movement detector using accelerometers [24–26], minimizing energy consumption within accuracy requirements [27], minimizing location error for a given energy budget [26, 28], and utilizing a prediction-based approach [29].

2.7 Network-aware Computing

Communication is a necessity for the human race; hence, with the course of time, the network, the Internet, and the high-speed networks had been invented. But as the rate of generated electronic data rose exponentially over the years, concomitant with the increase in storage capacity and network bandwidth, so did the network usage and traffic. This brought in a whole new set of challenges to be dealt with. It is quite interesting understand if we can use the same protocol and algorithms for both high-speed network and current Internet structure.

Balman et al. [30] classified the challenges in network-aware data management into increasing amounts of scientific data, need for efficient use of high-bandwidth networks, network provisioning and traffic isolation, novel data access mechanisms and models and ease-of-use and user needs.

Advanced services for optimization and tuning large-scale data move-

ment, end-to-end resource coordination and network provisioning, and novel abstraction techniques for data representation are essential in order to support the requirements of data-intensive applications these days. Since the amount of data is continuously growing, traditional techniques to manage data between distributed resources are not sufficient. Many scientific applications do poorly and fail to provide adequate use of the available bandwidth in an end-to-end context. The lack of network-aware advanced tools in the scientific community could cause ineffective use of the network.

However, during last years some interesting are emerged. The need for network-aware application placement on cloud computing infrastructures leads to *Choreo* framework [31]. As applications become more network-intensive, they can become bottlenecked by the network, even in well-provisioned clouds. Without a network-aware system for placing workloads, poor paths can be chosen while faster, more reliable paths go unused. By placing applications with the goal of minimizing the total completion time, Choreo is able to improve application-level performance. Choreo's placement also tends to place tasks that transfer large amount of data on the same machines if possible, avoiding any network transmission time, as well as avoiding slow paths in the network. The *PORTOLAN* system [32–34] is an Internet measurement system based on traceroute that aims at obtaining the Internet graph and building maps of the signal coverage through smartphone-based crowdsourcing. Basically, the system is composed by a client side, an Android application installed on several mobile phones, and a server side. The crowdsourcing encourages the best qualified and most creative participants to join in on a project and it can involve possibly millions of people in helping the scientific research. Furthermore PORTOLAN project follows a bottom-up approach where the phones are used as mobile monitors. In this approach the measurements are taken from the edge of the Internet, and not from the top.

Chapter 3

The ANARC System

3.1 Guiding Principles

We lack conceptual models and tools to support the rapid development of rich network-aware applications that might better inform the empirical investigation of interaction design and the social implications of network-aware computing. The work presented in this chapter attempts to enable a new phase of network-aware applications development. We want to help application developers understand what context is, what it can be used for, and provide concepts and practical support for the software design and construction of network-aware applications.

The ANARC system consists of a library installed on mobile devices that helps Android programmer to build quickly and easily mobile network-aware applications.

The ANARC system is designed around a small number of basic principles and assumptions:

- *Extreme portability*
  The device is designed to be carried or worn at all times.
- *Irregular connectivity.*
  The system assumes that smartphone isn't always connected to the Internet network.
- *Location reporting.*
  The device's location is always detectable.
- *Vast number of information sources*

ANARC is mainly a network-aware library, but it cannot neglect other context information, so it should exploit as many information sources as possible.

The system designers specifically avoided addressing certain issues, such as intermittent capability of geographical location detection dues to restrictive human user decision or, simply, dues to signal network absence. Restricting the system capabilities in this way let us conserve time without affecting the thesis's main goal.

The most significant advantage of assuming irregular connectivity is that the device is not obliged to use constantly a data/WiFi connection, indeed, it can do the majority of the permitted operations in offline mode. That is, instead of relying partly or entirely on remote processing the device interacts with its context and executes local elaborations. An equivalent library with a server-based processing would have been lighter at the expense of the server side which is more costly to maintain and less responsive. Furthermore, the device should be always connected and the user could incur in extra cost, due to internet data use.

3.1.1 Contextual information and function

Nowadays, mobile devices are usually used in changing environments, yet they do not adapt to those changes very well so devices are often left unaware of their surrounding environment. With the perspective of an application designer in mind, we wanted to provide a conceptual library that automatically supports all the tasks that are common across applications, requiring the designer to only provide support for the application-specific tasks. To this end, we have identified a number of requirements that the library must fulfill to enable designers to more easily deal with context. These requirements are:

Separation of concerns - One of the main reasons why context is not used more often in applications is that there is no common way to acquire and handle context. Application developers choose whichever technique is easiest to implement, at the expense of generality and reuse. We now look at a common way in which context has been handled: delegating to ANARC library the monitoring and analysis activities to hide implementation details. Ideally, we would like that

third part mobile applications are able to handle context in the same manner as they manage user input. We provide an important abstraction to enable designers to use input without worrying about how the input was collected.

Context interpretation - To support transparency, context must often be interpreted before it can be used by an application. An application may not be interested in the low-level information, and may only want to know when a high level event occurs. The interpretation, to be easily reusable by multiple applications, needs to be provided by the library. Otherwise, each application would have to reimplement the necessary implementation.

Resource Discovery - With a resource discovery mechanism, when an application is started, it could specify the type of context information required, e.g. geographical position as latitude and longitude, network information such as network operator name or mobile data connection type and so on. The mechanism would be responsible for finding any applicable components and for providing the application with ways to access them. For example, in the case of a simple "In/Out" geographical area application, rather than hard-coding the location sensing, the developer can indicate that the application is to be notified whenever any user enters or leaves the area. This information can then be provided by any source of location context.

As ANARC born as a network-aware library, it should implements and exposes the functions listed in Appendix A.3.

3.1.2 Context-Triggered Actions

One of ANARC strengths is allowing third part application freely to implement specific application actions. The library observes only what application asked and verifies when a context configuration meets a condition. The developer must specified as we will see in Sec.3.2, the string which ANARC should use in order to notify an event and to get some variable values. The programmer must make the action string unique, to do that we recommend use as prefix your package/app name, i.e., *com.example.mypackagename.myaction*. This helps avoiding a situation where other apps or system components might attempt to process it.

3.1.3 Information sources

Recall 3.1, one of our key challenge is tracking a vast number of information sources. Smartphone data can originate from many sources such as hardware sensor, online banks, social networking websites, users' behavior patterns, etc. We aim to achieve source completeness and accuracy of source information.

Smartphones have a diverse set of media capture capabilities. They not only function as a phone, but also as a camera phone, as a portable media player, and as an Internet client (email, web browsing, and data/Wi-Fi connectivity). Visual as well as audio media types can be processed and stored. In addition, smartphones often are able to process tactile information, for example, in form of a multi-touch screen that renders a virtual, rather than physical, keyboard. Based on Pitt et al. [35] mobile devices are equipped with an accelerometer, gyroscope, positioning capabilities, allowing the detection of the exact whereabouts of its owner. Geographical coordinates are obtained using a combination of GPS (Global Positioning System), the cellular infrastructure and Wi-Fi networks. Localization accuracy is typically within in the range 10 meters, most often even higher.

3.2 Rule definition language

One of our goals was to make possible for the programmer to easily communicate with the ANARC library. The *Extensible Markup Language* (XML) is a flexible way to create common information formats and share both the format and the data on the World Wide Web, intranets, and elsewhere. XML, a formal recommendation from the *World Wide Web Consortium*[1] (W3C), is similar to the natural language and contains markup symbols to describe data. This means that an XML file can be processed purely as data by a program. Such a standard way of describing information would enable a programmer to send effortlessly requests to ANARC. XML is "extensible" because the markup symbols are unlimited and self-defining.

In respect of this, we create a *XML Schema Definition*[2] (XSD) which use following tags:

[1] `http://www.w3.org`
[2] `http://www.w3schools.com/schema/`

RULELIST - it contains a group of RULE tags.

RULE - it contains one of the following combinations shown in List. 3.1, List 3.2 and List. 3.3.

WHEN - It wraps a *CONDITION* tag, a *DOIFTRUE* tag and/or a *DOIFFALSE* tag. Optionally, it can contain also an *ONCHANGE* element.

CONDITION - this tag identifies a boolean condition

DOIFTRUE - when the boolean condition specified among *CONDITION* tags become true, the ANARC's core launches the action defined among *DOIFTRUE* tags. This action will be launching just on the positive edge of transition.

DOIFFALSE - when the boolean condition specified among *CONDITION* tags become false, the ANARC's core launches the action defined among *DOIFFALSE* tags. This action will be launching just on the negative edge of transition.

ONCHANGE - It wraps a *PROPERTY* tag, a *DO* tag and, optionally, a *THRESHOLD* tag.

PROPERTY - these tags enclose the property name which the programmer wants to observes.

DO - every time the property's value changes, the ANARC core lunches the action specified among *DO* tags.

THRESHOLD - When a property belongs to a continuous range, the programmer can specify a threshold. Only when the change is greater than the threshold the ANARC core can launches the *DO* action.

PROPERTYLIST - The programmer can specify a list of property of which he wants to know the value every time the ANARC core launches an action in spite of the boolean condition status.

To be more specific, we provide the *XML Schema Definition* (XSD) in Appendix A.4.1 and three possible configuration of the tags inside a rule. In particularly, in List. 3.1 the rule specifics just a boolean condition and the actions to execute when it become true or false; otherwise, List.3.2 doesn't uses the boolean condition, it means that the developer is interested

to monitor every changes of the propriety encloses inside the *PROPERTY* tags; finally, in List. 3.3, it is provided a complete example that uses both boolean condition and on change monitoring.

```
<RULE>
    <WHEN>
      <CONDITION>
         boolean condition
      </CONDITION>
      <DOIFTRUE> action  </DOIFTRUE>
      <DOIFFALSE> action  </DOIFFALSE>
       </WHEN>
     <propertyList>
      <PROPERTY>
         property for which you want to know the value.
      </PROPERTY>
      <PROPERTY>
         another property
      </PROPERTY>
    </propertyList>
  </RULE>
```

Listing 3.1: XML Rule tags disposition using only a boolean condition

```
<RULE>
     <ONCHANGE>
      <PROPERTY>
         property to be monitored
      </PROPERTY>
      <DO> action  </DO>
    </ONCHANGE>
     <propertyList>
      <PROPERTY>
         property for which you want to know the value.
      </PROPERTY>
      <PROPERTY>
         another property
      </PROPERTY>
    </propertyList>
  </RULE>
```

Listing 3.2: XML Rule tags disposition using only an *onchange* condition

```
<RULE>
    <WHEN>
      <CONDITION>
         boolean condition
      </CONDITION>
      <DOIFTRUE> action  </DOIFTRUE>
      <DOIFFALSE> action  </DOIFFALSE>
      <ONCHANGE>
      <PROPERTY>
         property to be monitored
      </PROPERTY>
      <THRESHOLD>
         threshold value
      </THRESHOLD>
      <DO> action  </DO>
    </ONCHANGE>
       </WHEN>
     <propertyList>
      <PROPERTY>
         property for which you want to know the value.
      </PROPERTY>
      <PROPERTY>
         another property
      </PROPERTY>
    </propertyList>
  </RULE>
```

Listing 3.3: XML Rule tags disposition using both boolean condition and *onchange* condition

In order to clarify further the previous listings, we provide some rules as example. The rule in List. 3.4 permits to monitor the state of the Wi-Fi and to receive notification each time the boolean condition changes its state. List. 3.5 illustrates a rule that test if the device is using a data connection and if the signal strength is greater than 6. When that boolean condition becomes true the library start to check changes of cell ID value. ANARC notifies when the condition is not true anymore. This rule specifies, also, a set of extra key-value couples that the library must send anytime it launches an action (latitude, longitude, type of mobile connection and network operator name). Please note that it isn't specified the DOIFTRUE action.

```
<ruleList>
   <RULE>
         <WHEN>
         <!--
         Test if the wifi antenna is enabled and if the
             current connection type is equal to 'WIFI'. See
             Appendix A.3 for further function supported by the
             library and example of use.
         -->
            <CONDITION>
                  equals(hw.wifi.state, 'ENABLED') &&
                     equals(net.networkAccess.ConnectivityType,
                     'WIFI')
            </CONDITION>
            <DOIFTRUE>
                  com.example.actionTrue
            </DOIFTRUE>
            <DOIFFALSE>
                  com.example.actionFalse
            </DOIFFALSE>
         </WHEN>
   </RULE>
</ruleList>
```

Listing 3.4: XML Rule example 1

```
<ruleList>
   <RULE>
         <WHEN>
         <!--
         Test if the device is using a data connection and if
             the signal strength is greater then 6. See
             Appendix A.3 for further function supported by the
             library and example of use.
         -->
            <CONDITION>
                  equals(net.networkAccess.ConnectivityType,
                     'MOBILE') && num_gt(net.cellular.ASU, 6)
            </CONDITION>
            <!--
             When the boolean condition is true, no action
                 must be launched.
               -->
```

```
13            <DOIFFALSE>
14                 com.example.actionFalse
15            </DOIFFALSE>
16            <ONCHANGE>
17            <!--
18             When the boolean condition is true, start
                   monitoring cellID changes.
19              -->
20              <PROPERTY>
21                   net.cellular.cellID
22              </PROPERTY>
23              <DO>
24                   com.example.actionCellIDchanged
25              </DO>
26            </ONCHANGE>
27          </WHEN>
28          <!--
29          List of the additional couple key-value to return
               when the core launches an action (See Tables A.1,
               A.2, A.3, A.4  and A.5 inside Sec. A.5).
30           -->
31          <propertyList>
32            <PROPERTY>
33                   geo.position.latitude
34            </PROPERTY>
35            <PROPERTY>
36                   geo.position.longitude
37            </PROPERTY>
38            <PROPERTY>
39                   net.networkAccess.mobileConnectionType
40            </PROPERTY>
41            <PROPERTY>
42                   net.cellular.NetworkOperatorName
43            </PROPERTY>
44          </propertyList>
45    </RULE>
46 </ruleList>
```

Listing 3.5: XML Rule example 2

3.3 System architecture

The framework currently is formed by a *core module* and three modules.

In the first part of this chapter we describe the single components and their functionalities, while in the second part we illustrate in detail how these components interact to each other.

3.3.1 The Framework Core

Basically the *ANARC core* module acts as a middleware between Android applications and the framework modules. The core receives rules from one or more applications, properly parses them and then asks to the appropriate module to start monitoring and notify changes of those proprieties.

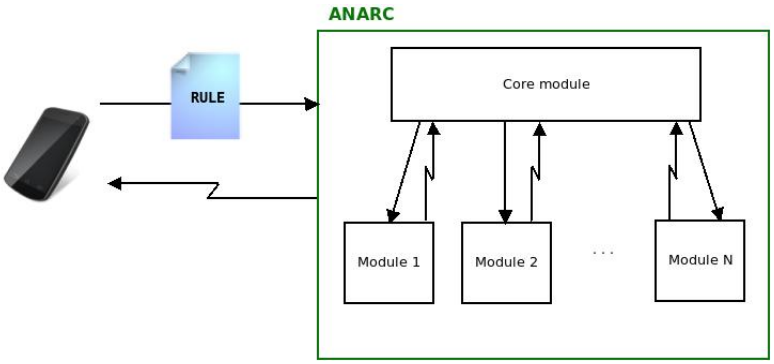


Figure 3.1: High level ANARC architecture schema

The core module is made up of classes showed in Fig. 3.2 and listed hereunder:

- Framework.java
- MultiCheck.java
- Rule.java
- PowerSaveTask.java

Android applications send one or more rules to the framework sending a broadcast intent. The Framework.java class uses, indeed, a broadcast receiver *FrameworkBroadcastReceiver* that handle intents with one of the following action, it interacts properly with the core:

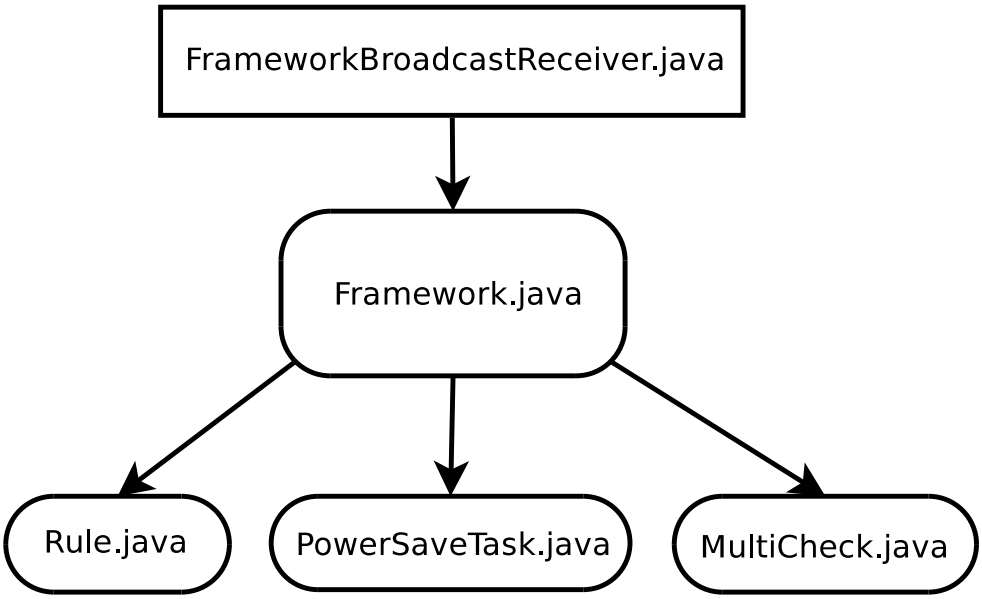


Figure 3.2: High level ANARC's core structure

- com.framework. START_FRAMEWORK
  It checks if an instance of Framework.java was previously created then send to the framework the rule list received from the Android application.
- com.framework.STOP_FRAMEWORK
  It removes all rules already submitted and asks to the core to stop itself and all the modules.
- com.framework.removeRULE
  It sends to the core the RID of the rule to remove.

When the constructor of Framework class is called, It instantiates all the modules and retrieves all the properties that can be monitored form the ANARC system. This step is necessary because it permits to check the consistency of the rules sent from third part Android application. When *FrameworkBroadcastReceiver* notifies the new rules arrive, the core parses the rule and populates a hash map that links each parsed rule to a unique *Rule IDentificator*.

Every time a property changes his value, the framework core executes some important optimizations: It creates a new thread for each property changed. Every thread will verify if the new values makes change the state

of one or more rule (from true to false or vice versa) then execute, if any, the action specified inside the rule for the positive or negative edge of the change. We should be aware that only rules that contain that property will be evaluated. Efficiency is obtained using hash maps; since it will be necessary search among all monitored properties we decided to use hash maps to optimize the search. It is a good solution because property is unique, dense in range and lie in a small range.

For evaluate expressions, we decide to use *JEVAL* an advanced library for adding high-performance, mathematical, boolean and functional expression parsing and evaluation. It gives also the possibility to use built-in function and custom function, too. The operating principles will be described in Appendix A.2.

Power consumption reduction was one of our mainly goals. The ANARC framework periodically tries to pause all unnecessary modules. A module can be paused when there no property to be monitored anymore. If a new rule will added the module is resumed.

In Appendix A.1 a more detailed analysis on core classes and methods.

3.3.2 The modules

The ANARC system must be easy to extend so we give to the programmer the possibility of built custom modules. Each module must implement the interface *ModuleInterface* that define those classes must be implemented:

- monitoredProperties
  It returns an ArrayList which contain all the property that is monitorable inside the module.
- PauseModule
  When it is invoked, this method executes the necessary instructions in order to pause the module.
- getStatusModule
  It returns a boolean value that represents the current status of the module:
  - running
  - paused

- unregister
  When it is invoked, this method executes the necessary instructions in order to stop properly the module (e.g. stop services and all monitoration activities).

As shown in Fig. 3.1, the module communicates following an asynchronous paradigm with the core. The core asks to be informed of every time a certain property changes its value. The module, on the other hand, starts to check the changes and sends a message to the core only when necessary. This scheme permits to distribute the workload among the modules without overcharging the core.

Inside ANARC framework there are three modules already connected to the core and so ready to use.

3.4 Component interaction

In this section, in light of the considerations that we have shown up to this point, we will now identify principal modalities of communication among modules, core and third part Android applications. Fig. 3.3 shows an high

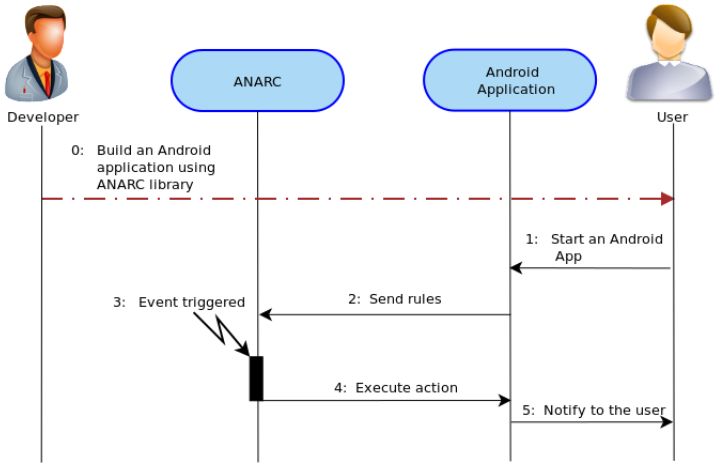


Figure 3.3: High level sequence diagram.

level sequence diagram. The Fig. 3.9 effectively explains how, over the time, they are interacting and communicating one another. In particular the previous diagram follows these steps:

1. The developer builds his own Android Application integrating ANARC library.
2. The user starts the third part Android application.
3. The third part Android application sends his rules.
4. If a rule become true or get back to false, the core send a broadcast intent to the third part application linked to the rule.
5. The Android application may notify the user.

3.4.1 Network Module

As ANARC system is a framework network-aware, the most important module is the *Network* one. It can detect changes of the properties listed in Table A.1.

As ANARC is a network-aware library, it will make an extensive use of Internet features. The *Internet Protocol* (IP) is the foremost communication protocol amongst all Internet protocol for relaying datagrams across the network. IP identifies hosts and, jointly TCP, delivers packets from a source host to a destination host. An IP address is typically a binary number expressed in a human readable dot-decimal numeric notation; in particular the:

Internet Protocol Version 4 (IPv4) addresses are commonly written using the quad-dotted notation of four decimal integers, ranging from 0 to 255 each. An Ipv4 address consists of 32 bits, which may be divided into four octets.

Internet Proptocol Version 6 (IPv6) An IPv6 address is represented as eight groups of four hexadecimal digits, each group representing 16 bits (two octets). The groups are separated by colons (:). An IPv6, in contrast to IPv4, addresses have a size of 128 bits. Therefore, IPv6 has a vastly enlarged address space compared to IPv4.

An IP address is a unique identifier given to a single device on an IP network.

On top of that, we define two important custom functions:

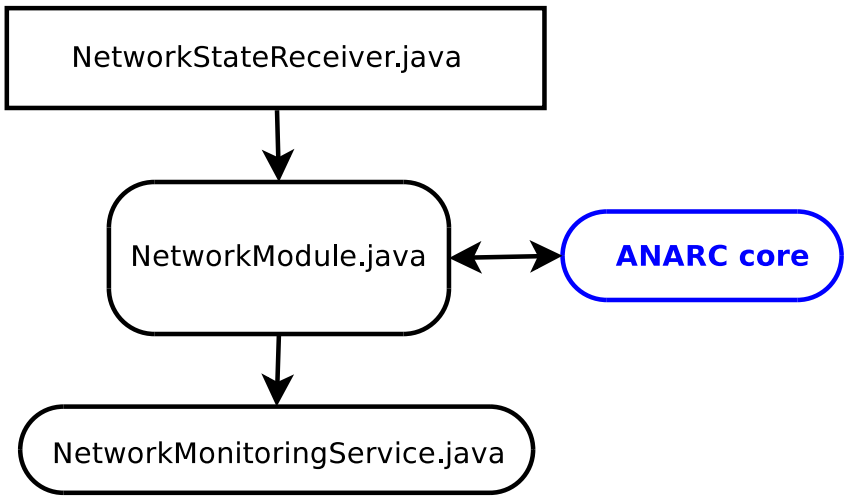


Figure 3.4: High level Network module structure

- **ipRange** function checks if *IP_to_check* belongs to a continuous IP address range or, dually, if it is between *IP_start* and *IP_end*. IP addresses need to be in dot-decimal notation and could be both IPv4 and IPv6.
- **Subnet** function checks if an IP address belongs to a given *subnet/-mask*. IP addresses need to be in dot-decimal notation and could be both IPv4 and IPv6.

The Fig. 3.4 describes the high level structure of the network module (for a more detailed structure see Appendix A.5.1). It is formed by three java classes:

- **NetworkModule**
  This is the main class of the module. It has to implements the interface *ModuleInterface* as pointed out in Sec. 3.3.2 in this way it can inherit the methods *monitoredProperties*, *PauseModule*, *ResumeModule* and *unregister*. When the default constructor is invoked, it starts the service *NetworkMonitoringService* and then it waits for messages from the broadcast receiver *NetworkStateRecevier*. This class communicates to ANARC's core, asynchronously, when a property changes its values, indeed, it receives a broadcast intent from *NetworkStatetReceiver* and then updates all variables related to network connectivity (See table in Appendix A.5.1.

- **NetworkMonitoringService**
  This service is started sticky from *NetworkModule* that means it can be explicitly started and stopped as needed from *NetworkModule*. Using a listener, the PhoneStateListener and a *TelephonyManager* object, both provided from Android API, the service is able to monitor the properties as shown in Tab. A.2. The *activator* method is invoked at the end of *onCreate* process.

- **NetworkStatetReceiver**
  This broadcast receiver receives intents from the Android system since it is registered inside the Manifest.xml for these actions: android.net.CONNECTIVITY_CHANG android.net.wifi.RSSI_CHANGED, android.net.wifi.SCAN_RESULTS.

3.4.2 Hardware Module

Hardware module permits to ANARC framework to monitor changes in hardware configuration and state.

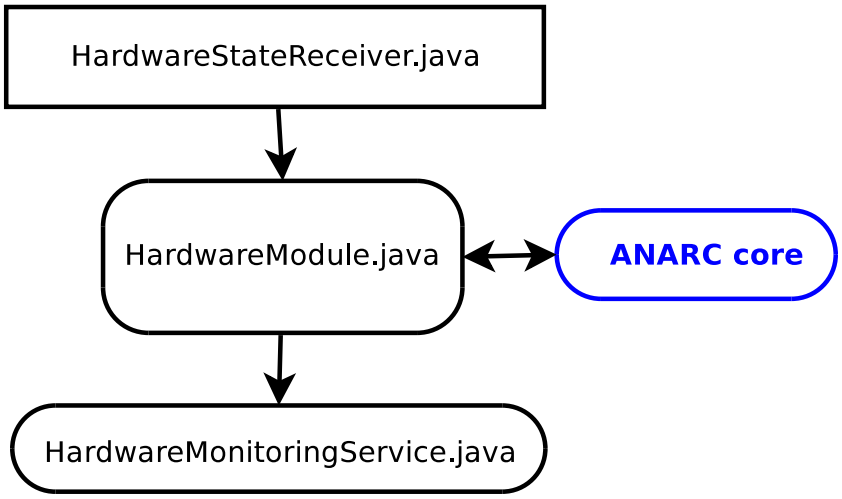


Figure 3.5: High level Hardware module structure

This module is formed by three java classes:

- **HardwareModule**
  This is the hardware module's main class. It implements the interface *ModuleInterface* as pointed out in Sec. 3.3.2 and inherits the methods *monitoredProperties*, *PauseModule*, *ResumeModule* and *unregister*.

When the default constructor is invoked, it starts the *HardwareMonitoringService* sticky service.

The methods *getHealtString*, *getPluggedString*, *getBatteryStatusString*, *getBluetoothString* and *getWifiString* are used to convert integer codes in human readable string. *checkHWChanges* checks all properties in Tab. A.3 and updates them values, if changed. This function is invokes every time the *HardwareStateReceiver* receives an intent from the system.

- **HardwareMonitoringService**
  This service is started sticky from *HardwareModule*. Using a *PhoneStateListener* and a *gpsListener* objects, both provided from Android API, the service is able to monitor the properties as shown in Tab. A.4.

- **HardwareStateReceiver**
  This broadcast receiver receives intents from the Android system since it is registered inside the Manifest.xml for these actions:
  android.net.wifi.WIFI_STATE_CHANGED,
  android.bluetooth.adapter.action.STATE_CHANGED and
  android.intent.action.BATTERY_CHANGED.

3.4.3 Geographical Module

If monitoring of geographical attributes is fundamental for a context-aware system, it is true even more for a network-aware system as ANARC.

Since Internet is a network-of-networks that consists of millions of private, public, academic, business, and government networks, of local to global scope, that are linked by a broad array of electronic, wireless and optical networking technologies is very much in the interest of ANARC analyzing these networks with regard to different geographical standpoints. Moreover, Smartphone applications made easy for users moving across Internet, collecting information and analyzing them in a real-time way.

The ANARC's geografical module structure is shown in Fig. 3.6.

The ANARC's geographical diagram class shown in Fig. A.4 shows these classes:

- GeoModule.java

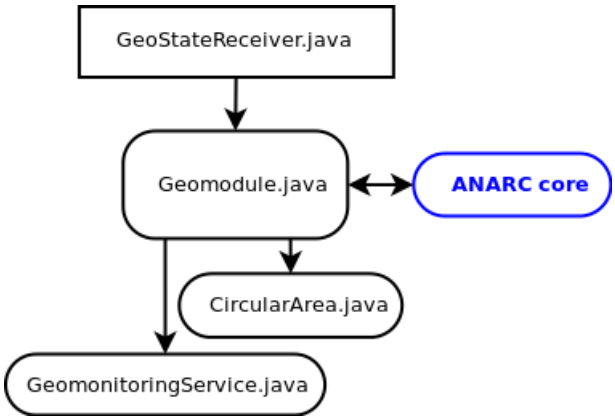


Figure 3.6: Geographical module diagram class

This is the geographical module's main class. It implements the interface *ModuleInterface* as pointed out in Sec. 3.3.2 and inherits the methods *monitoredProperties*, *PauseModule*, *ResumeModule* and *unregister*. When the default constructor is invoked, it starts the *GeoMonitoringService* service. Moreover, *GeoModule.java* implements some methods which are designed to dynamically start or remove circular target areas from the monitoring set.

- **GeoMonitoringService**
  This service is started sticky by the *GeoModule*, which means it can be explicitly started and stopped as needed by the *GeoModule*. Using jointly a LocationListener and the LocationManager provided from Android API, it updates the properties in Tab.A.5 and uses a smart update frequency choice as explained, as we'll see in due course, utilizing the *checkDistanceFromCircular* method.

- **GeoStateReceiver**
  This class communicates to ANARC's core, asynchronously, every time the Android operating system trigger a proximity alarm intent in this way ANARC core can be informed when the user cross a circular target area.

The ANARC's geographical module in addition to the control of properties (See Tab. A.5) such as position and speed adds, compared with network and hardware modules it gives an extra level of logic.

It gives the possibility to manage circular geographical areas of interest and trigger the entrance and the exit from them. The programmer musts simply specify just latitude and longitude of the center and the radius.

As previous pointed out in Section 3.3.1 the reduction of power consumption followed all the develop process. Saving power of smartphone's battery becomes important because of appearance of applications and technologies that consume more power such as GPS and Wi-Fi. Hereafter we will show a smart method to save power during position tracking, it tries to utilize GPS antenna only if the device is reasonably near to a circular area, and otherwise it utilizes network localization methods. The idea behind this service is to increase battery life by stopping power-consuming while user is very far from the border of any circular area monitored by the framework at the given time. Test conducted to evaluate the effectiveness of this service on Android Smartphones shows there is a slight increase in battery life.

Location Sampling Rate is the time interval between two device's location update. A smaller window in which you listen for location updates means less interaction location providers, thus, preserving battery life. But it also allows for fewer locations from which to choose a best estimate. In order to compensate for this situation, we focused on the following situation:

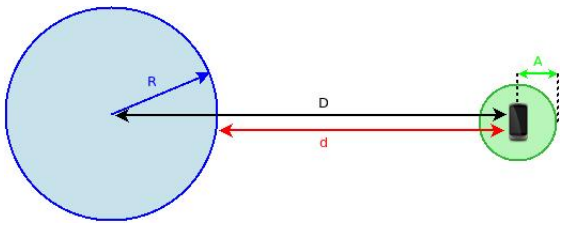


Figure 3.7: Distance from a circular area

In Fig. 3.7 A stand for location accuracy, D is the distance between the device and the center of circular area, d is the distance between the device and the target area's perimeter.

The aim is dynamically change the localization method depending on d value. Usually national air line flights minimum length is 400-500 km (depending on country), so if a user is 400 km far from all circular area and he catches a flight, he needs at least:

$$t_{lowerbound} = \frac{400km}{850\frac{km}{h}} \approx 30 \text{ minutes} \tag{3.41}$$

This also means that it is unnecessary, for at least 30 minutes, using a high accuracy localization method such as GPS. Despite the level of accuracy for different providers varies significantly depending on available cells/satellites in a geographical area, moving speed and environmental factors, it can be estimates as:

- network location provider
  - using cell location: accuracy varies from 200 meters to 2000 meters
  - using a Wi-Fi connection: estimated accuracy is $\sim$ 100 meters
- GPS location provider: from 3 meters to $\sim$ 50 meters.

$$provider = \begin{cases} \text{network} & \text{if d} \leq 2\cdot \text{ A} \\ \text{GPS} & \text{otherwise} \end{cases} \tag{3.42}$$

Bearing in mind Fig. 3.7, we create the pseudo-algorithm shown in List. 3.6.

```
1
2  if(GSP is disables){
3    foreach(CircularArea){
4          calculate d = |D − R|;
5          if(d ≃ A){
6                  turn on GPS;
7                  break;
8          }
9    }
10 }
```

Listing 3.6: Pseudo-algorithm for smart geolocation

We choice to activate GPS only when d is twice A for take in account the time required for a GPS receiver to acquire satellite signals.

To better understand the "Is inside Target Region" meaning, analyze the following diagram:

Looking at diagram in Fig.3.8, we can define that:

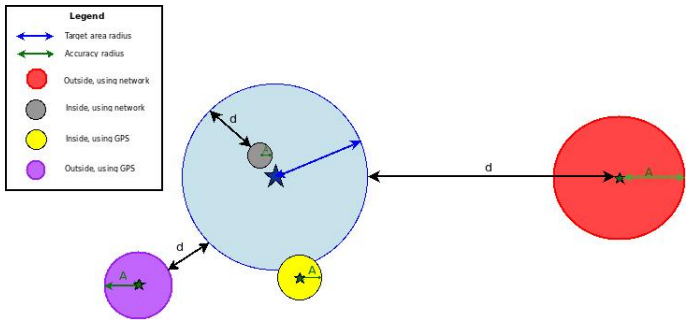


Figure 3.8: Monitored location trigger status and provider

Outside Region - Using network provider

When monitored location with accuracy radius is fully outside the target location with its proximity radius (red example above) a coarse accuracy is in line with Eq. (3.42) requirements.

Inside Region - Using GPS provider

When the current position's accuracy area intersects the target location with its proximity radius (yellow example above). According to Eq. (3.42) the use of GPS is unavoidable.

Outside Region - Using GPS provider

When monitored location with accuracy radius is fully outside the target area with its proximity radius (purple example above) but $d \ngeq 2 \cdot A$ the localization method must use GPS antenna.

Inside Region - Using network provider

When monitored location with accuracy radius is fully inside the target area with its proximity radius (grey example above) and $d \geq 2 \cdot A$ the network localization provider is able to function properly.

Due to the approximate nature of position estimation, if the device passes through the given area briefly, it is possible that no event will be fired. Similarly, an event could be fired if the device passes very close to the given area but does not actually enter it.

Android APIs define accuracy as the radius of 68% confidence. In other words, if you draw a circle centered at a location's latitude and longitude, and with a radius equal to the accuracy, then there is a 68% probability that the true location is inside the circle.

In statistical terms, it is assumed that location errors are random with a normal distribution, so the 68% confidence circle represents one standard deviation. Note that in practice, location errors do not always follow such a simple distribution.

This accuracy estimation is only concerned with horizontal accuracy, and does not indicate the accuracy of bearing, velocity or altitude if those are included in this Location.

3.5 Example of use

In this section, it will be provided a useful explanation for built your custom Android application.

Looking at Sections 3.3.1 and 3.3.2, now we can enhance the sequence diagram of Fig. 3.3. It is quite interesting understand the interaction between ANARC core and modules.

1. One or more third part Android applications send rules to the ANARC core.
2. The FrameworkBroadcastReceiver intercepts these messages and forwards them to the Framework class.
3. The Framework class parses the rules and asks to the appropriate modules to start monitoring the properties specified from the client applications.
4. When one of the monitored properties changes his values, the module sends an asynchronous message to the Framework class.
5. The Framework class asks to the evaluator to evaluate the boolean conditions (if any) which contain the changed property.
6. If necessary, the Framework class launches the action specified from the third part applications.

Following the communication schema in Fig. 3.9, an application that would use the ANARC library, inside *onCreate* method of the MainActivity must (refer to List. 3.7):

1. create an intent and set as action:

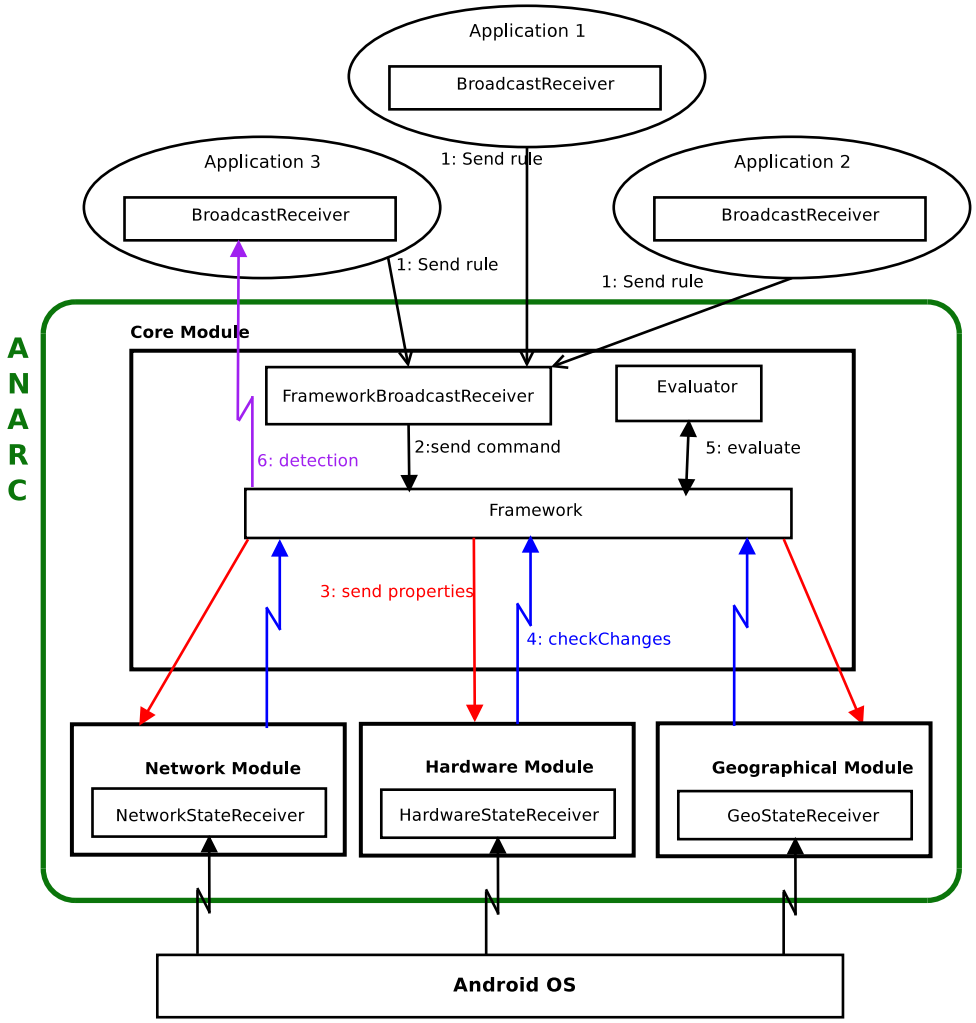


Figure 3.9: Component interaction diagram.

- *com.framework.START_FRAMEWORK* (line 44)
- or *com.framework.removeRULE* (line 65) or *com.framework.STOP_FRAMEWORK* (line 74).

2. put in the intent, as an extras, a string with one or more rule.
3. send a sendOrderedBroadcast for the *com.framework.START_FRAMEWORK* (line 55) or a sendBroadcast for the others actions.
4. create a BroadCastRecevier (lines 81-111). It is necessary to handle responses send from the ANARC library every time you send rules to it.

```
1  public class MainActivity extends Activity {
2
3  @Override
4  protected void onCreate(Bundle savedInstanceState) {
5          super.onCreate(savedInstanceState);
6
7          /*
8          * Create a string variable which contains the rules according
9          * the Rule Definition Language syntax.
10         */
11         String rules_to_send="<ruleList>
12                   <RULE>
13                     <WHEN>
14                       <CONDITION>
15                          boolean condition
16                       </CONDITION>
17                       <DOIFTRUE> action  </DOIFTRUE>
18                       <DOIFFALSE> action  </DOIFFALSE>
19                       <ONCHANGE>
20                       <PROPERTY>
21                          property to be monitored
22                       </PROPERTY>
23                       <DO> action  </DO>
24                     </ONCHANGE>
25                     </WHEN>
26                     <propertyList>
27                       <PROPERTY>
28                          property for which you want to know the value.
29                       </PROPERTY>
30                       <PROPERTY>
31                            //another property
32                       </PROPERTY>
```

```
33                  </propertyList>
34                </RULE>
35                <RULE>
36                        //another rule
37                </RULE>
38                </ruleList>";
39
40  /*
41  * Create an intent and set as action com.framework.START\_FRAMEWORK
42  * or com.framework.STOP\_FRAMEWORK or com.framework.removeRULE.
43  */
44  Intent intent_send= new Intent("com.framework.START_FRAMEWORK");
45  intent_send.putExtra("rulelist", rules_to_send);
46  /*
47  * Send an ordered broadcast, it allows you to receive data back
48  * from the broadcast. This is accomplished by supplying your
49  * own BroadcastReceiver when calling, which will be treated as a
50  * final receiver at the end of the broadcast, its
51  * onReceive(Context, Intent) method will be called with the result
52  * values collected from the other receivers.
53  */
54  AppBroadcastRCV appBroadcastRCV= new AppBroadcastRCV();
55  sendOrderedBroadcast(intent_send, null, appBroadcastRCV, null, 0 ,
        null,null);
56
57  // ......
58
59  /*
60  * Broadcast the given intent to all interested BroadcastReceivers.
61  * This call is asynchronous; it returns immediately, and you will
62  * continue executing while the receivers are run. No results are
63  * propagated from receivers and receivers can not abort the
        broadcast.
64  */
65   Intent intent_remove= new Intent("com.framework.removeRULE");
66   intent_remove.putExtra("RID", rid_to_remove);
67   sendBroadcast(intent_remove);
68
69  // ......
70   // DO SOMETHING
71   // .....
72
73   Intent intent_stop= new Intent("com.framework.STOP_FRAMEWORK");
74   sendBroadcast(intent_stop);
75
76   // .....
77
```

```
78
79  /*
80  * AppBroadcastRCV is your own BroadcastReceiver to treat as the
81  * final receiver of the broadcast.
82  */
83  public class AppBroadcastRCV extends BroadcastReceiver{
84
85  /*
86  * This method is called when the BroadcastReceiver is receiving an
87  * Intent broadcast. During this time you can use the other methods
88  * on BroadcastReceiver to view/modify the current result values.
89  * When it runs on the main thread you should never perform
90  * long-running operations in it (there is a timeout of 10 seconds
91  * that the system allows before considering the receiver to be
92  * blocked and a candidate to be killed).
93  */
94  @Override
95  public void onReceive(Context context, Intent intent) {
96
97   /*
98   * Retrieve the current assigned rule IDs as set by the
99   * previous receiver.
100  */
101  if(extras.containsKey("RIDS")){
102     ArrayList<Integer> RIDS = extras.getIntegerArrayList("RIDS");
103     if(!RIDS.isEmpty()){
104        for(int rid: RIDS){
105            Log.i("AppBroadcastRCV", "Received RID: " + rid);
106         }
107      }
108     else
109        Log.e("AppBroadcastRCV", "No RID received.");
110     }
111   }
112 }
```

Listing 3.7: Main activity structure example

3.6 Summary

This chapter has presented a network-aware library architecture consisting of two logical components: ANARC core and add-on modules. Each of these components is based on the dynamic environment surrounding the users. The core module interacts with other application installed on device and manages all modules activities. The second components in the architecture are the add-on module. They can easily build and link to the ANARC core. Three of them are already implemented: network, hardware and geographical module. One important attribute provided from a mobile deice is location. A smart algorithm helps to extend battery life. ANARC controls, also, delimited geographical areas. This chapter concluded with an explanation on how an Android programmer can develop an application which uses ANARC library.

Chapter 4

Applications

To show some possible usage of ANARC we illustrate two applications that we implemented for Android, and we describe a third application that we are planning to build. The first two applications are focused on the cellular network coverage, while the third one is an application that automatically logs in to known wireless networks that have an HTTP-based authentication. We show the rules that these applications submit to ANARC, along with some examples of execution.

4.1 Signal Coverage Map

Several applications exist on the Google Play Store, aiming at discovering and mapping the coverage of cellular networks. Examples are Portolan Network Tools [32], Root-Metrics CoverageMap [36] or NetRadar [37]. Typically all these application rely on the contribution of users, which manually run measurements that collect RSSI samples associated to GPS samples. This strategy is driven by the fact that the GPS unit usage is extremely expensive in terms of battery consumption, thus its control is left to the user. However, this can result in having areas that are oversampled and areas that are not sampled at all [38]. Moreover, in order to speed up the collection process, it would be desirable to automate the collection of samples in the areas of interest. With the help of ANARC we built a simple application that automatically collects RSSI samples associated to GPS samples when the smartphone is in one or more previously specified circular areas. Listing 4.1 contains the list of rules that the application submits to ANARC. The

rules specify that the application has to be notified when the smartphone is in the area of interest, each time that the cell identificator changes. ANARC must then return the current geographic position, cellID and ASU value. In order not to consume too much battery when monitoring the smartphone's position, ANARC implements an optimization based on a hybrid approach described in Sec. 3.4.3.

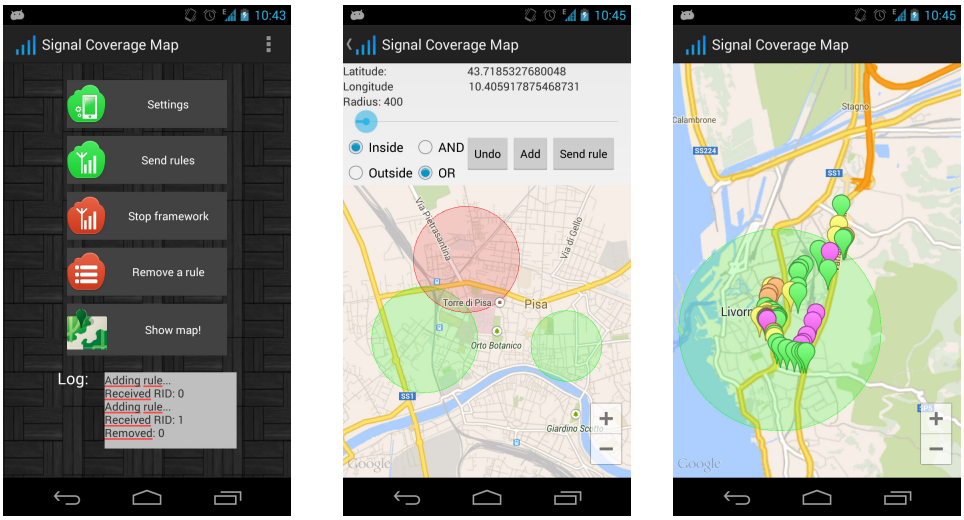
```
<!--
Refer to Sec. 3.2 for the strucuture of the rule list
-->
<ruleList>
   <RULE>
         <WHEN>
         <!--
         Test if the user is inside a circular area with
            radius 800 meters centred in Pisa or a a circular
            area with radius 3 kilometres centred in Livorno.
            See Appendix A.3 for further function supported by
            the library and example of use.
         -->
            <CONDITION>
                 CircularArea(43.721377,10.389909, 800) ||
                     CircularArea(43.5435, 10.325011, 3000)
            </CONDITION>
            <DOIFTRUE>
                 com.usingframework.RSSI.getAreaAlarmTrue
            </DOIFTRUE>
            <DOIFFALSE>
                 com.usingframework.RSSI.getAreaAlarmFalse
            </DOIFFALSE>
            <ONCHANGE>
            <!--
             When the boolean condition is true, start
                monitoring cellID changes.
              -->
              <PROPERTY>
                   net.cellular.cellID
              </PROPERTY>
              <DO>
                   com.usingframework.RSSI.getSignalASU
              </DO>
            </ONCHANGE>
```

```
30          </WHEN>
31          <!--
32          List of the additional couple key-value to return
                when the core launches an action (See Tables A.1,
                A.2, A.3, A.4  and A.5 inside Sec. A.5).
33            -->
34          <propertyList>
35             <PROPERTY>
36                     geo.position.latitude
37             </PROPERTY>
38             <PROPERTY>
39                     geo.position.longitude
40             </PROPERTY>
41             <PROPERTY>
42                     geo.position.accurancy
43             </PROPERTY>
44             <PROPERTY>
45                     net.signal.ASU
46             </PROPERTY>
47             <PROPERTY>
48                     net.cellular.NetworkOperatorName
49             </PROPERTY>
50          </propertyList>
51    </RULE>
52 </ruleList>
```

Listing 4.1: Signal Coverage Map rule

The main activity (Fig. 4.1a) allows the user to send rules to the framework, remove a rule previously submitted, show the coverage map and edit the circular target areas. The user, indeed, can decide how many areas must be or not monitored and the logical operator among them (Fig. 4.1b). Tapping on "Remove rule" button, the user can select which rule ANARC library will not monitor anymore. Selecting the "Show map" button, the application show the current coverage map (Fig. 4.1c). Each sample when tapped shows some useful information (cellID, ASU, Network operator name).

When the application receives am advice, it sends all the collected data to a remote database. It could be useful for infer some a posteriori statistic or information.



(a) Main activity (b) Areas editor (c) Sample map

Figure 4.1: Screenshots

4.1.1 MainActivity.java

In order to provide an additional example, in List. 4.2 we show how we customized the general structure in List. 3.7 aim to reach our purposes.

The main activity of Coverage Map App contains a set of buttons as show in Fig. 4.1a, so it is necessary to use an OnClickListener (referring to List. 4.2 lines 15-23, 25-31, 33-42, 44-56 and 58-68); in that way, the buttons can react when user clicks on them.

```
1  public class MainActivityTest extends Activity {
2
3    //    .......
4
5    String rules_to_send;
6
7    @Override
8    protected void onCreate(Bundle savedInstanceState) {
9      super.onCreate(savedInstanceState);
10
11     rules_to_send= /* rule as described in List. 4.1 */ ;
12
13
14       /*
15      * When the user clicks on the "Settings" button the
16      * MainActivity launches the AddCircleActivity activity
17      * for which it would like a result when it finished. See the
            screenshot in Fig. 4.1b
18      */
19     settingsButton.setOnClickListener(new OnClickListener() {
20       Intent intent= new
             Intent(MainActivityTest.getIstance().getApplicationContext(),
21                                                 AddCircleActivity.class);
22
23       @Override
24       public void onClick(View v) {
25         startActivityForResult(intent, 1);
26       }
27     });
28
29
30      /*
31      * When the user click on the "Send Rules" button, the
32      * MainActivity send the rules to ANARC library using
33      * the sendRuleToFramework method.
34      */
35     startButton.setOnClickListener(new OnClickListener() {
36
37       @Override
38       public void onClick(View v) {
39         sendRuleToFramework();
40       }
41     });
42
```

```
43
44        /*
45        * When the user click on the "Stop Framework" button, the
46        * MainActivity broadcast the intent_stop to all interested
47        * BroadcastReceivers and then remove all the previously
48        * assigned RIDs (line 54) .
49        */
50      endButton.setOnClickListener(new OnClickListener() {
51        Intent intent_stop= new Intent("com.framework.STOP_FRAMEWORK");
52
53        @Override
54        public void onClick(View v) {
55          registered_rid.clear();
56          mContext.sendBroadcast(intent_stop);
57          editText.setText("Stop framework\n");
58        }
59      });
60
61        /*
62        * When the user click on the "Remove Rule" button, the
63        * MainActivity invokes the showListRule method. It shows
64        * a pop-up list by which the user can choice the rule RID
65        * to remove.
66        */
67      removeButton.setOnClickListener(new OnClickListener() {
68
69        @Override
70        public void onClick(View v) {
71          int dim;
72          if((dim=registered_rid.size())>0){
73            showListRule();
74          }
75          else{
76            editText.setText("");
77          }
78        }
79      });
80
```

```
81
82    /*
83       * When the user click on the "Show Map" button, the
84       * MainActivity launches the AddCircleActivity activity.
85       * It shows to the user a map with the Circular Area
86       * monitored (if any) and the collected samples.
87       * See the screenshot in Fig. 4.1c
88       */
89     mapButton.setOnClickListener(new OnClickListener() {
90
91       @Override
92       public void onClick(View arg0) {
93         try{
94           Intent intent_map= new
                 Intent(MainActivityTest.this.getApplicationContext(),
                 RSSI_map.class);
95           startActivity(intent_map);
96         }
97         catch(Exception e){
98           e.printStackTrace();
99         }
100      }
101    });
102  }
103
```

```
104 /*
105  * This method is used to send new rules to the ANARC
106  * library when the user presses the "Send Rule" button
107  * in the MainActivity.
108  */
109 private void sendRuleToFramework(){
110   Intent intent= new Intent("com.framework.START_FRAMEWORK");
111   if(!locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)){
112     alertGps();
113   }
114   else{
115
116     /*
117      * If the user chose some custom circular area inside the
118      * AddCircleActivity, the variable "result" contains them,
119      * otherwise it is empty. In this case the default settings
120      * will be used.
121     */
122       if(!result.equals("")){
123           "<ruleList>"
124           + "<RULE>"
125           + " <WHEN>"
126           + " <CONDITION>"
127              // Use custom Circular Areas
128           +        result
129           + " </CONDITION>"
130           + " <DOIFTRUE>"
131           + "   com.usingframework.RSSI.getAreaAlarmTrue"
132           + " </DOIFTRUE>"
133           + " <DOIFFALSE>"
134           + "   com.usingframework.RSSI.getAreaAlarmFalse"
135           + " </DOIFFALSE>"
136           + "   <ONCHANGE>"
137           + "    <PROPERTY>"
138           + "      net.cellular.cellID"
139           + "    </PROPERTY>"
140           + "    <DO>  "
141           + "       com.usingframework.RSSI.getSignalRSSI "
142           + "    </DO>"
143           + "   </ONCHANGE>"
144           + " </WHEN>"
145           + " <propertyList>"
146           + "   <PROPERTY>"
147           + "     geo.position.latitude"
148           + "   </PROPERTY>"
149           + "   <PROPERTY>"
150           + "     geo.position.longitude"
151           + "   </PROPERTY>"
152           + "      <PROPERTY>"
```

```
153             + "         geo.position.accurancy"
154             + "       </PROPERTY>"
155             + "       <PROPERTY>"
156             + "         net.cellular.ASU"
157             + "       </PROPERTY>"
158             + "       <PROPERTY>"
159             + "         net.networkAccess.RoamingState"
160             + "       </PROPERTY>"
161             + "       <PROPERTY>"
162             + "         net.cellular.NetworkOperatorName"
163             + "       </PROPERTY>"
164             + " </propertyList>"
165             + " </RULE>"
166             + "</ruleList>";
167
168       }
169
170       /*
171       *  Send the rules to the ANARC library
172       */
173       intent.putExtra("rulelist", rules_to_send);
174
175       AppBroadcastRCV appBroadcastRCV= new AppBroadcastRCV();
176       editText.append("Adding rule... \n");
177       sendOrderedBroadcast(intent, null, appBroadcastRCV, null, 0 ,
              null,null);
178     }
179   }
180
181   /*
182   * This method show a dialog which permits to user to
183   * choose which rule must be removed or "All" .
184   */
185   private int showListRule(){
186     int ret=-1;
187     if(registered_rid.size()>0){
188       final String[] items = new String[registered_rid.size()+1];
189       items[0]="All";
190       int i=1;
191       for(int rid: registered_rid){
192         items[i]=String.valueOf(rid);
193         i++;
194       }
195
```

```
196       AlertDialog.Builder builder = new AlertDialog.Builder(this);
197       builder.setTitle("Make your selection");
198       builder.setItems(items, new DialogInterface.OnClickListener() {
199
200         public void onClick(DialogInterface dialog, int item) {
201         Toast.makeText(MainActivityTest.getIstance().getApplicationContext(),
              "Selected: " + items[item], Toast.LENGTH_SHORT).show();
202           //If "item==0" the user selected the "All" option.
203           if(item==0){
204
205              //Broadcast an intent for each rule identificator to
                   remove.
206             for(int rid : registered_rid){
207               Intent remove_frame_intent= new
                    Intent("com.framework.removeRule");
208               remove_frame_intent.putExtra("RID", rid);
209               editText.append("Removed: " + rid + "\n");
210               registered_rid.remove(registered_rid.indexOf(rid));
211               // Ask to ANARC libary to remove the rule with ID "rid"
212               MainActivityTest.getIstance().sendBroadcast(remove_frame_intent);
213             }
214           }
215           else{
216              // The user selected a specific RID, broadcast an
                   intent to remove it.
217               Intent remove_frame_intent= new
                    Intent("com.framework.removeRule");
218             remove_frame_intent.putExtra("RID",
                  Integer.parseInt(items[item]));
219             editText.append("Removed: " + items [item] + "\n");
220             registered_rid.remove(registered_rid.indexOf(Integer.parseInt(items[item])));
221             MainActivityTest.getIstance().sendBroadcast(remove_frame_intent);
222           }
223         }
224       });
225       AlertDialog alert = builder.create();
226       alert.show();
227     }
228     else
229       Toast.makeText(this, "no rules to remove",
            Toast.LENGTH_SHORT).show();
230     return ret;
231   }
232
233 // ..............
234
```

```
235
236   /*
237    * When the Coverage Map App is terminating, it broadcasts
238    * as many intents are necessary for removing all active rules.
239    */
240   @Override
241   protected void onDestroy(){
242     super.onDestroy();
243
244     for( Integer rid_elem : registered_rid){
245       Intent remove_frame_intent= new
              Intent("com.framework.removeRule");
246       remove_frame_intent.putExtra("RID", rid_elem);
247       editText.append("Removed: " + rid_elem + "\n");
248       registered_rid.remove(rid_elem);
249       sendBroadcast(remove_frame_intent);
250     }
251   }
252
253    /*
254    * The onActivityResult is called when an activity you launched
255    * exits, giving you the requestCode you started it with, the
256    * resultCode it returned, and any additional data from it.
257    * The resultCode will be RESULT_CANCELED if the
258    * AddCircleActivity explicitly returned that, didn't return any
259    * result, or crashed during its operation. You will receive this
260    * call immediately before onResume() when the MainActivity
261    * is re-starting.
262    */
263   @Override
264   protected void onActivityResult(int requestCode, int resultCode,
          Intent data) {
265       if (requestCode == 1) {
266         if(resultCode == RESULT_OK){
267         result=data.getStringExtra("result");
268         ArrayList<Area> lista_aree= (ArrayList<Area>)
              data.getSerializableExtra("lista_aree");
269
270         //.............
271
272         sendRuleToFramework();
273       }
274       else if (resultCode == RESULT_CANCELED) {
275         //Write your code if there is no result
276         // ..........
277         // Reset to default settings
278       }
279
280   // ....................
```

```
281
282   /*
283   *  If the GPS localization methos is disable, this method
284   *  asks to the user to enable it.
285   */
286   public synchronized void alertGps(){
287     AlertDialog.Builder builder = new AlertDialog.Builder(this);
288     builder.setMessage("Enable GPS first, please!")
289     .setCancelable(false)
290     .setPositiveButton("Yes", new DialogInterface.OnClickListener()
          {
291       public void onClick(DialogInterface dialog, int id) {
292         Intent intent = new
              Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
293         startActivity(intent);
294       }
295     })
296     .setNegativeButton("No", new DialogInterface.OnClickListener() {
297       public void onClick(DialogInterface dialog, int id) {
298         Intent backtoHome = new Intent(Intent.ACTION_MAIN);
299         backtoHome.addCategory(Intent.CATEGORY_HOME);
300         backtoHome.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
301         startActivity(backtoHome);
302         dialog.cancel();
303
304       }
305     });
306     AlertDialog alert = builder.create();
307     alert.show();
308   }
309 }
```

Listing 4.2: Main activity source code

Chapter 5

Conclusion

5.1 Contributions

The purpose of this thesis has been to implement an Android library that facilitates the developers during the creation of network-aware applications. We reach our goals proving ANARC system of two major features.

The first one is to perform a reactive and homogeneous monitoring of a heterogeneous resource set. Modern smartphones are equipped with a mixed set of sensors. So it is possible to analyze the environment surrounding the user (light sensor, microphone, etc.), the network connections (Wi-Fi antenna, Bluetooth antenna, etc.), the geographical position (e.g. GPS antenna) or the hardware state (battery health, battery state, etc.). ANARC allows developer to retrieve information from the sensors using a simple high level language. The mainly advantage is that using this approach the developer can receive information in the same way from each resource. In other words, he can retrieve data from the light sensor in the same way he retrieves data from the Wi-Fi antenna.

The second feature is the ability to react in real-time to the environmental changes. Normally, an Android application uses a polling approach: it periodically senses the environment in order to detect changes and to react to them. This approach causes, in some cases, the loss of relevant data or, in other cases, the creation data redundancy. The ANARC reactive approach, instead, allows a constantly monitor the environment changes and reacting to them immediately. This behavior permits to generate data only when it is really necessary.

To our knowledge, no other open-source Android library provides these functionalities.

5.2 Future Work

Future improvements are possible for this library through different points of view. First, it could be interesting to design and to implement additional modules, e.g.:

Place - A place is similar to a *Point Of Interest* (POI) which describes information about locations such as name, category, unique identifier, or civic address. To be more precise, a place is a human construct which typically has a coarse level of spatial granularity. Places are generally larger scale administrative constructs, either informally or formally defined. Countries, states, counties, districts, neighborhoods and postal codes or telephone area codes are all places. Places are also informal or colloquially defined, such as the Bay Area in the United States, home or office. Places have spatial relationships with parents, children, and adjacencies and contained by semantics. The Place module should be able to assign a human readable label to geographical areas interesting for the user. For instance, if a device every morning moving between a set of tower cell, the system must be able to infer that the user is moving from *home* to the *office* using that specific route.

Time - The module should implement some mechanisms in order to permit the scheduling of specific tasks.

Second, it should be useful studying the feasibility of the ANARC implementation using other operating systems, e.g. iOS. It is possible that different operating systems can permit a different access to the sensor and their information.

Third, ontology languages could be introduced in way of providing vocabulary and structure for describing information sources.

Appendix A

Implementation details

A.1 ANARC library classes and methods

The UML diagram class in Fig. A.1 provides an overview of the ANARc's core by describing the objects and classes inside the system and the relationships between them.

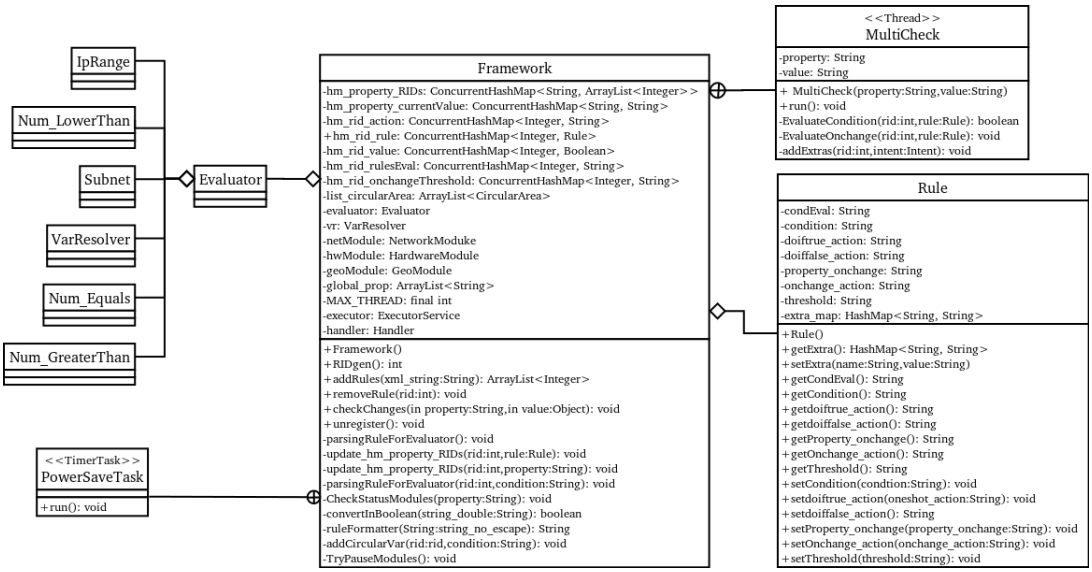


Figure A.1: Core class

Most significant methods used in *Framework* class are:

- *ArrayList¡Integer¿ addRules(String xml_string)*:
  Adds rules read from an xml formatted string and assigns a RID to each rule read.

- *int RIDgen()*:
  It have the task of generate unique integer identificator for each rule received from customer Android applications.
- *void parsingRuleForEvaluator(int rid, String string_rule)* :
  It parses the parameter *string_rule* as JEVAL evaluator required.
- *void removeRule(int RID)*:
  removes the rule identified by RID.
- *void checkChanges(String property, Object value)*:
  checks if the value of the property has changed, in that case evaluates, also, all the rules that contain *property*.
- *void unregister()*:
  stops all modules' activity.

The explanation of some data structured used is given hereunder:

- *boolean frameworkStatus*
  specifies if the framework is running or not.
- *ConcurrentHashMap¡String, ArrayList¡Integer¿¿ hm_property_RIDs*
  links each property with the RIDs in which it appears.
- *ConcurrentHashMap¡String, String¿ hm_property_currentValue*
  links each property with his current value.
- *ConcurrentHashMap¡Integer, Boolean¿hm_rid_value*
  links each RID with the current boolean value of the rule.
- *ConcurrentHashMap¡Integer, Rule¿ hm_rid_rule*
  links each RID to a *Rule* objects.
- *ConcurrentHashMap¡String, String¿hm_property_currentValue*: links each property with his current value.
- *ArrayList¡CircularArea¿ list_circularArea*
  list of current CircularArea object.
- *ConcurrentHashMap¡Integer, String¿ hm_rid_onchangeThreshold*
  links each RID to a threshold, if any.

A.2 JEVAL evaluator library

JEVAL uses as operator precedence (from highest to lowest):

- + unary plus, - unary minus, ! boolean not
- $*$ multiplication, \division, % modulus
- + addition, - subtraction
- < less than, <= less than or equal, > greater than, >= greater than or equal
- = equal, ! = not equal
- && boolean and
- —— boolean or

JEVAL evaluator follow these main giudeline in order to evaluate an expression:

- Spaces are ignored when parsing.
- The expression is evaluated as one or more sub-expressions.
- Sub-expressions within open parentheses and closed parentheses are evaluated before other parts of the expression.
- Inner most sub-expression are evaluated first working outward.
- Sub-expressions at the same level are evaluated from left to right.
- When evaluating expressions and sub-expressions, operators are evaluated with the following precedence listed before.
- Operators with the same precedence are evaluated from left to right.
- Once the expression is parsed, Variables are replaced with their values. The evaluator has its own internal variable map that it used to resolve variable values. It's possible to choose to set you own variable resolver on your evaluator instancein this case variables resolved by a custom resolver will override any variables in the evaluator's inner variable map.
- Functions are executed and replaced with their results. Function arguments are each individually evaluated as sub-expressions that are

comma separated. This gives you the ability to use nested functions in your expressions.

- Once all variables and functions are resolved, then the parsed expression and sub-expressions are evaluated according to operator precedence.

Function and variable names can not break any of the following rules:

- can not start with a number
- can not contain an operator (see the above list of operators)
- can not contain a quote character - single or double
- can not contain a brace character - open or closed
- can not contain one of the following special characters: #, ~, ^, !

Expressions can contain different types of expressions but numeric and string types can not be mixed in a left / right operand pair. Also, if an expression does not change, it will not be parsed each time an evaluation is required. Therefore, variables values can change so the expression can be evaluated without having to re-parse it. The string used to start variables, "#", can not appear in an expression.

A.3 ANARC Functions

As ANARC born as a network-aware library, it implements and exposes the following functions:

- EqualsIgnoreCase
  usage: ***equalsIgnoreCase(string_one, string_two);***
  example: *equalsIgnoreCase('hello world', 'HELLO WORLD');*
  return: true or false
- Equals
  usage: ***equals(string_one, string_two);***
  example: *equals('hello world', 'hello world');*
  return: true or false
- Num_GreaterThan
  usage: ***num_gt('numeric_value_one', 'numeric_value_two');*** example: ***num_gt('3','2');***
  return: true or false

- Num_LowerThan
  usage: ***num_lt('numeric_value_one', 'numeric_value_two');*** example: *num_lt('3','2');* return: true or false

- Num_Equals
  usage: **num_equals('numeric_value_one, numeric_value_two');**
  example: num_equals('3', '2'); return: true or false

- IpRange
  Checks if *ip_to_check* belongs to a continuous IP address range and if it is between *ip_start* and *ip_end*.
  usage: ***ipRange(ip_to_check, ip_start, ip_end);*** example: *ipRange('10.0.2.15', '10.0.2.0', '10.0.2.30')*; return: true or false

- Subnet
  Checks if *ip_to_check* belongs to a given *subnet/mask*. usage: ***subnet(ip_to_check, subnet, mask);*** example:
  subnet('10.0.2.15', '10.0.2.0', '24'); return: true or false

- CircularArea
  usage: ***CircularArea(center_latitude, center_longitude, radius);***
  example: *CircularArea(43.721387,10.389901, 2000);*
  return: true if the user is inside the specified area, false otherwise.

A.4 Rule Definition Language

A.4.1 Schema Definition Language

```
1  <?xml version="1.0" encoding="UTF-8"?>
2   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       elementFormDefault="qualified"
       attributeFormDefault="unqualified">
3   <xs:element name="ruleList">
4    <xs:complexType>
5     <xs:sequence>
6      <xs:element name="RULE">
7       <xs:complexType>
8        <xs:sequence>
9         <xs:element name="WHEN" minOccurs="0">
10         <xs:complexType>
```

```
11          <xs:sequence>
12           <xs:element name="CONDITION"
                 type="xs:string"></xs:element>
13           <xs:element name="DOIFTRUE" type="xs:string"
                 minOccurs="0"></xs:element>
14           <xs:element name="DOIFFALSE" type="xs:string"
                 minOccurs="0"></xs:element>
15           <xs:element name="ONCHANGE" minOccurs="0">
16            <xs:complexType>
17             <xs:sequence>
18              <xs:element name="PROPERTY"
                    type="xs:string"></xs:element>
19              <xs:element name="DO"
                    type="xs:string"></xs:element>
20              <xs:element name="THRESHOLD" type="xs:string"
                    minOccurs="0"></xs:element>
21             </xs:sequence>
22            </xs:complexType>
23           </xs:element>
24          </xs:sequence>
25         </xs:complexType>
26        </xs:element>
27        <xs:element name="propertyList" minOccurs="0">
28         <xs:complexType>
29          <xs:sequence>
30           <xs:element name="PROPERTY" maxOccurs="unbounded"
                 type="xs:string"></xs:element>
31          </xs:sequence>
32         </xs:complexType>
33        </xs:element>
34       </xs:sequence>
35      </xs:complexType>
36     </xs:element>
37    </xs:sequence>
38   </xs:complexType>
39  </xs:element>
40 </xs:schema>
```

Listing A.1: list:xsdRulefile

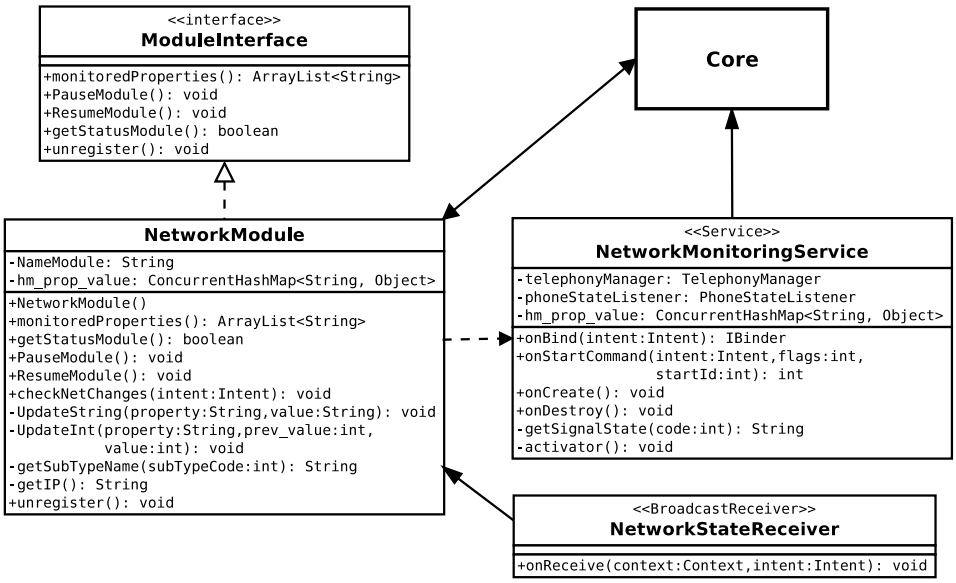


Figure A.2: Network module class diagram

A.5 The modules

A.5.1 Network Module

The class diagram in Fig. A.2 shows the building blocks of the Network module and relationships between its classes.

The NetworkModule class implements the following methods:

- checkNetChanges
  When *BroadcastStateReceiver* receives an intent from the system, it invokes this function. *checkNetChanges* checks all properties in Tab. A.1 and updates them values, if changed.
- UpdateString(String property, String value)
  It is called by previous method when it needs to update a string property.
- UpdateInt(String property, int prev_value, int value)
  It is called by *checkNetChanges* when it needs to update an integer property.
- getSubTypeName(int subTypeCode)
  It converts *subTypeCode*, that is a a network-type-specific integer de-

scribing the sub-type of the network, in an human readable string. See property *net.networkAccess.mobileConnectionType* in Tab.A.1

- getIP()
  It gets current IP address and converts it into a human readable string.

A.5.2 Hardware Module

The class diagram in Fig. A.3 specifics attributes and methods of the hardware module's components previously presented in 3.4.2.

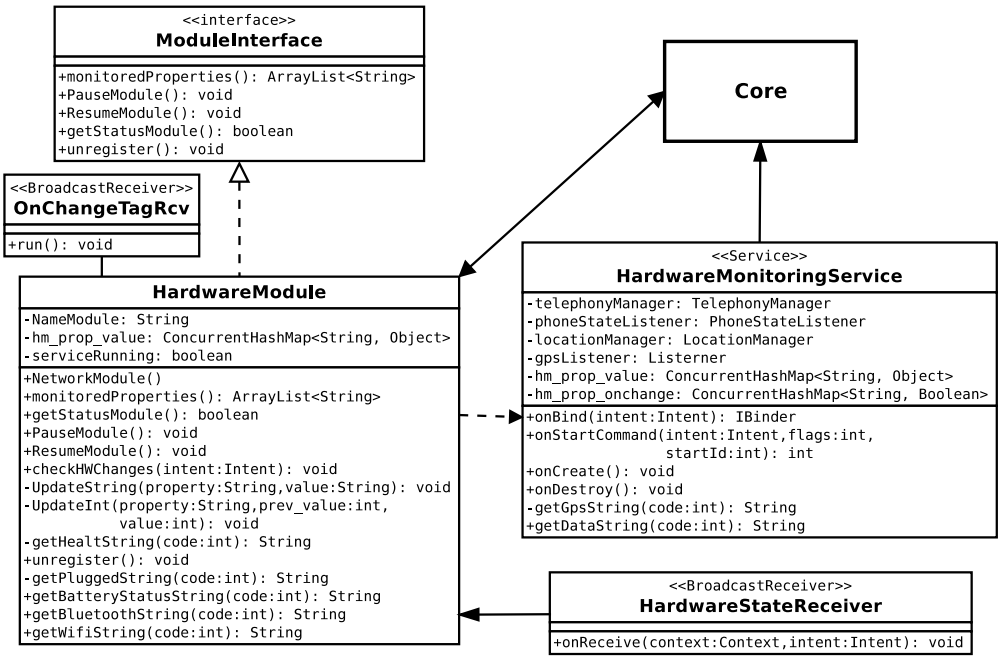


Figure A.3: Hardware module class diagram

The HardwareModule is able to monitor the properties show in Tab. A.3.

The Tab. A.4 shows the HardwareMonitoringService monitorable properties.

A.5.3 Geographical Module

In addition to Sec. A.5.3 description, we show the UML class diagram (Fig. A.4) and the description of two inner class of GeoMonitoringService.java:

Property name	Values
net.ip.value	Current IP address value assigned to the device.
net.networkAccess.status	CONNECTED CONNECTING DISCONNECTED DISCONNECTING SUSPENDED UNKNOWN
net.networkAccess. connectivityType	WIFI MOBILE
net.networkAccess. mobileConnectionType	1xRTT HSUPA CDMA UMTS EDGE EHRPD EVDO_0 EVDO_B EVDO_A HSPAP GPRS IDEN HSDPA UNKNOWN HSPA
	true/false
net.wifi.rssi	Current Wi-Fi RSSI value. It isn't normalized (but should be). Each vendor use a difference range.
net.wifi.ssid	Current Wifi SSID value.

Table A.1: Network module monitorable properties and values

Property name	Values
net.cellular.cellID	Current GSM Cell ID (CID)
net.cellular.LAC	Current Location Area Code)
net.cellular.cdma.sysID	Current system ID for CDMA networks.
net.cellular.cdma.netID	Current network ID for CDMA networks.
net.cellular.cdma.bsID	Current base station ID for CDMA networks.
net.cellular.RSSI	Current RSSI value.
net.cellular.ASU	Current ASU value.
net.cellular.status	STATE_EMERGENCY_ONLY STATE_IN_SERVICE STATE_OUT_OF_SERVICE STATE_POWER_OFF

Table A.2: NetworkMonitoringService monitorable properties and values

Property name	Values
hw.wifi.state	DISABLED DISABLING ENABLED ENABLING UNKNOWN ERROR
hw.bluetooth.state	OFF TURNING_OFF ON TURNING_ON ERROR
hw.battery.state	CHARGING DISCHARGING FULL NOT_CHARGING STATUS_UNKNOWN ERROR
hw.battery.level	Current battery level value.
hw.battery.health	COLD DEAD GOOD OVER_VOLTAGE OVERHEAT UNKNOWN UNSPECIFIED_FAILURE: ERROR
hw.battery.plugged	AC USB WIRELESS ERROR

Table A.3: Hardware module monitorable properties and values

Property name	Values
hw.data.state	ACTIVITY_DORMANT CONNECTED CONNECTING SUSPENDED DISCONNECTED ERROR
hw.gps.status	FIRST_FIX SATELLITE_STATUS STARTED STOPPED ERROR

Table A.4: HardwareMonitoringService monitorable properties and values

- **PauseLocationUpdateTask**
  In accordance with Eq. 3.41, It sends a message to the *handler* after 30 minutes have passed, in order to resume location tracking.

- **mainTask**
  Two minutes later GPS antenna enabling event, this method checks if GPS reached the satellites fix.

The GeoModule class can monitor the properties listed in Tab. A.5.

Property name	Values
geo.position.latitude	Current latitude value.
geo.position.longitude	Current longitude value.
geo.position.altitude	Current altitude value.
geo.position.speed	Current speed value.

Table A.5: Geographical module monitorable properties and values

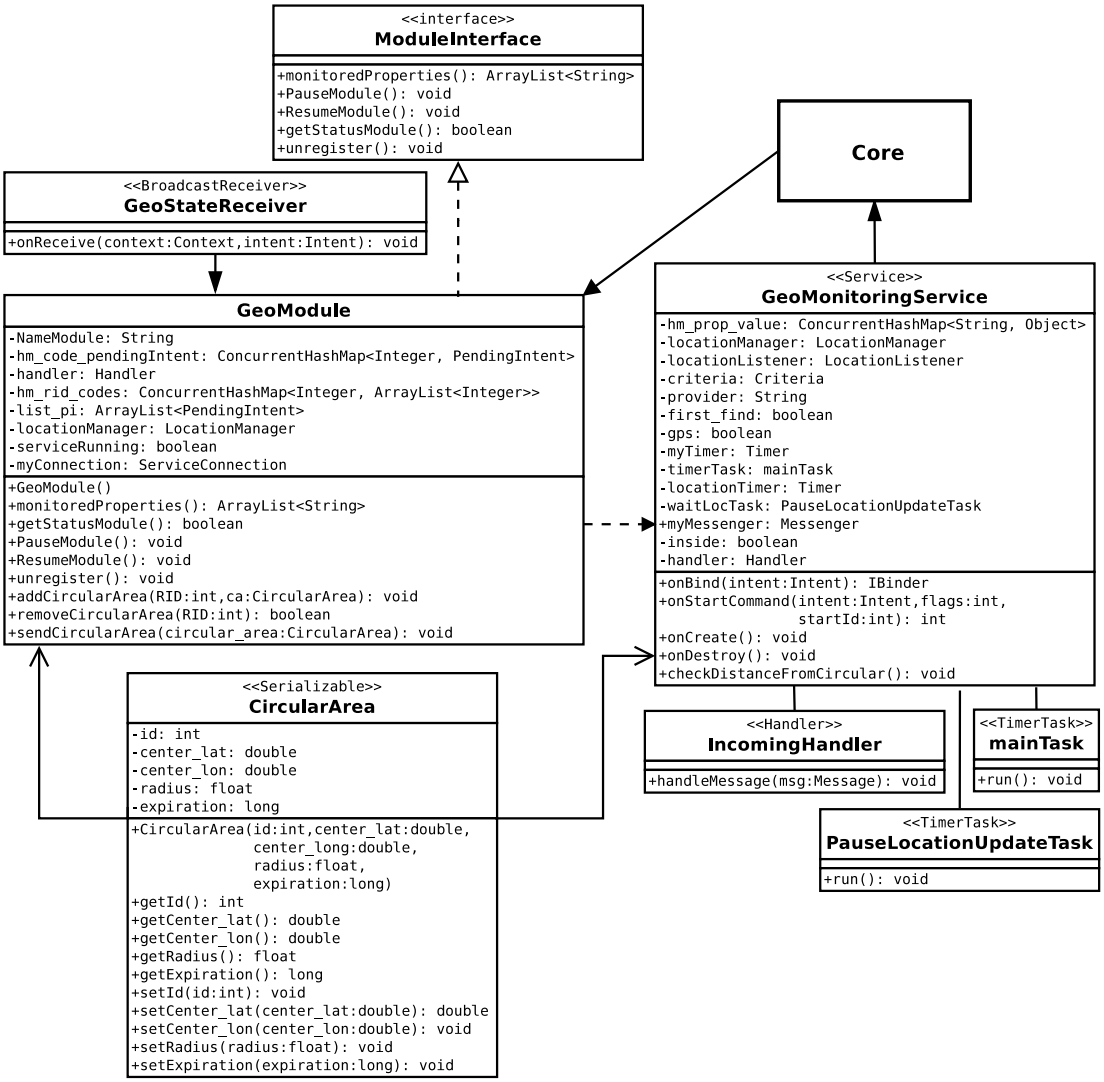


Figure A.4: Geographical module diagram class

Bibliography

[1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Handheld and ubiquitous computing*, pp. 304–307, Springer, 1999.

[2] P. Makris, D. Skoutas, and C. Skianis, "A survey on context-aware mobile and wireless networking: On networking and computing environments' integration," 2012.

[3] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, "A survey of context modelling and reasoning techniques," *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161–180, 2010.

[4] M. Raento, A. Oulasvirta, R. Petit, and H. Toivonen, "Contextphone: A prototyping platform for context-aware mobile applications," *Pervasive Computing, IEEE*, vol. 4, no. 2, pp. 51–59, 2005.

[5] S. B. and T. M., "Disseminating active map infromation to mobile-hosts," *IEEE Network*, 1994.

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

[7] S. Hattori, T. Tezuka, and K. Tanaka, "Activity-based query refinement for context-aware information retrieval," in *Digital Libraries: Achievements, Challenges and Opportunities*, pp. 474–477, Springer, 2006.

[8] K. Lyytinen, Y. Yoo, U. Varshney, M. S. Ackerman, G. Davis, M. Avital, D. Robey, S. Sawyer, and C. Sorensen, "Surfing the next wave:

design and implementation challenges of ubiquitous computing environments," *Communications of the Association for information systems*, vol. 31, no. 2004, pp. 697–716, 2004.

[9] K. Lyytinen and Y. Yoo, "Ubiquitous computing," *Communicationsof the ACM*, vol. 45, no. 12, p. 63, 2002.

[10] B. N. Schilit and D. Duchamp, "Adaptive remote paging for mobile computers," 1991.

[11] Z. Drey and C. Consel, "A visual, open-ended approach to prototyping ubiquitous computing applications," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, pp. 817–819, IEEE, 2010.

[12] L. A. Suchman, *Plans and situated actions: the problem of human-machine communication*. Cambridge university press, 1987.

[13] M. Weiser, "Some computer science issues in ubiquitous computing," *Communications of the ACM*, vol. 36, no. 7, pp. 75–84, 1993.

[14] R. Want, A. Hopper, V. Falcão, and J. Gibbons, "The active badge location system," *ACM Transactions on Information Systems (TOIS)*, vol. 10, no. 1, pp. 91–102, 1992.

[15] A. Harter and A. Hopper, "A distributed location system for the active office," *Network, IEEE*, vol. 8, no. 1, pp. 62–70, 1994.

[16] C. Qin, X. Bao, R. Roy Choudhury, and S. Nelakuditi, "Tagsense: a smartphone-based approach to automatic image tagging," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pp. 1–14, ACM, 2011.

[17] C. Spelta, V. Manzoni, A. Corti, A. Goggi, and S. M. Savaresi, "Smartphone-based vehicle-to-driver/environment interaction system for motorcycles," *Embedded Systems Letters, IEEE*, vol. 2, no. 2, pp. 39–42, 2010.

[18] A. Beach, M. Gartrell, S. Akkala, J. Elston, J. Kelley, K. Nishimoto, B. Ray, S. Razgulin, K. Sundaresan, B. Surendar, *et al.*, "Whozthat? evolving an ecosystem for context-aware mobile social networks," *Network, IEEE*, vol. 22, no. 4, pp. 50–55, 2008.

[19] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton, "Cyberguide: A mobile context-aware tour guide," *Wireless networks*, vol. 3, no. 5, pp. 421–433, 1997.

[20] A. LaMarca, Y. Chawathe, S. Consolvo, J. Hightower, I. Smith, J. Scott, T. Sohn, J. Howard, J. Hughes, F. Potter, *et al.*, "Place lab: Device positioning using radio beacons in the wild," in *Pervasive Computing*, pp. 116–133, Springer, 2005.

[21] M. Azizyan, I. Constandache, and R. Roy Choudhury, "Surroundsense: mobile phone localization via ambience fingerprinting," in *Proceedings of the 15th annual international conference on Mobile computing and networking*, pp. 261–272, ACM, 2009.

[22] D. H. Kim, Y. Kim, D. Estrin, and M. B. Srivastava, "Sensloc: sensing everyday places and paths using less energy," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, pp. 43–56, ACM, 2010.

[23] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, pp. 21–21, 2010.

[24] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell, "The jigsaw continuous sensing engine for mobile phone applications," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, pp. 71–84, ACM, 2010.

[25] J. Paek, J. Kim, and R. Govindan, "Energy-efficient rate-adaptive gps-based positioning for smartphones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pp. 299–314, ACM, 2010.

[26] Z. Zhuang, K.-H. Kim, and J. P. Singh, "Improving energy efficiency of location sensing on smartphones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pp. 315–330, ACM, 2010.

[27] K. Lin, A. Kansal, D. Lymberopoulos, and F. Zhao, "Energy-accuracy trade-off for continuous mobile device location," in *Proceedings of the*

8th international conference on Mobile systems, applications, and services, pp. 285–298, ACM, 2010.

[28] I. Constandache, S. Gaonkar, M. Sayler, R. R. Choudhury, and L. Cox, "Enloc: Energy-efficient localization for mobile phones," in *INFOCOM 2009, IEEE*, pp. 2716–2720, IEEE, 2009.

[29] Y. Ma, R. Hankins, and D. Racz, "iloc: a framework for incremental location-state acquisition and prediction based on mobile sensors," in *Proceedings of the 18th ACM conference on Information and knowledge management*, pp. 1367–1376, ACM, 2009.

[30] M. Balman and S. Byna, "Open problems in network-aware data management in exa-scale computing and terabit networking era," in *Proceedings of the first international workshop on Network-aware data management*, pp. 73–78, ACM, 2011.

[31] K. LaCurts, S. Deng, A. Goyal, and H. Balakrishnan, "Choreo: network-aware task placement for cloud applications," in *Proceedings of the 2013 conference on Internet measurement conference*, pp. 191–204, ACM, 2013.

[32] "Portolan network sensing architecture." `http://portolan.iet.unipi.it/`.

[33] A. Faggiani, E. Gregori, L. Lenzini, S. Mainardi, and A. Vecchio, "On the feasibility of measuring the internet through smartphone-based crowdsourcing," in *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2012 10th International Symposium on*, pp. 318–323, IEEE, 2012.

[34] E. Gregori, L. Lenzini, V. Luconi, A. Vecchio, and D. di Ingegneria dell'Informazione, "Sensing the internet through crowdsourcing,"

[35] L. F. Pitt, M. Parent, I. Junglas, A. Chan, and S. Spyropoulou, "Integrating the smartphone into a sound environmental information systems strategy: Principles, practices and a research agenda," *The Journal of Strategic Information Systems*, vol. 20, no. 1, pp. 27 – 37, 2011.

[36] "Rootmetrics coverage map." `http://www.rootmetrics.com/`.

[37] "Netradar." `http://www.netradar.org/`.

[38] A. Faggiani, E. Gregori, L. Lenzini, V. Luconi, and A. Vecchio, "Lessons learned from the design, implementation, and management of a smartphone-based crowdsourcing system," in *Proceedings of First International Workshop on Sensing and Big Data Mining*, SENSEMINE'13, (New York, NY, USA), pp. 2:1–2:6, ACM, 2013.