POLITECNICO DI TORINO

Master degree in Computer Engineering

Master Degree Thesis

Recommending Trigger-Action Rules for the IoT

Supervisors:
Fulvio CORNO
Luigi DE RUSSIS
Alberto MONGE ROFFARELLO

Candidate:
Alessia MANTOVANI

December 2018

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Internet of Things (IoT) is the Internet extended into to the physical world. Its function is to collect data from IoT objects and transform them into useful information. These objects can be standard devices, such as computers, smartphones and tablets, but also online services or any kind of non-internet-enabled physical device and everyday objects.

Although IoT objects do most of the work without any human intervention, people can interact with them for example to set them up, to give them instructions or simply to access the data. For empowering users to program their IoT objects, a promising approach has been adopted (i.e. EUD). End-User Development (EUD) refers to activities and tools that allow end users (i.e., people who are not professional software developers) to program computers.

One of the most used format in EUD is Trigger-Action programming paradigm. It is a paradigm where the user must define rules by means of "if-then" constructs, to create event-action rules. In the last years, an evolution of this area was the use of mobile devices to support end users development activities. Although EUD in the IoT has recently gained interest, interoperability and scalability challenges remain. With the spread of new smart "things", the amount of information may become too high and cluttered [1].

Therefore the most important concepts of the just mentioned paradigm are:

- Trigger: an event that, when detected, causes the execution of an Action;
- Action: an operation that is executed as a consequence of a Trigger;
- Rule: the association of a Trigger with an Action (an example of a rule could be: IF I go into my home, THEN turn on smartphone Wi-fi);

This thesis wants to use the EUD approach and the Trigger-Action programming paradigm, adding a system of automatic recommendations instead of having

the user create her rule manually. For simplicity, this was done focusing on mobile and web worlds (the IoT objects were user smartphone and an online service), allowing an easiest "in-the-wild" evaluation with end users. This system involves the development of an Android application (called "TriggerAction Rules") which studies end-users' habits of using their Android devices with the aim of proposing simple associations of events (i.e., trigger-action rules) to them. Thus they no longer have to create these rules manually neither know the characteristic of every single object they may encounter before creating their rules, but they have only to decide if keep or discard the recommendation suggested to them.

1.1 Problem definition

As the popularity of the Internet of Things (IoT) is expanding, even its complexity is growing and as a consequence, the number of possible combinations between different devices and services is increasing exponentially. Thus, new needs emerged and rules definition and reuse may become challenging without proper suggestions. However existing recommendation techniques could be useful to help end users to use EUD systems, but have not yet been consistently explored. Moreover, recommendation technologies have mainly been studied for assisting developers instead of end users.

Nowadays, many applications which implement trigger-action programming are being created with the aim of helping end users to manage their IoT objects (both devices feature and online services). Thanks to application like IFTTT[1], people can define, share, and reuse trigger-action rules. However, these kind of applications allow users to create chains of simple conditional statements, but they do not provide any kind of recommendation on which statement they have to associate. In this way, users have to search manually in the triggers and actions huge list and create one or more trigger-action rule. Users must be aware of every single object they may encounter before creating their rules and they often do not have the necessary knowledge to really understand all the possible implications of the selected associations. This procedure, indeed, involves that they could define several similar rules performing the same logical operation.

The reasons just mentioned led to the development of EUPont. This ontology can support end users in the choice of the desired objects in order to exploit their functionality and improve the rule composition process. Thanks to EUPont it has been possible to developing the Android application described in this thesis.

[1]IFTTT (if-this-then-that) is a free service that allows users to get all their apps and devices talking to each other. Available: https://ifttt.com

1.2 Thesis goal

The goal of the thesis is to design, develop, and evaluate a mobile-based application that suggests and actually executes trigger-action rules based on contextual information and user preferences, with the aim of assisting end users in customizing their smartphone functionalities and online services.

In this way, the user's commitment regarding the composition of the rules can be greatly reduced, as well as the infinite possibilities of interconnection of the features that their IoT objects can provide to them. Therefore the process of "learning" and the creation of rules is automated, without depriving the user of control.

In particular, this application can facilitate end users in this scenario, by recommending rules according to their smartphone usage habits, which they can evaluate and subsequently decide whether to activate them or not.

The research questions which guided this work were:

- Is it feasible to automate and assist end users in the creation of trigger-action rules?
- Is it possible to realize a mobile-based application which can suggests Trigger-Action Rules to the users without having users create them?

Once the application has been designed and implemented, the following aspects concerning the measure useful to evaluate the previous questions, were considered:

- Understandability: Is the mobile application understandable enough for end users?
- Usefulness: Does the mobile application effectively help end users in Trigger-Action Rules creation?

1.3 Thesis structure

The initial step of this work was the research and the scouting of the main features and events that the system should support. Subsequently, the second step was to develop the application which supports the selected elements and finally, an evaluation was proposed to a selected number of users.
More in details, the thesis is composed by eight chapters which actually reflect the performed work: Introduction, Background, Analysis, Design, Implementation, Testing, Results and Conclusions.

Background In this chapter, the main concept regarding the approach used (EUD), the ontology by which the application is supported (EUPont) and some tools and libraries of Android (e.g. Room Database, Data bindings), are presented in-depth for the reader contextualization. EUPont get

while the main goal of the thesis is the creation of a mobile application, such an app required additional information coming from either the cloud or a custom-made server. Therefore,

Analysis This chapter, describes how the research and the scouting of the main features and events that the system should support, have been done.
First of all, it has been defined a complete list of elements that the system will allow to customize: smartphone features (supported Android functionalities) and online services (e.g. Facebook, Instagram, Twitter).

Then, it has been defined which events can be considered triggers and which actions and according to the type of the event considered, it has been done a differentiation for customizing the chosen smartphone features and online services.

Design Since this mobile application can not reach the objective above mentioned without additional information coming from either the cloud or a custom-made server, the system follows a client-server architecture and its composition is described in this chapter. Two examples of rule creation for Android functionality and for Facebook service, respectively, are presented and illustrated.

Implementation This chapter describes both how the data coming from the server are translated in the application and the implementation of the mobile application. It describes the communication, how the algorithm works and how the part of online service (Facebook) has been developed.

This phase of the work has been characterized by the development of an Android application able to suggest and execute trigger-action rules that replicates the operations (defined in the previous phase) commonly performed by the user on her smartphone or online services. In particular, the application allows the user to activate a suggested rule and browse the activated rules. It uses the EUPont ontology to save the controlled devices, trigger-action rules and contextual information;

User Evaluation In this chapter, an "in-the-wild" evaluation of the developed application with a selected number of users, is described. Two phases have been performed: a first evaluation has been done for revising some bugs of the application and to correct the algorithm depending on the user feedback; in the second phase, instead, the evaluation has been only oriented to collect data and information about the measures of interest.

Results In this chapter, results obtained from the "in-the-wild" evaluation are described and commented.

Conclusion Final description and discussion on the work done and illustration of possible future improvements.

Chapter 2

Background

In this section the main concepts, systems and tools used for this work, will be presented in-depth for the reader contextualization.

2.1 EUPont ontology

EUPont (End-User Programming Ontology) is an ontology that provides a high-level representation for the end-user programming in the IoT. EUPont, available at http://elite.polito.it/ontologies/eupont.owl, is an extensive ontology for a specific use case. It has been designed for describing and supporting the execution of IoT applications developed by end users to customize their IoT objects (i.e. IoT entities). It can serve as the base layer for end user customization solutions in the IoT, since it defines a structured vocabulary for abstract and generic trigger-action rules that can be automatically adapted to different contextual situations by run-time environments [2].

The main concepts modeled by the Trigger-Action Programming are: Rule (must have at least one Trigger and one Action), Trigger and Action (both are hierarchically organized on the basis of their final goal, in such a way, users can define rules in abstract terms, or they can be more specific).

Devices and services offered to the user are often complex and can offer several capabilities. Trying to resolve this complexity, it was modeled the IoT ecosystem as a set of IoT entities (IoTEntity) able to expose one or more services (Service). Each service describes a capability of an IoT entity, and may have commands (Command) to perform some actions on the related IoT entity, or notifications (Notification) to register event listeners on the related IoT entity.

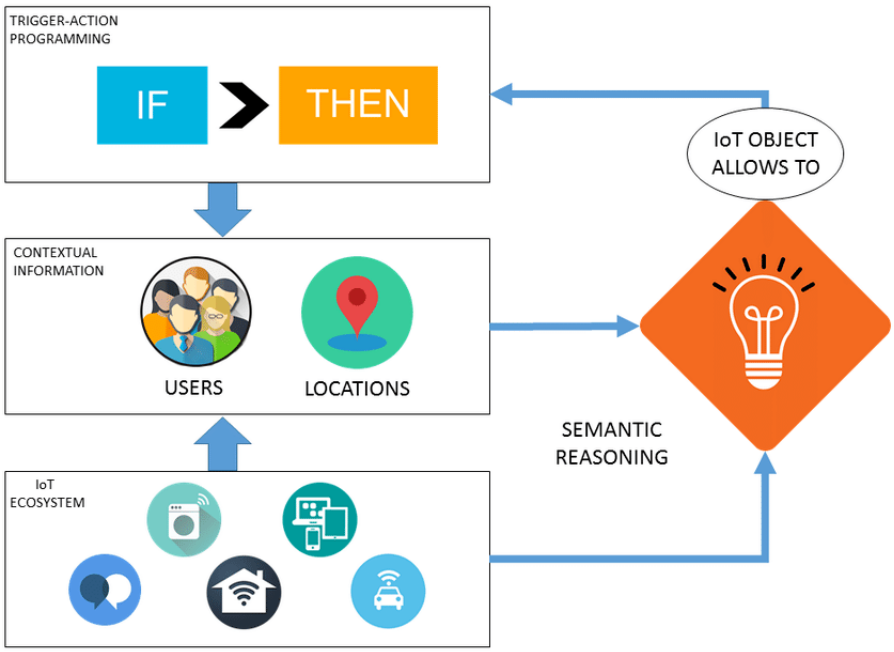

Figure 2.1. EUPont structure

The ontology is composed of four main blocks (Figure 2.1):

- **Trigger-Action Programming** block allows the definition of abstract and technology/brand independent trigger-action rules. Triggers and actions are organized in a hierarchy;

- **IoT Ecosystem** block models IoT devices and services as entities that can offer one or more functionality. Each functionality may have commands to perform some actions, or notifications to register event listeners and both of them include the features needed to interact with the specific technology;

- **Contextual Information** block describes locations and users that act as restrictions for trigger-action rules, and the contextual information of the IoT Ecosystem. Thanks to this layer, the ontological representation provides a strong support for executing the defined end-user rules;

- **Semantic Reasoning** block automatically maps the defined trigger-action rules with devices and services in the IoT Ecosystem that allow to reproduce the desired behaviors, by keeping into account the current context, dynamically.

2.2 Android

The main part of this work has been the development of an Android application that can exploit the previous concepts and systems. It was necessary to intercept events fired by the user and collect all the information about them, in order to implement an algorithm that can be able to provide trigger-action association according to the analysis of the data collected. The following paragraphs will present the libraries and tools that have been used in the application for the implementation of events interception, local database and data bindings.

Events interception In a software architecture, publish–subscribe is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers, but instead categorize published messages into classes without knowledge of which subscribers, if any, there may be. Similarly, subscribers express interest in one or more classes and only receive messages that are of interest, without knowledge of which publishers, if any, there are. Publish–subscribe is a sibling of the message queue paradigm, and is typically one part of a larger message-oriented middleware system. Most messaging systems support both the pub/sub and message queue models in their API, e.g. Java Message Service (JMS). This pattern provides greater network scalability and a more dynamic network topology, with a resulting decreased flexibility to modify the publisher and the structure of the published data.

Android provides various classes to intercept events and all of them have been used in this work:

- Broadcast Receiver: similar to the publish-subscribe design pattern, these broadcasts are sent when an event of interest occurs. Applications can receive broadcasts in two ways: manifest-declared receivers (the system launches application when the broadcast is sent) or context-registered receivers (receive broadcasts as long as their registering context is valid).
- Service: is an application component that can perform long-running operations in the background, and it doesn't provide a user interface;
- Content Observer: is a software design pattern that establishes a one-to-many dependency between objects, anytime the state of one of the objects (the "subject" or "observable") changes, all of the other objects ("observers") that depend on it are notified;
- Sensor Listener: most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions, this interface is used for receiving notifications from the SensorManager when there is new sensor data.

Local Database The Android Room library was used for collecting all information about the events and thanks to this data, the application is able to create a Rule, associating a Trigger to an Action that will then be sent to the server.

The Room persistence library provides an abstraction layer over SQLite to allow for more robust database access while harnessing the full power of SQLite. The library helps the developer to create a cache of his application's data on a device that's running his application. This cache, which serves as the application's single source of truth, allows users to view a consistent copy of key information within the application, regardless of whether users have an internet connection. There are 3 major components in Room library:

- Database: Contains the database holder and serves as the main access point for the underlying connection to the application's persisted, relational data and include the list of entities associated with the database within the annotation;
- Entity: Represents a table within the database;
- DAO: Contains the methods used for accessing the database.

Data bindings Regarding the user interface, it was used the Data Binding Library. It is a support library that allows to bind UI components in layouts to data sources in the application using a declarative format rather than programmatically. Layouts are often defined in activities with code that calls UI framework methods. Binding components in the layout file lets remove many UI framework calls in activities, making them simpler and easier to maintain. This can also improve the application's performance and help prevent memory leaks and null pointer exceptions.

The expression language allows to write expressions that connect variables to the views in the layout. The Data Binding Library automatically generates the classes required to bind the views in the layout with desired data objects. The library provides features such as imports, variables, and includes that can be used in layouts. These features of the library coexist seamlessly with existing layouts. For example, the binding variables that can be used in expressions are defined inside a data element that is a sibling of the UI layout's root element. Both elements are wrapped in a layout tag.

The Data Binding Library provides classes and methods to easily observe data for changes. You don't have to worry about refreshing the UI when the underlying data source changes. Variables or their properties can be made observable through this library as well as objects, fields, or collections.

Chapter 3

Analysis

This chapter explains how the events, which subsequently has been implemented in the system, have been selected. First of all, a list of Android features has been drawn up. After a careful selection of the most important and used features, a sub list of these has been extracted.

3.1 Research stage

In this phase of the work, a list of all operations that users can perform with their smartphones has been compiled, by making a conceptual differentiation between trigger and action. Thus it has been searched for all the activities that can be considered as trigger and all the operations that can be considered as action. Some of the events found can be both triggers or actions, but others can only be triggers, for example:

- Profile mode: this event is detected when the user changes notification profile mode. In this case it acts as a trigger and it can be the one that causes the execution of an action (e.g. IF I change my phone profile to Vibration, then turn on bluetooth). But the same event (when it acts as an action) can be the consequence of another operation (e.g. if I go to the cinema, THEN change my phone profile to Silent).
- Time: this type of events can be only used as triggers, because they can happen but they can not be executed: IF every day at 8, then activate Wi-fi;

This research has been performed by searching the Web (consulting IFTTT site) for information about possible events for every feature and then these data have been compared with those on Android official documentation. From this, a first list has been filled (Tables from 3.6 to 3.12). These tables have four columns that

indicate the macro category and the category to which an event belongs (e.g. event: enable Wi-fi, category: Wi-fi, macrocategory: Connectivity), the type of the event (i.e. trigger or action) and optional parameters.

Some of the most important features described in the tables are:

- Battery level and device under charge or not, with battery level as parameter;
- Opening, closing, installing and uninstalling application, with application name as a parameter;
- Enabling, disabling, connecting and disconnecting Wi-fi, bluetooth, mobile data and airplane mode functionalities, with the name of the network/bluetooth device to which connecting of from which disconnecting, as a parameter;
- All events about the screen such as brightness setting, rotation, night mode, screen lock/unlock;
- Notification profile mode, jack plugged/unplugged, audio/music play/stop;
- Image/video/photo taken;
- Alarm/timer management;
- All notification received with the contact from which it comes and the content as parameters;
- Walking, running, moving on a vehicle and on a bicycle;
- Events repeated over time (e.g. every day at 10).

The process that the application uses to intercept operations, has to be automatic. Users have not to choose a trigger and pair it with an action, the application must do this for them. It has to detect every activity that users perform on their devices and store information about them.

This involves that every event has to be initially considered as a trigger even if it is not. Android application has to be able to get information about what has just happened whenever user performs an operation. In this case the trigger is any event occurring and the action is data storage (every information about the event is stored in a local database). Afterward the application has to analyze the collected data and finally it creates recommendations (if any) consistent with the just detected event.

3.2 Android features

After drawing up the list, it has been possible to start to collect information on Android Broadcast Receiver, Services, Listener and Content Observer. At the end of this refining stage, a subset of features that has been decided to implement for a first version of the application, has been extracted. However, this subset has changed during the development because of Android policy. Especially the actions list has been reduced because many of them require the root permission to be programmatically executed (Table 3.1 shows actions discarded).

Table 3.1. Android Actions not supported

Device	Reboot Restart Shutdown
Power save	Activate Deactivate
Video	Enable recording Disable recording
Photo	Take a photo Take a screenshot
Audio	Enable recording Disable recording
Screen	Unlock screen
Airplane mode	Activate Deactivate
Mobile data	Activate Deactivate

Table 3.2. Android Actions supported

Wifi	Connect to Disconnect from Activate Deactivate
Bluetooth	Connect to Disconnect from Activate Deactivate
Location	GPS activate
Profile	Normal Vibration Silent
Screen	Lock
Brightness	Set max lighting Set min lighting
Rotation	Activate Deactivate
Contact	Add Delete
Night mode	Activate Deactivate
Application	Launch Kill

For example, referring to Table 3.1, both airplane mode and mobile data features require explicit user authorization to be enabled/disabled, thus it can not be programmatically executed because the necessary permission can only be obtained by a system application.

Tables 3.2 and 3.3 show the sublist of triggers and actions finally supported.

Table 3.3. Android Triggers supported

Notification	From contact From app
Application	App launched App killed
Wifi	Connected to Disconnected from Activated Deactivated
Bluetooth	Connected to Disconnected from Activated Deactivated
Mobile data	Activated Deactivated
Airplane mode	Activated Deactivated
Screen	Locked Unlocked
Brightness	Level increased Level decreased
Night mode	Activate Deactivate
Rotation	Activated Deactivated

Profile	Normal Vibration Silent
Photo-Video	Photo taken Video taken Screenshot taken
Location	GPS activated GPS deactivated Enter a place Exit a place
Contact	Added Deleted
Power save mode	Activated Deactivated
Battery	Plugged Unplugged
Device	Turned on
Jack	Plugged Unplugged
Activity	Walking Running On bike On vehicle
Time	Every day at

3.3 Facebook service

The same process just described, has been done for online services, which first were Facebook, Instagram, Twitter, Whatsapp and Telegram. It has been decided to focus only on the Facebook service for a first version of the application.

Services such as Instagram, Telegram and Whatsapp have been detected only with notification managing (application name and optionally contact name have been extracted from notification). This can be done only for messages received and push notification.

The Facebook service has been developed differently from the previous online services because not all available events provide a notification to the user (e.g. when the user posts a status, any notification is received on her smartphone and on some devices, even posting a photo does not cause any notification). For this reason Facebook SDK has been used and its webhook has been implemented (more information can be found in Section 5.3).

Tables 3.4 and 3.5 show the sublist of triggers and actions finally supported.

Table 3.4. Facebook Triggers supported

Facebook	link posted
	photo posted
	video posted
	status posted
Messenger	message received

Table 3.5. Facebook Actions supported

Facebook	post link
	post photo
	post video
	post status

Every time the user posts something (link, photo, video, status) on her Facebook page or she receives a private message (as shown in Table 3.4), any other action can be executed as a consequence (according to the algorithm analysis which can be found in Section 5.2). In the same way, whenever an event triggers a Facebook action, the application can execute one of the operation shows in Table 3.5.

Chapter 4

Design

The work plan for this thesis was: designing the system, implementing and finally testing the application; after this steps new needs emerged from users feedback, for this reason some changes have been made and the process just described has been repeated. The algorithm which the application uses for suggesting rules has been revised, the code has been updated and new tests has been done.

The system follows a client-server architecture because the application which has to be developed, needs additional information, and its composition is described in this chapter. The main part of the project is also explained and it is described how the system architecture is composed and how the application is structured.

4.1 System architecture

Since this mobile application cannot reach the objective above mentioned without additional information coming from either the cloud or a custom-made server, the system follows a client-server architecture and it is composed by: a server, a mobile application as a client, one or more online services (e.g., Facebook), an online storage, and a cloud messaging platform. As shown in Figure 4.1, the server, in particular, hosts the EUPont ontology and a remote database while the mobile application is an Android application that hosts an additional local database.

After the user performs an operation on her smartphone, the application collect information locally and when is able to create an association between a Trigger and an Action, asks to the user if the association has to be saved or deleted. In the first case the application sends a request to the server and stores the information on an online database for analysis purpose. Regarding online services, instead, the work has been done for Facebook only[1].

[1] As said before, for a first version of the application it has been decided to focus on one service

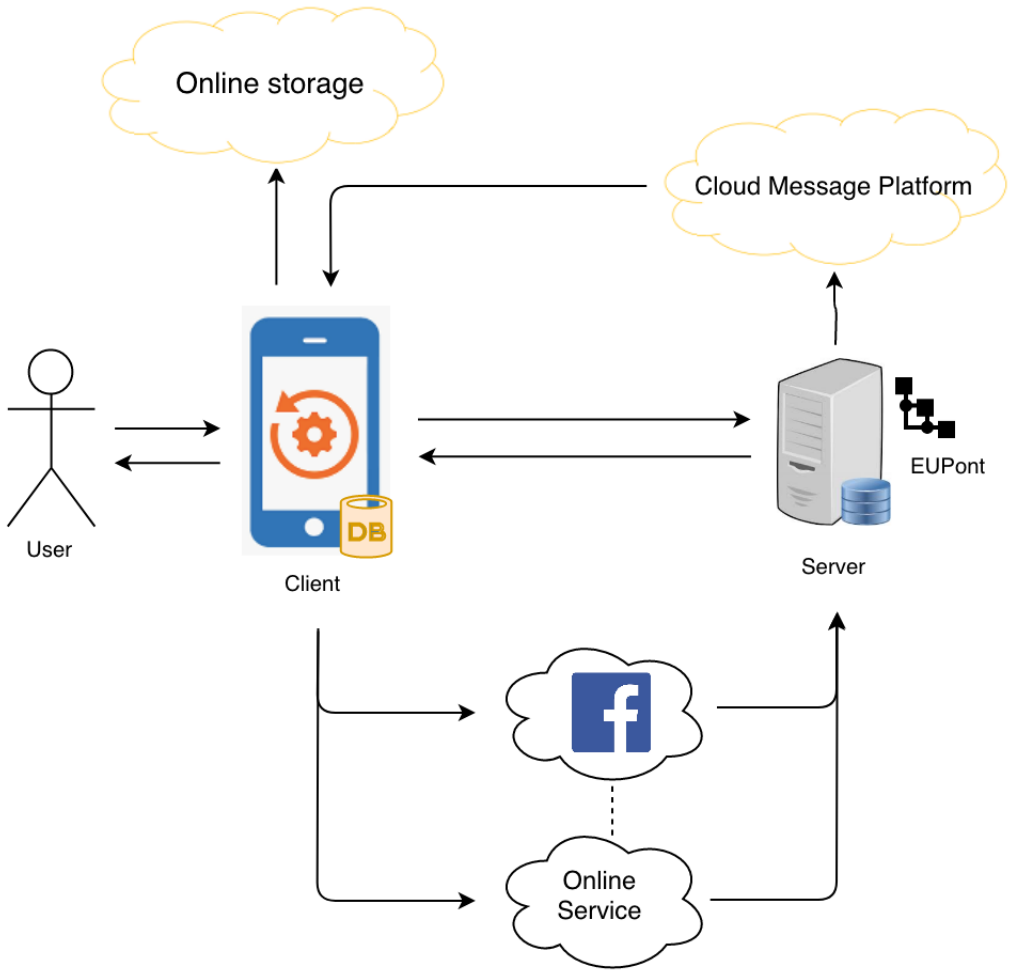

Figure 4.1. General architecture

Server It is responsible of managing requests coming from Android application, and storing data collected in the database. It also host the EUPont ontology interface which can support end users in the choice of the desired objects in order to exploit their functionality and improve the rule composition process.

Client It is the Android application that host a local database in which stores all data collected from every event occurring. When the algorithm studied for creating a trigger-event association creates a new rule, the application sends data about this rule to the server which stores them remotely.

Online services They can be services such as Facebook, Twitter, Instagram. In this work in particular, has been considered Facebook only. It is supported by the

server on which are managed requests sent by the callbacks provided by Facebook SDK (i.e. Webhook).

Cloud Messaging Platform It is used for sending messages to the application. This messages are sent by the Webhook and contain information about an event that the user has just made on Facebook. Thanks to them, the application can obtain this data and integrate them with all the others occurred previously on the smartphone.

Online storage It is used only for statistical purpose. For every user, information about rules saved/ignored/deleted are stored here and some analysis are performed.

4.2 Rule creation processes

There are two different operating mode for respectively the events associating to the device (i.e., Android) and online services (here exemplified for Facebook, but easily extendible to similar online services). This is because Android provides built-in tools for detecting events on the device, while for Facebook a Webhook is necessary.

Android features The process used for managing Android features rules creation, needs the server[2] for the communication of the initial information (user credential for registering/logging) and for the rules management (save/delete). Online storage is used only for collecting analysis information.

Below rule creation process is described in details, according to Figure 4.1:

1. The process starts with the registration (only the first time) or login (eventually for the following times) of the user through the application. After this, the server sends to the application a file with all data listing services and its triggers and actions. The application processes this data and performs the association with integer tags (more information can be found in Section 4.2).

2. Now the user can use her device and the application will register and store her operation on the local database.

3. When the application build a new rule, it warns the user with a notification.

4. If the user decides to save the suggested rule, its data will be sent to the server (which store them in the remote database).

[2]Available: https://dog.polito.it:8443/mobileEUP

5. Finally all information about rules saved/ignored/deleted are stored on the online storage for the analysis of the final results.

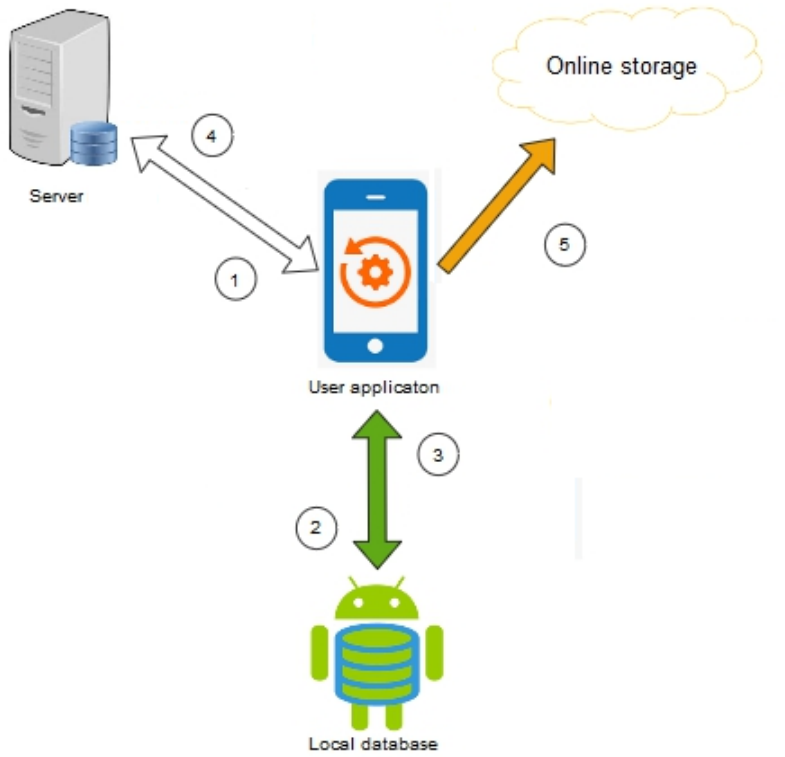

Figure 4.2. Android features process

Online service This process unlike the previous, uses the server even in the composition of a rule including the online serivce events. It is more complex because it includes an Application Server that can build message and send push notification to Android Client through a Cloud Message Platform. Messages are used for passing information to the application that stores them locally with the others previously collected on the smartphone. After the user registered and logged in the app, she has to connect her online service account to the app. Then the application can start collecting information from it. In this case, Facebook service is described in details (Figure 4.2) because is the only one considered in this study, but others services works in the same way, for example Twitter needs a webhook too.

Rule creation process for Facebook service:

1. It starts with the user that shares a post on Facebook;
2. The Webhook provided by Facebook SDK is activated and send a callback request to the server;
3. The server will send a request to the Cloud Message Platform and it will subscribe to a topic which is specific for every user, in this way the notifications are personal because referred to the user ID.
4. Cloud Message Platform sends a notification to the application, this is useful both for the rule creation process and when a Facebook trigger occurs.
5. Now the application can store Facebook operation data on the local database (point 2 of Figure 4.2).
6. When the application build a new rule, it warns the user with a notification (point 3 of Figure 4.2);
7. If the user decides to save the suggested rule, its data will be sent to the server (point 4 of Figure 4.2).
8. Finally all information about rules saved/ignored/deleted are stored on the online storage for the analysis of the final results (point 5 of Figure 4.2).

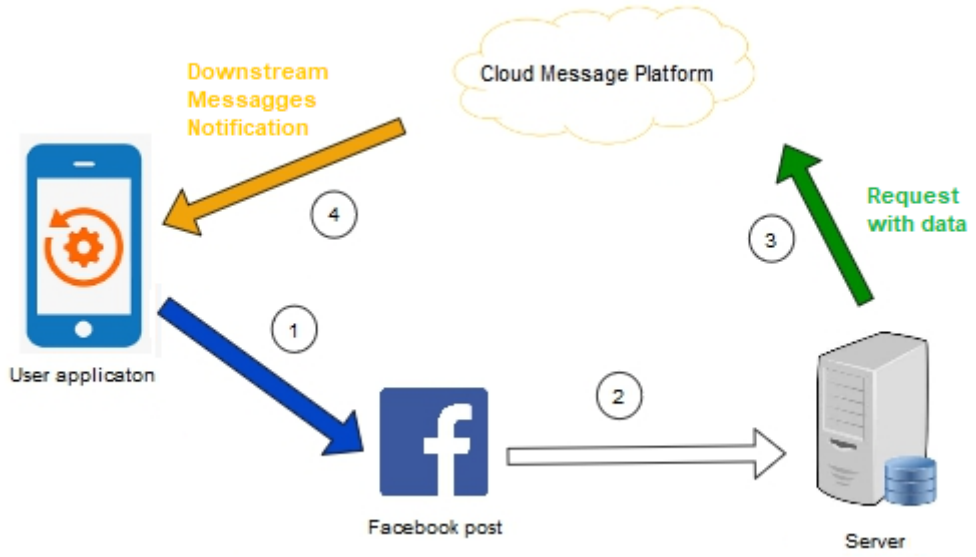

Figure 4.3. Facebook service architecture

When the user saves and activates a rule including a Facebook event:

- If the rule has a Facebook event as a trigger, then the corresponding action will be automatically executed when the application receives the notification from the Cloud Message Platform;
- If the rule has a Facebook operation as an action, then when the trigger is detected, the application will share a post/video/image/link on Facebook automatically.

4.3 Application structure

The most important element of this work is the Android application, and now its structure will be explained.

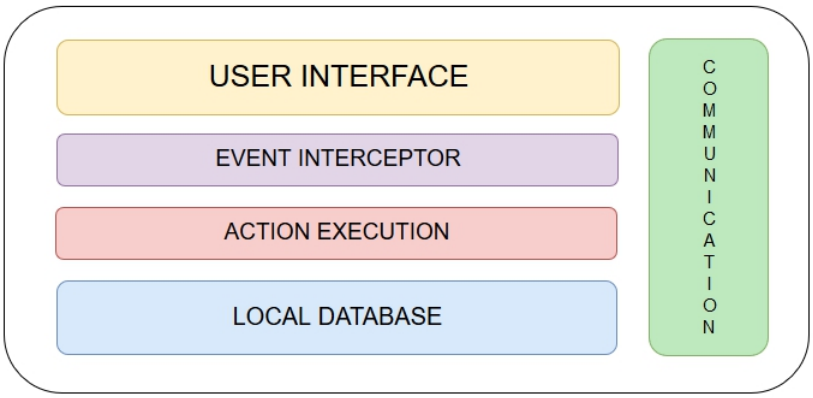

Figure 4.4. Application structure

The application is composed by five modules, referring to Figure 4.4:

User Interface: it is the front-end interface with which the user interacts, it allows her to insert her credentials for logging in the application, to navigate saved rules, to save new ones and to set preferences; it is very simple because the application largely runs in background, there are four views (login, home for rule list visualization, details of a selected rule, and settings where users can set preferences such as adding places to be monitored)[3];

Event Interception: it is responsible for detecting users activities (trigger) on their smartphone (more information can be found in Section 2.3)[4];

[3]Some images can be found in Appendix A

[4]in Android is implemented with Broadcast Receiver, Service, Content Observer

Action Execution: it is the module which is responsible for the execution an action.

Local Database: it is used for collecting all the information about the activities performed by users. It is hosted on the user device and it includes the following tables:

Table 4.1. Local Database Tables

Tables	**Details**
EventDB	timestamp, day of the year, hour of the day, eventName, param1
TriggerActionDB	uid, triggerName, param1T, actionName, param1A, quantity
RuleDB	uid, ruleID, ruleName, triggerActionId, attiva
ContactDB	uid, name, priority

Communication: this module is characterized by HTTP Requests, exchanged between server and client (it will be deepened in the following chapter) and it uses a Remote Database for data storage. The application receives objects in which are contained data that are stored in the remote database. Such data consists of information about the user (e.g., her username and her name), the available IoT devices and services, with the related details like the available triggers and actions.

Chapter 5

Implementation

According to the general architecture described in the previous chapter, the implementation of its components can now be described in-depth: Android application that manages operation made by the user and analyzes data collecting from these operations, through the algorithm which is its most important part; the communication component which describes the requests exchanged between client and server; Facebook Webhook as the implemented online service, that is managed by a Spring Boot application Server.

The server, instead, is composed by a Spring application connected to the EU-Pont ontology and to a remote database, but this elements were already present before the beginning of the work for this thesis;

5.1 Android application

This application wants to automatically create trigger-action rules, analyzing information that it collects from the use that the users make of their smartphone. It consists of a front-end part represented by the user interface and a second back-end part represented by the algorithm that is at the base of the application's operation. It underlies the application and it is responsible for the creation of the rules.

5.1.1 User interface

It allows a user to easily insert her credentials for logging in the application, to navigate the saved rules, to save new rules and to set preferences (e.g., the places to be monitored, Figure 5.4). It is very simple because the application largely runs in background. There are four main views: login (FIgure 5.1), home (for rule list visualization, shown in Figure 5.2), details for a selected rule (Figure 5.3) and settings.

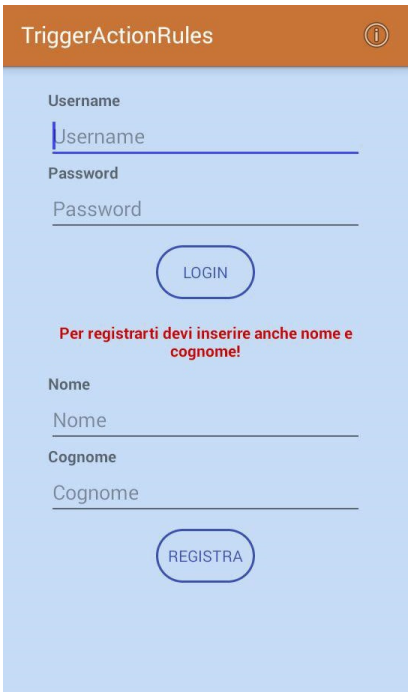

Figure 5.1. Application login

Figure 5.2. Application places list

Figure 5.3. Application rule details

Figure 5.4. Application home

5.1.2 Algorithm

This algorithm is responsible for the trigger-action association and, as a consequence, for the creation of the rules. It has been studied with the aim of being automatic, thus the user has not to chose which trigger associates to which action. Through an automatic evaluation of the data collected by the application, it is able to suggest a potentially significant rule.

After a first "release" of the application, it has been given to two users that have first tried it for a week. Thanks to their feedback, the algorithm has been slightly revised and some parameters has been modified. Thus, the need to add some specific feature emerged. The suggested rules were a lot but not very indicative, because the majority of them were focused for example on the switching on and off on the screen coupled with every other event. Furthermore it was modified the management of location events. It was initially searched during all day another equivalent event, once it was found, the counter was increased, then this approach has been changed and a parameter for time interval searching has been inserted.

It is now presented the final version of the just mentioned algorithm and its changes.

```
int MAX_QUANTITY_GENERAL = 4;
int MAX_QUANTITY_TIME = 2;
int MAX_QUANTITY_LOCATION = 2;
long TIME_INTERVAL_SEARCHING = 20 * 1000;
long TIME_INTERVAL_SEARCHING_LOCATION = 60 * 60 * 1000;
int CONTACT_PRIORITY = 50;
```

Referring to the code, this parameters have been modified:

- *MAX_QUANTITY_GENERAL* represents the quantity of repeated events for generic features, beyond which the suggestions are triggered, it has been raised from 2 to 4 because users complained about the high frequency of the suggested rules notification;

- *MAX_QUANTITY_TIME* and *MAX_QUANTITY_LOCATION* represents the amount of repeated events for time and location features respectively, beyond which the suggestions are triggered, they were created because time and location features needs a lower threshold respect to the generic ones (e.g. if the user enter to the same place twice, this can be significant but if the user turn on the screen twice, this is not so much relevant);

- *TIME_INTERVAL_SEARCHING* represents the interval in which generic events are searched, it was greater because it has been changed its conceptual

meaning[1];

- *TIME_INTERVAL_SEARCHING_LOCATION* represents the interval in which location events are searched, it has been added after this first test stage;
- *CONTACT_PRIORITY* represents the quantity of events collected for the "notification from a specific contact" feature, this parameter has been added because the users complained about the fact that a contact rarely contacted was treated like a frequent contacted one.

The following algorithm was used to gather information on user habits in order to set up statistics for creating rules.

Every time the user performs an action on his device (and this action is part of the collection of supported features[2]), the algorithm performs the following operations:

1. The information are stored in the EventDB[3] table;
2. Some checks for filtering events that are fired twice, are made;
3. The presence of rules previously activated, are checked and if there are triggers corresponding to the operation just performed by the user, all relative actions are executed;
4. A research within a set time interval is performed and the algorithm checks if:
   - The event just occurred is actually a trigger[4]
   - The first event found is actually an action[4]
   - The action found does not below to the same category[5] of the trigger one

   In this case it stores the trigger-action correlation, in the TriggerActionDB[6] table and if the occurrences of the correlation are higher than a threshold[7] (different if the event is related to time, location or all the others[8]), a new rule is suggested to the user;

[1]More details can be found in Chapter 4

[2]More information can be found in Chapter 3

[3]Defined in the previous chapter

[4]This is done through the data association, see section 5.3

[5]All categories are reported into tables from 3.6 to 3.12 of chapter 3

[6]Defined in the previous chapter

[7]See different parameters in the previous page

[8]See different implementation in the code in the next pages for more details

5. Upon reaching the pre-set threshold, the user is notified of the suggestion for the new rule and depending on the utility of this rule, he can:
   - Ignore it (and it will be temporarily excluded from the count of that association occurrences)
   - Save it on device (in RuleDB table) and on the server, then:
     * Turn it on
     * Turn it off
     * Delete it

The application uses the EUPont ontology accessible through RESTful API, to store rules, control devices and other contextual information for login (e.g. username, password and data registration).

timestamp	day	hour	eventName	param1
1537360038373	262	14	96	-
1537360079632	262	14	66	com.google.android.youtube
1537360090925	262	14	63	com.google.android.youtube
1537360103682	262	14	53	-
1537360215625	262	14	59	-
1537360244934	262	14	60	eduroam
1537360318173	262	14	69	Silenzioso

Figure 5.5. EventDB Room table

For example, observing Figure 5.2, when the user sets the profile tone to silent, the algorithm watches the previous event which is "connect to eduroam", it checks if it belongs to different category (the trigger belongs to wifi category, while the action belongs to profile category), in this case an association can be created in the second table (Figure 5.3).

The previous implementation is referred to events included in Android features. For time, location and fitness events there are different implementations of the algorithm for counting events.

- Time: recurring events (e.g. Every day at 10), whenever an event is detected, it is searched into local database if that particular event is present in previous days in the same hour interval, if it is a new rule is created.

uid	triggerName	param1T	actionName	param1A	quantity
1	63	com.shazam.android	69	Silenzioso	1
2	63	com.shazam.android	69	Normale	1
3	63	com.shazam.android	69	Vibrazione	1
4	64	com.spotify.music	69	Normale	1
5	64	com.spotify.music	69	Vibrazione	1
6	69	Silenzioso	60	eduroam	1

Figure 5.6. TriggerActionDB Room table

- Location: enter to/exit from a place indicated by the user, whenever an event is detected, it is verified if in the previous time interval a location event is present, if it is, the event detected will be the action and the location event will be the trigger[9].
- Activities (i.e. fitness in the code): walking/running/cycling/on vehicle events, whenever an event is detected, it is searched into local database if a fitness event is present in the previous hour[10].

The following code shows how this different implementation have been developed, the complete code can be found in Appendix A.4.2.

```
private void eventsAssociation() {
   // controllo che sia effettivamente un'azione
   String parentEventActionName = Dictionary.getActionID(getEventName());
   if (!parentEventActionName.equals("")) {
      // EVENTI TEMPO
      // dal db elenco gli eventi che hanno lo stesso orario di quello
          appena intercettato ma in giorni precedenti
      if (App.getDB().getEventsFromHour(getEventName(), getParam1(),
          getDay(), getHour()).size() > MAX_QUANTITY_TIME) {
         EventDB timeEvent = new EventDB(System.currentTimeMillis(),
             Dictionary.DICT_TIME_EVERY_DAY_AT, getHour() + "");
         addTriggerActionInDB(timeEvent, this, MAX_QUANTITY_TIME);
      }
```

[9]Google GeoFence API has been used for detecting the entries and exits

[10]Android ActivityTransition API has been used for detecting user activity

```
        // EVENTI LOCATION
        // escludo gli eventi della stessa categoria
        if (!parentEventActionName.split("_")[2].contains("gps")) {
           // cerco informazioni sull'entrata o l'uscita da un luogo
              nell'ultima ora
           EventDB e = App.getDB().getOnOffEvent(Dictionary.DICT_GPS_ENTER,
              Dictionary.DICT_GPS_EXIT,
                getTimestamp() - TIME_INTERVAL_SEARCHING_LOCATION,
                   getTimestamp());
           if (e != null) {
              addTriggerActionInDB(e, this, MAX_QUANTITY_LOCATION);
           }
        }
        // EVENTI FITNESS
        // escludo gli eventi della stessa categoria
        String split = parentEventActionName.split("_")[2];
        if (!split.contains("walking") && !split.contains("running") &&
            !split.contains("cycling") && !split.contains("vehicle")) {
           // cerco informazioni sull'inizio o la fine di una attivita'
              nell'ultima ora
           associateOnOffEvents(Dictionary.DICT_FITNESS_CAMMINATA_ON,
              Dictionary.DICT_FITNESS_CAMMINATA_OFF);
           associateOnOffEvents(Dictionary.DICT_FITNESS_CORSA_ON,
              Dictionary.DICT_FITNESS_CORSA_OFF);
           associateOnOffEvents(Dictionary.DICT_FITNESS_BICICLETTA_ON,
              Dictionary.DICT_FITNESS_BICICLETTA_OFF);
           associateOnOffEvents(Dictionary.DICT_FITNESS_MACCHINA_ON,
              Dictionary.DICT_FITNESS_MACCHINA_OFF);
        }
        // EVENTI FUNZIONALITA'
        assignTriggersAndActions(parentEventActionName);
     }
     // pulisco un po' la tabella di log, eliminando tutti gli eventi piu'
        vecchi di 72 ore
     App.getDB().deleteLog(getTimestamp() - TIME_INTERVAL_DELETING);
  }
```

5.1.3 Data association

When a new event is detected (as described in the algorithm section) after user interaction, it can not be known if that event is a Trigger or an Action, this distinction is applied later. For this reason, it has been necessary to create a common tag between the two types of event. Referring to the code[11], for example, *DICT_BLUETOOTH_ON* is related to the Trigger "if I turn bluetooth on" and to the Action "then turn bluetooth on".

```
public static final int DICT_BLUETOOTH_ON = 7;
public static final int DICT_BLUETOOTH_OFF = 8;

TRIGGERS.add(new Pair<>(DICT_BLUETOOTH_ON,
    ":trigger_android_bluetooth_enabled"));
TRIGGERS.add(new Pair<>(DICT_BLUETOOTH_OFF,
    ":trigger_android_bluetooth_disabled"));

ACTIONS.add(new Pair<>(DICT_BLUETOOTH_ON,
    ":action_android_bluetooth_enable"));
ACTIONS.add(new Pair<>(DICT_BLUETOOTH_OFF,
    ":action_android_bluetooth_disable"));
```

This association was performed with a series of sequential IDs, in this way every trigger-action pair is represented by a different integer. So when the user perform an operation on his device, the algorithm can store that event in the database and later it will decide if that event is a trigger or an action.

After this translation, every string with trigger name from EUPont ontology is paired to the integer tag and the same is done for the actions. As explained in Chapter 3, not every trigger can be an action, for this reason the two lists are not equivalent. Every time a rule is composed, first of all a research is performed in order to confirm that the trigger is included in the corresponding triggers list and the action is included in the corresponding actions list.

5.2 Communication

Since the server is a RESTful one, the database queries have been implemented with simple HTTP requests to the address *https://dog.polito.it:8443/mobileEUP*. In the application these request have been implemented using AsyncTask class that allows to perform background operations and publish results on the UI thread.

[11]The complete list for translation is reported in Appendix A.4.1

Figure 5.1 shows the sequence of requests done when the user register into the application for the first time or when she login into it. Request 3 after register/login return a file in JSON format which contains a list of all data about triggers and actions that the application needs.

Once the user is logged, the application continues to run in background and the operations in Figure 5.1 are not repeated until the user logout. Every time the user decides to save a rule, a POST Request is sent to the server, as well as a DELETE Request is sent when the user wants to delete a rule saved. The final Request is necessary if the user forgets her credentials and logs in the application with others, in this case data are bind to the device through the imei number and they are retrieved to the user when she log in.

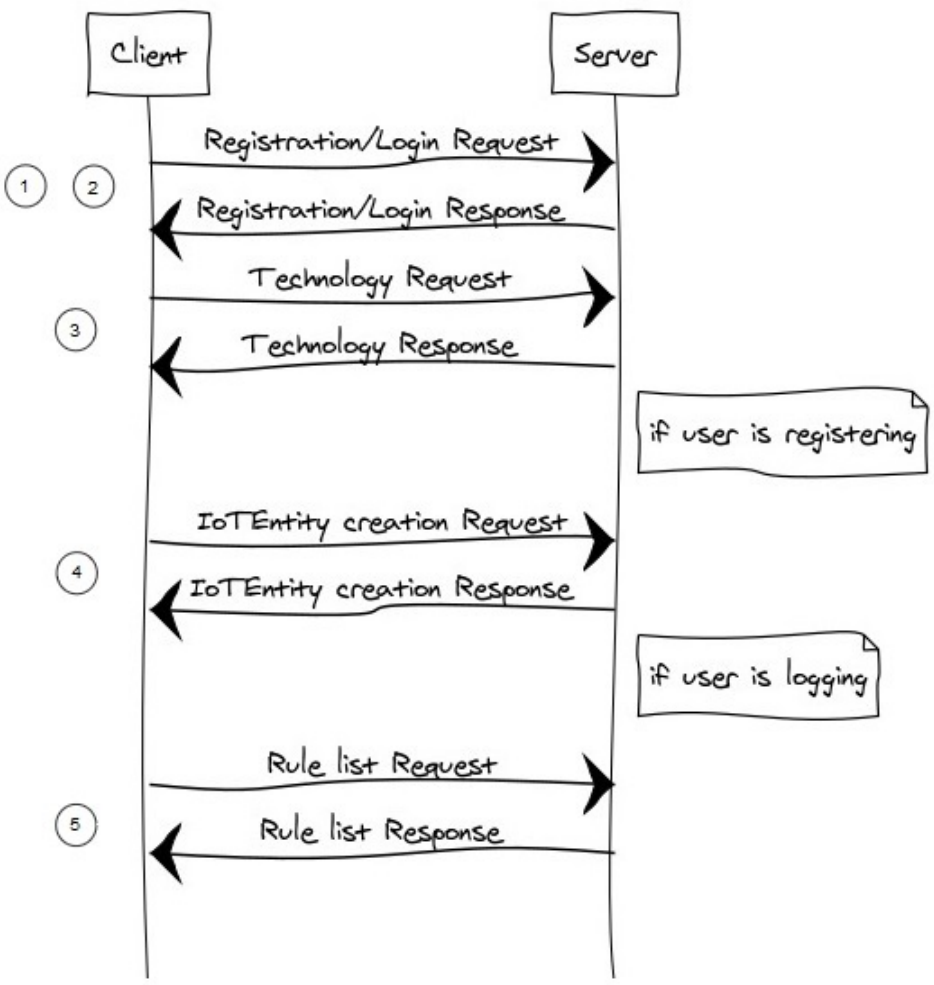

Figure 5.7. Registration/Login process

The application receives objects in which are contained data that are stored in the remote database which has the following tables structure:

- User: it describes a user who during the registration defines username, password, name, lastName;
- IOTEntity: it describes an entity with the list of all the name of the services;
- Technology: it has a list of services for a particular device or online service and their details;
- Service: it represents the list of Triggers and Actions for a particular technology;
- Trigger and Action which have the same structure: the entity which belongs to, the service which belongs to, a name, a short description for the user and optional parameters;
- Detail: it describes the optional parameters for Triggers and Action;
- Rule: it represents the association of a Trigger with an Action.

5.3 Webhook

As it can be read from Facebook documentation "Webhooks allows you to receive real-time HTTP notifications of changes to specific objects in the Facebook Social Graph". For example, a notification can be sent when a user share a photo. This prevents from having to query the Graph API for changes to objects that may or may not have happened.

When a Webhook is configured, an object type must be chosen and it must be subscribed to specific fields for that object type because different objects have different fields. Whenever there's a change to the value of any object field that has been subscribed to, a notification will be sent.

Apps can request approval for specific login permissions, which control the types of data the app can access when using the Graph API. The app must be approved for the login permission that corresponds to that type of data, and the object that owns the data must grant your app permission to access that data (e.g., a User allowing your app to access their feed). Although apps in development mode are automatically approved for all login permissions, they will not receive live Webhooks notifications unless the person who installed the app has an Admin, Developer, or Tester role on the app.

To use Webhooks, an endpoint must be set up on a server, then add and configure the Webhooks product in the app's dashboard. The endpoint[12] must be able to process two types of HTTP requests: Verification Requests and Event Notifications.

Verification Requests Anytime the Webhooks product is configured in App Dashboard, a GET request will be sent to the endpoint URL. Verification requests include the following query string parameters, appended to the end of the endpoint URL. Whenever the endpoint receives a verification request, it must:

- Verify that the *hub.verify_token* value matches the string set in the Verify Token field when configuring the Webhooks product in App Dashboard.
- Respond with the *hub.challenge* value.

```
@RequestMapping(value = "/webhook", method = RequestMethod.GET)
public ResponseEntity<String> verifyWebhook(
      @RequestParam(value = "hub.mode") String mode,
      @RequestParam(value = "hub.verify_token") String verify_token,
      @RequestParam(value = "hub.challenge") String challenge) {

   if (mode.equalsIgnoreCase("subscribe") &&
       verify_token.equalsIgnoreCase(TOKEN)) {
      return new ResponseEntity<>(challenge, HttpStatus.OK);
   }
   return new ResponseEntity<>("Push Notification ERROR!",
       HttpStatus.BAD_REQUEST);
}
```

Event Notifications Subscribing to specific fields on an object type, notifications will be sent as HTTP POST requests and will contain a JSON payload that describes the change, whenever there is a change to one of these fields. The endpoint should respond to all Event Notifications with 200 OK HTTPS. In this specific implementation, the part for Firebase Cloud Messaging is included. *Id* and *Time* values are extracted from JSON, the former is user identification code and it is used for the topic and the latter is the timestamp for the request identification. Then a notification is build with these information and it is sent to Firebase which is responsible to deliver the message to the user device through the application.

[12] *https://dog.polito.it:8443/mobileEUP/webhook*

```
@RequestMapping(value = "/webhook", method = RequestMethod.POST)
public ResponseEntity<String> sendPushNotification(
      HttpServletRequest request, @RequestBody String data)
      throws Exception {

   String firebaseResponse;

   JSONObject json_data = new
       JSONObject(data).getJSONArray("entry").getJSONObject(0);

   String field =
       json_data.getJSONArray("changes").getJSONObject(0).getString("field");
   Long time = json_data.getLong("time");
   String userID = json_data.getString("id");

   JSONObject body = new JSONObject();
   body.put("to", TOPIC + userID);
   body.put("priority", "high");

   JSONObject notification = new JSONObject();
   notification.put("title", field);
   notification.put("body", time);

   body.put("notification", notification);

   HttpEntity<String> httpRequest = new HttpEntity<>(body.toString());
   CompletableFuture<String> pushNotification =
       androidPushNotificationsService.send(httpRequest);
   CompletableFuture.allOf(pushNotification).join();
   try {
      firebaseResponse = pushNotification.get();
   } catch (Exception e) {
      e.printStackTrace();
      return new ResponseEntity<>("Push Notification ERROR!",
          HttpStatus.BAD_REQUEST);
   }

   System.out.println("GCM Notification is sent successfully: "
        + userID + " " + field);
   return new ResponseEntity<>(firebaseResponse, HttpStatus.OK);
}
```

Chapter 6

In-the-wild evaluation

This section presents the end-user evaluation procedure. After fixing some bugs and revising the application with the suggestion of a first couple of users, the real in-the-wild evaluation began.

6.1 Study procedure

During this step, the application was given to end users and the study of its behavior in the real world began.

The 6 participants has been recruited without any particular peculiarity, by telephone contact. The only characteristic that caused the restriction of user selection was the Android version. The application developed in this thesis supports up to Android Nougat (version 7.1.2, API level 25), because in Android Oreo (version 8, api level 26) important limitations on background processes have been added and causes the killing of the just mentioned processes, if not properly managed. Since it is based on background processes, this fact would have had a huge impact on the evaluation. However for a correct management of this problem, the application would have been completely rewritten according to the new specifications. For this reason it has been decided to make a first evaluation without this improvement and analyze the results, but reserving to decide to add it in the future.

After contacting the users, the introductory material has been sent to them: a link to an initial questionnaire, an explanation document, the apk of the application to be installed and, at the end of the test, the link to a final questionnaire. Both questionnaires have been done online, through Google Forms. The protocol followed provided that the user had to:

1. Read the document with the explanation of the scope of the application and some general concepts (Appendix A)

2. Complete the initial questionnaire (Appendix B) where they were asked about the functionalities which they most use, about their smartphone level usage and some personal information
3. Install the application and keep it for about 2 weeks
4. Complete the final feedback questionnaire (Appendix C) where they were asked to express their satisfaction about the application and optional feedback and comments for the application improvement

Users simply have to install the application, which after a period of "study", will begin to suggest rules that the user will have to accept or not, depending on the usefulness of them.

The period of "study" that the application requires, consists in storing the actions that each user performs with her device (according to the supported features) and put them in relation to each other two by two so that an action is consequent to an event, also taking into account the temporal component, because the associated events must have been performed in the same time interval, decided a priori. Once a certain threshold (even this decided a priori) of repetitions of the same pair has been exceeded, the user can be notified of the possibility of creating a new rule.

6.2 Measures

At the end of the test the user has been asked to fill in a questionnaire for any feedback and the following parameters has been evaluated:

- Total number of saved/deleted/ignored rules
- Number of saved/deleted/ignored rules per user
- Description of saved/deleted/ignored rules

These parameters have been analyzed in order to evaluate the real utility of the application, obtaining a feedback on the total distribution of the rules saved, deleted and ignored, while for the statistics on those per user, they have been integrated with the answers of the initial questionnaire in which you are asked to give preferences on the most used features. In both cases it is useful to know also the definition of these rules to understand which may be more useful and which less, therefore it is necessary to keep track of the names of the events from which they are composed. Data will also be collected by the final questionnaire, regarding:

- Possible problems and critical issues;
- Utility and usability of the application.

Chapter 7

Results

In this section the results of the work will be presented. This data have been collected with the help of Firebase Real Time Database and Google Forms.

7.1 Data discussion

As can be seen from Figure 7.1, the general distribution of the rules has been divided into two of the three possibilities (saved/deleted/ignored). As it is shown, no rule has been deleted and the trend is almost equally partitioned between rules saved and ignored. Considering Figure 7.2, where is represented the distribution of rules per user, it can be seen that even in this graph the trend is maintained except for two of the five users.

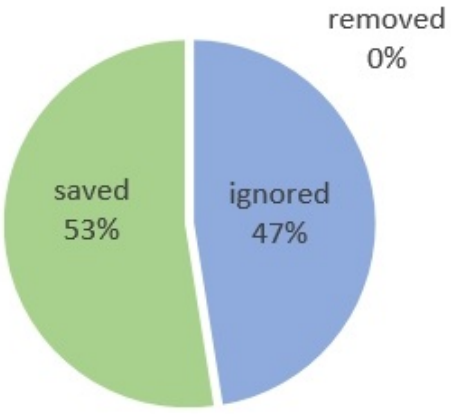

Figure 7.1. Distribution total rules

In the initial questionnaire was asked to the user how much she think to use her smartphone (on a scale from 1 to 5, where 1 is rarely used and 5 is very used) and comparing these answers with this graph, it can be observed that they are well integrated with the previous data (e.g. user 1 has answered 2 and user 5 has answered 4).

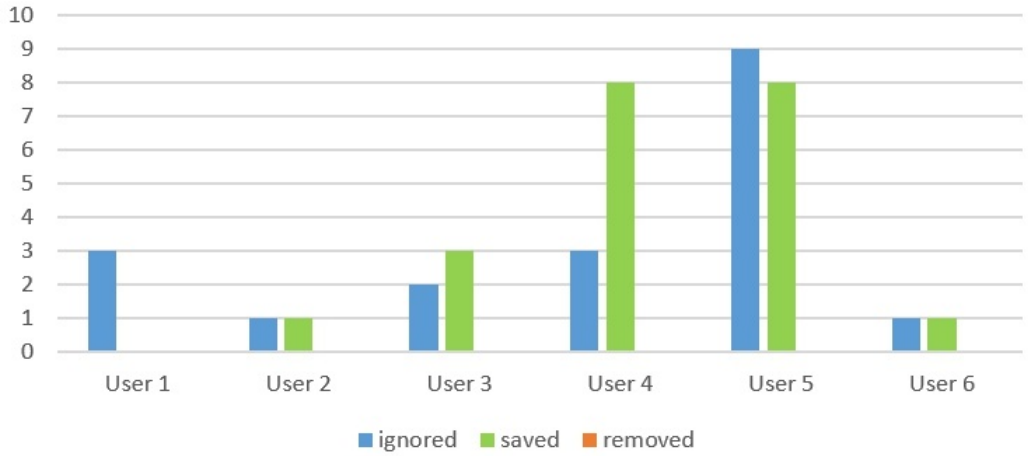

Figure 7.2. Distribution rules per user

Others observation have been done on the distribution of the features characterizing the rules monitored. Generally triggers are more scattered than actions, this can be brought back to the fact that actions list is reduced compared to triggers one[1].

In particular, considering Figure 7.3, triggers are mostly spread over wifi, gps, application and screen functions while actions are mostly spread over music, wifi and gps. This data are consistent with Figure 7.4, where users were asked to indicate the features that they mostly use (in the initial questionnaire). Wifi, Gps, Application (opening/closing application) in fact, emerge among these choices.

[1]More information can be found in Chapter 3

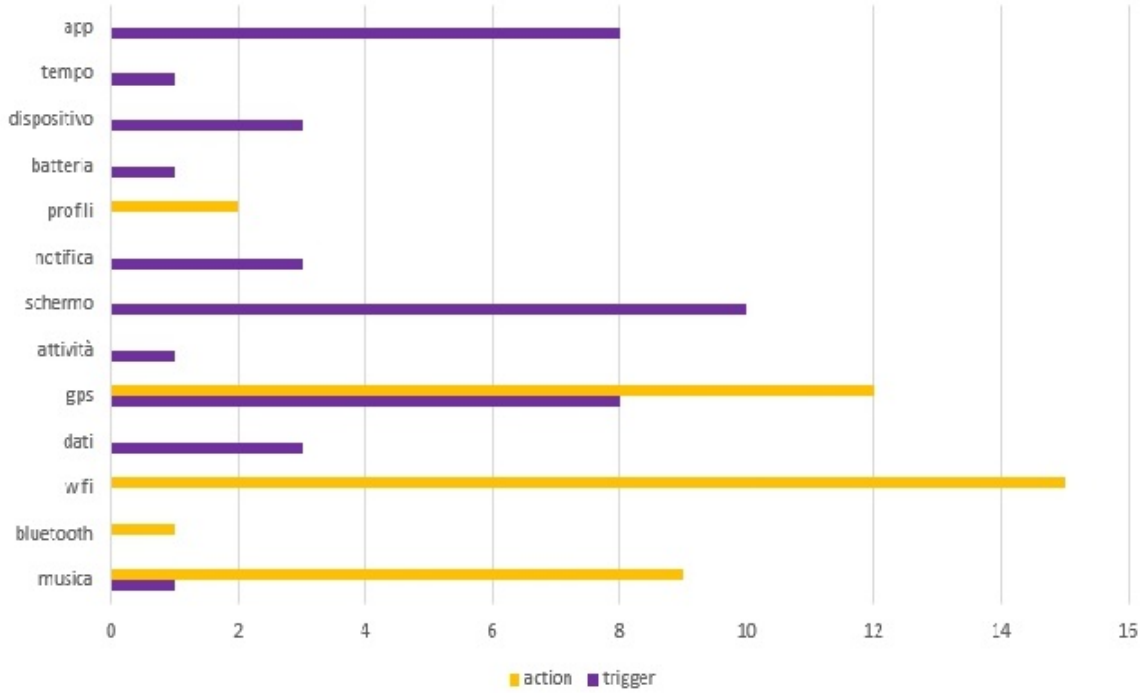

Figure 7.3. Distribution features: data form the evaluation

Figure 7.4. Distribution features: data from initial questionnaire

In the end, observing users answer, it can be seen that this kind of application has been rather appreciated (Figure 7.5) and they would recommend it to their friends (83.3% yes and 16.7% no), so this study can act as a basis for further expansion of this application.

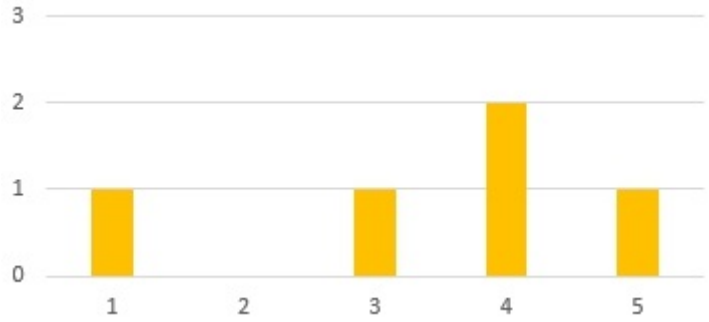

Figure 7.5. Application utility (1 = not useful, 5 = very useful)

The real results of all this thesis work are represented by Tables 7.1 and 7.2, where the detailed composition of every rule suggested by the application during the evaluation, is documented. For each and every user participating to the evaluation, the automatic trigger-action association made by the application has been stored with the judgment that the user has given about them (i.e. saved/ignored/deleted). Rules which have been saved from the user are highlighted to see which of them are considered useful by users.

It can be observed that among these users:

- the type of suggested rules changes in relation to their habits, there are no equally recurring rules for each user;
- the features which mostly recurs are Wifi one and GPS one and the rules built with them are the most appreciated;
- screen features (both lock and unlock) have always been ignored;
- enter/exit a place features have always been saved;
- the most significant and useful rules that users saved, were:
  - If I enter a place, then connect WiFi to;
  - If Mobile data Disabled, then WiFi Enable;
  - If I Enter a place, then Set audio profile to;

Another curious data is that although users have had the opportunity to connect their account to Facebook (in this way they could use Facebook webhook to monitor their activity on the social network), none of them chose to connect the profile.

Table 7.1. End-user evaluation results - part 1

Utente	Trigger	Action	type
1	Screen Unlocked	Connect WiFi to	ignored
	App killed	Connect WiFi to	ignored
	Screen Unlocked	Deactivate GPS	ignored
2	Device Turned on	Deactivate GPS	saved
	Notification from app	Stop Music	ignored
3	Screen Unlocked	Stop Music	ignored
	Device Turned on	Deactivate GPS	saved
	Smartphone recharged	Connect WiFi to	ignored
	App launched	Set audio profile to	saved
	App killed	Connect Bluetooth to	saved
	Screen Locked	Connect WiFi to	ignored
	App launched	Connect WiFi to	ignored
	App killed	Bluetooth Enable	ignored
4	Screen Unlocked	Deactivate GPS	ignored
	Walking	Deactivate GPS	saved
	Enter a place	Stop Music	saved
	Device Turned on	Deactivate GPS	ignored
	App killed	Activate GPS	saved
	App killed	Deactivate GPS	ignored
	Music On	Connect WiFi to	saved
	GPS deactivated	Music On	saved
	App killed	Deactivate GPS	saved
	Notification from contact	Stop Music	saved
	Notification from app	Stop Music	saved

Table 7.2. End-user evaluation results - part 2

Utente	Trigger	Action	type
5	Enter a place	Stop Music	saved
	Screen Unlocked	Activate GPS	ignored
	Enter a place	Connect WiFi to	saved
	Device Turned on	Deactivate GPS	ignored
	Screen Unlocked	Connect WiFi to	ignored
	Screen Unlocked	Stop Music	ignored
	App killed	WiFi Enable	saved
	Mobile data Disabled	Connect WiFi to	saved
	Enter a place	WiFi Enable	saved
	Mobile data Disabled	Connect WiFi to	ignored
	App killed	Deactivate GPS	ignored
	Mobile data Disabled	WiFi Enable	saved
	Every day at	Stop Music	ignored
	Screen Unlocked	Connect WiFi to	ignored
	Enter a place	Connect WiFi to	saved
	Enter a place	WiFi Enable	saved
	Screen Unlocked	WiFi Enable	ignored
6	Enter a place	Set audio profile to	saved
	App launched	Connect WiFi to	ignored

Chapter 8

Conclusions

The popularity of the Internet of Things (IoT) is expanding, even its complexity is growing and as a consequence, the number of possible combinations between different devices and services is increasing exponentially. Existing recommendation techniques could be useful to help end users without programming skills to use EUD systems, but have not yet been consistently explored. Recently researches has certainly studied recommendation technologies with the aim of assisting developers, but they have not deeply explored the possibility to extend their attention to assisting end users.

With this thesis it has been possible to analyze this second approach. End users have been involved into the "in-the-wild" evaluation of the application and they have been part of the improvement process, because of their feedback in the final stage of the work. It has been possible to explore which rules are largely composed, which rules do not raise interest in the user and if the user is well disposed towards the use of a mobile application which propose this kind of suggestion.

The final opinion was that this application would have great potential, because it has been largely appreciated. Users were enthusiast to try it and other people that were not participating to the study, were interested in possible future releases.

The main drawback is that since the application needs to monitor all activities which users perform on their device, it requires many services running in the background and this causes an intense battery consumption. Moreover this was even the main reason for which it was decided not to support Android version 8 (Oreo), because it would be necessary to change the implementation of the services according to the new standard of this new version, due to the fact that Android Oreo imposes important limits to background services for resource and battery saving. For this reason, the work has been focused on previous version and one of the possible improvements would be the optimization of this aspect and the integration with Android latest version.

8.1 Future works

Thanks to the work done with this thesis it will be possible to realize the integration of the developed application with the RecRules recommender system (a hybrid and semantic recommender system for EUD interfaces).

RecRules suggests trigger-action rules on the basis of their final functionality, to help end users to discover IoT applications. The recommendation process of RecRules united with the semantic information offered by EUPont, is used to model trigger-action rules in terms of functionality[10]. Recommendations may be used to suggest relevant smart "things" based on user preferences and interests and may be useful to optimize the time and cost of using IoT in a particular situation[9], RecRules suggest trigger-action rules that relate couple of IoT objects and this suggestions can be provided by EUD interfaces to facilitate end users in customizing their devices and services.

The integration with RecRules will allow to compute suggestions based on trigger-action rules previously activated by the user (content-based information), and based on trigger-action rules activated by other users (collaborative information).

Appendix A

Appendix

A.1 User information document

Grazie di aver accettato di prendere parte a questo studio. Ti chiedo gentilmente di compilare il questionario iniziale e dopo aver provato l'applicazione compilare il questionario finale con eventuali suggerimenti e commenti.

L'Internet of Things (IoT) è il modo in cui si chiama l'evoluzione di Internet, che da rete di computer diventa rete di oggetti, i quali si collegano, comunicano scambiandosi informazioni ed agiscono anche senza il controllo di un essere umano. Oggigiorno stanno nascendo sempre più applicazioni che propongono all'utente di gestire i propri oggetti (dispositivi e servizi IoT) tramite la creazione di "Regole" che implicano l'accoppiamento di un Evento ed una conseguente Azione, questa Azione è automaticamente eseguita nel momento in cui l'Evento desiderato viene rilevato (un esempio di regola potrebbe essere: Se entro in casa -> allora accendi wifi).

L'obiettivo di questo studio è quello di valutare un sistema di raccomandazione che suggerisca ed esegua delle Regole Evento-Azione basate sulle informazioni e le preferenze dell'utente, con lo scopo di assisterlo, in questo modo egli non si deve più preoccupare del sempre più complesso sistema di gestione delle funzionalità che sta alla base della creazione di queste regole e che spesso può risultare ostico per utenti senza alcun tipo di interesse o abilità nella programmazione, ma dovrà solamente decidere quale delle regole suggerite gli sono più utili.

Questa applicazione in particolare, studierà e terrà traccia delle azioni compiute dall'utente, in modo da creare delle statistiche che utilizzerà per suggerire delle regole. Una volta raggiunta una certa soglia di ripetizioni effettuate sulla stessa coppia evento-azione, all'utente verrà notificata la regola (Figura A.1) e verrà chiesto se la vuole salvare (nel caso la reputi utile) oppure ignorare (nel caso non avesse senso). Successivamente, se l'utente sceglie di salvare la regola, potrà vedere un elenco di tutte quelle salvate in precedenza ed ancora decidere quali di queste attivare

o disattivare (Figura A.2). Dal momento in cui le regole saranno attive, verranno eseguite in automatico, senza che il proprietario del dispositivo debba più compiere nessuna azione.

La prova durerà 2 settimane, come prima cosa ti sarà chiesto di acconsentire a dei permessi, poi dovrai crearti un'utenza, quindi scegliere username, password ed inserire nome e cognome. Dopo aver fatto ciò dovrai inserire almeno un indirizzo per permettere all'applicazione di monitorare entrata/uscita da tale luogo. Mi raccomando, controlla che l'icona del servizio sia sempre attiva nella barra delle notifiche (se hai installato qualche applicazione per il monitoraggio della batteria o dei servizi in background, potrebbe decidere di chiudertela) e se decidi di associare il tuo profilo Facebook, avvisami perchè devi essere abilitato/a!

Di seguito sono riportate le funzionalità supportate (che si possono trovare anche nella sezione "informazioni" dell'applicazione):

- **Trigger**
    - Facebook (post condiviso)
    - Attività (camminata/corsa/in macchina/in bicicletta)
    - Batteria (connesso/non connesso alla corrente)
    - Bluetooth (on/off/connesso a/disconnesso da)
    - Camera (nuovi foto/screenshot/video)
    - Contatti (nuovi/aggiornati/rimossi)
    - Cuffie (inserite/rimosse)
    - Location (entrata/uscita)
    - Gps (on/off)
    - Mobile data (on/off)
    - Music (on/off)
    - Notifiche da contatti
    - Notifiche applicazioni (Facebook/Whatsapp/Telegram/Instagram/Gmail)
    - Profili audio (normale/vibrazione/silenzioso)
    - Applicazioni (installata/disinstallata/aperta/chiusa)
    - Blocco schermo (on/off)
    - Modalità aereo (on/off)
    - Rotazione schermo (on/off)
    - Sms (ricevuti/inviati)

- Wifi (on/off/connesso a/disconnesso da)
- Tempo (ogni giorno alle)

- **Action**
  - Facebook (condividi post/foto/video)
  - Bluetooth (on/off/connetti a/disconnetti da)
  - Contatti (aggiungi/rimuovi)
  - Profili audio (normale/vibrazione/silenzioso)
  - Rotazione (on/off)
  - Wifi (on/off/connetti a/disconnetti da)

L'applicazione chiederà di mandare una mail con il log degli eventuali errori che causeranno dei crash.

Ecco il link al questionario iniziale: https://goo.gl/forms/mgFqDLsBS5T7spcI2

Figure A.1. Application rule details

Figure A.2. Application home

A.2 Initial questionnaire

Da compilare prima dell'instalazione dell'applicazione

1. Età:
2. Genere:
    - o Uomo
    - o Donna
3. Occupazione:
4. Grado di utilizzo del cellulare (1= lo uso molto poco, 5 = lo uso tanto):
    - o 1
    - o 2
    - o 3
    - o 4
    - o 5
5. Seleziona le funzionalità che usi di più:
    - o Facebook
    - o Batteria
    - o Bluetooth
    - o Camera
    - o Contatti
    - o Cuffie
    - o Gps
    - o Mobile data
    - o Musica
    - o Notifiche
    - o Profili audio
    - o Applicazioni
    - o Blocco schermo
    - o Modalità aereo

- o Rotazione schermo
- o Sms
- o Wifi

6. Hai l'applicazione ufficiale di Facebook installata?

   - o Si
   - o No

A.3 Final questionnaire

Da compilare una volta finita la prova dell'applicazione

1. Username (scelto nella registrazione dell'app):
2. Pensi che le funzionalità supportate si sposino con le tue abitudini di utilizzo?
    - o Si, per la maggiorparte
    - o No, molte funzionalità mancano
    - o No, le funzionalità supportate non mi sono utili
3. Quanto hai trovato utile questa applicazione? (1= poco, 5 = tanto):
    - o 1
    - o 2
    - o 3
    - o 4
    - o 5
4. Consiglieresti questa applicazione ai tuoi amici?
    - o Si
    - o No
5. Suggerimenti per altre funzionalità:
6. Altri commenti:

Appendix B

Tables

Table B.1. Triggers first list - part 1

Macro category	Category	Trigger	Parameter
Battery	Battery	Low battery level	Level
		Device is unplugged	-
		Device is plugged in	-
		App consumes	Level
Applications	Application	App installed	Application name
		App uninstalled	Application name
		App launched	Application name
		App killed	Application name
Connectivity	Wifi	Connected to	Name
		Disconnected from	Name
		Activated	-
		Deactivated	-
		Scan	-
	Bluetooth	Connected to	Name
		Disconnected from	Name
		Activated	-
		Deactivated	-
		Scan	-
	Mobile data	App consumes	Level
		Activated	-
		Deactivated	-
	Airplane mode	Activated	-
		Deactivated	-

Table B.2. Triggers first list - part 2

Macro category	**Category**	**Trigger**	**Parameter**
Display	Screen	Locked	-
		Unlocked	-
	Lighting	Level increased	-
		Level decreased	-
	Night mode	Set	-
		Unset	-
	Rotation	Activated	-
		Deactivated	-
		Device wallpaper updated	-
Sound	Profile	Set to	Profile
	Jack	Plugged in	-
		Plugged off	-
	Audio	Audio recorded	-
		Music in progress	-
		Music stopped	-
Media	Photo-Video	Photo taken	-
		Screenshot taken	-
		Video recorded	-
		Image saved	-
Storage	Storage	File Saved on device	-
		Low phone storage space	-
	Drive	Document added	-
	Dropbox	Photo added	-
		Document updated	-
Organizer	Alarm	Added	-
		Deleted	-
	Timer	Added	-
		Deleted	-
	Calendar	Event added	-
		Event deleted	-
		Event updated	-
Locations	Location	GPS activated	-
		GPS deactivated	-
		Enter a place	Place
		Exit a place	Place

Table B.3. Triggers first list - part 3

Macro category	Category	Trigger	Parameter
Transportation	Move	On bike	-
		On car	-
		On foot	-
Activities	Walking	Start	-
		Stop	-
	Running	Start	-
		Stop	-
	Cycling	Start	-
		Stop	-
Communication	Notifications	Notification app received	App name
		Notification contact received	Contact name
	Not disturb	Enable	-
		Disable	-
	Social	new link post	hashtag
		new photo post	hashtag
	Facebook	new video post	hashtag
	Instagram	new post	-
	Twitter	new private message	mittente, stringa
		new message sent	destinatario, stringa
		You are tagged in a photo	-
		Private message contains	-
	Messaging	New SMS	-
	Whatsapp	SMS contains	Value
	Telegram	New Email	-
		New Email Attachment	-
		Email contains	Value
		New chat message	-
		Chat message sent	-
		Chat message contains	Value
		New photo from a chat	-
		New audio from a chat	-
		New video from a chat	-

Table B.4. Triggers first list - part 4

Macro category	**Category**	**Trigger**	**Parameter**
	Phone call	Call answered	-
		Call rejected	-
		Call muted	-
		Call started	-
		Call ended	-
		Speakerphone activated	-
		Call missed	-
Contacts	Contacts	New contact added	-
		Contact deleted	-
Device	Device	Turned on	-
	Power save	Activated	-
		Deactivated	-
News	News	News available	-
		News matches search	Value
		Tomorrow's weather report	-
		Current temp drops below	Value
		Current temp rises above	Value
		Current condition changes to	Value
		Tomorrow's high rises above	Value
		Tomorrow's low drops below	Value
		Tomorrow's forecast calls for	Value
Environment	Ambient light	Rises above	Value
		Drops below	Value
	Temperature	Rises above	Value
		Drops below	Value
Time	Time	Every day at	Time
		Every hour at	Time
		Every week at	Time
		Every month on the	Time
		Every year on	Time

Table B.5. Actions first list - part 1

Macro category	**Category**	**Action**	**Parameter**
Applications	Application	Launch	Name
		Kill	Name
Notifications	Not disturb	Enable	-
		Disable	-
	Notification	Send autonotification	-
		Notification from application	Application name
Connectivity	Wifi	Enable	-
		Disable	-
		Connect to	Name
		Disconnect from	Name
	Bluetooth	Enable	-
		Disable	-
		Connect to	Name
		Disconnect from	Name
	Airplane mode	Enable	-
		Disable	-
	Mobile data	Enable	-
		Disable	-
Display	Rotation	Enable	-
		Disable	-
	Screen	Lock screen	-
		Unlock screen	-
		Set lighting level	Level
		Set night mode	-
		Update wallpaper	Image
		Blink the notification led	Color
Sound	Volume	Set media volume	Volume
		Set call volume	Volume
		Set ringtone volume	Volume
		Set alarm volume	Volume
	Audio profile	Set to	Profile

Table B.6. Actions first list - part 2

Macro category	**Category**	**Action**	**Parameter**
Audio	Audio	Enable recording	-
		Disable recording	-
	Music	Enable	-
		Disable	-
		Play specific song	Song
		Next song	-
		Previous song	-
Photo-Video	Video	Enable recording	-
		Disable recording	-
	Photo	Take a photo	-
		Take a screenshot	-
Storage	Storage	Move app to ext storage	App name
		Clear Download folder	-
		Create Document	Doc name
		Append to a document	Doc name, Text
		Add row to a spreadsheet	Doc name
		Delete file	Doc name
		Share	Doc name
Social	Facebook	New status	Text
		Upload photo	Image
		Upload video	Video
		Send private message	Text, Receiver
	Twitter	New status	Text
		Upload photo	Image
		Hashtag	Text
	Instagram	Upload photo	Image
		Upload video	Video
	Spotify	Add track to playlist	Track
Messaging	Email	Send email	Text, Mail
	Sms	Send SMS	Text, Number
	Whatsapp	Send chat message	Text, App, Number
	Telegram	Send photo	Text, App, Number
		Send video	Text, App, Number
		Send audio	Text, App, Number

Table B.7. Actions first list - part 3

Macro category	Category	Action	Parameter
Phone call	Call	Answer	-
		Reject	-
		Mute the incoming call	-
		Start	Number
		End	-
		Block calls	Number
		Active speakerphone	-
Organizer	Alarm	Enable	Time
		Disable	Time
	Timer	Enable	Duration
		Disable	Duration
	Countdown	Enable	Duration
		Disable	Duration
	Calendar	Add a calendar event	Details (Title, Time, ...)
		Delete a calendar event	Title
Locations	GPS	Activate	-
		Deactivate	-
		Get traffic information	Position
		Get weather forecast	Position
Contacts	Contact	Create new contact	Name, Number
		Delete old contact	Name
Device	Device	Reboot	-
		Restart	-
		Shutdown	-
	Power save	Enable	-
		Disable	-
Utility	Note	Create note	Title
		Append note	Text, Title
		Delete note	Title

B.1 Code

B.1.1 Dictionary

```
public class Dictionary {

    private static final Set<Pair<Integer, String>> TRIGGERS = new
        HashSet<>();
    private static final Set<Pair<Integer, String>> ACTIONS = new
        HashSet<>();
    private static final Set<Pair<Integer, String>> PROFILES = new
        HashSet<>();
    private static Set<Service> SERVICES = new HashSet<>();
    public static Set<Technology> TECNOLOGIES = new HashSet<>();
    public static Set<CustomPlace> AREA_LANDMARKS = new HashSet<>();

    public static final int DICT_NOTIFICATIONS_ON = 1;
    public static final int DICT_NOTIFICATIONS_OFF = 2;
    public static final int DICT_NOTIFICATIONS_SEND = 3;
    public static final int DICT_NOTIFICATIONS_RECEIVED_FROM_APP = 4;
    public static final int DICT_NOTIFICATIONS_RECEIVED_FROM_CONTACT = 5;
    public static final int DICT_BLUETOOTH_ON = 7;
    public static final int DICT_BLUETOOTH_OFF = 8;
    public static final int DICT_BLUETOOTH_DISCONNECTED_FROM = 9;
    public static final int DICT_BLUETOOTH_CONNECTED_TO = 10;
    public static final int DICT_BLUETOOTH_SCAN = 11;
    public static final int DICT_TIMER_ADDED = 12;
    public static final int DICT_TIMER_DELETED = 13;
    public static final int DICT_TIMER_ON = 14;
    public static final int DICT_TIMER_OFF = 15;
    public static final int DICT_ALARM_ADDED = 16;
    public static final int DICT_ALARM_DELETED = 17;
    public static final int DICT_ALARM_ON = 18;
    public static final int DICT_ALARM_OFF = 19;
    public static final int DICT_PHOTO_TAKEN = 20;
    public static final int DICT_VIDEO_TAKEN = 21;
    public static final int DICT_SCREENSHOT_TAKEN = 22;
    public static final int DICT_SMS_IN = 23;
    public static final int DICT_SMS_OUT = 24;
    public static final int DICT_SCREEN_LOCKED = 26;
    public static final int DICT_SCREEN_UNLOCKED = 27;
    public static final int DICT_NIGHT_MODE_ON = 28;
    public static final int DICT_NIGHT_MODE_OFF = 29;
```

```
public static final int DICT_ROTATION_ON = 30;
public static final int DICT_ROTATION_OFF = 31;
public static final int DICT_WALLPAPER = 32;
public static final int DICT_SET_BRIGHTNESS = 33;
public static final int DICT_BRIGHTNESS_INCREASED = 34;
public static final int DICT_BRIGHTNESS_DECREASED = 35;
public static final int DICT_DATA_HIGH_CONSUMPTION = 36;
public static final int DICT_DATA_ON = 37;
public static final int DICT_DATA_OFF = 38;
public static final int DICT_DEVICE_ON = 39;
public static final int DICT_DEVICE_RESTART = 40;
public static final int DICT_DEVICE_REBOOT = 41;
public static final int DICT_DEVICE_SHUTDOWN = 42;
public static final int DICT_STORAGE_HIGH_CONSUMPTION = 43;
public static final int DICT_STORAGE_FILE_SAVED = 44;
public static final int DICT_STORAGE_DELETE_FILE = 45;
public static final int DICT_STORAGE_IMAGE_SAVED = 46;
public static final int DICT_STORAGE_CLEAR_DOWNLOAD = 47;
public static final int DICT_STORAGE_MOVE_APP = 48;
public static final int DICT_STORAGE_SHARE_FILE = 49;
public static final int DICT_LIGHTING_RISES = 50;
public static final int DICT_LIGHTING_DROPS = 51;
public static final int DICT_GPS_ON = 52;
public static final int DICT_GPS_OFF = 53;
public static final int DICT_GPS_EXIT = 54;
public static final int DICT_GPS_ENTER = 55;
public static final int DICT_AIRPLANE_ON = 56;
public static final int DICT_AIRPLANE_OFF = 57;
public static final int DICT_WIFI_ON = 58;
public static final int DICT_WIFI_OFF = 59;
public static final int DICT_WIFI_CONNECTED_TO = 60;
public static final int DICT_WIFI_DISCONNECTED_FROM = 61;
public static final int DICT_WIFI_SCAN = 62;
public static final int DICT_APP_INSTALLED = 63;
public static final int DICT_APP_LAUNCHED = 64;
public static final int DICT_APP_KILLED = 65;
public static final int DICT_APP_UNINSTALLED = 66;
public static final int DICT_TEMPERATURE_RISES = 67;
public static final int DICT_TEMPERATURE_DROPS = 68;
public static final int DICT_NOTIFICATIONS_PROFILE_SET = 69;
public static final int DICT_VOLUME_SET_PROFILE = 69;
public static final int DICT_VOLUME_SET_CALL = 70;
public static final int DICT_VOLUME_MUTE_PHONE = 71;
```

```
public static final int DICT_VOLUME_SET_RINGTONE = 72;
public static final int DICT_VOLUME_SET_MEDIA = 73;
public static final int DICT_VOLUME_SET_ALARM = 74;
public static final int DICT_BATTERY_PLUGGED = 75;
public static final int DICT_BATTERY_UNPLUGGED = 76;
public static final int DICT_BATTERY_POWER_MODE_ON = 77;
public static final int DICT_BATTERY_POWER_MODE_OFF = 78;
public static final int DICT_BATTERY_LOW = 79;
public static final int DICT_BATTERY_HIGH_CONSUMPTION = 80;
public static final int DICT_TIME_EVERY_DAY_AT = 81;
public static final int DICT_TIME_EVERY_WEEK_AT = 82;
public static final int DICT_TIME_EVERY_YEAR_AT = 83;
public static final int DICT_TIME_EVERY_MONTH_AT = 84;
public static final int DICT_SPEAKERPHONE_ON = 85;
public static final int DICT_SPEAKERPHONE_OFF = 101;
public static final int DICT_CALL_MUTED = 86;
public static final int DICT_CALL_REJECTED = 87;
public static final int DICT_CALL_ANSWERED = 88;
public static final int DICT_CALL_STARTED = 89;
public static final int DICT_CALL_ENDED = 90;
public static final int DICT_CALL_BLOCKED = 91;
public static final int DICT_CALL_MISSED = 92;
public static final int DICT_AUDIO_RECORDED = 93;
public static final int DICT_MUSIC_ON = 94;
public static final int DICT_MUSIC_OFF = 95;
public static final int DICT_JACK_ON = 96;
public static final int DICT_JACK_OFF = 97;
public static final int DICT_PREVIOUS_SONG = 98;
public static final int DICT_NEXT_SONG = 99;
public static final int DICT_PLAY_SONG = 100;
public static final int DICT_CONTACT_ADD = 110;
public static final int DICT_CONTACT_UPDATE = 111;
public static final int DICT_CONTACT_DELETE = 112;
public static final int DICT_FITNESS_CAMMINATA_ON = 113;
public static final int DICT_FITNESS_CAMMINATA_OFF = 114;
public static final int DICT_FITNESS_CORSA_ON = 115;
public static final int DICT_FITNESS_CORSA_OFF = 116;
public static final int DICT_FITNESS_BICICLETTA_ON = 117;
public static final int DICT_FITNESS_BICICLETTA_OFF = 118;
public static final int DICT_FITNESS_MACCHINA_ON = 119;
public static final int DICT_FITNESS_MACCHINA_OFF = 120;

public static final int DICT_FACEBOOK_NEW_POST = 200;
```

```
public static final int DICT_FACEBOOK_NEW_POST_LINK = 201;
public static final int DICT_FACEBOOK_NEW_POST_VIDEO = 202;
public static final int DICT_FACEBOOK_NEW_POST_PHOTO = 203;
public static final int DICT_FACEBOOK_PRIVATE_MSG_IN = 204;
public static final int DICT_FACEBOOK_PRIVATE_MSG_OUT = 205;

public static void initialize() {
    triggers();
    actions();
    ringerMode();
    places();
    jsonInit();
}

public static void jsonInit() {
    DeviceEntity d = App.get().readFromJson();
    if (d != null) {
        TECNOLOGIES.clear();
        SERVICES.clear();
        for (Technology tech : d.getTecnologies()) {
            TECNOLOGIES.add(tech);
            SERVICES.addAll(tech.getServices());
        }
    }
}

private static void triggers() {
    TRIGGERS.clear();
    TRIGGERS.add(new Pair<>(DICT_NOTIFICATIONS_RECEIVED_FROM_APP,
        ":trigger_android_notifications_received_from_app"));
    TRIGGERS.add(new Pair<>(DICT_NOTIFICATIONS_RECEIVED_FROM_CONTACT,
        ":trigger_android_notifications_received_from_contact"));
    TRIGGERS.add(new Pair<>(DICT_BLUETOOTH_ON,
        ":trigger_android_bluetooth_enabled"));
    TRIGGERS.add(new Pair<>(DICT_BLUETOOTH_OFF,
        ":trigger_android_bluetooth_disabled"));
    TRIGGERS.add(new Pair<>(DICT_BLUETOOTH_DISCONNECTED_FROM,
        ":trigger_android_bluetooth_disconnected_from"));
    TRIGGERS.add(new Pair<>(DICT_BLUETOOTH_CONNECTED_TO,
        ":trigger_android_bluetooth_connected_to"));
    TRIGGERS.add(new Pair<>(DICT_PHOTO_TAKEN,
        ":trigger_android_media_photo_taken"));
```

```
TRIGGERS.add(new Pair<>(DICT_VIDEO_TAKEN,
    ":trigger_android_media_video_taken"));
TRIGGERS.add(new Pair<>(DICT_SCREENSHOT_TAKEN,
    ":trigger_android_media_screenshot_taken"));
TRIGGERS.add(new Pair<>(DICT_SMS_IN,
    ":trigger_android_sms_received"));
TRIGGERS.add(new Pair<>(DICT_SMS_OUT,
    ":trigger_android_sms_sent"));
TRIGGERS.add(new Pair<>(DICT_SCREEN_LOCKED,
    ":trigger_android_screen_locked"));
TRIGGERS.add(new Pair<>(DICT_SCREEN_UNLOCKED,
    ":trigger_android_screen_unlocked"));
TRIGGERS.add(new Pair<>(DICT_NIGHT_MODE_ON,
    ":trigger_android_screen_night_mode_enabled"));
TRIGGERS.add(new Pair<>(DICT_NIGHT_MODE_OFF,
    ":trigger_android_screen_night_mode_disabled"));
TRIGGERS.add(new Pair<>(DICT_ROTATION_ON,
    ":trigger_android_screen_rotation_enabled"));
TRIGGERS.add(new Pair<>(DICT_ROTATION_OFF,
    ":trigger_android_screen_rotation_disabled"));
TRIGGERS.add(new Pair<>(DICT_BRIGHTNESS_INCREASED,
    ":trigger_android_screen_brightness_increased"));
TRIGGERS.add(new Pair<>(DICT_BRIGHTNESS_DECREASED,
    ":trigger_android_screen_brightness_decreased"));
TRIGGERS.add(new Pair<>(DICT_DATA_ON,
    ":trigger_android_data_enabled"));
TRIGGERS.add(new Pair<>(DICT_DATA_OFF,
    ":trigger_android_data_disabled"));
TRIGGERS.add(new Pair<>(DICT_DEVICE_ON,
    ":trigger_android_device_on"));
TRIGGERS.add(new Pair<>(DICT_GPS_ON,
    ":trigger_android_gps_enabled"));
TRIGGERS.add(new Pair<>(DICT_GPS_OFF,
    ":trigger_android_gps_disabled"));
TRIGGERS.add(new Pair<>(DICT_GPS_EXIT,
    ":trigger_android_gps_exit"));
TRIGGERS.add(new Pair<>(DICT_GPS_ENTER,
    ":trigger_android_gps_enter"));
TRIGGERS.add(new Pair<>(DICT_AIRPLANE_ON,
    ":trigger_android_airplane_enabled"));
TRIGGERS.add(new Pair<>(DICT_AIRPLANE_OFF,
    ":trigger_android_airplane_disabled"));
```

```
TRIGGERS.add(new Pair<>(DICT_WIFI_ON,
    ":trigger_android_wifi_enabled"));
TRIGGERS.add(new Pair<>(DICT_WIFI_OFF,
    ":trigger_android_wifi_disabled"));
TRIGGERS.add(new Pair<>(DICT_WIFI_CONNECTED_TO,
    ":trigger_android_wifi_connected_to"));
TRIGGERS.add(new Pair<>(DICT_WIFI_DISCONNECTED_FROM,
    ":trigger_android_wifi_disconnected_from"));
TRIGGERS.add(new Pair<>(DICT_APP_INSTALLED,
    ":trigger_android_app_installed"));
TRIGGERS.add(new Pair<>(DICT_APP_LAUNCHED,
    ":trigger_android_app_launched"));
TRIGGERS.add(new Pair<>(DICT_APP_KILLED,
    ":trigger_android_app_killed"));
TRIGGERS.add(new Pair<>(DICT_APP_UNINSTALLED,
    ":trigger_android_app_uninstalled"));
TRIGGERS.add(new Pair<>(DICT_NOTIFICATIONS_PROFILE_SET,
    ":trigger_android_notifications_profile_set"));
TRIGGERS.add(new Pair<>(DICT_BATTERY_PLUGGED,
    ":trigger_android_battery_plugged"));
TRIGGERS.add(new Pair<>(DICT_BATTERY_UNPLUGGED,
    ":trigger_android_battery_unplugged"));
TRIGGERS.add(new Pair<>(DICT_BATTERY_POWER_MODE_ON,
    ":trigger_android_battery_power_mode_enabled"));
TRIGGERS.add(new Pair<>(DICT_BATTERY_POWER_MODE_OFF,
    ":trigger_android_battery_power_mode_disabled"));
TRIGGERS.add(new Pair<>(DICT_BATTERY_LOW,
    ":trigger_android_battery_low"));
TRIGGERS.add(new Pair<>(DICT_TIME_EVERY_DAY_AT,
    ":trigger_android_time_every_day_at"));
TRIGGERS.add(new Pair<>(DICT_MUSIC_ON,
    ":trigger_android_audio_music_on"));
TRIGGERS.add(new Pair<>(DICT_MUSIC_OFF,
    ":trigger_android_audio_music_off"));
TRIGGERS.add(new Pair<>(DICT_JACK_ON,
    ":trigger_android_audio_jack_on"));
TRIGGERS.add(new Pair<>(DICT_JACK_OFF,
    ":trigger_android_audio_jack_off"));
TRIGGERS.add(new Pair<>(DICT_FITNESS_CAMMINATA_ON,
    ":trigger_android_walking"));
TRIGGERS.add(new Pair<>(DICT_FITNESS_CAMMINATA_OFF,
    ":trigger_android_walking"));
```

```
        TRIGGERS.add(new Pair<>(DICT_FITNESS_CORSA_ON,
            ":trigger_android_running"));
        TRIGGERS.add(new Pair<>(DICT_FITNESS_CORSA_OFF,
            ":trigger_android_running"));
        TRIGGERS.add(new Pair<>(DICT_FITNESS_BICICLETTA_ON,
            ":trigger_android_cycling"));
        TRIGGERS.add(new Pair<>(DICT_FITNESS_BICICLETTA_OFF,
            ":trigger_android_cycling"));
        TRIGGERS.add(new Pair<>(DICT_FITNESS_MACCHINA_ON,
            ":trigger_android_vehicle"));
        TRIGGERS.add(new Pair<>(DICT_FITNESS_MACCHINA_OFF,
            ":trigger_android_vehicle"));

        TRIGGERS.add(new Pair<>(DICT_FACEBOOK_NEW_POST,
            ":trigger_facebook_new_post"));
        TRIGGERS.add(new Pair<>(DICT_FACEBOOK_NEW_POST_VIDEO,
            ":trigger_facebook_new_video_post"));
        TRIGGERS.add(new Pair<>(DICT_FACEBOOK_NEW_POST_PHOTO,
            ":trigger_facebook_new_photo_post"));
    }

    private static void actions() {
        ACTIONS.clear();
        ACTIONS.add(new Pair<>(DICT_BLUETOOTH_ON,
            ":action_android_bluetooth_enable"));
        ACTIONS.add(new Pair<>(DICT_BLUETOOTH_OFF,
            ":action_android_bluetooth_disable"));
        ACTIONS.add(new Pair<>(DICT_BLUETOOTH_DISCONNECTED_FROM,
            ":action_android_bluetooth_disconnect_from"));
        ACTIONS.add(new Pair<>(DICT_BLUETOOTH_CONNECTED_TO,
            ":action_android_bluetooth_connect_to"));
        ACTIONS.add(new Pair<>(DICT_NIGHT_MODE_ON,
            ":action_android_screen_night_mode_enable"));
        ACTIONS.add(new Pair<>(DICT_NIGHT_MODE_OFF,
            ":action_android_screen_night_mode_disable"));
        ACTIONS.add(new Pair<>(DICT_SET_BRIGHTNESS,
            ":action_android_screen_set_brightness"));
        ACTIONS.add(new Pair<>(DICT_GPS_ON, ":action_android_gps_enable"));
        ACTIONS.add(new Pair<>(DICT_WIFI_ON,
            ":action_android_wifi_enable"));
        ACTIONS.add(new Pair<>(DICT_WIFI_OFF,
            ":action_android_wifi_disable"));
```

```
        ACTIONS.add(new Pair<>(DICT_WIFI_CONNECTED_TO,
            ":action_android_wifi_connect_to"));
        ACTIONS.add(new Pair<>(DICT_WIFI_DISCONNECTED_FROM,
            ":action_android_wifi_disconnect_from"));
        ACTIONS.add(new Pair<>(DICT_VOLUME_SET_PROFILE,
            ":action_android_volume_set_profile"));
        ACTIONS.add(new Pair<>(DICT_MUSIC_ON,
            ":action_android_audio_start_music"));

        ACTIONS.add(new Pair<>(DICT_FACEBOOK_NEW_POST,
            ":action_facebook_new_staus"));
        ACTIONS.add(new Pair<>(DICT_FACEBOOK_NEW_POST_PHOTO,
            ":action_facebook_photo_post"));
        ACTIONS.add(new Pair<>(DICT_FACEBOOK_NEW_POST_VIDEO,
            ":action_facebook_video_post"));
    }

    private static void ringerMode() {
        PROFILES.clear();
        PROFILES.add(new Pair<>(0, "Silenzioso"));
        PROFILES.add(new Pair<>(1, "Vibrazione"));
        PROFILES.add(new Pair<>(2, "Normale"));
    }
}
```

B.1.2 EventListened

```
public class EventListened extends EventDB {

   // regola ignorata dall'utente
    public static final int RULE_IGNORED = -100;
   // regola eliminata dall'utente
    public static final int RULE_ELIMINATED = -101;
   // regola memorizzata dall'utente
    public static final int RULE_SAVED = -102;
   // quantita' di eventi ripetuti per determinare i suggerimenti
    private static final int MAX_QUANTITY_GENERAL = 4;
   // quantita' di eventi tempo ripetuti per determinare i suggerimenti
    private static final int MAX_QUANTITY_TIME = 2;
   // quantita' di eventi location ripetuti per determinare i suggerimenti
    private static final int MAX_QUANTITY_LOCATION = 2;
   // tempo di ricerca eventi simili (20 sec)
    private static final long TIME_INTERVAL_SEARCHING = 20 * 1000;
   // tempo di ricerca eventi location (1 ora)
    private static final long TIME_INTERVAL_SEARCHING_LOCATION = 60 * 60
        * 1000;
   // tempo cancellazione tabella di log (72 ore)
    private static final long TIME_INTERVAL_DELETING = 72 * 60 * 60 *
        1000;
    private static String LAST_EVENT_NAME;
    private Context context;

    public EventListened(Context c, final int nome) {
        super(System.currentTimeMillis(), nome, "-");
        this.context = c;
        newEvent();
    }

    public EventListened(Context c, final int nome, final String
        nameDetail) {
        super(System.currentTimeMillis(), nome, nameDetail);
        this.context = c;
        newEvent();
    }

    private void newEvent() {
        final String this_event_name = getParam1().equals("-") ?
            getEventName() + "" : getEventName() + ":" + getParam1();
```

```
    if (LAST_EVENT_NAME == null ||
        !LAST_EVENT_NAME.equals(this_event_name)) {
        // run the sentence in a new thread
        new Thread(new Runnable() {
            @Override
            public void run() {
                // memorizzo l'ultimo evento presente nel db cosi' da
                    filtrare un po' di eventi doppi
                EventDB lastEvent = App.getDB().getLastEventTime();
                // controllo che l'evento precedente non sia lo stesso
                    di quello attuale
                boolean eventDuplicate = lastEvent != null &&
                        lastEvent.getEventName().equals(getEventName())
                            &&
                        lastEvent.getHour().equals(getHour()) &&
                        lastEvent.getParam1().equals(getParam1());
                if (!eventDuplicate) {
                    App.getDB().insert(EventListened.this);
                    checkActiveRules();
                    eventsAssociation();
                }
            }
        }).start();
    }
}

private void checkActiveRules() {
    // controllo se ci sono regole attive coerenti con l'azione appena
        fatta dall'utente
    List<TriggerActionDB> ruleList =
        App.getDB().getRuleT(getEventName());
    for (TriggerActionDB ta : ruleList) {
        CompleteRuleDB rule = App.getDB().getRule(ta.getUid());
        if (rule != null && rule.getAttiva().equals("true")) {
            if (getParam1() == null) {
                // se ci sono e non hanno parametri, non sevono
                    ulteriori controlli
                GenericAction.newInstance(ta.getActionName(),
                    ta.getParam1A(), context);
            } else if (ta.getParam1T() != null &&
                ta.getParam1T().equals(getParam1())) {
                // altrimenti devo controllare se i parametri sono
                    uguali
```

```
                GenericAction.newInstance(ta.getActionName(),
                    ta.getParam1A(), context);
            }
        }
    }
}

private void eventsAssociation() {
    // controllo che sia effettivamente un'azione
    String parentEventActionName =
        Dictionary.getActionID(getEventName());
    if (!parentEventActionName.equals("")) {
        // EVENTI TEMPO
        // dal db elenco gli eventi che hanno lo stesso orario di
            quello appena intercettato ma in giorni precedenti
        if (App.getDB().getEventsFromHour(getEventName(), getParam1(),
            getDay(), getHour()).size() > MAX_QUANTITY_TIME) {
            EventDB timeEvent = new
                EventDB(System.currentTimeMillis(),
                Dictionary.DICT_TIME_EVERY_DAY_AT, getHour() + "");
            addTriggerActionInDB(timeEvent, this, MAX_QUANTITY_TIME);
        }

        // EVENTI LOCATION
        // escludo gli eventi della stessa categoria
        if (!parentEventActionName.split("_")[2].contains("gps")) {
            // cerco informazioni sull'entrata o l'uscita da un luogo
                nell'ultima ora
            EventDB e =
                App.getDB().getOnOffEvent(Dictionary.DICT_GPS_ENTER,
                Dictionary.DICT_GPS_EXIT,
                    getTimestamp() - TIME_INTERVAL_SEARCHING_LOCATION,
                        getTimestamp());
            if (e != null) {
                addTriggerActionInDB(e, this, MAX_QUANTITY_LOCATION);
            }
        }

        // EVENTI FITNESS
        // escludo gli eventi della stessa categoria
        String split = parentEventActionName.split("_")[2];
```

```
        if (!split.contains("walking") && !split.contains("running")
            && !split.contains("cycling") &&
            !split.contains("vehicle")) {
            // cerco informazioni sull'inizio o la fine di una
                attivita' nell'ultima ora
            associateOnOffEvents(Dictionary.DICT_FITNESS_CAMMINATA_ON,
                Dictionary.DICT_FITNESS_CAMMINATA_OFF);
            associateOnOffEvents(Dictionary.DICT_FITNESS_CORSA_ON,
                Dictionary.DICT_FITNESS_CORSA_OFF);
            associateOnOffEvents(Dictionary.DICT_FITNESS_BICICLETTA_ON,
                Dictionary.DICT_FITNESS_BICICLETTA_OFF);
            associateOnOffEvents(Dictionary.DICT_FITNESS_MACCHINA_ON,
                Dictionary.DICT_FITNESS_MACCHINA_OFF);
        }

        // EVENTI FUNZIONALITA'
        assignTriggersAndActions(parentEventActionName);
    }
    // pulisco un po' la tabella di log, eliminando tutti gli eventi
        piu' vecchi di 72 ore
    App.getDB().deleteLog(getTimestamp() - TIME_INTERVAL_DELETING);
}

private void associateOnOffEvents(int on, int off) {
    EventDB e = App.getDB().getOnOffEvent(on, off, getTimestamp() -
        TIME_INTERVAL_SEARCHING, getTimestamp());
    if (e != null && e.getEventName() == on) {
        addTriggerActionInDB(e, this, MAX_QUANTITY_GENERAL);
    }
}

private void assignTriggersAndActions(String actionName) {
    Integer[] listName = {
            getEventName(),
            Dictionary.DICT_GPS_ENTER,
            Dictionary.DICT_GPS_EXIT,
            Dictionary.DICT_FITNESS_CAMMINATA_ON,
            Dictionary.DICT_FITNESS_CAMMINATA_OFF,
            Dictionary.DICT_FITNESS_CORSA_ON,
            Dictionary.DICT_FITNESS_CORSA_OFF,
            Dictionary.DICT_FITNESS_BICICLETTA_ON,
            Dictionary.DICT_FITNESS_BICICLETTA_OFF,
            Dictionary.DICT_FITNESS_MACCHINA_ON,
```

```
            Dictionary.DICT_FITNESS_MACCHINA_OFF
    };
    Long time = getTimestamp();
    // prendo il primo evento immediatamente precedente a quello
       attualmente scatenato e che sia un trigger
    for (EventDB trigger : App.getDB().getTriggerOfEvent(listName,
       time - TIME_INTERVAL_SEARCHING, time)) {
        // controllo che sia effettivamente un trigger
        String triggerName =
            Dictionary.getTriggerID(trigger.getEventName());
        if (!triggerName.equals("")) {
            if (filter(triggerName, actionName)) {
                addTriggerActionInDB(trigger, this,
                    MAX_QUANTITY_GENERAL);
                LAST_EVENT_NAME = actionName + (getParam1() != null ?
                    ":" + getParam1() : "");
                break;
            }
        }
    }
}

private boolean filter(String triggerName, String actionName) {
    boolean filter1 = triggerName.contains("airplane") &&
            (actionName.contains("wifi") ||
                actionName.contains("bluetooth") ||
                actionName.contains("data"));
    boolean filter2 = actionName.contains("airplane") &&
            (triggerName.contains("wifi") ||
                triggerName.contains("bluetooth") ||
                triggerName.contains("data"));
    // controllo che le associazioni non appartengano alla stessa
       classe di eventi
    // (per esempio: evento wifi_on -> trigger wifi_off)
    boolean filter3 = triggerName.contains(actionName.split("_")[2]);
    boolean filter4 = triggerName.equals("");

    return !filter1 && !filter2 && !filter3 && !filter4;
}

private void addTriggerActionInDB(EventDB trigger, EventDB action,
    int quantityLimit) {
```

```
    TriggerActionDB ta = new TriggerActionDB(trigger.getEventName(),
        action.getEventName(), trigger.getParam1(),
        action.getParam1());
    try {
        App.getDB().insert(ta);
        ta = App.getDB().getTriggerActionUID(trigger.getEventName(),
            action.getEventName(), trigger.getParam1(),
            action.getParam1());
    } catch (Exception ex) {
        // elemento gia' inserito quindi incremento soltanto la
            quantita' dopo aver ricavato il UID
        ta = App.getDB().getTriggerActionUID(trigger.getEventName(),
            action.getEventName(), trigger.getParam1(),
            action.getParam1());
        // ma soltanto se il trigger non e' stato gia' attivato
        if (ta.getQuantity() != RULE_SAVED && ta.getQuantity() !=
            RULE_ELIMINATED) {
            ta.incrementQta();
            App.getDB().update(ta);
        }
    }
    // controllo se le quantita' superano un certo limite, oltre il
        quale si puo' suggerire una regola
    if (ta != null && ta.getQuantity() > quantityLimit) {
        sendNotification(ta);
    }
}

private void sendNotification(TriggerActionDB ta) {
    // siamo pronti a creare una nuova regola suggerita!
    Intent i = new Intent(context, MainActivity.class);
    i.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
        Intent.FLAG_ACTIVITY_CLEAR_TASK);

    // controllo anche che l'utente sia loggato (che abbia le
        credenziali memorizzate)
    if (!App.get().getSP().getString("user", "").equals("") &&
            !App.get().getSP().getString("pass", "").equals("")) {
        int notificationID = ta.getUid().intValue();
        i.setAction(notificationID + "");
        String title = Dictionary.getTriggerName(ta.getTriggerName())
            + " -> " + Dictionary.getActionName(ta.getActionName());
```

```
            new MyNotification(context, title, i,
                notificationID).sendNotification();
        } else {
            new MyNotification(context, "Devi effettuare il login!!!", i,
                App.NOTIFICATION_LOGIN).sendNotification();
        }
    }
}
```

Bibliography

[1] F. Corno, L. De Russis, A. Monge Roffarello, *A Semantic Web Approach to Simplifying Trigger-Action Programming in the IoT*, 2017

[2] F. Corno, L. De Russis, A. Monge Roffarello, *EUPont: High Level Representation for End User Programming in the IoT*, 2016

[3] F. Corno, L. De Russis, A. Monge Roffarello, *RecRules: Recommending Trigger-Action Rules for End-User Development in the IoT*

[4] Firebase, *www.firebase.google.com*

[5] Facebook Developers, *www.developers.facebook.com*

[6] Facebook API Graph, *www.developers.facebook.com/tools/explorer/*

[7] IFTTT, *www.ifttt.com*

[8] Android Developers, *www.developer.android.com*

[9] Lina Yao, Quan Z. Sheng, Anne H.H. Ngu, Helen Ashman, and Xue Li. 2014. Exploring Recommendations in Internet of Things. In *Proceedings of the 37th International ACM SIGIR Conference on Research; Development in Information Retrieval (SIGIR '14)*. ACM, New York, NY, USA, 855–858.

[10] Fulvio Corno, Luigi De, and Alberto Monge Roffarello. 2017. A High-Level Approach Towards End User Development in the IoT. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '17)*. ACM, New York, NY, USA, 1546–1552.

[11] B. Ur, E. McManus, M. Pak Yong Ho and M. L. LIttman, "Practical Trigger-Action Programming in the Smart Home," in *Proceedings of the SIGCHI*

Conference of Human Factors in Computing Systems, CHI '14, 2014.

[12] H. Lieberman, F. Paternò, M. Klann and V. Wulf, "End-User Development: An Emerging Paradigm", in *End User Development*, 2006, pp. 1-8.