UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DESIGN AND DEVELOPMENT OF A MOBILE APPLICATION FOR ACTIVITY MONITORING IN AN INTELLIGENT ENVIRONMENT

JUAN JOSÉ HERRERO BERMEJO

ENERO 2020

TRABAJO DE FIN DE GRADO

Título:	Diseño y desarrollo de una aplicación de móvil para monitorizar la actividad en un entorno inteligente
Título (inglés):	Design and development of a mobile application for activity monitoring in an intelligent environment
Autor:	Juan José Herrero Bermejo
Tutor:	Carlos Ángel Iglesias Fernández
Departamento:	Departamento de Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	—
Vocal:	—
Secretario:	—
Suplente:	—

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO FIN DE GRADO

DESIGN AND DEVELOPMENT OF A MOBILE APPLICATION FOR ACTIVITY MONITORING IN AN INTELLIGENT ENVIRONMENT

Juan José Herrero Bermejo

Enero 2020

Resumen

El proyecto que se muestra a continuación explica detalladamente el desarrollo de una nueva versión de la aplicación para smartphone Ewe Tasker, desarrollada en Android, que permite la automatización de tareas a través del servidor web Ewe Tasker, perteneciente al grupo GSI de la Escuela Técnica Superior de Ingenieros de Telecomunicaciones.

Activity Recognition es la aplicación principal de esta nueva versión de la app Ewe Tasker y su función consiste en reconocer, almacenar, analizar y procesar la información de la actividad de un usuario a tiempo real, ya sea andar, correr, ir en un vehículo, etc con el fin de monitorizar su actividad, ofrecerle un entorno que muestre estadísticas sobre su actividad y generar eventos de actividad de tal modo que se puedan utilizar para la automatización de tareas desde el servidor Ewe Tasker a través de la app.

Además, incluye una segunda aplicación, integrada de antiguas versiones de la app, que ofrece un entorno inteligente que permite ejecutar tareas de automatización creadas previamente en el servidor Ewe Tasker. Esta aplicación incluye la generación de eventos de detección de movimiento mediante beacons, implementa el envío de los nuevos eventos de actividad y permite la interacción con distintos actuadores del entorno inteligente, ofreciendo múltiples posiblidades a la hora de automatizar tareas.

Como resultado, este proyecto ofrecerá al usuario una app para tener acceso a información sobre su actividad en todo momento y ejecutar reglas de automatización personalizadas según sus necesidades, tanto en un entorno inteligente, como en su día a día.

Palabras clave: Android, Java, Reconocimiento de actividad, Automatización, EWE, Internet de las cosas, Google Play Services, Beacons

Abstract

The project shown below explains in detail the development of a new version of the Ewe Tasker smartphone application, developed in Android, which allows the automation of tasks through the Ewe Tasker web server. This server belongs to the GSI group of the Higher Technical School of Telecommunications Engineering.

Activity Recognition is the main application of this new version of the app and its function is to recognize, store, analyze and process user activity information in real time in order to monitor his activity, offer user an environment that shows statistics about his activity and generate activity events in such a way that they can be used to automate tasks from Ewe Tasker server through the app.

In addition, it includes a second application, integrated from old versions of the app, which offers an intelligent environment that allows users to execute automation tasks previously created on the Ewe Tasker server. This application includes the generation of motion detection events through beacons, implements the sending of generated activity events and allows interaction with different actuators in a intelligent environment, offering multiple possibilities when automating tasks.

As a result, this project provides user an app to have access to information about their activity at all times and allows to execute custom automation rules according to their needs, both in an intelligent environment and in another context.

Keywords: Android, Java, Activity Recognition, Automation, EWE, IoT, Google Play Services, Beacons

Agradecimientos

En primer lugar, quería darle las gracias a Carlos Ángel por darme la oportunidad de formar parte del GSI y de contribuir en él desarrollando una nueva versión de una aplicación del grupo, además de agradecer la ayuda y la facilidades que me ha proporcionado.

No obstante, quería agradecer a todo el grupo del GSI la ayuda que en cada momento me han dado y el gran conocimiento que me han ofrecido, sobre todo, a Sergio, por todo lo que me ayudó en los momentos más difíciles de este proyecto.

También quería dar la gracias a mis compañeros de despacho y de oficina, con los que compartí tanto horas de trabajo como momentos de risas y diversión.

Por otro lado, tengo que agradecer a muchas de las personas que han hecho posible todo lo que he conseguido en mi vida, y lo que me ha hecho ser como soy actualmente:

Mi familia, por su apoyo incondicional, su cariño y su confianza, que son uno de los pilares que me sostienen.

Mis amigos, que me apoyan, me ayudan, me cuidan... Gracias por estar ahí siempre.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Context

Over the last few years, terms such as “smart-home” [1], “intelligent environments” [2] or “Internet of Things” [3] are reaching a lot of importance in the technological world. This popularity could be due to the increase in the use and number of electronic devices which incorporate different wireless networks, either WiFi or Bluetooth. All these wireless connections have a point in common, which allow connection between devices. The possibilities these technologies based on connected devices offers are uncountable, since they can be applied to any field, such as medicine or security, and they are still under development, as new applications are still being discovered [3].

One of the most innovative features in new term that these technologies develop, is the notion of automation, which is getting a lot of importance in simplifying people’s life [4]. Automation replaces the actuation of people when performing tasks by a device, that entails many-fold benefits for them, such as saving on time or greater comfort in his life. In addition, automation has benefits for the system, such as allowing activity logging, very useful for many systems when it comes to finding possible failures. In fact, the possibility of monitoring, controlling and automating the different acts of daily life such as turning on the light, opening the door, pulling down the shutters... in an intelligent environment is being developed for a long time [5] and now it is an available technology presents in all kinds of current projects [6]. Moreover, automation is quite demanded nowadays because it can be applied in a lot of fields, from in health care [7] to in security systems [8], and it offers multiple opportunities. One of them that is the cause of this project, is its integration with human activity recognition software.

Activity recognition can reveal the activities the user is carrying out, such as riding a bike, going in a vehicle, walking, running... [9] to monitoring the user activity, among many other uses. Nowadays, this technology is present in a lot of different areas, such as in health care, where it can infer users diary of physical activities and energy expenditure based on Metabolic Equivalents [10]; in security, detecting abnormal activities [11]; or in entertainment, improving the 3D data acquisition techniques [12]. This technology, of course, detects user activity using a device that user carries.

The mobile phone is the ideal device, because it has become one of those essentials that we carry with us everywhere, and, also, it includes different sensors that allow the rotation, location and even acceleration detection. Therefore, it has been developed different activity recognition software using the mobile phone sensors in last years [13], especially using the accelerometer, which ensure a high percentage of accuracy [14], but also the gyroscope [15]. In fact, it has been studied that the use of both sensors together gets better accuracy in the activity recognition, making the recognition process more reliable [16]. Some multinational companies like Google, has developed its own activity recognition software, offering users an API [1] to the free application development using activity recognition [17].

In conclusion, to use the technologies explained before, specially automation tasks and activity recognition, this project will try to integrate them and combine them in a mobile application in order to monitor human activity and make automation task using activity events.

[1] `https://developers.google.com/location-context/activity-recognition`

1.2 Project goals

The main goal of this project is developing an Android application that detects, analyzes and monitors human activity. This application will be integrated into a semantic event-based automation framework so that it can enable activity-based automation.

This main objective includes:

• Develop and design a useful and simple user interface that allows users accessing to both their activity recognition information and an intelligent automation environment.

• Develop a user interface that shows statistics and graphics about the user detected activity information.

• Develop an intelligent automation environment that allows the use of automation rules using detected activity events.

• Integrate a real-time database that allows to manage the detected activity data at all times.

• Connect with a rule engine which is able to run the event driven rules and handle its response

• Connect with action triggers in order to run actions generated by the rule engine.

1.3 Structure of this document

In this section we provide a brief overview of the chapters included in this document. The structure is as follows:

Chapter 1 is the introduction for the project, that explain the reasons why the project was chosen and the main goals are described.

Chapter 2 describes the main standards and technologies on which this project is based.

Chapter 3 offers an overview of the main use cases, explaining the different system actors and how interact with it.

Chapter 4 provides a full description of the project architecture and all the components that includes.

Chapter 5 It provides a description of the study cases in this project. This chapter tries to explain how a user can use the app since the start, using all available functionalities.

Chapter 6 concludes the project, explain the problems encountered and the future work to develop in the app.

CHAPTER 2

Enabling Technologies

This chapter offers a brief review of the main technologies that have made possible this project. Firstly, the technologies of Google Play Services are explained and then, the task automation platform and the ontology implemented for its operation are presented.

2.1 Google Play Services

Google Play Services [1] is a library developed by Google to allow developers use different Google Services, that offers fold-many utilities such as, user identification or location detection, to use free in a mobile application.

Nowadays, the Google Services are being used in many different projects [18] because its useful and easy handling. In addition, although Google offers fast and frequent updates on these services, it guarantees users its use in non-updated versions, which makes it sustainable to use these services in any app and getting similar behavior across different versions of Android and devices. Even the older versions of Android, nowadays Android Jellybean 4.1, has still access to the last novelties. The use of

[1] https://developer.android.com/distribute/play-services

Google Services versus using other external or own services, provides advantages for both the user and the developer.

- For a developer, these services involve a standardized and abstract way of accessing advanced functions such as location, maps or user login without the need to include long lines of code that perform complex functions with worse results than Google Services, which would greatly increase the size and complexity of any project.
- For an user, although an app which use Google Play Services implies to need internet connection, the Google services use has benefits in phone resources [19]. In fact, the most app and devices includes Google services and apps even in the native configurations of the device [20]. Therefore, when an application includes Google Play Services, it usually does not imply a new service that requires installing other services, since it already includes the entry point to communicate with Google.

  By the reasons explained before, use Google Services is usually more efficient when it comes to ensuring our battery and other resources on the smart-phone, as well as in the improvement in the software, versus the other service use.

2.1.1 Activity Recognition

Activity Recognition, as we have explained, can reveal the activities the user is carrying out to monitor the his activity and use information extracted from his activities to perform statistics, among other uses. There are different systems of human activity recognition currently available. They differ substantially among themselves by the complexity and precision in detecting activities and by the different activities that each one can detect, but also, by the process followed to obtain it. However, they agree to use the phone's sensors, especially the gyroscope and accelerometer, to detect activities.

Because human activities are complex and sensor signals have varying amounts of noise, classification algorithms are almost always probabilistic. Activity recognition systems typically have three main components [21]:

- a low-level sensing module that continuously gathers relevant information about activities using microphones, accelerometers, light sensors, and so on.

- a feature processing and selection module that processes the raw sensor data into features that help discriminate between activities.

- a classification module that uses the features to infer what activity an individual or group of individuals is engaged such as walking, cooking, or having a conversation.

2.1.1.1 Activity Recognition API

The Activity Recognition API [2] of Google is the activity recognition software developed by Google. It is built on top the different sensors available in a device, specially accelerometer and gyroscope, which provides insights into what users are currently doing. The API works detecting activities by periodically reading short bursts of sensor data and processing them using machine learning models. To optimize resources, the activity reporting is stopped if the device has been still for a while, and uses low power sensors to resume reporting when it detects movement. Moreover, Google Service removes the need to have a service constantly running in the background for activity detection purpose, as older systems used to. This implies much better management of system resources, especially the battery.

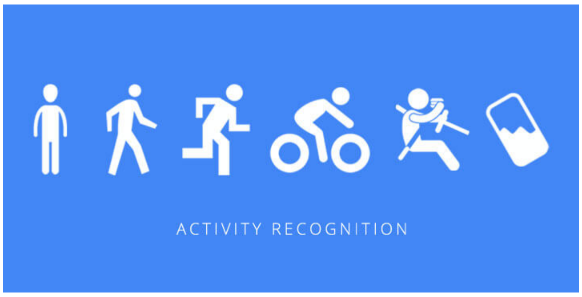

Figure 2.1: Activity Recognition API

2.1.1.2 Activity Detection

The obtainment of detected activities and the activity events consists in:

The app, that implements Google Services, extract from the Activity Recognition software a list of detected activities, each of which includes confidence and type properties. Type property means the name of the activity that user is undertaking

[2] "https://developers.google.com/location-context/activity-recognition"

and confidence is a number that indicates the likelihood that the user is performing the activity represented in the result.

Activity detection is not an exact science, so rather than returning a single activity that the user is definitely performing, the Activity Recognition API returns a list of activities that the user may be performing, with the confidence property for each activity.

The possible activities that Google API can detect are:

- Still (not moving), including standing and seating.
- On foot, when the user is moving on foot, whether walking or running.
- Walking.
- Running.
- In a vehicle, such as a car, a train, a bus, etc.
- On a bicycle.
- Tilting, that is detects when the device angle relative to gravity changed significantly. This often occurs when a device is picked up from a desk or a user who is sitting stands up.
- Unknown activity. It happens when it is unable to detect the current activity.

When an activity is accompanied by a high percentage of confidence property, such as 90%, it indicates certainly that the user is currently doing the corresponding detected activity. However, when “unknown activity” contains the higher confidence percentage, indicates that there are not enough information to detect the activity and it needs more time to track it.

In order to ensure smooth operation of the software, we had to decide the interval of time to execute activity recognition tasks. We decided to choose a low interval of 3 seconds, because it guarantees a quick activity recognition, but it is not so low as to continuously detect a different activity and consume too many resources.

2.1.1.3 Activity Recognition Transition

On the other hand, Google offers the Activity Recognition Transition API that allows to know when a user start or stops a specify activity. The app subscribes to a transition in activities of interest and the API notifies your app only when needed, avoiding the detecting activities periodically. Accordingly, it needs less resources in the device than detected recognition API and it can be useful, for example, automating actions after state transitions.
Nevertheless, this API do not allow the monitoring of detected activity in real time, so this project does not focus on this API.

2.1.2 Firebase

Firebase [3] is a mobile and web application development platform developed by Google that provides developers with a plethora of tools and services to help them develop high-quality apps, grow their user base, and earn more profit. The main features that make Firebase so attractive to developers are the following:

- Easily synchronize your project data without having to manage connections or write complex synchronization logic, in the same way than other Google Play Services make it.
- Use a multiplatform toolset: it integrates easily for web platforms and mobile applications. It is compatible with many platforms, such as IoS, Android, web applications, Unity, C++...
- Use Google's infrastructure and automatically scale for any type of application, from the smallest to the most powerful.
- Create projects without a server: The tools are included in the SDKs for mobile and web devices, so it is not necessary to create a server for the project.

Thanks to all these functionalities, any developer can combine and adapt the platform according to their needs.

One of the strength services that Firebase offers is its storage service. It supports two ways to save the data into a real time database or into depending of the ap-

[3] https://firebase.google.com/

plication that developer want to give his data, can result interesting one or another one.

2.1.2.1 Realtime Database

Firebase Realtime Database [4] is a powerful but simple, NoSQL database hosted in the cloud that store the data in JSON format and they are synchronized in real time with each connected client. When developers compile cross-platform applications with iOS, Android and JavaScript SDK, all their clients enter an instance of Realtime Database and receive updates automatically with the latest data. In other words, the real-time activation of this database allows users to access their data information from any device in real time, sharing an instance of Realtime Database, and each time a user makes a modification to it, it is stored in the cloud and the other devices are automatically notified.

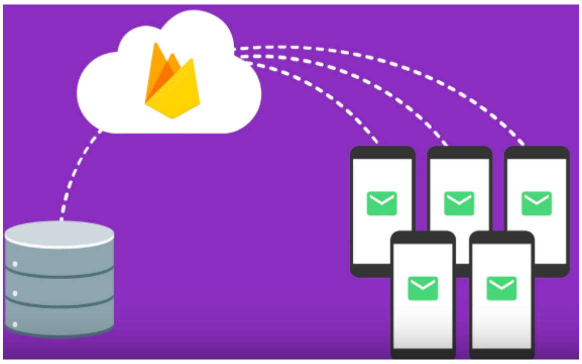

Figure 2.2: Firebase Realtime Database

This database includes some interesting key functions that should be explained:

- Real time: Instead of typical HTTP requests, Firebase Realtime Database uses data synchronization (every time data changes, connected devices receive that update in milliseconds). It provides collaborative and enveloping experiences without thinking about the network code.
- Offline support: Firebase apps continue to respond, even offline, as the Firebase Realtime Database SDK makes your data persist on disk. When the connection is reestablished, the client device receives the missing changes and synchronizes them with the current state of the server.

[4] `https://firebase.google.com/docs/database`

- Access from client devices: Firebase Realtime Database can be accessed directly from a mobile device or a web browser; an application server is not required. Security and data validation are available through the Firebase Realtime Database security rules: rules based on expressions that are executed when data is read or written.

- Scaling in several databases: With Firebase Realtime Database developers can meet the data needs of the large-scale app: they can divide the information into several database instances within the same Firebase project. In addition, they can use Firebase Authentication, explained in next lines, to optimize the authentication process in the project. It allows authenticate users in various instances of the database and control access to information in each database. To do this, Firebase Realtime Database includes custom rules in each of the instances of the database.

  Firebase features make it an ideal database to our project to store activity in real time in order to monitor user activity. Now, we are going to describe another alternative that also we studied, Cloud Firestore.

2.1.2.2 Cloud Firestore

Cloud Firestore is another real time storage alternative based on Google Cloud Storage [5], officially brought out of beta on January 31, 2019. This database is a powerful flexible, scalable and cloud-based NoSQL database to store and synchronize data for client and server side programming. Firebase SDKs for Cloud Storage add Google security to file upload and download operations for Firebase apps, regardless of network quality. Besides, developers can use its SDK to store images, audio, video and other types of user-generated content. On the server, it be can used Google Cloud Storage to access the same files.
It is the successor to Firebase's original databasing system, Real-time Database, and allows for nested documents and fields rather than the tree-view provided in the Real-time Database.

In conclusion, having chosen this database would also have been a good option, but finally, we chose the Realtime Database because it still offers an efficient and low-latency solutions and it is more manageable and simple than this database.

[5] `https://firebase.google.com/docs/storage/`

2.1.2.3 Firebase Auth

It's a cloud-hosted NoSQL database that lets store and sync between users, saving data as JSON objects that the developers can manage in real-time, making easy to access the data from any device.

Firebase Authentication provides easy-to-use back-end services, SDKs, and UI libraries already developed to authenticate users in your app. It supports authentication through passwords, phone numbers, popular federated identity providers, such as Google, Facebook and Twitter, and much more.

This service includes two ways to authenticate users:

- FirebaseUI Auth: FirebaseUI provides a direct authentication solution that controls UI flows for users who access with email addresses and passwords, phone numbers and popular federated identity providers, including Google Access and Facebook Access. The FirebaseUI Auth component implements recommendations for authentication on websites and mobile devices, which can maximize the conversion of access and registration of your app. It also handles extreme cases, such as recovery and linking accounts, which can have security implications and be prone to errors when trying to handle them correctly. Moreover, FirebaseUI can be easily customized to fit the rest of the visual style of the app and is open source.

- Firebase SDK Authentication: In addition to direct authentication, this service includes integration with federated identity providers, such as Google, Facebook, Twitter y GitHub, authentication by phone number, integration with custom authentication systems or even anonymous authentication, that consist in create temporary anonymous accounts to allow the use of functions that require authentication.

Between both services, we chose to authenticate our users with FirebaseUI Auth because we don't consider necessary more complex ways to authenticate users than the method of this service offers. This function is still in development and it will be integrated as soon as is possible in Ewe Tasker app.

2.2 Rule Automation

Rules is a way to represent the knowledge with conditions in the scope of logic. A rule consist in If-then clauses to express logical functions and operations, expressed in rule language. Generally, it is formed by antecedence and consequence containing clauses, and logical quantifiers used to quantify possible domains of variables. The antecedence contains conditions using logical operators, while the consequence part contains actions. If the conditions are matched, the actions are operated. In the form of subject-relation-object of a clause, the subject and object can be variables, individuals, literal values or other data structures. The parameter used in the rules can be defined in different variety of languages and formats [22].

In recent days, Rule-based task automation is being developed in many web sites and mobile or desktop applications. Generally, platforms like this allows to defining the custom rules, executing an specific action when some specific event is triggered. Thus, it could be possible to create rules such as: "When i get in my car, turn on the Bluetooth and connect with the car" or "When i sit at my desk, turn on the light of the work lamp". This services are called Task Automation Services (TASs) and offers endless possibilities.

2.2.1 Task Automation Services

Task Automation Services provide users a visual programming environment to manage their own personal automation task. The rules are defined by Event-Condition-Action, executing an action when a certain event is triggered. For example, in the example "When i get in my car, turn on the Bluetooth and connect with the car", when the event of get in the car is triggered, the turn on Bluetooth action will be executed.

In last years, the number of TASs is growing intensely highlighting some of them, such as Zapier, IFTTT or Integromat. They differentiates between them providing different options and functions, trying offer its own particular mark [23]. The reason of this variety of servicies could be linked to following TASs's important features:

- Usability, offering a intuitive, simple but powerful intuitive interface for creation of task automation.
- Customizability, allowing users to choose between different event triggered and

actions executed.

- Integration with existing Internet services[24].

On the other hand, all TASs follows the same kind of thinking about rule-based reasoning, a sequence of 'when this then that' steps. Besides, they follows a steps in a task automation process, from the definition of the task, a schedule of when the task is done and its duration and what resources, equipment are needed to perform it to the track system to control and monitor it.

They uses expressive engines that allow to maximizing configurability. One of the languages that stands out from the rest is Notation3 [6].

2.2.2 Notation3

As mentioned before, the rules are defined using Notation3 (N3), a Semantic Web Logic based on the use of triples called Turtle [25]. Turtle is an RDF data serialization format, capable both of describing everything using triples to describing rules to be executed on those triples.
The World Wide Web Consortium (W3C) defined as targets of N3 [26]: to optimize expressions of data and logic in the same language, to allow RDF to be expressed, to allow rules to be integrated smoothly with RDF, to allow quoting so that statements about statements can be made, to be as readable, natural and symmetrical as possible. Besides the language achieves these with following features: URI abbreviation using prefixes which are bound to a namespace (using @prefix) a bit like in XML; repetition of another predicate for the same subject using a semicolon ";" and another object for the same subject and predicate using a comma ","; Bnode syntax with a certain properties just put the properties between [and]; formulae allowing N3 graphs to be quoted within N3 graph using and; variables and quantification to allow rules, etc to be expressed; a simple and consistent grammar.

2.2.3 EYE

EYE(Euler YAP Engine) [27] is a high-performance reasoning engine that supports the Semantic Web layers and it is capable of evaluating, according to what is specified in the rules, whether the conditions for launching the actions have been met. It is

[6]https://www.w3.org/TeamSubmission/n3/

written in Prolog and supports, among others, all built-in predicates defined in the Prolog ISO standard

EYE can be configured with many options of reasoning, e.g., not proving false model, output filtering, and providing useful information of reasoning, e.g., proof explanation, debugging and warning logs.

The inference engine also supports using user-defined plugins. In addition to the main supported language, Notation3, EYE also supports the RIF-BLD(Basic Logic Dialect) rule language.

2.2.4 EWE ONTOLOGY

Evented WEb Ontology (EWE) [28] is a standardized data schema, typically referred as "ontology" or "vocabulary", designed to describe elements within TASs enabling rule interoperability.

- Enable to publish raw data from Task Automation Services (Rules and Channels) online and in compliance with current and future Internet trends.
- Enable Rule interoperability.
- Provide a base vocabulary for building domain specific vocabularies.

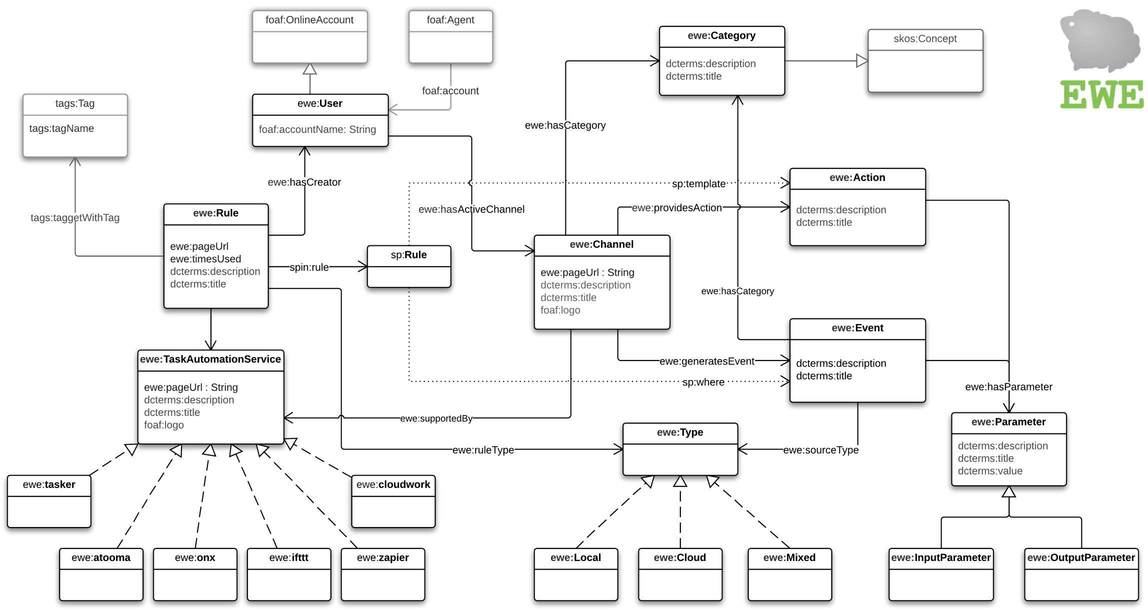

Figure 2.3: EWE Class Diagram

As shown in the above scheme, EWE ontology is formed by four main classes:

- Channel: It defines individuals, including sensors and actuators, which generate events, provide actions or both.
- Event: This class defines the realization of a process that have no duration over time. Events are generated by a certain Channel, and they are triggered by the occurrence of a process which defines them. Their parameters can be modeled as input or output parameters: The output parameters are configured within Rules to customize Actions. However, the input parameters allows users configure the events, describing under which conditions should they be triggered. Different services may generate the same event.
- Action: It defines an operation or process whose nature depend on itself that may be provided by a Channel. For example, an action could produce logs, modify states on a server or even switch on a light. Actions can be configured by means input parameters to react according to a specific Event, whose data are the output parameters.
- Rule: Rule class defines an "EVENT-CONDITION-ACTION" (ECA) rule. An action is executed when a rule is triggered by an event. Rules aim is to define the connections between instances of Event and Action classes, including the configuration parameters: output from Events to input of Actions.

2.2.5 Data Storage Technologies

2.2.5.1 Fuseki

Fuseki [29] is a SPARQL server based on Apache Jena framework. It is known to be a solution as a storage layer for ontologies. To do so, it provides the SPARQL 1.1 protocols for query and update as well as the SPARQL Graph Store protocol. Among its advantages stand out its security, by using Apache Shiro's framework 9 , and its interface which allows server monitoring and administration.

This tool has been developed by Jena, so it incorporates Jena's text query and spatial query. In addition, its uses are diverse. It can be used as an operating system service, as a Java web application, and as a standalone server. In some cases, Fuseki is used to provide the protocol engine for other RDF query and storage systems.

In this project, Fuseki is used by EweTasker platform to store the definition of channels, automation rules and devices in N3 through Ewe ontology.

2.2.5.2 MongoDB

MongoDB [30] is a NoSQL database, in other words, a non-relational database. The way it works is simple. The database contains collections which are made up of documents. These documents are, in turn, composed of fields, where information is stored.

As we explained, it is oriented to documents and more specifically to JSON 6 documents. This is an advantage because JSON format is widely used in web applications. Moreover, the treatment of these documents is done in a flexible way, being able to modify each data entry in the document dynamically. Another advantage is that it facilitates data analysis using ad hoc queries, indexing, and real time aggregations. In addition, it is a distributed database, providing horizontal scaling and high availability.

This database is one of the most used storage technologies, with more than 9 million of downloads around the world. It is a free and open-source tool. Newer versions are published under the Server Side Public License (SSPL) v1 7 and it provides integration for more than ten programming languages as Python or Java.

In this project, MongoDB is used in Ewetasker platform to store user data.

2.2.5.3 Elastic Search

Elasticsearch [31] is a distributed, RESTful search and analytic engine highly scalable, designed to be a central data store and generally used as the underlying search engine that powers applications that have complex search features and requirements. Elasticsearch popular use is due to the fact that provides a HTTP web interface to interact through queries based on JSON and responeses based in schemafafree JSON documents.

Elasticsearch works with a complex search, but it provides great performance. In order to get such a high performance algorithm, this tool is formed by indices which are divided into shards. The shards are stored in distributed nodes, and each shard can be replicated more than once. These nodes act as a coordinator to delegate operations involving different shards. This search server, which is based on Lucene 8 library (Java), provides clients in the main programming languages such as Java, SQL, .NET, PHP or Python.

In this project, we use Elasticsearch in Ewetasker platform to store usage data.

2.3 Beacons

A beacon is an intentionally conspicuous device smart little devices based in Bluetooth technology that emits a signal that uniquely identifies each device without a previous synchronization need. This signal is emitted constantly in a broadcast Interval and can be received by another device such as a smarthphone. Beacons are usually used to detect people presence and generate certain actions in consequence. The main features that they incorporate are:

- There are small devices so that they can be easily placed at any point.
- Beacon are base in Bluetooth technology so that can detect users presence without internet connections. This feature is really interesting when it comes to use it in environments with poor coverage or difficulty in accessing internet.
- They ensure a low consumption. The batteries that beacons incorporates has a very long life, up to two years with a simple button cell.
- The signal emitted contain a unique number that identifies the beacon. Therefore, their signal are unique and cant be confused with another one. Besides, this signal allows the simultaneous connection with different devices, so that different users can use the beacons utilities in the same time without connection problems.
- The broadcast Interval to send the signal usually can be configurable, depending on whether you want more battery life or better accuracy.
- Long dynamic range when Beacons detect the user distance, allowing to detect the user presence in a relatively large space.

CHAPTER 3

Requirement Analysis

In this chapter the requirements analysis are required using different situations. In order to understand the software developed completely, it is necessary to make a detailed analysis of the possible use cases. To be able possible explain the possible use cases, we chose to provide a standard way to visualize the design of the system using the Unified Modeling Language (UML)[32].

Therefore, this chapter will give an idea about the actors of the system and their interactions, being able to understand the importance of each one in each case of use.

3.1 Use cases

The next chapter will explain the uses cases of this Project, defining the requisites, the system uses and the actors that intervenes.
The next diagram defines the interactions between actors, which will be explained in next lines.

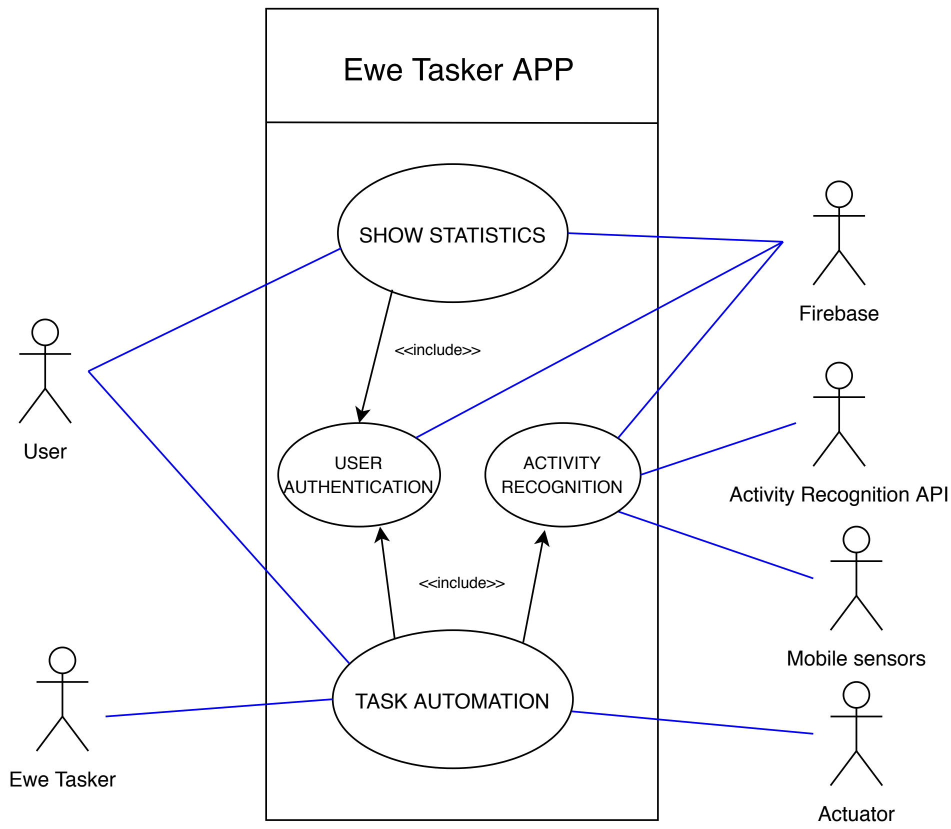

Figure 3.1: Use cases

- Show statistics: This is one the main use case in the system. The final and main actor is user, who access to this function to analyze his activity information downloaded from Firebase in form of graphs, that shows the time that user spent in each activity in last day, last week and last month. This use case use both secondary uses cases, activity recognition and user authentication.

- Task Automation: Task automation system based on rules is clearly another main use case, whose aim is to allow user use automation rules with the mobile phone and interacting with the different actuators, which play a role of secondary actors. User still being the main actor because is the responsible to create his own rules and use it.

- User Authentication: User authentication is a function that includes both task automation and show statistics.

- Activity Recognition: This secondary use case explains the process to detect the activity that user is carrying out. This use case is the responsible to transform the information provided by mobile sensors in probabilities of each activity and consequently, the software obtains the activity carried out by user.

3.1.1 System actors

- User: The user that carries a smartphone is the one main actors of the app and the final user of the system. It uses the app in its daily routine to get automation task in intelligent environment and interacts with it through the app to interact with different actuators through the Ewe Tasker Server. Moreover, he access to the statistic channel to show his activity information.

- Activity Recognition API: This is a secondary actor. It uses the mobile phone sensors, such as, accelerometer and gyroscope, to detect the activity that user is undertaking at all time. This API intervenes in all cases of study, both sending activity events in automation task and showing statistics about user activity.

- Ewe Tasker: This server is based on the automation task, thus it is an important secondary actor in automation tasks. It allows the connection between the mobile app and the actuators in order to realize an automation task. Ewe Tasker receives events generated by actuators and users and sends them the actions according the rules created.

- Firebase: Firebase is also a really important secondary actor which is encountered in all situations of this Project. On the one hand, it is the responsible to monitor the activity, because it store all the information data in real time to provide it when user requires. Therefore, it receives continuously activity data from the app and it sends activity information packages when the app request it. On the other hand, it provides the service to register and login users in activity recognition application.

- Actuators (Smart devices): Smart devices form an important secondary actors whose aim is, on the one hand, to send events to Ewe Tasker, such as presence detection from beacons, and on the other, to carry out an action such as turning on a light from a Smart light.

- Mobile phone sensors: Accelerometer and gyroscope have the main responsible in the activity recognition. They provides Google API information about the orientation, position, and acceleration of mobile phone and consequently the user, very useful when detecting user activity. These sensors form the last secondary actor.

3.1.2 Conclusions

After this section, we can understand how the system works and how its elements interact between them. In the Case Study Chapter we study in details a series of study cases that have been carried out. Theses cases combine the different use cases in different contexts to get a different functionalities of the Mobile APP.

CHAPTER 4

Architecture

In this chapter, we cover the design phase of this project, as well as implementation details involving its architecture. Firstly, we present an overview of the project, divided into several modules. This is intended to offer the reader a general view of the architecture. After that, we present each module separately and in much more depth.

4.1 Overview

The system is composed of the following several modules, that they are explained in details later:

- Android Mobile App:

  This mobile app is the main module of this project and it focus into the activity recognition, as well as the connection task using activity events. Basically, the mobile app process a list of detected activities coming from activity recognition software of Google, in order to get the most probably activity in every moment and generate activity events. These events are sent to EWE Tasker Server, allowing the use of activity events in rules creation. Moreover, the app collects and saves relevant information of the detected activities into a Firebase realtime database to offers a series of graphs and statistics on user activity at different time scales. On the other hand, the app provides the Beacon Channels utilities, created in an old version of the EWE Tasker app[33].

- EWE Tasker:

  The main aim of this Task Automation Server is to handle events, such as activity events, and to trigger accordingly an action generated by a rule engine in order to sent it to the mobile app or even another device. It include functions for creating and managing rules and for user authentication, in addition to display several user information dashboards. Ewe Tasker is composed of four sub-modules: Rule Engine, Rule Administration, Channel Administration and Action Trigger.

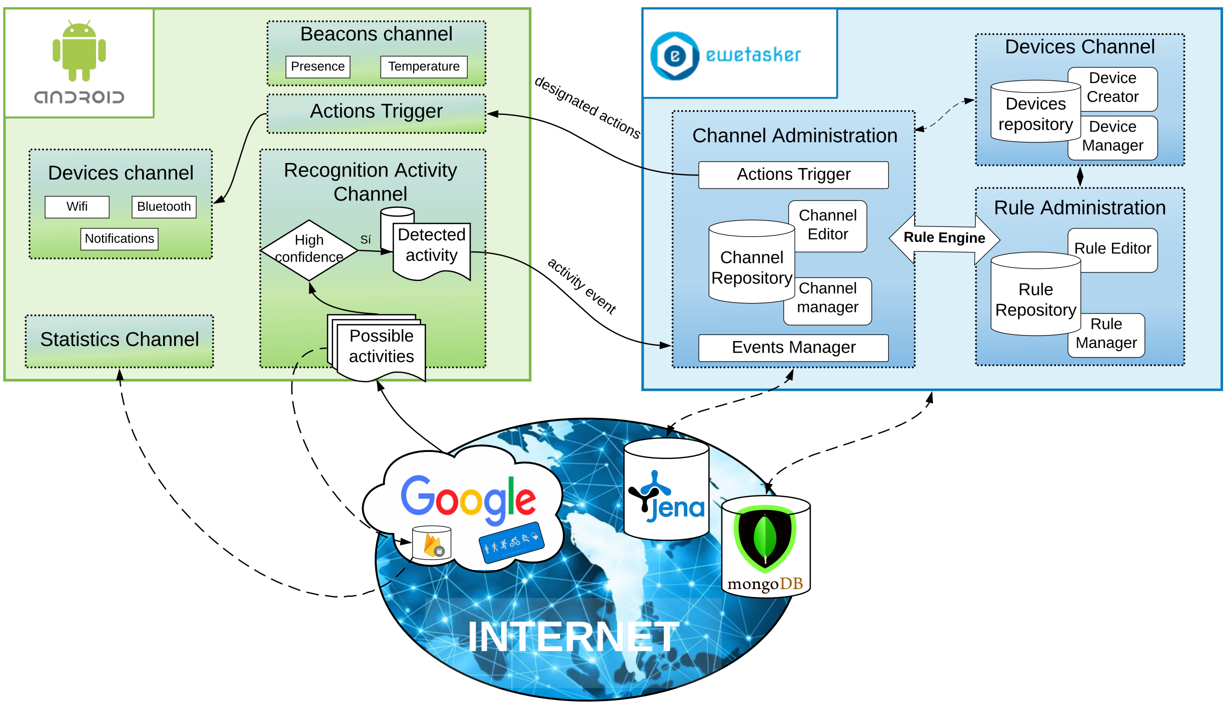

Figure 4.1: Architecture

4.2 Android Mobile APP Module

The mobile APP module is the main pillar of this project. It is a new version of EweTasker mobile app [34] developed in Android with Java. This version is centered in the activity recognition, so that it includes different functionalities using the activity information and it allows the sending of activity events in the EWE Tasker Server to use them in automation tasks. The app provides three different functionalities:

- Firstly, generating and sending activity events to Ewe Tasker. In order to generate the events, it handles a list of detected activities coming from activity recognition software of Google, processing it to get the most probably activity in every moment. If this activity has a high percentage of confidence is sent as an activity event to Ewe Tasker.
- Secondly, triggering and execute returned actions by Ewe Tasker. The actions which Ewe returns, such as turn Blueetooth or Wifi on or show a notification, are executed by the smartphone.
- Finally, providing user information about the activity tracked in every moment, and offers graphics and statistics about user activity at different periods of time.

Activity data generated by activity recognition software are handled by "Activity Recognition Channel" sub-module. After being processed, they are sent to EWE Tasker. The server generates actions according a rules before designed, and it send to the Mobile App. These actions are triggered by the Actions Trigger sub-module. Finally, the device channel, which receive the actions and understand it, is the responsible to execute the action.

4.2.1 Activity Recognition Channel

In order to detect the activity of user, the app starts the activity tracking of the Activity Recognition API of Google. To do it, the app needs to icorporate the Google Play Services, providing Google permission to use the smartphone sensors such as the gyroscope and the accelerometter, among others permissions. Then, this Google Service starts to detect changes in the smartphone sensors, getting activity results every few seconds.

When a new activity result is detected (the user activity has changed), the API subtracts the list of activities from the result. This list of activities includes the respective confidence percentage of each activity. Then, the app proceed to save the list in Firebase real time database with the detection date (useful for the statistics channel). Besides, when a specific activity has more than 80% of confidence percentage, it is saved as the activity that the user is carrying out in the smartphone "shared preferences", replacing the previous one. Finally, the module makes an HTTP connection to the EWE Tasker server to send the activity.

4.2.2 Statistics Channel

This channel provides user the information about his activity in the current day, in last week, or in last month, as well the current activity list tracked.
In order to get that information, the app access to the Firebase Realtime Database to download the activity information in the corresponding time interval. Then, it calculates the time used in carrying out each activity by the user in that time interval, and generates a graphics showing it.

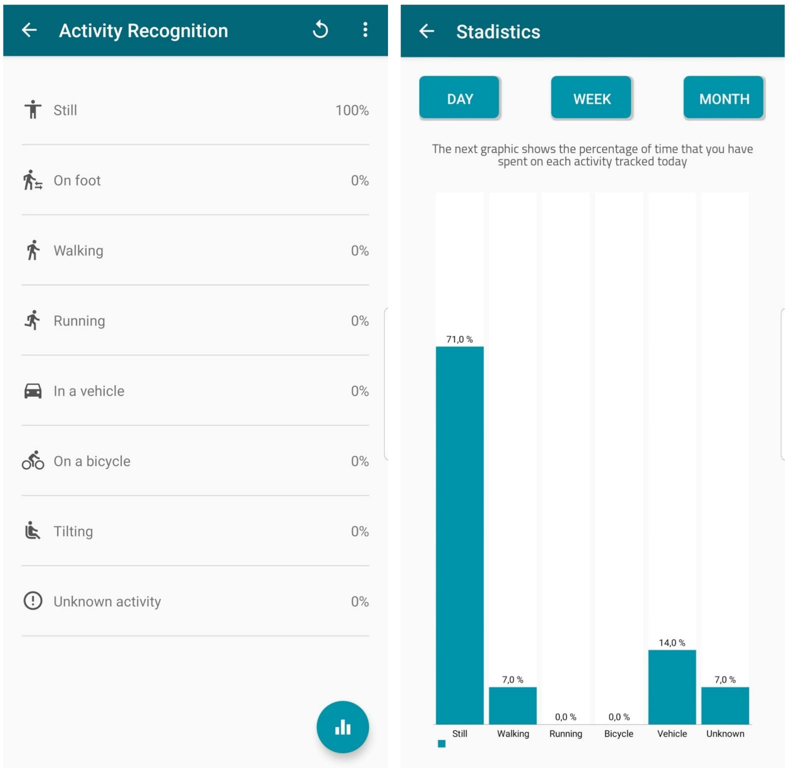

Figure 4.2: Statistics Channel

4.2.3 Actions Trigger

This sub-module goal is to trigger actions after EWE Tasker sends them to the app mobile. The actions that are received by the smart phone are specified in the smart phone channel inside the Ewe Tasker and they includes technologies that mobile phone incorporates such as: Wifi, Bluetooth, notifications tools, etc.

4.2.4 Devices Channel

The app device channel is a sub-module whose aim is to access to the different functionalities of the technologies that mobile phone incorporates, such has Wifi, Bluetooth, toast notification, etc to execute the actions triggered by the actions trigger module, so that it turn Wifi or Bluetooth on, it show a toast notification in the screen, it start a call to someone...

4.2.5 Beacons Channel

Additionally, this projects also includes an updated beacon channel, whose module has been incorporated from the last version of EWE Tasker APP, as we explained before. The aim of this channel is to allow the connection between beacons and the mobile app, so that users can use beacons events in their rules and creates automation task with them. The beacons interface 4.3 consist in an intelligent environment to listen beacons near user. When the app detects one, the beacons events are sent to the EWE Tasker. If exists rules configured by that beacon, the actions are returned to the mobile phone.

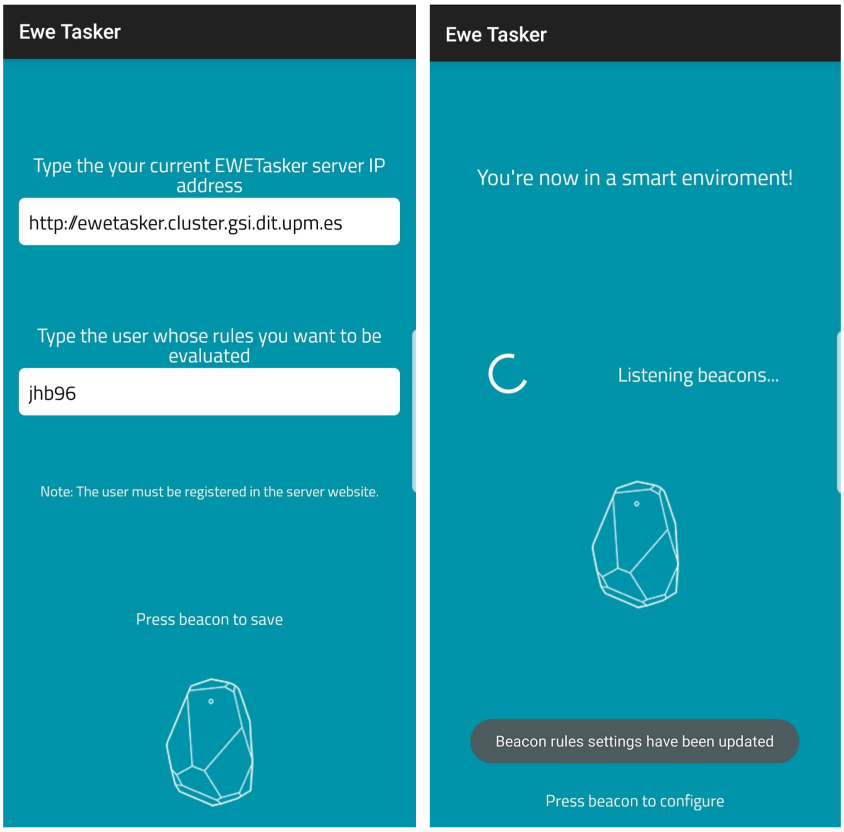

Figure 4.3: Beacons Interface

4.3 EWE Tasker Server Module

This web server developed by GSI manages the rules defined by the user, allowing the task automation between the activity events and different actions, customized by the user. A user has two possibilities to choose the executable actions when he creates a rule using activity events: It can to choose actions from the smartphone channel or another channel defined in EWE Tasker. In the first case, the module send the actions back to the mobile phone when a detected activity is received, so that the mobile phone can execute them. In the other case, the module send the actions to the device which the respective channel refers, such as a smart light, a beacon, a "chrome cast", etc.

4.3.1 Rule Engine

The Rule Engine is a module based ontology model whose goal is the rules creation, so that it is an essential module in EWE Tasker. This module architecture is divided into two parts: EYE Server and EYE Helper.

- Eye Server is an Euler Yap Engine reasoner implemented in JavaScript whose aim is process the events and rules written in Notation3 and generates a response with an action in text format.
- Eye Helper is a module implemented in PHP responsible for the reception of events from the Channel Administration and loading of stored rules in the Rules Administration module. After retrieves the events and rules, it sends them to EYE Server for being processed. Once Eye process them, it send back the response, containing the actions that must be triggered and EYE Helper sends to Channel Administration.

The process of handling events and triggering actions consist in:

EWE Tasker server receives a new event which is sent to Eye Helper. This module captures it and loads the available rules that uses this event from the Rule Administration module. Then, those rules and events are sent to the EYE Server[35], that analyze and obtain conclusions from them using the ontology model inferences. These conclusions are the actions represented by triples. The response is generated using Notation3 in a text format, so it is parsed later. It's important to know that the EYE Server is stateless. Thus, all events and rules are loaded before a new interference and removed after the inferences have been made. The same happens with actions when they are inferred.

4.4 Rule Administration

The main goal of this module is to offer users an automation rule creator and editor that allows to configure custom rules using the available events and actions in the existing channels. Besides, Rule Administration store these rules and it provides them to Rule Engine. The three main parts of this module are: Rule Manager, Rule Editor, Rule Repository.

4.4.1 Rule Editor

The rule editor provides a graphical interface to the creation, edition and removing rules, as shown figure 4.4. In order to make the process easy and intuitive the interaction is based on icons and "drag and drop" actions.

As explained in chapter 2.2, the rule creation process follows the structure "If this then that". "This" correspond to the chosen event available in a channel and "That" means the action existing in another or same channel. The module allows to create all possible rules between the available events and actions.

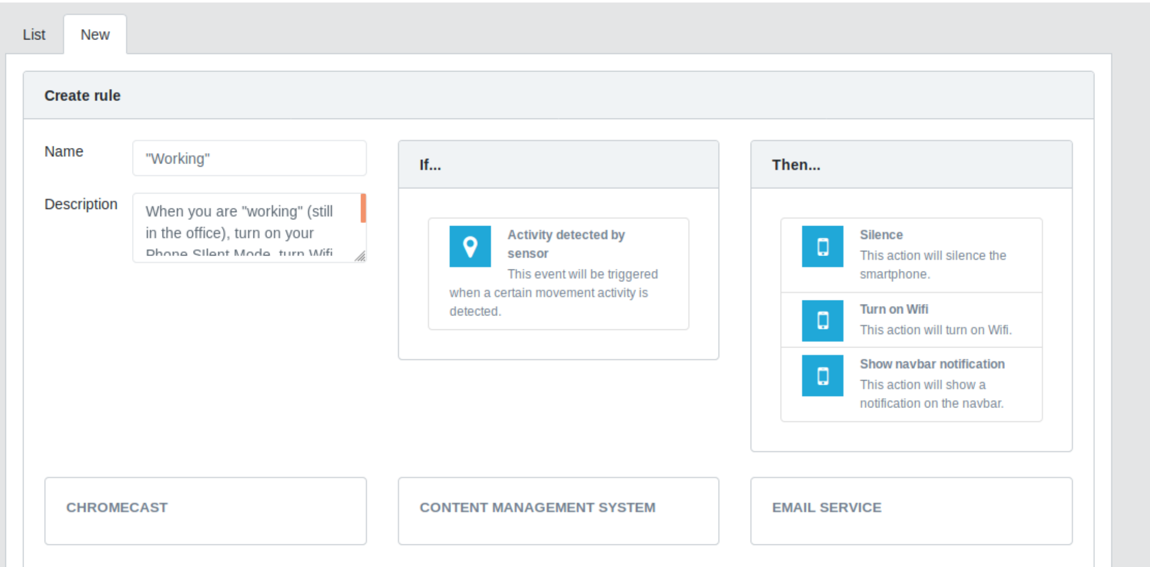

Figure 4.4: Rule Editor Interface

The parameters required to create a new rule are specifics of the channel and the device that the action or event belongs. Consequently, the parameter "user name" is required to create a Spotify Action or the "activity detected name" is required to create the specific activity event. For example, if i want to create the automation rule: "When i am running, play a song in Spotify", i should drag the event icon into the activity recognition channel and drop into the area reserved to events, writing "Running" in the event parameter space and choosing the activity recognition sensor device.

Then , i perform the same for action "play a song in Spotify": I drag and drop the play song event icon from Spotify channel to the actions area, specifying the Spotify parameters, such as a song, and the Spotify device assigned to my Spotify user name.

Finally, i can set the name and description of the rule before push the submit button to create the rule, which will be registered in the Rule Repository.

4.4.2 Rule Manager

Rule manager is a module created to manage the user rules and developed in PHP following the MVC patterns, whose architecture follows a three layered architecture: Model, View, Controller. User can access the pages that the View includes. These pages offers all the functionalities to create, edit and remove rules, processed by the Controller. The Controller manage all the data provided by the View and the Model. Also, this layer manage the requests sent from the user: requests made to the REST interface and the View pages requests. The Model is an abstract information layer whose aim is to access the information under the Controller request, so that create a connection to the corresponding database, extracting the information and returning it to the Controller. Rule manager graphic interface includes the existing rules including their details as shown the figure 4.5.

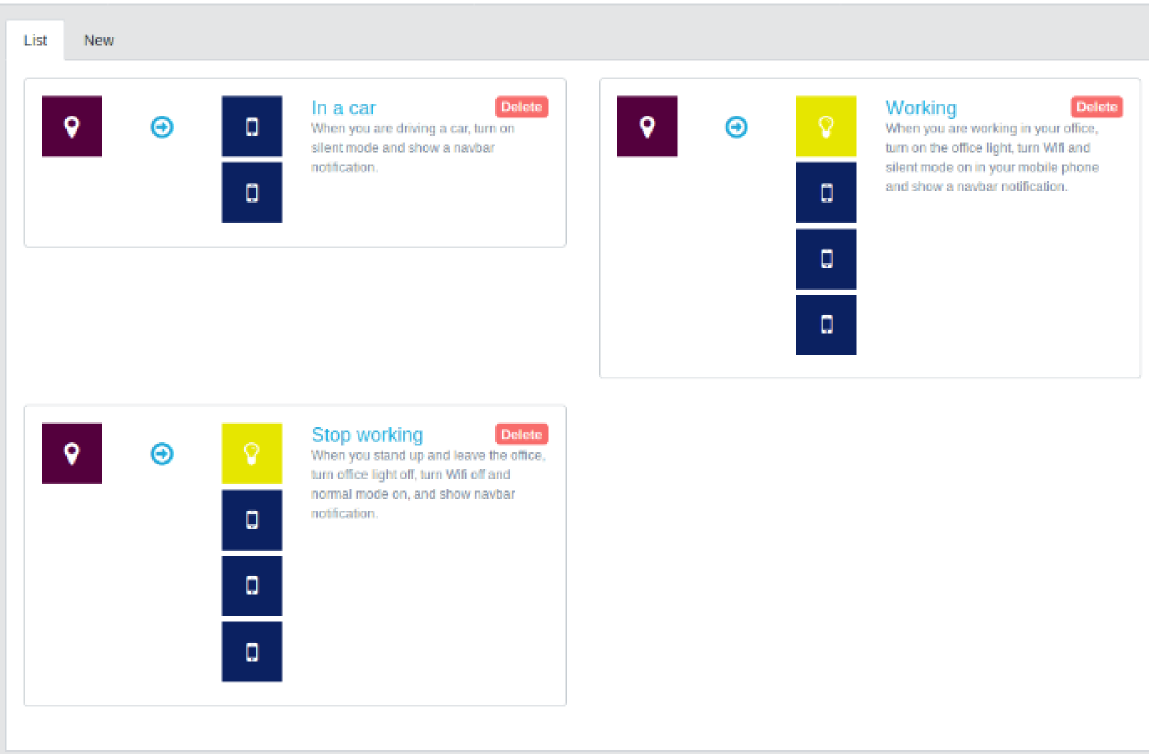

Figure 4.5: Rule Manager Interface

The interface functionalities consist in:

- Create rules, allowing to access to the rule editor to create a new rule from scratch.
- Edit rules, accessing to the rule editor of an existing rule. There the user can edit some parameters or change the rule completely.
- Remove rules using the “delete” button in a existing rule.

4.4.3 Rule Repository

All the rules are stored in a Fuseki Database [1] via N3 using EWE ontology, whose data is accessible through the SPARQL queries defined in the server.

4.5 Channel Administration

Channel Administration is a module which provides user an interface to visualized all created channels. Unlike the Rule Administration, this interface does not allows to create or delete channels. These functions are supported by Fuseki database server. This channel takes care of both providing the channels and events to and to triggering the actions coming from Rule Engine.

4.5.1 Channels Interface

The Channel Interface is based on a set of extendable boxes representing each channel, which offers the available events and actions for each one as is shown in figure 4.6. The channels can include events, actions or both, depending of the type of channel. For example, a sensor channel usually provides only events, or a Spotify Channel just offers actions. However, there are channels that includes both of them, such has Smartphone Channel.

[1] https://jena.apache.org/

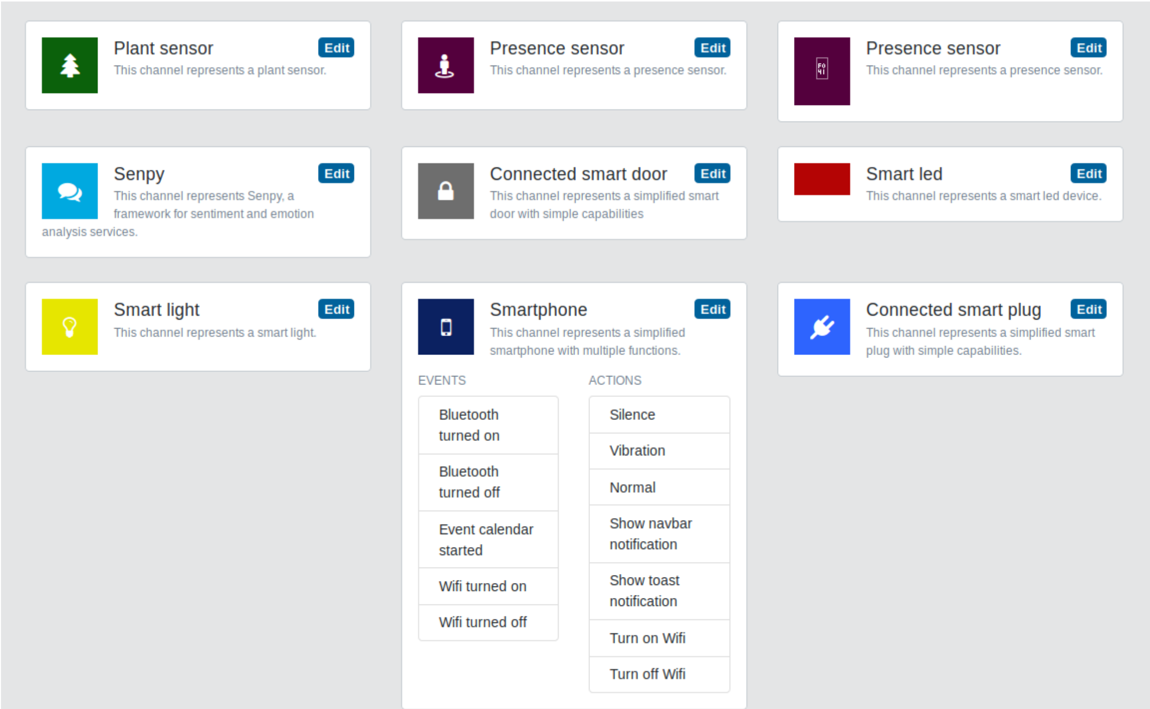

Figure 4.6: Channels Interface

4.5.2 Channel Creation

As we explain above, the creation of channels in EWE Tasker is handled by the Fuseki Server using N3 files. N3 files follows a EWE ontology structure as shown in the example 4.7. In order to create the Activity Recognition Channel of EWE Tasker, three elements have been defined:

- Activity Recognition Sensor which represents the human activity recognition channel in EWE Tasker.
- Detected Activity Event as Event definition. This sensor defines the activity events defining detected activity as output parameters. Activity recognition channel provides events but no actions. Thus, the detected activity sensor is only created to trigger activity events.
- Detected Activity as Parameter definition, which defines the activity event as the detected activity: still, in a vehicle, walking...

```
###################################
# Channel definition
###################################
ewe:ActivityRecognitionSensor a owl:Class ;
    rdfs:label "Activity Recognition Sensor" ;
    rdfs:comment "This channel represents an human activity recognition sensor" ;
    foaf:logo "fas fa-walking" ;
    dbo:colour dbo:Purple ;
    ewe:generatesEvent ewe:DetectedActivitySensor ;
    ewe:hasCategory ewe-device:Device ;
    rdfs:subClassOf ewe:Channel .
###################################
# Events definition
###################################
ewe:DetectedActivityEvent a owl:Class ;
    rdfs:label "Detected activity event" ;
    rdfs:comment "This event will be triggered when an activity is detected" ;
    ewe:hasOutputParameter ewe:DetectedActivity ;
    rdfs:subClassOf ewe:Event ;
    rdfs:domain ewe:ActivityRecognitionSensor .
###################################
# Parameters definition
###################################

ewe:DetectedActivity a owl:Class ;
    rdfs:label "Detected activity" ;
    rdfs:comment "This parameter represents the detected activity: still, walking, etc" ;
    rdf:datatype xsd:string ;
    rdfs:subClassOf ewe:Parameter ;
    ewe:operation string:equalIgnoringCase ;
    rdfs:subClassOf onyx:Activity;
    rdfs:domain ewe:ActivityRecognitionSensor .

###################################
# Actions definition
###################################
# This channel has not actions
```

Figure 4.7: Channel N3 File

4.5.3 Channel Repository

In the same way as rules, the channels are stored in Fuseki database via N3 using EWE ontology. In order to create new channels it is necessary to access to the Fuseki database server 4.8 and upload the N3 files defining the new channel, so that users can use their actions and/or events. Everytime that Ewe Tasker is loaded, the server request all the channel information from the Fuseki server.

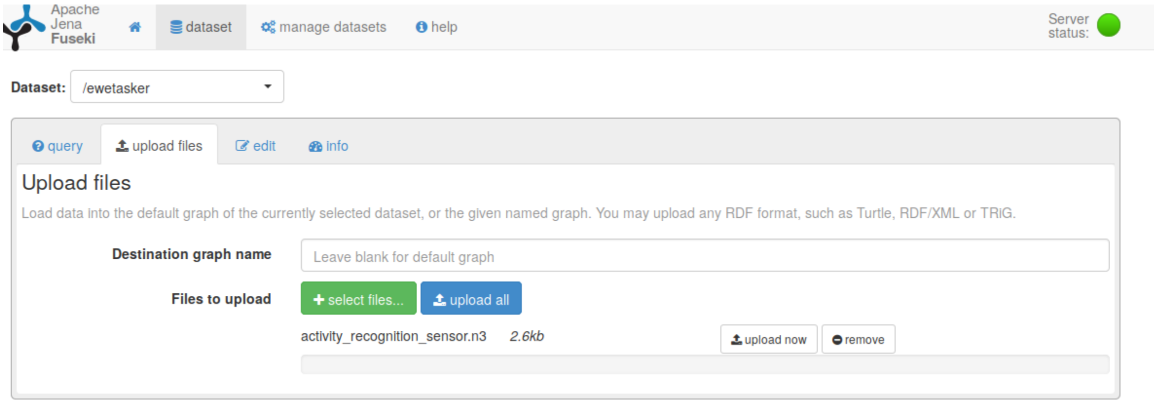

Figure 4.8: Fuseki Server

4.5.4 Events Manager

The Event Manager aim is to handle events from any device, in this project, from the smartphone and send them to the Rule Engine.

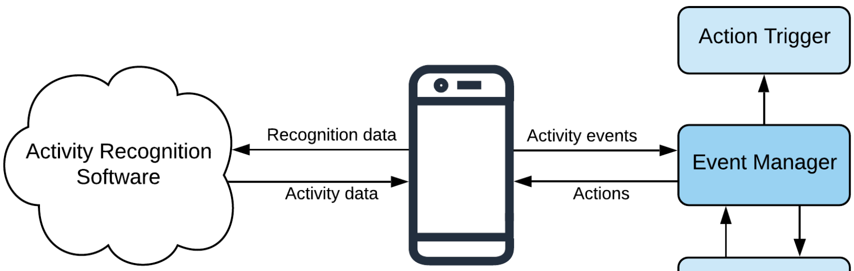

Figure 4.9: Events Manager Interconnection

4.5.5 Action Trigger

The Action Trigger module handles the actions that comes from Rule Engine and send them to any device, such as a smart light or the Smartphone. The actions returned to Smartphone may belong to the Smartphone Channel, such as turn on Wifi or show a toast notification, that uses the mobile phone internal services. However, there are actions belongs other channels that uses another application in the mobile phone, such as, play a song in Spotify (In Spotify Channel) or send a WhatsApp Message (in WhatsApp Channel).

4.5.6 Device Administration

The same way as the device channel module from phone module, this module incorporates all the devices created by the user, such as the user smartphone, the activity recognition sensor, a Spotify device with user credentials...

4.5.7 Device Manager

The devices interface includes functionalities to create, edit or remove devices, so that users can manage the devices they need use it. The delete button allows to remove a device.

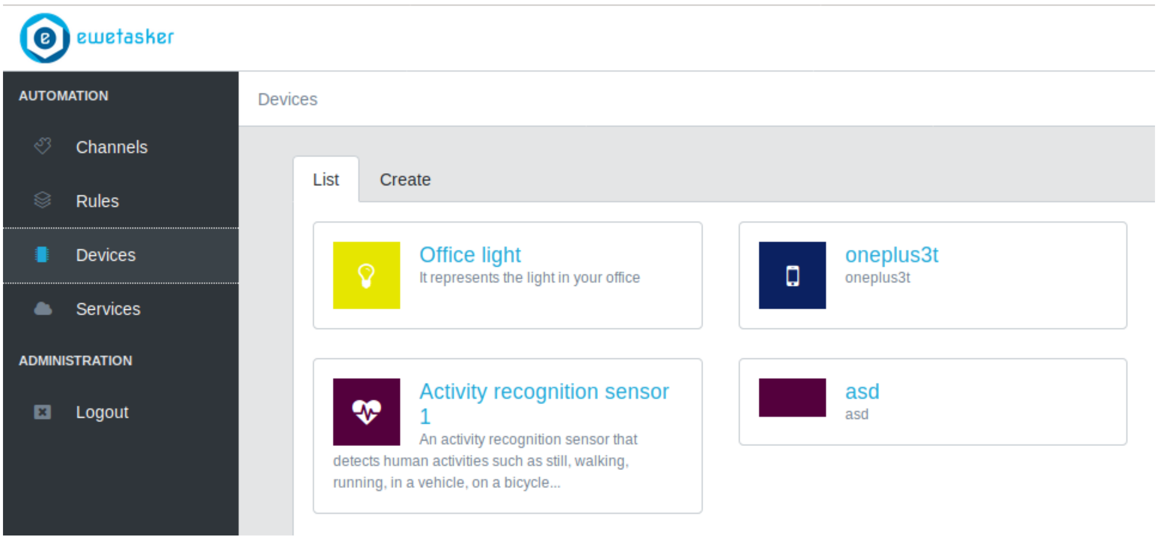

Figure 4.10: Device Manager Interface

- Push a device allow to access to the editor interface so that user cam edit device information
- Edit button redirects user to the creation interface where user can create a new device.
- Delete button allows to remove the selected device.

4.5.8 Device creator

The device creator goal is to allow users the devices creation. As we explained in the rule creation section 4.4.1, before use a channel in a rule, users must create a personal device assigned to a available channel. Thus, for example, users create a device representing an activity recognition sensor before use the activity events, as shown in figure 4.11, or they create a device assigned to a smartphone channel in order to use the smartphone actions, such as turn on Wifi. The channel name and the description are the only parameters required so that users can define the new device created and difference it from other devices of the same channel.

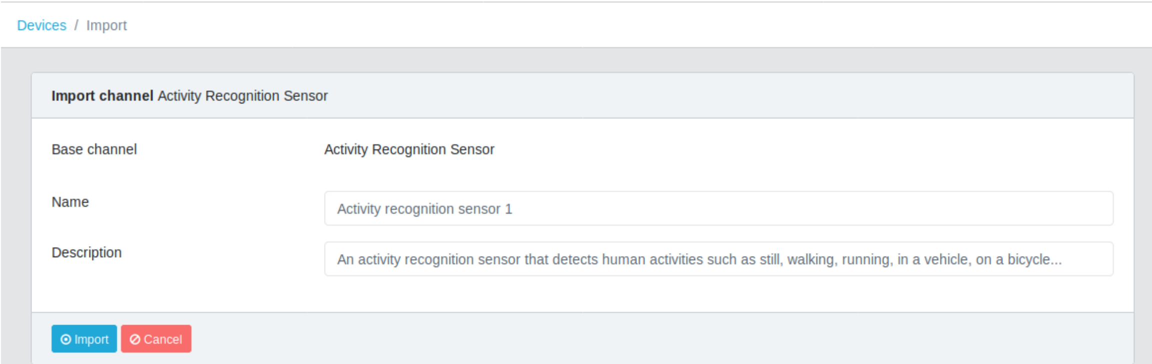

Figure 4.11: Device Creation Interface

4.5.9 Device Repository

The devices are stored in Fuseki database. The devices that user creates are unique and exclusive of the user. Thus, when the user connects he can access to the own devices but not access to devices which other users create.

CHAPTER 5

Case study

5.1 Introduction

In this chapter we are going to describe the study cases. This description will cover the main features, and its purpose is to understand the utility of both the activity recognition and the automation tasks as well as the activity monitoring in different contexts. The main actor in this project is the user, as we explained in chapter 3.1, whose aim is divided in two:

- to automate tasks using activity and beacons events and other actuators in a intelligent environment.
- to use the activity recognition to register his activity and know information and statistics about his activity both in the his work routine and in a normal activity routine, as well as automate custom rules using the activity events.

To achieve these objectives, two study cases have been tested. The first one recreates a intelligent environment in a office where user uses automation tasks that interacts with different actuators and his activity in work is monitored. The second situation proves the utility of monitor user activity and uses automation tasks in a real daily routine.

5.2 Automation Rules creation

Firstly, in order to start using automation tasks, user has to create the rules that he wants to manage in the Ewe Tasker Server [1]. He can access to the server from app or to any device with internet.

5.2.1 Login and Register

On the access to Ewe Tasker, the server will request a user login or register. User can creates a new account or just login if he already has a existing account. User authentication is need by the fact that the server stores personal devices and rules which only own user must use.

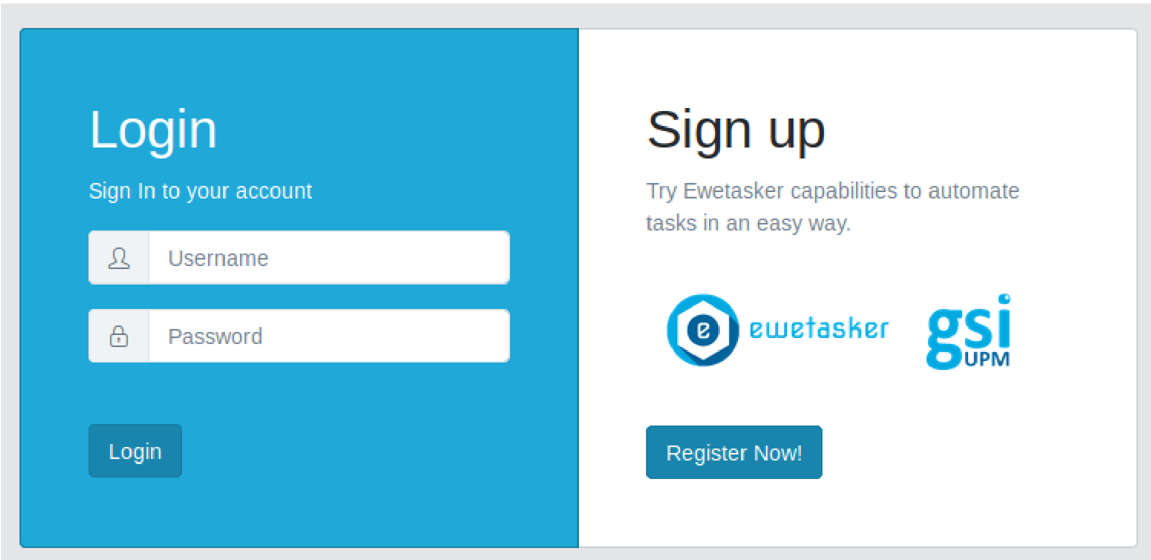

Figure 5.1: Ewe Tasker Login and Register

[1] http://ewetasker.gsi.upm.es/

5.2.2 Devices and rules creation

One he access the server, he proceeds to create the devices that he want to use (Beacons, a smartphone, smart lights and the activity recognition sensor), specifying parameters required such as name and description.

After do this, the system will interpret that the devices exists and user will be able to use it to create rules. Thus, user access to the rules manager interface and starts new rule creation. He creates a list of rules to automate tasks in his work routine, as is shown in figure 5.2. Finally, user can use the EWE Tasker service and the rules that he has created from Ewe Tasker app. To create other rules, the process followed is the same as the one explained here, but choosing other actions, events and devices.

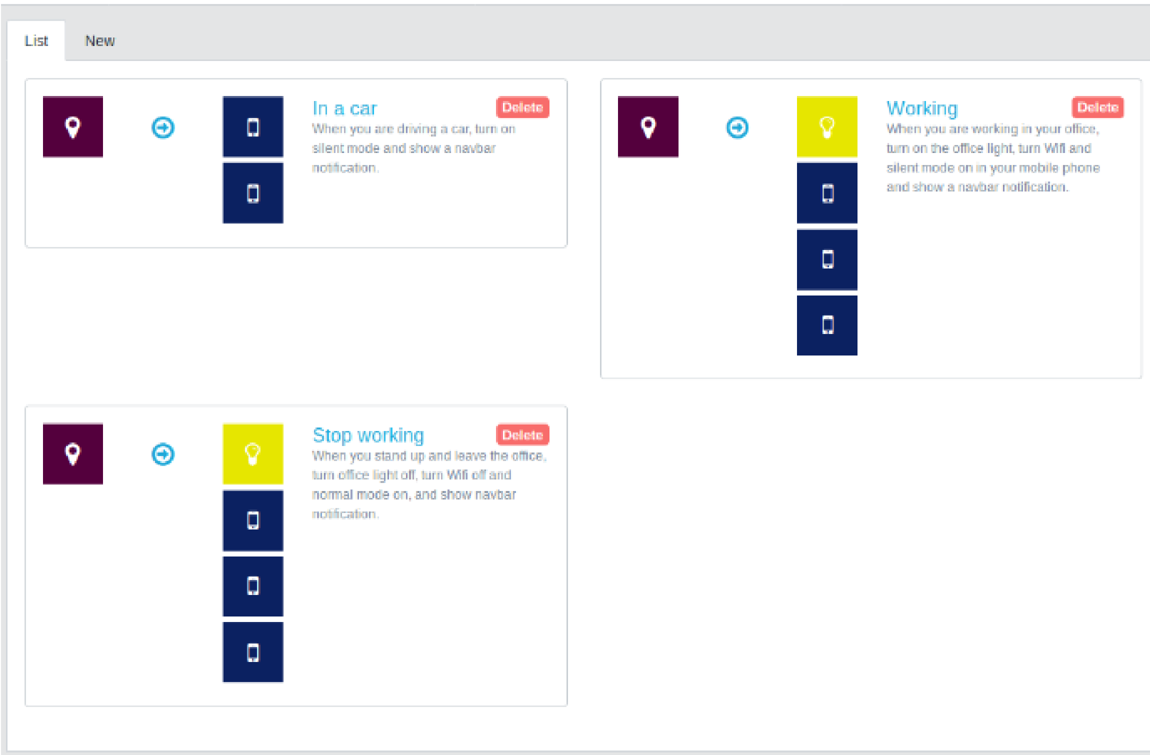

Figure 5.2: Rules Created List

5.3 Intelligent Office

This use case recreates an user that wants to automate tasks in his office in order to make more comfortable staying there, and also, to monitor his work activity so that he can know the spent time in the office working, resting or just making other things.

5.3.1 Environment description

This environment has been recreated in the laboratory of the Intelligent Systems Group [2] (GSI), that offers users an automation environment providing actuators such as beacons, smart lights or even, a smart door.

Figure 5.3 shows the described laboratory that includes different actuators: 3 beacons, 3 smart lamps and a smart door; and the different rooms where the user can work or rest.

The rooms defined are:

- Hall. It is the entrance of the office. This room incorporate a smart main door which only open with identification; the ”hall beacon” which detects the users who are going to enter the office; and the ”Hall Light” which turn on when someone is in hall.
- Rest room. It is the rest area where workers drink coffee and chat. The room includes the ”Rest Beacon” to detect users resting.
- Work room. The room where workers spend more time working. It incorporates a beacon and a smart light in each desk, called “Work Beacon” and “Work Light”.

[2] http://www.gsi.dit.upm.es/

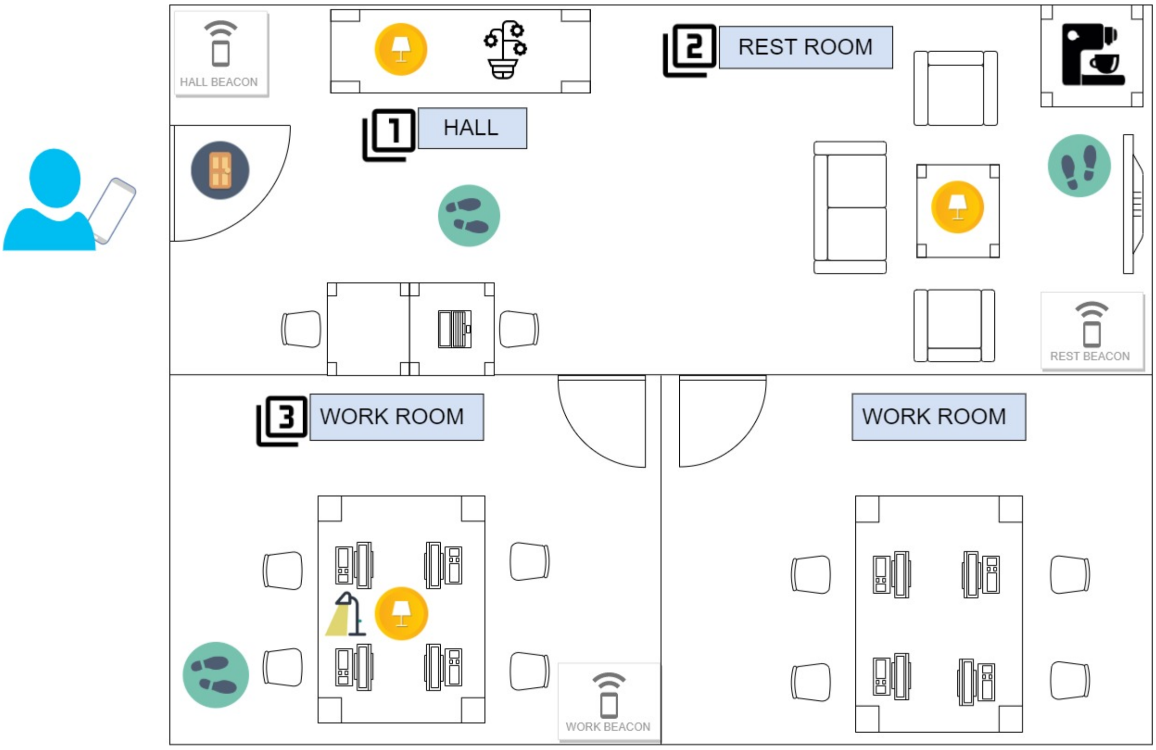

Figure 5.3: Intelligent Office Environment

5.3.2 Use case development

The use case starts when user arrives to the office and starts the EWE Tasker app, starting both the intelligent environment and the activity recognition. In order to open the main door, mobile phone receives a signal from the "Hall Beacon" and send the beacon event to Ewe Tasker, who process it and trigger the action associated to the rule "presence detected at door", allowing to open the door. Besides, another rule is executed, causing the "hall Smart light" turn on.

Once inside, user goes to the "rest room" to take a coffee and chat, when app detects the "Rest Beacon" signal and start tracking his activity in resting room and counting how long the user is there. After he drinks his coffee, he walks to the "Work Room" to start working.

In the "Work Room", the "Work Beacon" sends a signal that mobile phone detects, and it starts the activity recognition to monitor user work activity. In this room, user has got a personal smart lamp in his desk. In order to turn on the lamp, user has to be in the Work Room still in his desk during a time (a "still" activity event is required), so the beacon event is not enough to turn light on. The Ewe Tasker rule forces him to stand still for a while to prevent that the light is turning on and off continuously or it is turning on when he is doing something outside his desk. Finally, this rule causes that the mobile phone turn on silent mode, so that notifications do not bother the user while he is working.

5.4 Daily Routine

This scenario represents a user who uses the app in his diary routine, making it more comfortable. Firstly, he wants to know information about his physical activity, such as how much time he has ran or he has ridden his bicycle, or about how long he has driven his car today. Secondly, he wants to automate tasks in his daily routine in order to avoid small actions that may result annoying, such as turn on silent mode in his mobile phone when he is sleeping, and turn off when he wakes up or turn Wifi off when he leaves home.

Before starting his daily routine, the user created the following rules in Ewe Tasker in order to use automation tasks in his daily routine:

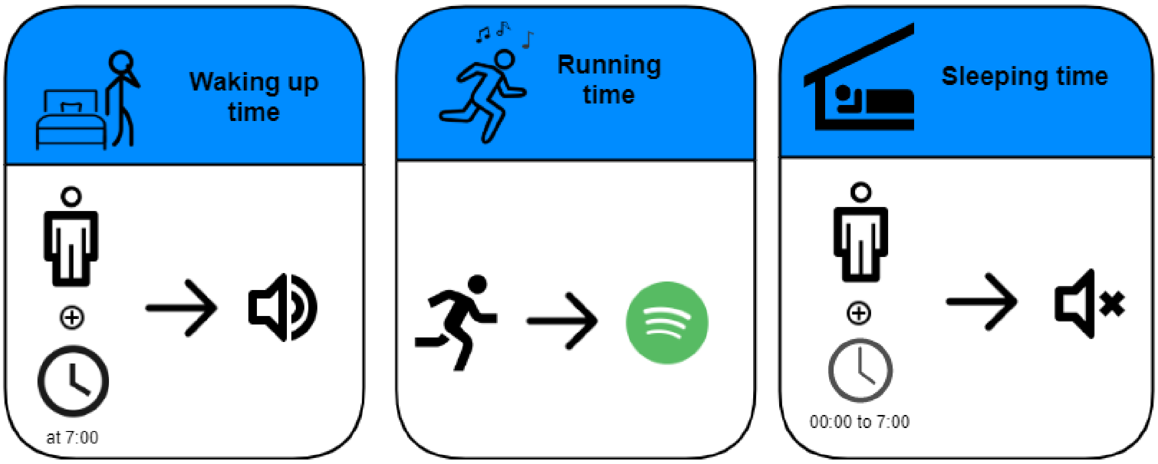

Figure 5.4: Daily Routine Rules

Once the rules have been created, user starts monitoring his daily routine. To do it, he initializes the app at morning, before go to work. In order to arrive at the office, he walks ten minutes to get the subway station, where he stay 30 minutes, approximately, and walks other 10 minutes since the station from the building where he works. He finishes working at 2 o'clock and goes back on the same way to home.

In the afternoon he decides to go to run for a while. He usually goes to a park

that is a little far from home to go walking, so he rides his bike for ten minutes to get the park. How he always listens music while he is running, he created a rule in EWE Tasker, in order to play music in Spotify when he starts to run. After run approximately 40 minutes, he comes back home ridding his bike again.
At night he watches the statistics and graphics of his diary routine and checks the carried out time in each activity today. Moreover, he access to the total information of his activity in the day, week and month to evaluate if the time he spends in physical activity has increased. Finally, he falls asleep from tiredness, but the “sleeping time rule” caused that mobile phone put the silent mode on from 00:00 o’clock to 7 o’clock, as he programmed.

The figure below shows the mobile app interface where the statistics about the activity which app tracked and collected in the day, in the week, and in the month, respectively, are shown.

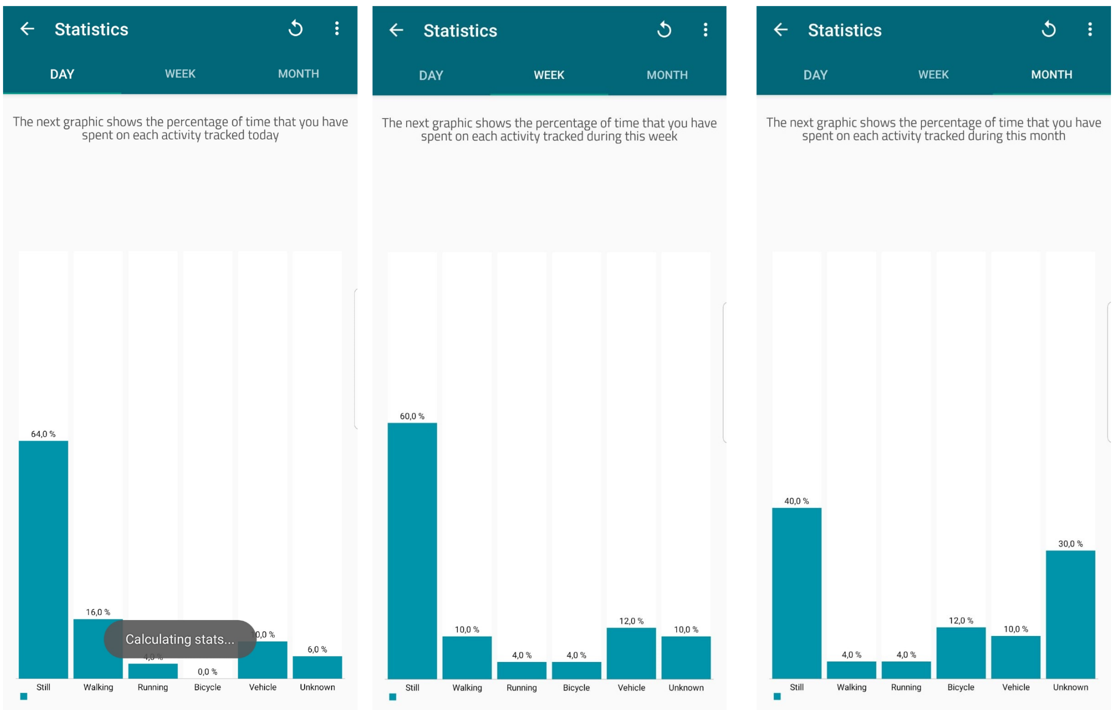

Figure 5.5: User Activity Statistics

5.5 Conclusions

Although this chapter don't show all the possible uses of the app, it give us an idea of the different contexts where the system can be applied. Should be added that the two main studied cases, in a smart office and in daily routine complement each other, since the smart office use case can be given in the daily routine.

Therefore, we can understand that the app can be used at all times and in all environments, whether it's smart environment or not despite some features are not available when the user is outside an environment intelligent.

CHAPTER 6

Conclusions and future work

In this chapter we will describe the conclusions extracted from this project, problems encountered, and the thoughts about future work.

6.1 Conclusions

Concluding all that we have explained in this document, a new version of Ewe Tasker app have been developed. The new version includes the activity recognition application, to monitor user activity, allow automation task using activity events and show statistics about user activity information. Moreover, it includes an updated version of Beacon application, to allow the automation task using beacons in a intelligent environment.

To make this application, the system have covered several different technologies, from powerful services of Google to web server technologies. The technologies used and studied in order to make the project have been:

- Web technologies: Java, Python, PHP, HTML5 in the Ewe Tasker server.

- Database technologies: MongoDB, Fuseki, Elastic Search, Firebase real time Database in order to save user data, server elements definitions, usage data and activity information, respectively.
- Semantic technologies: Notation3, EYE, RDF in the rules and channels definition.
- Mobile development technologies: Java, Objective-C.

It is important to remember that a part of this project has a important influence of the software architecture developed in a older project cited several times in this document [36].

Finally, in the next steps, we will summarize the achieved goals and the problems faced in the Project, as well as the future work and possible improvements in this Project.

6.2 Achieved goals

In the following section the features and goals achieved in this Project are explained:

Detect the activity that users are carrying out at all time using an activity recognition software. The Activity Recognition API of Google integrated in the mobile app allows to send Google sensor and recognition data in order to receive the tracked activities information from the API.

Generate the activity events to send it to Ewe Tasker. After detects the activity, the mobile app process the list of activities to get the most probably activity and if it has a high confidence the activity event is generated.

Create the activity recognition channel into Ewe Tasker in order to all users can use the activity events to make automation rules. Ewe Tasker recognizes this events and allows both the creation of recognition sensor devices and the incorporation of activity events into rule creation process.

Store activity data into a real time database so that it can be accessible at all time by the mobile app, making possible the activity monitoring.

Create a user graphic interface to display activity information. The mobile app offers users information about their activity at all time, both the activity tracked in each time and activities undertaking by the user in the last day, week or month via bar graphics.

Integrate and update the beacon task automation platform in the mobile app. Beacons are a really useful tool in a office to detect the user at all time. We decided to keep it in the mobile app so that user can totally monitor his activity at work.

6.3 Problems Faced

During the development of the project we had to face some problems. These problems are brie described in this section:

Rules storage. The rules storage has been changed over the last years. They were stored using Linked Data Fragments and even in a MongoDB database. Both ways had different problems. Thus, in this project we use Fuseki database to store rules written in N3, similar than channels, and MongoDB to store user information. This made difficult to retrieve the functionality of create and manage rules in the own mobile app.

Activity Recognition accuracy. As we explained before, the activity recognition needs to configure the activity detection interval of time. When users carry out an specific activity for a long time, such as running or even in a vehicle, they probably stop doing frequently because any factor, such as, a signal stop or traffic. In these cases, software works definitely better with slow Interval of time, such as 5 seconds because it waste much less resources and it get better accuracy. On the other hand, when the activity is changing constantly, such as in the office, the detection interval need to be really short, because a long Interval make really difficult to track the carried out activity. Finally we decided to take 3 second because get good results, as we explained before.

Beacons channel integration. The several changes in the updated versions Android made difficult to adapt the application. So, the Ewe Tasker version has not been migrated to Android X. This will be studied in future work.

6.4 Future work

In this section, some possible future features that could be developed will be explained in order to improve the Ewe Tasker application.

- Develop more sophistic graphics and statistics offering more details about the user dairy activity, such as specific graphics of each activity which shows the time per hour that user carried out the activity.
- Develop the iOS version of the application, allowing users who own an Iphone or any other IoS device to use EWE Tasker automation service.
- Implement the Google Play location API and integrate it into the Activity Recognition application, thus improving accuracy when user activities are detected and being able to create new automation tasks, including tasks combined with activity recognition, such as executing a specific action only if user is running in a specific place.
- New Channel Integration: In the current version, there are some internet channels in EWE Tasker such as Twitter or Spotify. It would be really interesting add more channels such as Instagram or LinkedIn, even Trello allowing new automation tasks.
- Migrate app to Android X to get the last version of Android.

APPENDIX A

Impact of the project

In this appendix, we are going to talk about the possible social, economic and environmental impact that this project could have in sections below. We will also give the possible ethical and professional implications of such project.

A.1 Social Impact

The social impact of this project can be measured in terms of user experience. These terms, extrapolated to the working context, imply an improvement in working results and a easier working experience adapted to the nowadays jobs.

In physical exercise context, this app can motivate people to walk and run, exciting them to get better time registers of both exercise.

Besides, it offers drivers to know the time they spent in a long travel and help them in order to make a rest stop. For example, they can program automated notifications to notify them when they are driving more than 2 hours.

A.2 Economic Impact

The economic impact for the business is clearly an improvement by the fact that they could monitor the worker activities with this app. A company could impose minimum requirements on the time workers are working to avoid wasting more time than necessary. This would entail to a higher worker performance and in consequently, greater benefits for the company.

A.3 Environment Impact

The development of this project and the subsequent implementation of the resulting system do not have a direct impact on the environment.
However, many times we spend light stupidly because we forget to turn it off, a fact that in an intelligent environment could be avoided. On a large scale like a company, it would reduce the total amount of energy consumed, therefore which would have an impact on the environment if it is a important company.

A.4 Ethical and Professional Implications

The ethical implications of this project are related to data collection. Data collection should always be carried out with users consent once they are informed of the purpose. The terms of use, when using free software platforms such as Moodle, are defined by the entities that implement these platforms in accordance with the legislation of each country and the treatment that will be made of the data. However, our system only collects user activity information when the user wish it and they can anytime decide about when we are tracking it.

APPENDIX B

Economic budget

In this appendix, we are going to resume the possible costs involved in this project development. Firstly, in first section we are going to calculate the human resources needed for this master thesis design and development. In addition, in next section the costs of the material resources needed for this project are going to be described. Finally, the licenses used in this project are going to be described in last section.

B.1 Human resources

In this section, we will take into account the time employed in the designing, developing and testing this system.

In this section, we will take into account the time employed in the designing, developing and testing this system. We will give an approximation based on the average salary of a Telecommunication Engineer, to find the cost of the development of the project. The estimate of the working time used to carry out this project has been calculated on the basis of ECTS credits 1. A master's thesis consists of 12 ECTS credits, each representing 25 to 30 hours of work. This makes a total of 360 hours of work, or two months of half-time work. If we consider a gross salary of 1500 euros per month, the cost of the project amounts to 2400 euros. This cost includes the design, development and testing tasks for the creation of the system. The cost of system maintenance and operation is not considered very high.

B.2 Material resources

The following material resources have been used to carry out this project. Firstly, a personal computer in which the design, development and testing tasks can be carried out. Secondly, a smartphone which has not to be necessarily powerful or modern.

The personal computer on which the project has been developed costs approximately 900 euros and the mobile phone cost 300 euros. Both of them has got good technical characteristics. The laptot features are:

- CPU: Intel Core i5 2.7GHz
- Memory: 16GB RAM DDR4
- Hard Disk: 250GB SSD

The mobile phone characteristics are:

- Ram: 6GB RAM DDR4
- Rom: 64 GB

B.3 Licenses

The software used in the development of the project is open-source software.

Bibliography

[1] Abhiditya Jha, Jess Kropczynski, Heather Richter Lipford, and Pamela J Wisniewski. An exploration on sharing smart home devices beyond the home. In *IUI Workshops*, 2019.

[2] Catia Prandi, Lorenzo Monti, Chiara Ceccarini, and Paola Salomoni. Smart campus: Fostering the community awareness through an intelligent environment. *Mobile Networks and Applications*, pages 1–8, 2019.

[3] Petar Radanliev, David De Roure, Jason RC Nurse, Razvan Nicolescu, Michael Huth, Stacy Cannady, and Rafael Mantilla Montalvo. New developments in cyber physical systems, the internet of things and the digital economy–discussion on future developments in the industrial internet of things and industry 4.0. 2019.

[4] Chien-Yuan Liu. A smart home automation system. In *Proceedings of the 3rd International Conference on Intelligent Technologies and Engineering Systems (ICITES2014)*, pages 381–388. Springer, 2016.

[5] Christopher Miller, Wende Dewing, Karen Haigh, David Toms, Rand Whillock, Christopher Geib, Stephen Metz, Rose Richardson, Stephen Whitlow, John Allen, et al. System and method for automated monitoring, recognizing, supporting, and responding to the behavior of an actor, February 12 2004. US Patent App. 10/341,335.

[6] Pascal Dresselhaus, Sven Goldstein, Hans Beckhoff, and Ralf Vienken. Connection unit, monitoring system and method for operating an automation system, January 31 2019. US Patent App. 16/149,988.

[7] Smail Benzidia, Blandine Ageron, Omar Bentahar, and Julien Husson. Investigating automation and agv in healthcare logistics: a case study based approach. *International Journal of Logistics Research and Applications*, 22(3):273–293, 2019.

[8] Nishad Joshi and Nikita Virkud. Gsm based security automation system for building entry management. 2019.

[9] Duc Ngoc Tran and Duy Dinh Phan. Human activities recognition in android smartphone using support vector machine. In *2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*, pages 64–68. IEEE, 2016.

[10] Kunlun Zhao, Junzhao Du, Congqi Li, Chunlong Zhang, Hui Liu, and Chi Xu. Healthy: A diary system based on activity recognition using smartphone. In *2013 IEEE 10th International Conference on Mobile Ad-Hoc and Sensor Systems*, pages 290–294. IEEE, 2013.

[11] Jie Yin, Qiang Yang, and Jeffrey Junfeng Pan. Sensor-based abnormal human-activity detection. *IEEE Transactions on Knowledge and Data Engineering*, 20(8):1082–1090, 2008.

[12] Jake K Aggarwal and Lu Xia. Human activity recognition from 3d data: A review. *Pattern Recognition Letters*, 48:70–80, 2014.

[13] Muhammad Shoaib, Stephan Bosch, Ozlem Durmaz Incel, Hans Scholten, and Paul JM Havinga. A survey of online activity recognition using mobile phones. *Sensors*, 15(1):2059–2085, 2015.

[14] Song-Mi Lee, Sang Min Yoon, and Heeryon Cho. Human activity recognition from accelerometer data using convolutional neural network. In *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 131–134. IEEE, 2017.

[15] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *Esann*, 2013.

[16] Muhammad Shoaib, Hans Scholten, and Paul JM Havinga. Towards physical activity recognition using smartphone sensors. In *2013 IEEE 10th international conference on ubiquitous intelligence and computing and 2013 IEEE 10th international conference on autonomic and trusted computing*, pages 80–87. IEEE, 2013.

[17] Giuseppe Cardone, Andrea Cirri, Antonio Corradi, Luca Foschini, Raffaele Ianniello, and Rebecca Montanari. Crowdsensing in urban areas for city-scale mass gathering management: Geofencing and activity recognition. *IEEE Sensors Journal*, 14(12):4185–4195, 2014.

[18] May Thet Htar Nyo and Win Zaw Hein. Design and construction of navigation based auto self-driving vehicle using google map api with gps. *Int. J. Trend Sci. Res. Dev*, 3:65–68, 2019.

[19] Giuseppe Cardone, Andrea Cirri, Antonio Corradi, Luca Foschini, and Rebecca Montanari. Activity recognition for smart city scenarios: Google play services vs. most facilities. In *2014 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE, 2014.

[20] Xiami includes more Google Services. `https://elandroidelibre.elespanol.com/2020/01/los-moviles-de-xiaomi-usaran-de-serie-mas-apps-de-google.html`. El android libre; accessed 3 November 2019.

[21] Tanzeem Choudhury, Gaetano Borriello, Sunny Consolvo, Dirk Haehnel, Beverly Harrison, Bruce Hemingway, Jeffrey Hightower, Karl Koscher, Anthony LaMarca, James A Landay, et al. The mobile sensing platform: An embedded activity recognition system. *IEEE Pervasive Computing*, 7(2):32–33, 2008.

[22] Thanyalak Rattanasawad, Kanda Runapongsa Saikaew, Marut Buranarach, and Thepchai Supnithi. A review and comparison of rule languages and rule-based inference engines for the semantic web. In *2013 International Computer Science and Engineering Conference (ICSEC)*, pages 1–6. IEEE, 2013.

[23] Amir Rahmati, Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Ifttt vs. zapier: A comparative study of trigger-action programming frameworks. *arXiv preprint arXiv:1709.02788*, 2017.

[24] Miguel Coronado, Carlos A Iglesias, and Emilio Serrano. Modelling rules for automating the evented web by semantic technologies. *Expert Systems with Applications*, 42(21):7979–7990, 2015.

[25] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. Rdf 1.1 turtle. *World Wide Web Consortium*, 2014.

[26] Tim Berners-Lee and Dan Connolly. Notation3 (n3): A readable rdf syntax. w3c team submission 28 march 2011, 2011. *Last Accessed*, 24, 2016.

[27] Thanyalak Rattanasawad, Marut Buranarach, Kanda Runapongsa Saikaew, and Thepchai Supnithi. A comparative study of rule-based inference engines for the semantic web. *IEICE TRANSACTIONS on Information and Systems*, 101(1):82–89, 2018.

[28] M Coronado, CA Iglesias, and E Serrano. Ewe ontology specification, 2015.

[29] Apache Jena. Apache jena fuseki. *The Apache Software Foundation*, 2014.

[30] Mongo DB. `https://www.mongodb.com/es`. MongoDB, accessed 10 October 2019.

[31] Elastic Search. `https://www.elastic.co/es/products/elasticsearch`. Elastic Search, accessed 10 October 2019.

[32] UML. `https://en.wikipedia.org/wiki/Unified_Modeling_Language`. Unified Modeling Language, Wikipedia; accessed 5 December 2019.

[33] Sergio Munoz López. *Development of a Task Automation Platform for Beacon Enabled Smart Homes*. PhD thesis, Master's thesis, ETSI Telecomunicación, Universidad Politécnica de Madrid, 2016.

[34] EWE Tasker. `https://github.com/gsi-upm/ewe-tasker-android`. Ewe Tasker GitHub.

[35] R. Verborgh and J. De Roo. Drawing conclusions from linked data on the web: The eye reasoner. *IEEE Software*, 32(3):23–27, May 2015.

[36] Sergio Muñoz, Antonio F Llamas, Miguel Coronado, and Carlos Angel Iglesias. Smart office automation based on semantic event-driven rules. In *Intelligent Environments (Workshops)*, pages 33–42, 2016.

[37] Oscar Araque. Design and Implementation of an Event Rules Web Editor. Trabajo fin de grado, Universidad Politécnica de Madrid, ETSI Telecomunicación, July 2014.

[38] J. Fernando Sánchez-Rada. Design and Implementation of an Agent Architecture Based on Web Hooks. Master's thesis, ETSIT-UPM, 2012.

[39] Master's thesis.

[40] Allan Askar. Internet of things (iot) device registration, May 14 2019. US Patent 10,291,477.

[41]

[42] Jayeeta Saha, Arnab Kumar Saha, Aiswarya Chatterjee, Suyash Agrawal, Ankita Saha, Avirup Kar, and Himadri Nath Saha. Advanced iot based combined remote health monitoring, home automation and alarm system. In *2018 IEEE 8th annual computing and communication workshop and conference (CCWC)*, pages 602–606. IEEE, 2018.

[43] Ozlem Durmaz Incel, Mustafa Kose, and Cem Ersoy. A review and taxonomy of activity recognition on mobile phones. *BioNanoScience*, 3(2):145–171, 2013.

[44] Mohammad Derawi and Patrick Bours. Gait and activity recognition using commercial phones. *computers & security*, 39:137–144, 2013.

[45] Jordan Frank, Shie Mannor, and Doina Precup. Activity recognition with mobile phones. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 630–633. Springer, 2011.

[46] Mikko Rinne, Seppo Törmä, and D Kratinov. Mobile crowdsensing of parking space using geofencing and activity recognition. In *10th ITS European Congress, Helsinki, Finland*, pages 16–19, 2014.

[47] Jad Helmy and Ahmed Helmy. The alzimio app for dementia, autism & alzheimer's: Using novel activity recognition algorithms and geofencing. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–6. IEEE, 2016.

[48] Sergio Muñoz, Oscar Araque, J Sánchez-Rada, and Carlos Iglesias. An emotion aware task automation architecture based on semantic technologies for smart offices. *Sensors*, 18(5):1499, 2018.

[49] Yadid Ayzenberg, Javier Hernandez Rivera, and Rosalind Picard. Feel: frequent eda and event logging–a mobile social interaction stress monitoring system. In *CHI'12 extended abstracts on human factors in computing systems*, pages 2357–2362. ACM, 2012.

[50] Firebase. `https://firebase.google.com/?hl=es-419`. Firebase, accessed 5 September 2019.