Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (30th cycle)

IoT Notifications: from disruption to benefit

Architectures for the future of notifications in the IoT

By

Teodoro Montanaro

******

Supervisor(s):
Fulvio Corno, Supervisor
Pino Castrogiovanni (TIM), Co-Supervisor

Politecnico di Torino
2018

Declaration

I would like to dedicate this thesis to my parents, my brother, and all the people (uncles, aunts, cousins, friends and grandmother) who believed in me.

Acknowledgements

I know that it is not a good practice to use emoticons in a doctoral thesis, but I think I can do it in the acknowledgments paragraph.

As almost everyone knows, I love to speak :-). I'm not as good as my brother to establish a conversation with every person, animal or object present on the earth XD, but I cannot declare that I can stay more that a minute without speaking. Now that it is clear, I will use the same approach I usually use when I am near a person (and I am not tired, this is the only situation in which I could be a silent person) to say thanks to the people who helped me during my Ph.D.

At first, I want to thank all the people that guided me and "walked" near me during the Ph.D.! Specifically, I want to thank Fulvio Corno and Luigi De Russis, my mentors. Their patience and support allowed me to grow, improve my skills, and, most important, arrive at the end of the Ph.D. :-) In addition, I would like to thank them, together with Laura Farinetti, a teacher of rare goodness, for the opportunity they gave me to discover my passion in teaching. They involved me in the teaching activities of the two courses "Ambient Intelligence" and "Social Networking: technologies and applications" letting me improve my teaching skills and, maybe, find my future way.

Then, I would like to thank all the other members and alumni of the e-Lite research group for their support: Sebastian Aced Lopez, who convinced me to continue my training with a Ph.D., Alberto Monge Roffarello and Juan Pablo Sáenz that started their Ph.D. in our research group one year before my departure, and Fabio Ballati who started a few days before my last day.

Moreover, I'm indebted with all the TIM members: Pino Castrogiovanni, my industrial tutor and Claudio Borean who guided me in cooperating within the JOL Swarm team. Also, I would like to thank Dario, Ennio, Roberta, Giuseppe and

Alfonso who cooperated with me in some projects and that taught me a lot of industrial tricks that I am appreciating now in my new job.

Furthermore, I want to thank all the people I have met in these years, even if only for a few moments, in the department, in the laboratory ("lab6" has been my second house during the Ph.D., so Alessandro, Andrea, Francesco, Roberta, Alberto and Giuseppe were my second family) and also all around the Politecnico.

And it is not finished: I would like to thank Andrea Marcelli for involving me in some of his projects and supporting me in almost all the machine learning experiments, and also, Edoardo Fadda, Giovanni Zenezini, Maliheh Ghajargar and Rosario Scatamacchia for their patience in the collaborations we made during some extra projects carried on during the Ph.D.

I'm proud of have been part of the DAUIN department at Politecnico di Torino but also of the TIM JOL Swarm team and I hope to continue to meet all of you in the future! ;-)

Finally, I would like to thank all the people who helped me by testing the XDN framework: Alberto Monge Roffarello, Luca Venturini, Alberto Cannavò, Fabio Ballati, Federica Bazzano, Federico Salaroglio, Gianpaolo Paterno, Giovanni Piumatti, Giuseppe Ministeri, Juan Pablo Sáenz, Luca Mannella, Orazio Scivolone, Tamer Saadeh and Mohammad Ghazi Vakili.

Ok, if you arrived here and you are not yet in the list, the answer to your question is: NO, I did not forget you. However, I wanted to leave you at the end to oblige you to read everything XD!

I am forever grateful to my parents, Rosario and Francesca, my brother Andrea, all the members of my big family (aunts, uncles, cousins, and grandma) and all my friends (you are too many to be mentioned in a few lines, so I will thank you personally after the discussion XD) who have always supported and encouraged me.

And last, but not the least, I would like to thank the reviewers of this thesis: even though I will discover who you are only when I will have finished to write the thesis, I would like to thank you for your work and your suggestions.

Abstract

The growing number of mobile and IoT devices able to generate and show incoming notifications is fostering the spread of notifications in people lives. Nonetheless, although users are getting used to them, their presence is not always perceived as a benefit by recipients. With the aim of improving user experience with notifications, two different approaches are presented in this dissertation. The former acts at the distribution level, i.e., notifications are intercepted and then a system decides if, when, and how to show them; while the latter acts at the design level, i.e., notifications and their distribution strategies are designed with the aim of reducing user disruption and exploiting all the benefits that the availability of multiple devices could bring.

An IoT architecture is proposed for each approach: the Smart Notification System that relies on machine learning algorithms to adequately manage incoming notifications, and the XDN (Cross-Device Notification) framework that assists developers in creating cross-device notifications by scripting. The modular nature of both architectures allowed the simultaneous development and test of different independent but compatible subsystems and their exploitation in preliminary deployment sessions. The results, feedbacks and lessons learned from such sessions can foster the development of future solutions in the IoT notifications field and related domains.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Thanks to the continuous introduction of new devices and services, technology has become part of citizens' lives during the last decades and it is really difficult to think at any moment of the day in which it has not been transferred. Thanks to the useful services that they provide, in fact, tablets, smartwatches, smartphones, smart TVs, smart fridges, smart washing machines and other common devices became part of our lives and changed our habits and behaviors. Indeed, for example, since a few decades ago, no one would have imagined at the possibility of controlling the house heating system by her voice except in science fiction. In addition, since a few decades ago, our parents used to consult a paper map before every travel, while now we are accustomed to real-time positioning systems able to suggest the shortest path, also depending on the current traffic condition.

All these examples are only some of the innovations that are fostering the spread of one of the most pronounced concept of the last decades: the Internet of Things (IoT). It is the network of physical objects that are always connected to the Internet with the aim of sharing services and information with other connected "things".

The next paragraphs present some details about this innovative network of objects also reporting the most important elements that constitute an IoT system and some example of application of the IoT.

1.1.1 Internet of Things (IoT)

One of the hottest topic of the last decade is certainly the Internet of Things (IoT). Its spread is demonstrated by the huge number of different synonyms that were already introduced to represent the same concept. Internet of Everything (IoE), Industrial Internet, Pervasive Computing, Pervasive Sensing, Ubiquitous Computing, Cyber-Physical Systems (CPS), Wireless Sensor Networks (WSN), Smart Objects, Cooperating Objects, and Machine-to-Machine (M2M) are only a minimal part of the list of synonyms that compose the so called "terminology zoo" surrounding IoT.

Cisco, one of the biggest companies that is investing in IoT in recent years, declares that one of the first use of the term "IoT" dates back to 1999[1]. In fact, in a 1999 article for the RFID Journal, Ashton Kevin wrote the following visionary paragraph: *If we had computers that knew everything there was to know about things using data they gathered without any help from us, we would be able to track and count everything, and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best. We need to empower computers with their own means of gathering information, so they can see, hear and smell the world for themselves, in all its random glory. RFID and sensor technology enable computers to observe, identify and understand the world without the limitations of human-entered data.*[2]

Due to the early implementation of some technologies and the absence of some others, this declaration only represented the visionary point of view of a society that was starting to think at the possibility of exploiting the Internet to develop new devices and services. Although, since then, a lot of obstacles have been solved and, thanks to the rapid proliferation of connectivity and the introduction of sensors and actuators in everyday physical devices, the Internet of Things (IoT) is becoming reality in recent years [3].

As described by Cisco, IoT represents systems in which "things" (i.e., objects in the physical world) are connected to the Internet via wireless and wired Internet connections with the aim of collecting and sharing data and services. To better understand this definition, the next paragraphs present both the essential blocks that compose an IoT object/system and some examples of application in the IoT domain.

[1]https://www.cisco.com/c/dam/en_us/solutions/trends/iot/introduction_to_IoT_november.pdf
[2]http://www.rfidjournal.com/articles/view?4986

1.1.2 IoT building blocks

This section presents the main building blocks that an object, a sensor, or a system should possess to act in the IoT domain. They are presented with the aim of helping readers in understanding what the IoT is.

As effectively summarized by Atzori et al. [4], the actualization of the IoT concept into the real world is possible through the integration of several enabling technologies:

- Identification, sensing and communication technologies; they represent all the components that are encouraging the adoption of new objects in people daily routine actually leading to the IoT concept.
  - Identification: as declared by Al-Fuqaha et al. [5], it is essential for every IoT object to be uniquely identified within a communication network. For this purpose, two different properties may be exploited: the "object ID", that represents the name assigned to the object and allows humans to identify the object, and the "object address", that allows to reach the object and its exposed services within the network in which it is installed.
  - Sensing: another important property that could be adopted by an IoT object resides in the possibility of gathering data from related objects within the network and sending it back to a data warehouse, database, or cloud. In general, the collected data is analyzed to take specific actions based on required services. The sensors can use various types of local area connections (e.g., RFID, NFC, Wi-Fi, Bluetooth, and Zigbee) or wide area connectivity (e.g., GSM, GPRS, 3G, and LTE) to transmit or receive data to/from the connected other things. In addition, due to the high variability of both data and connectivity used to transmit it, such data differs from traditional computing ones because of their small size and the high frequency used for its transmission.
  - Communication: as declared by Al-Fuqaha, the IoT communication technologies connect heterogeneous objects together to deliver specific smart services. Typically, the IoT nodes should operate using low power in the presence of lossy and noisy communication links.

- Middleware; it represents a software layer that hide the details of different technologies that act behind the scene. It aims at abstracting the devices functionalities and communications capabilities, providing a common set of services. As summarized by Atzori et al. [4], it relies on 5 main components.
  - “Applications”: this component represents all the elements that export the system’s functionalities to the final user and let her interact with them. These elements include both software elements, e.g., graphical user interfaces or notifications, and physical devices, e.g., smart lamp or interactive touch screen that let the user interact with the system.
  - “Service composition”: it represents all the functionalities aimed at the composition of single services offered by networked objects then used to build specific applications. At this level, there is no notion of devices and the only visible assets are services.
  - “Service management”: it includes all the functions that are expected to be available for each object and that enable their management in IoT scenarios.
  - “Object abstraction”: IoT relies on a vast and heterogeneous set of objects, each one providing specific functions accessible through its own dialect. Thus, there is the need to harmonize the access to the different devices with a common language and procedure.
  - “Trust, privacy and security management”: the deployment of automatic communication of objects represents a danger for people future. The middleware must consequently include functions related to the management of the trust, privacy and security of all the exchanged data.

Even though, in some situations, these elements could be distributed among different devices, they represent the essential requirements for an object or a system that acts in the IoT domain.

1.1.3 IoT applications

In recent years, the interest in IoT is growing and the range of domains in which it is applied is raising in parallel. This subsection presents an updated list of some of the most explored areas in which IoT has been fostered up to the moment of writing.

It is inspired by the two works proposed by Kamilaris et al. [3] and by Rehman et al. [6]. The first is mainly focused on mobile phones, but considering that the IoT and mobile domains are strictly related, the contained information are used to enrich the effective analysis proposed by Rehman et al. [6].

- Participatory Sensing: it represents all the activities and the projects that encourage users to record and share information with the aim of cooperating to a general public shared objective. An example in such domain is represented by the use of bikes as IoT probes to collect data for monitoring the city air quality.
- Eco-Feedback: this category represents all the projects in which technology is used to give users a feedback about their personal impact on public environmental phenomena or events. For instance, the user personal smartwatch could be used to understand how much time the user usually spends in driving and how this time is influencing the city air quality.
- Actuation and Control: it represents cases in which technology is used to control physical devices. A smart home heating system is the most common example of such domain: the user, for instance, can set the system to have a constant temperature in the house.
- Health: it represents all the circumstances in which electronic devices are used to monitor the user's health. All the modern smart bracelets and smartwatches are, for example, designed to monitor user heart rate or sleep quality. Such information can be used to feed a smart system able to suggest which is the best moment to drink or to have a rest.
- Sports: This domain represents all the systems that use a combination of various sensors (e.g., heart rate, foot counter, etc.) during sport activities to record various information and help users to improve their performance.
- Agriculture: it is related to smart farming practices aiming at improving productivity, management of livestock and increase consumer satisfaction and transparency.
- Gaming: this area is about virtual games which foster the physical presence or status of the user to enhance the gaming experience.

- Transportation: it represents all the projects in which sensing features are harnessed to foster and improve user driving experience and/or parking.
- Interaction With Things: this category represents all the efforts focused on interacting with physical entities that are located near the user. An example of such area is a smart exhibition area like the one we presented in [7] that, for instance, allows visitors to interact with exposed innovative IoT devices (e.g., car robot).
- Social Interactions With People: this area groups all the projects that foster data extracted from online social networks or similar platforms to provide innovative services that, for example, are able to predict a user behavior.
- Security and Surveillance: security has been underestimated for a lot of years but, recently, it has become one of hottest topic treated in both research and industry sectors. Technologies proposed by the IoT domain, in fact, provide new ways and opportunities to adopt security measures in different domains. As an example, IoT can be applied to a house door by the use of IoT door locks to allow users to close the door remotely, but also by the use of smart power appliances that can avoid fire in case of user oversights (e.g., coffee pot forgotten on the stove).

1.1.4 IoT notifications

Since the early introduction of mobile devices in our society, notifications have earned an important role in providing to users the needed information in the right moment (e.g., the reminder of a business call sent 10 minutes before the call). However, since the introduction of IoT, their role has been changing.

In this thesis, the terms “common notifications” and “mobile notifications” will be used as synonyms to indicate notifications usually generated and/or notified on the devices used before the advent of IoT (e.g., smartphones or PC), while the term “IoT notifications” will be used to indicate the notifications usually generated and/or notified on all the existing IoT devices, in addition to all the devices already used by “common notifications”.

The term notification is usually used to depict all the messages and actions sent or performed to notify useful information to a receipt (e.g., a user or a system). Each notification is mainly composed by four important components:

- the source of the notification;
- the destination of the notification;
- the content of the notification;
- the modality used to show the arrival of the notification.

The differences among "common" and "IoT" notifications are equally distributed in all the components.

At first, the introduction of IoT added a huge number of heterogeneous sources and destinations for notifications. New devices and services are in fact introduced every day with the aim of helping people in their daily activities and almost all of them are able to generate and/or receive notifications. A smart toothbrush, for example, can generate a notification about its battery status or also the high pressure that the user is putting on her teeth. And, at the same time, it could vibrate as soon as a new notification arrives to the user while the user is cleaning her teeth.

In addition, also the content of the notification has drastically changed with the introduction of IoT. Looking at the previous example, in fact, the notifications generated by the smart toothbrush are really different from the ones generated before the advent of the IoT (e.g., a calendar reminder).

Finally, IoT has also introduced new ways of notifying people about incoming notifications. As an example, the red color of a smart lamp can now be used to warn the user about an event that requires her maximum attention (e.g., alarms).

1.2 Thesis Motivation

As depicted in the previous section, one of the main IoT enabling technologies is the middleware that is mainly exposed to end users through the "Applications", specific features, devices or services with which user can interact.

One of the features provided by almost all the IoT devices is the possibility to generate notifications and, in some cases, to receive and show notifications sent by other services/devices. As a consequence, the number of notifications received every day by users is raising together with the growth of services (e.g., instant messaging apps, cloud services) able to send and/or receive notifications in almost every moment of the day. As declared by Weber et al [8], in fact, "the ongoing wave of smart devices makes it possible to reach the user through multiple devices at once" increasing the number of notifications received by each user.

Such a widespread use of notifications can be perceived both as an advantage or a disadvantage: users can appreciate or criticize the arrival of a notification depending on different factors.

Several researches investigated the effects of notifications on users: almost all of them reveal that the first sensation that people feel thinking at notifications is often frustration. As declared by [9], in fact, notifications often arrive at inconvenient moments causing user frustration and/or annoyance.

Bailey et al. [10], for example, demonstrate that interruptions, e.g., those caused by notifications, have a disruptive impact on completion time and error rate for primary tasks. The conducted experiment uses a sample of primary and peripheral tasks representative of those often performed by users and measures the effects of interruption on task completion time, error rate, annoyance, and anxiety. In addition, Kushlev et al. [11] demonstrate that higher levels of user inattention and hyperactivity is mainly revealed when alerts are on than when alerts are off. Their work, in fact, investigates inattention and hyperactivity possibly caused by smartphones' interruptions. In the performed experiments, participants were asked to maximize phone interruptions for a week (by keeping notification alerts on and their phones within their reach/sight) and then minimize phone interruptions for another week (by keeping alerts off and/or their phones away). As additional contribution, the study reveals one of the first (and easiest) solutions that would solve the problem: the manual deactivation of notifications. Nonetheless, nowadays, notifications are essential for users and, in fact, other existing studies demonstrate that this solution is not applicable. In this context, Iqbal et al. [12] investigate how user task-execution patterns can be affected by email notifications. Results obtained by their field study in the workplace mainly show, again, that user focus on primary tasks is largely unaffected if notifications are disabled. However, the most important result is that the value that users assign to the awareness provided by notifications is enough to let them declare that they are willing to incur some disruption to maintain that awareness. Similarly, Adamczyk et al. [13] present another work that analyzes effects of interrupting a user at different moments within task execution in terms of task performance, emotional state, and social attribution. Results confirm that a system should enable a user to maintain a high level of awareness while mitigating the disruptive effects of interruption.

1.3 Goal

The research goal of this dissertation regards the investigation of the intelligence component in Internet of Things (IoT) architectures and applications. The research activity aimed at the study, definition, and prototyping of intelligent distributed architectures, and their main software components that may extract additional value and intelligent behaviors for end users.

Specifically, the distribution and customization of notifications in the IoT domain has been treated as an example of possible future IoT scenarios.

With the aims of reducing the disruption caused by notifications to end-user and, at the same time, allowing developers to exploit the spread of notifications to enhance their services, tools and applications, in this research, two different approaches have been adopted to reduce the disruption caused by notifications.

The first one focuses on users' needs and, by acting at the distribution level (i.e., notifications are intercepted and then the system decides if, when, and how to show them) investigates techniques and methodologies able to directly enhance user experience with notifications (e.g., through Machine Learning techniques).

Instead, the second one focuses on developers' needs and acts at the design level, i.e., it allows developers to design notifications and their distribution strategies with the aim of reducing user disruption and fully exploiting the advantages brought by the possibility of distributing them among different devices.

Two innovative independent IoT architectures are proposed as solutions for the approaches: the SNS (Smart Notification System) system [14] that is able to manage notifications using machine learning algorithms, and the XDN (Cross-

Device Notification) framework [15] that assists developers in creating cross-device notifications by scripting.

The user-centered design methodology is adopted in the design of such architectures: it is a problem-solving process that involves users in all the design phases. In fact, it not only requires designers to analyze and envision the way users are likely to consume a product, but also to validate their assumptions with regard to the user behavior in real world tests.

Such methodology, together with the modular nature of both architectures allowed the simultaneous development of the two architectures and the different independent but compatible submodules that compose them. Within the whole work, different simultaneous tests and preliminary deployment sessions were organized with the aim of a) collecting data for future experiments, b) verify the feasibility of the system, c) receive users' feedback about the designed system, and, finally, d) foster the development of future solutions in the IoT notifications' field and related domains. Obtained results and feedbacks demonstrate the effectiveness and the usefulness of the proposed approaches and pose the basis for future works in such domain.

1.4 Thesis organization

The remainder of the thesis is organized as follows. Chapter 2 presents the SNS system, with all its related works and the details of the independent but compatible internal modules. Then, Chapter 3 presents the XDN Framework and the results obtained by testing the system with real users. Finally, Chapter 4 concludes the thesis and discusses future developments.

1.5 Original contribution

Part of the work described in Section *2* has been previously published in [14, 16, 17], however it was revised and extended in the present dissertation. Specifically, the architecture of the *Smart Notification System* (presented in Section 2.5) and the initial experiments with the prototype of the *Decision Maker* module (presented in Section 2.7) were already described in [14]. In addition, the *SmartCity Collector* contribution (described in section 2.8.4) and the *Location Estimation* extra work (presented in Section 2.9.1) were already described in [16, 17]. Instead, the *IoT Collector Server* (presented in Section 2.8.1), the *Mobile Collector* module (presented in Section 2.8.2), and the *SmartHome Collector* module (presented in Section 2.8.3) are original contributions of the present thesis.

Furthermore, part of the work described in Section *3* has been also published in [15], but it was revised and extended in the present dissertation: all the tests with real users and the corresponding results, reported in Section *3.6* are original contributions of this thesis.

Chapter 2

SNS (Smart Notification System)[1]

2.1 Introduction and Motivation

As already depicted in the Introduction Chapter, notifications are overwhelming people lives and the benefit of receiving them on almost every mobile and IoT device has slowly became a problem for almost every user.

Such a problem is supported by different studies. One of them is the study conducted by Church et al. [18] on the reasons and perceptions of WhatsApp, a popular mobile messaging application, in which some interviewees declared they were annoyed with the amount of notifications received by mobile messaging applications in general. Moreover, the large-scale assessment of mobile notifications made by Sahami et al. [19] demonstrates that, even though not all the notifications are equally valued by users, they are becoming invasive and sometimes they reduce users' overall performance distracting them from other tasks. Furthermore, they observed that the users' reaction to notifications changes according to what she was doing before it (e.g., during a voice chatting the notification is temporarily ignored), the context in which the user was involved (e.g., if the user was at work, she used notifications to keep in contact with everything she did) and obviously on her habits.

[1]Part of the work described in this chapter has been previously published in [14, 16, 17]. Specifically, the architecture of the proposed system was revised to introduce new, more detailed, components and allow the simultaneous development of preliminary prototypes. In addition, in this thesis, the preliminary implementation of the most important modules of the SNS system are presented and the results obtained through some preliminary deployment sessions are discussed.

This chapter presents the Smart Notification System (SNS), a modular architecture that was designed to deal with notifications at the distribution level, i.e., notifications are intercepted and then the system decides if, when, and how to show them. It uses machine learning algorithms to manage incoming notifications according to context awareness and users habits.

The SNS system is based on supervised machine learning algorithms and is composed of different modules that cooperate to monitor both environment and users to make decisions on a) who should receive an incoming IoT notification; b) what is the best moment to show the notification to the chosen user(s); c) on which device(s) the chosen user(s) should receive the notification; d) which is the best way to notify the incoming notification (e.g., vibration, light, sound). With the aim of obtaining incremental preliminary results and feedbacks, the system is designed as a group of collaborative but independent modules aiming at fostering the simultaneous independent design and implementation of each single module and, in the meantime, generating initial results and feedbacks.

For this reason, the remainder of the chapter will be organized as follows. The first four Sections describe the whole SNS Architecture: the *Related works* Sections present all the works that inspired its design; the *Architecture* Section introduces all the details of the designed Architecture. Then, the following Sections, i.e., *Prototypes: design, implementation and preliminary deployment* with its *Decision Maker*, *Collectors*, and *Context Analysis* subsections, present all the independent but compatible modules that were designed and prototyped within the present work. At the end of each subsections the results and feedback obtained by the independent experiments performed for each submodule are discussed and, finally, the *Discussion and Conclusion* concludes the chapter.

2.2 Background

2.2.1 Machine Learning Approach

As already depicted, SNS is based on machine learning algorithms.

Machine learning is based on algorithms able to learn from and make predictions on data: they use sample inputs, called training sets, to build a model that is then exploited to make predictions [20]. Two different main machine learning algorithm categories can be identified: supervised learning and unsupervised learning algorithms. The difference between them is in the used dataset: supervised learning algorithms use data that have already been classified and to which labels have already been assigned. This kind of data are called labeled data and helps the algorithm to make the same prediction in similar situations. Unsupervised learning algorithms, instead, try to find hidden structure in unlabeled data by creating groups of data with similar properties.

Among the two different existing machine learning algorithm categories, SNS is based on supervised machine learning algorithms.

In addition, as effectively described by Gareth et al. [21], two main phases characterize the creation of a machine learning model: the training and the validation phases. As the names suggest, in the training phase the model is created, while in the validation phase, it is tested and validated through the analysis of some measures (that will be discussed at the end of this paragraph) representative of the effectiveness of the obtained model.

The most important element of machine learning approaches is the data: as already discussed, in fact, the algorithm learns from existing data to predict future behaviors. To be sure that the model is properly trained and that predictions are enough accurate as needed, the data used to train the model should not be the same used for the tests. However, considering that it is not always possible to have two separate datasets for the two phases, typically, the same dataset is used for both phases but it is randomly divided into two parts, a training set and a validation set. Thus, the model is fit on the training set, and the obtained fitted model is, then, used to predict the responses for the observations in the validation set.

Different techniques exists to divide the dataset into two parts. One of the most used [21] is the **k-fold cross-validation** technique. This approach involves the random division of the set of observations into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the model is fit on the remaining k-1 folds. A measure representative of the effectiveness of the obtained model is then computed on the observations contained in the validation fold. This procedure is repeated k times; each time, a different group of observations is treated as a validation set and finally a mean value of the k validations' measure is reported [21].

The most used measures to represent the effectiveness of the model are the "**accuracy**", the "**precision**" and the "**recall**" values [22]. The following description presents such values in a "single class" model, a model in which the Machine Learning algorithm makes predictions by choosing among only two possible values (also called "labels"): positive and negative (e.g., in medical domain, the algorithm could have to decide if the patient is affected or not by a specific illness). However, the explanation can be extended to cases with more labels: usually, a weighted value of repeated single class classification is computed.

As accurately described by Bužić et al [23], the calculation of the accuracy, precision and recall is based on four main parameters that are extracted from each validation phase:

- true positive (TP),
- true negative (TN),
- false positive (FP),
- false negative (FN).

As the names suggests, they represent the effectiveness of the model in predicting the corresponding labels: all the correct (True) estimations of a Positive label are counted in the TP value, while all the correct (True) estimations of a Negative label are counted in the TN value, and so on. From such values it is possible to estimate the accuracy, precision and recall:

- Accuracy, that represents the percentage of correct estimations, can be calculated through the following formula:

$$Accuracy = \frac{(TP+TN)}{(TP+TN+FP+FN)} \tag{2.1}$$

- Precision, that is a measure of the ability to detect positive cases (i.e., the result relevancy), can be calculated through the following formula:

$$Precision = \frac{TP}{(TP+FP)} \tag{2.2}$$

	Classified positive	**Classified negative**
Positive class	10 (TP)	15 (FN)
Negative class	25 (FP)	100 (TN)

Table 2.1 Results obtained in the example used to present the accuracy paradox

	Classified positive	**Classified negative**
Positive class	0 (TP)	25 (FN)
Negative class	0 (FP)	125 (TN)

Table 2.2 Results obtained by the "dumb" classifier in the example used to present the accuracy paradox

- Recall, that is a measure of the ability to avoid incorrect detection of negative cases (i.e., the result sensitivity), can be calculated through the following formula
$$Recall = \frac{TP}{(TP+FN)} \tag{2.3}$$

Analyzing the reported formulas and description it could be deducted that the accuracy measure is enough to estimate the effectiveness of a machine learning model: the higher the accuracy value, the more the model is able to correctly make predictions. However, it is common to fall into the "**accuracy paradox**". To explain such concept an example is introduced: a classifier is trained to do spam filtering and, after a training and validation phases the results reported in table 2.1 are obtained.

In this case, accuracy is equal to 73.3% and could lead to the conclusion that the classifier is working fine. However, by comparing it with a dumb classifier that always says "no spam" and returns the results reported in table 2.2, it could be deducted that the second one is better than the first one.

This phenomena is called "accuracy paradox": when TP < FP, then accuracy will always increase when a "always output negative" classification rule is choosen. For this reason the "precision" and"recall" values are introduced. In fact, as already discussed, they are respectively representative of the result relevancy and the result sensitivity and are complementary: by increasing one measure it is likely to decrease another (or, at best, another will remain the same) [23]

In the ideal case, if the classifier does not make mistakes, then precision = recall = 1. But in real world it is almost impossible to achieve. Even though, in fact, it would be easy to construct a completely useless classifier which would classify all cases as positive, making the recall measure perfect 1, this will, in turn, make the precision be very small.

In addition to the "accuracy paradox", another phenomenon could be lead in training the model: the "**overfitting**". It is defined as the "production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably"[2]. Therefore, even though the results of the validation phase reports good values for accuracy, recall and precision, the model is actually only too close to the data used in the training phase and it will not work effectively in real cases. The presented k-fold cross validation technique, is one of the techniques commonly used in literature to avoid this situation.

All the experiments performed within the current dissertation use the k-fold cross-validation technique to validate the designed and implemented models. In addition, the accuracy, precision and recall measures are used as representative of the effectiveness of the model.

2.2.2 REST API

In the following paragraphs the architecture of the SNS system will be presented and all the details of the modules that compose it will be discussed. One of the characteristics that is shared among almost all the modules that compose the SNS system is the design style used to design the interaction among all the modules: the REpresentational State Transfer (REST) architectural style [24].

As described by Costa et al [24], it consists of a set of constraints applied to elements within the architecture itself:

- *Client–server*

  The uniform interface separates clients from servers. This separation means that, for example, clients are not concerned with data storage, which remains

[2]https://en.oxforddictionaries.com/definition/overfitting, last visited on July 10, 2018

internal to each server and servers are not concerned with the user interface or user state, which remains internal to each client.

- *Stateless*

  It means that the necessary state to handle the request is contained within the request itself, whether as part of the URI, query-string parameters, body, or headers.

- *Cacheable*

  As on the World Wide Web, clients can cache responses. Responses must therefore, implicitly or explicitly, define themselves as cacheable, or not, to prevent clients reusing stale or inappropriate data in response to further requests.

- *Layered system*

  A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. Intermediary servers may improve system scalability by enabling load-balancing and by providing shared caches. Layers may also enforce security policies.

- *Uniform Interface*

  The uniform interface constraint defines the interface between clients and servers. It simplifies and decouples the architecture, which enables each part to evolve independently.

- *Code on demand* (optional)

  Servers are able to temporarily extend or customize the functionality of a client by transferring logic that it can execute to it. Examples of this may include compiled components such as Java applets and client-side scripts such as JavaScript.

2.3 Related works

This section discusses the related works referred to the whole SNS system leaving the discussion about the single independent modules to each dedicated section.

Feature	**Description**
F1	Exploit user information and/or habits in decision process
F2	Exploit user preferences in decision process
F3	Exploit context information in decision process
F4	Distribute notifications to multiple devices
F5	Distribute notifications to multiple users
F6	Threat heterogeneous notifications
F7	Exploit ML algorithms in decision process
F8	Support IoT notifications
F9	Exploit notification receipt time in decision process
F10	Exploit notification effects (e.g., notification, sound) in decision process

Table 2.3 Summary of features provided by related works

Related work	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**
Bohmer et al. [1]	N	N	N	N	Y	N	N	N	N	N
Ardissono et al. [25]	Y	Y	Y	N	N	N	N	N	N	N
Roecker et al. [26]	N	Y	Y	Y	N	N	N	N	N	Y
Leonidis et al. [27]	N	Y	Y	N	N	N	N	N	N	N
Banerjee et al. [28]	Y	Y	Y	N	N	Y	N	Y	Y	N
Etter et al. [29]	N	Y	Y	N	N	Y	N	N	Y	N
Poppinga et al. [2]	Y	Y	N	N	N	N	Y	N	Y	N
Arlein et al. [30]	N	Y	Y	Y	N	Y	Y	Partially	Y	N
Mehrotra et al. [31]	Y	Y	N	N	N	Y	Y	Partially	Y	N

Table 2.4 Summary of related works' features

Intrusive, annoying and repetitive notifications have been treated in several existing works and projects and several specific solutions acting at the distribution level can be found in literature. However, only a few of them uses machine learning (ML) to approach the problem.

A comparison of all the contributions presented in this section is reported in Table 2.4 that summarizes all the presented related works with respect to the exposed features and services reported in Table 2.3 (these features are extracted from the presented related works).

One of the first works to propose a solution to address the disrupting notification problem is the one proposed by Bohmer et al. [1]. In their work an initial introduction carefully explains how the abrupt full-screen notifications used since a few years ago to alert user(s) about incoming calls forcibly interrupted whatever activity the user

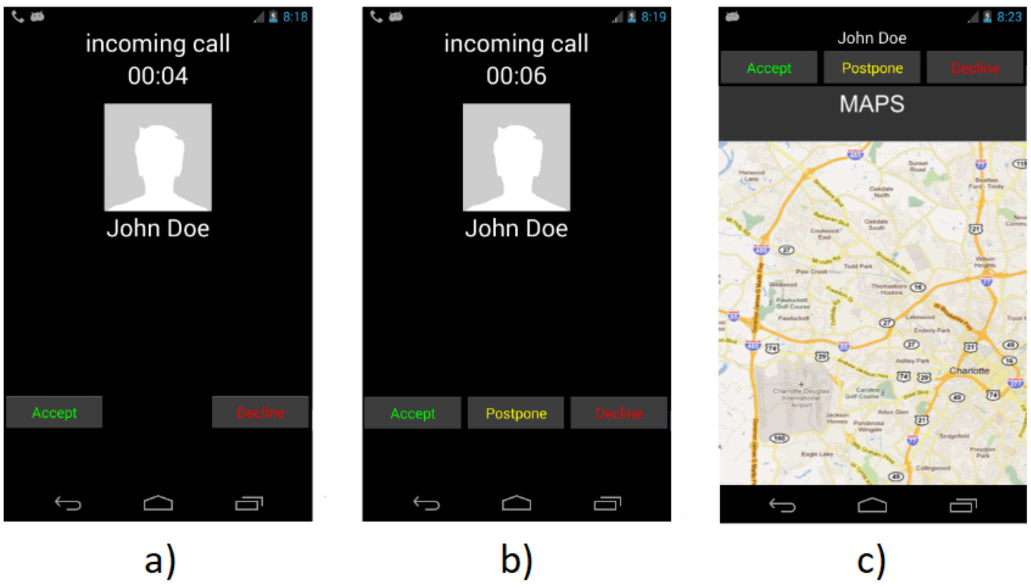


Fig. 2.1 Different ways of showing notifications. Image from [1]

was already engaged in. They compare three different ways of notifying the user about an incoming call.

Figure 2.1 shows all the three modalities: a and b represent abrupt full-screen notifications with different proposed actions, while c shows the proposed multiplex UI, a smaller partial-screen notification. Even though their solution does not use any machine learning approach, the proposed multiplex UI is very similar to the one that was, then, introduced in all the smartphones' operating systems and that is currently used by every people all around the world. Moreover, Ardissono et al. [25] propose a notification model to reduce the disruptive effect of notifications on user attention. Their model predicts user activities using information received by different heterogeneous Web applications and uses these predictions to decide whether to show, postpone or delete received notifications based on user preferences. Even though this work does not use machine learning techniques to manage incoming notifications, it follows an approach that is similar to the one presented in this thesis: user and environment context information are used to distribute notifications.

In addition, some other related projects could inspire or may be integrated in the next versions of our system, however, only few of them uses machine learning techniques due to the reported description. Roecker et al. [26], for instance, explore alternative approaches and strategies for email filtering and notification with the rationale of developing an unobtrusive notification interface that can be adapted to the user's context. Their goal was to continuously inform the user about relevant emails while trying to minimize the distractions through the use of peripheral interfaces that are more manageable and effective than traditional desktop pop-up windows. They act into two different ways: by distributing notification effects among different devices and interfaces (e.g., PDAs and ambient displays) and by categorizing incoming emails according to their importance. They demonstrate that notification effects can be distributed among different devices due to their importance to enhance user experience. Leonidis et al. [27] present a semantics-based, context-aware notification system that provides personalized university alerts (e.g., reminders about timetable) to graduate students based on their preferences. The "AlertMe" system takes advantage of semantic technologies in order to make context-dependent and policy based decisions. The available context information is modeled through an ontology and the final decision is drawn by a reasoner, a rule engine that is based on policies expressed as a set of SWRL rules. Banerjee et al. [28] present a UNS (Universal Notification System) that provides a framework to developers that want to develop an app that should be able to distribute notifications based on context information. The presented UNS provides to registered applications information about connected solutions, context awareness, and user preferences and let them generate and distribute smart city notifications. Even though this work would be used as starting point for our SNS system, unfortunately, no source code was found and it was only used as inspiration for the design of our architecture. Etter et al. [29] propose a similar service: the Awareness and Notification Service (ANS), that makes applications aware of context changes by notifying them with appropriate rendering of intensity. ANS takes a rule based approach based on the event-condition-action pattern: users specify when and what should be notified to them by using a convenient ANS rule language. This flexible mechanism, that demands decisions to users, allows to rapidly develop applications that provide context-aware notifications without the need to write programming code to activate rules, nor to implement personalized notifications. Furthermore, some commercial products have already been developed: one of them is Google Inbox[3] that scans user email accounts to identify important and similar information aiming at presenting what it considers the most important parts of the email first and group similar emails.

[3]http://www.google.com/inbox/, last visited on March 22, 2017

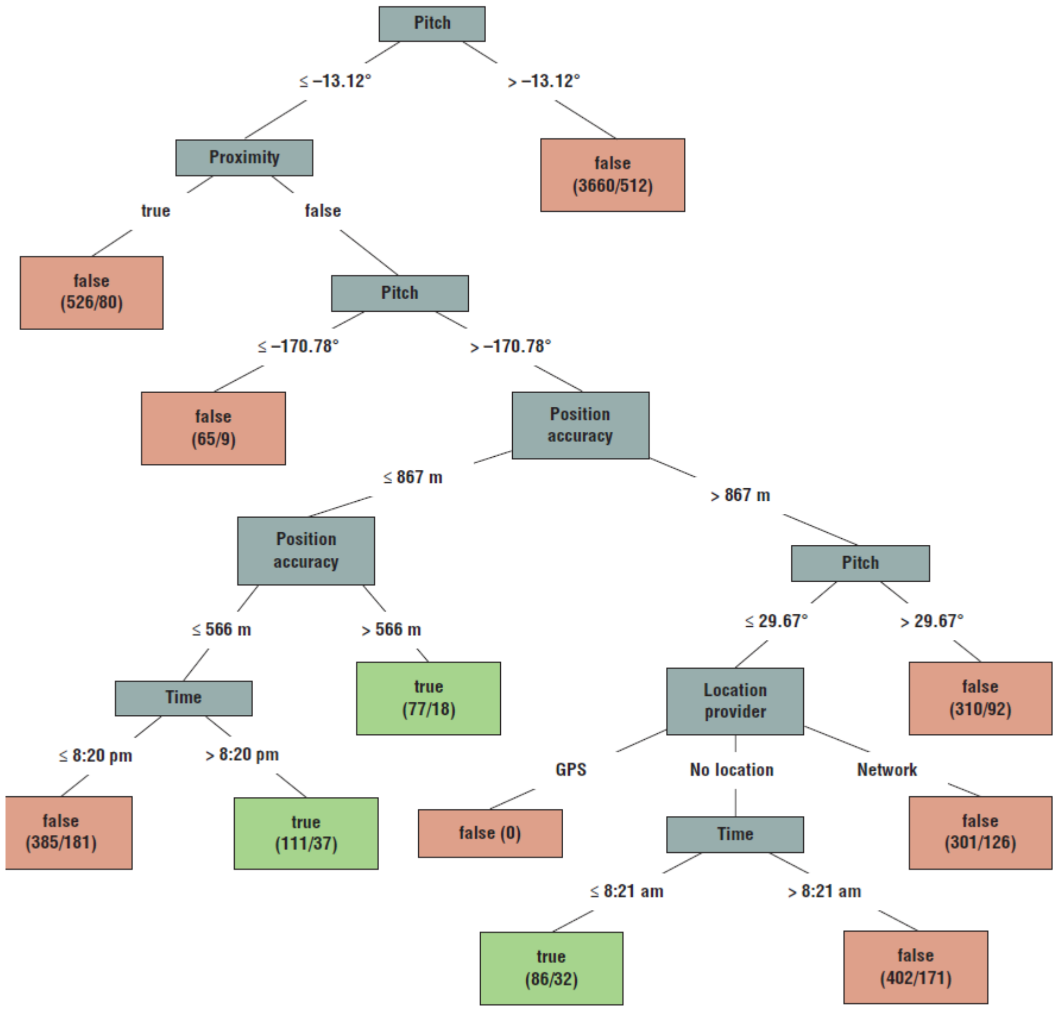


Fig. 2.2 The resulting C4.5 decision tree obtained by Poppinga et al. [2]. It consists of 20 elements and 11 leaves. It classified whether a notification should be issued (true) or not (false) with an accuracy of 77.85 percent. Image from [2]

Moreover, Poppinga et al. [2] conducted a large-scale, longitudinal study, collecting 6,581 notifications from 79 different users over a 76-day time period, aiming at developing a model for predicting suitable moments for issuing notifications.

By using a machine learning approach based on decision tree algorithm they derived the model reported in Figure 2.2. As shown in the reported figure, it exploits user context information and notification receipt time to predict opportune moments to issue notifications with approximately 77 percent accuracy. Even though they declare that their findings could lead to intelligent strategies to issue unobtrusive notifications on today's smartphones at no extra cost, unfortunately, the collected notifications dataset contains only information about users reaction to a notification (immediately answer or not) without any information about the type of received notification and/or the user that received it.

Finally, two works that are very similar to our SNS system are the ones proposed by Arlein et al. [30] and Mehrotra et al. [31]. Arlein et al. [30] propose a notification framework architecture that is similar to our SNS system: in fact, it allows notifications to be efficiently and effectively distributed and displayed in diverse new environments. However, it is different from our work due to a different approach to the problem. The first assumption that they do is related to the replication of the system: they assume that the system should be replicated for each user considering only devices in the decision process instead of the combination of both users and devices. Moreover, even though both the architectures accept notifications over multiple protocols, the architecture proposed by Arlein uses a deterministic approach to make decisions: a sequence of conditions are evaluated and, consequently, the system does not learn behaviors from previous detections (like we do by using machine learning algorithm). In addition, they do not consider context aware and user habits: contexts are only handled in the adaptation process in which the system chooses to adapt the notification as a summary or a complete notification.

Moreover, Mehrotra et al. [31], based on results of their previous works [32, 9, 33] design, implement and evaluate PrefMiner, a novel interruptibility management solution that learns users' notification preferences based on automatic extraction of rules. Rules are mined through a machine learning approach based on Association Rules and are able to evaluate user interaction with mobile phones to determine a behavior that users usually perform when a notification arrives. Such rules are then shown to users who might decide to accept or discard them at run-time and, only if the rule is accepted, it is then applied whenever a new notification arrives. This work is different from our work for two main reasons. The first one regards the involved notifications: their work consider only mobile notifications, while the SNS system is designed to work with both mobile and IoT notifications. In addition, their work does not consider context information about the environment and/or the user except the ones that can be collected through the user smartphone.

2.4 Scenario

A typical intended usage of the SNS system is described in the following scenario, that will be used as a running example: *a user (let's call her Maria) is in her house and it is 10 o'clock in the morning. She owns a smartphone, a smart washing machine, a smart fridge and a smart TV. While Maria is having a shower, the washing machine ends its washing cycle and sends a notification to Maria. Considering that she is involved in an activity that does not let her use any available device, the system decides to postpone the notification of the message by 15 minutes, the predicted moment in which she will exit the bathroom and take her smartphone (the prediction of the delay time is based on her habits: the activities performed in the bathroom usually last 15 minutes). Furthermore, due to the current time and the available notification modes, a sound and a message on the phone are chosen as methods to deliver the notification. Therefore, the smartphone is chosen as the receiver of the notification and the end of the shower is chosen as the right moment to notify it with a sound and a message on the phone.*

2.5 Architecture

The SNS system is a modular architecture that deals with notifications at the distribution level and uses machine learning algorithms to adequately manage incoming notifications according to context awareness and users habits.

Figure 2.3 shows the architecture of the proposed system: it accepts notifications from different external sources through its *Notification Collector* module that is also responsible for the conversion of the incoming notifications into a common format containing the information discussed in the following paragraphs. Converted notifications are then sent to the *Decision maker* that is aware of environment status (e.g., weather information, current date and time), user context (e.g., location, status, current activity), and user habits and uses them to choose the best devices and the best ways (e.g., vibration, sound, or a light signal) to present the received notifications. The decision is then attached as an additional information to the notifications and the result (called "Converted Notifications + LABELS" in Figure 2.3) is finally sent to the *Dispatcher* module that adapts the notifications to the chosen target devices and actually sends them.

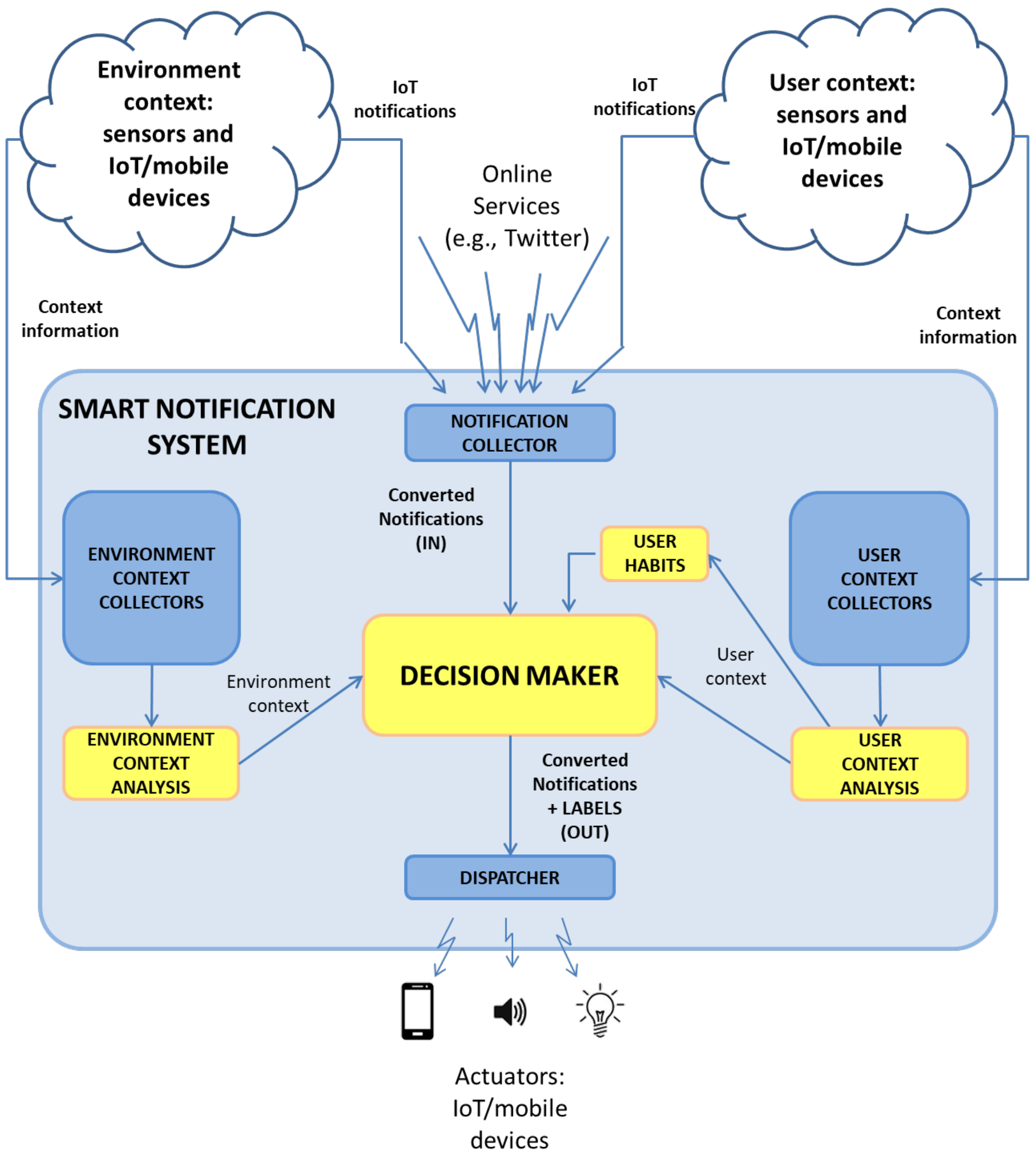

Environment context: sensors and IoT/mobile devices
IoT notifications
IoT notifications
User context: sensors and IoT/mobile devices
Online Services (e.g., Twitter)
Context information
Context information
SMART NOTIFICATION SYSTEM
NOTIFICATION COLLECTOR
Converted Notifications (IN)
USER HABITS
ENVIRONMENT CONTEXT COLLECTORS
USER CONTEXT COLLECTORS
DECISION MAKER
Environment context
User context
ENVIRONMENT CONTEXT ANALYSIS
USER CONTEXT ANALYSIS
Converted Notifications + LABELS (OUT)
DISPATCHER
Actuators: IoT/mobile devices

Fig. 2.3 Architecture design

The following description depicts the details of the most important architectural elements.

The ***Environment Context Collectors*** and ***User Context Collectors*** modules are responsible for retrieving information shown in the following Table 2.5 about environment and user context, but also about the status of the involved devices. These data are acquired from external blocks such as sensors, cloud services (e.g., calendar), and IoT devices. Then, the ***Environment Context Analysis***, the ***User Context Analysis*** and the ***User Habits*** modules are responsible for elaborating the information retrieved by the two Collectors to support the *Decision maker* in making decisions.

The ***Decision maker*** is the core of the system being responsible for all the decisions: every time it receives a notification, it takes the information from the two context analysis components and from the habits module, and decides:

- who should receive the incoming notification (one or more users);
- what is the best moment to show the notification to the chosen user(s) (i.e., a notification can be postponed or ignored if it does not contain important information);
- on which device(s) the chosen user(s) will receive the notification;
- which is the best way to notify the incoming notification.

The most important characteristic of this module is the way used to make decisions: a supervised learning classification algorithm is responsible for all the described choices.

As already explained, supervised machine learning algorithms are trained through labeled datasets, so we defined the essential characteristics and information that the system should manage to make the right decision:

- Table 2.5 shows the list of context information needed by the algorithm to make decisions, such as information related to available IoT devices (such as device owner, device current location and device current status), user context and environment context (e.g., user current position or current timestamp);
- Table 2.6, instead, lists the information related to both incoming (i.e., the “Converted Notifications” of Figure 2.3) and outgoing notifications (i.e., the “Converted Notifications + LABELS” of Figure 2.3).

Every time a notification arrives, the *Decision maker* collects available context data and, using the trained algorithm, considers incoming information to make a decision.

User context	Current activity
	Current location
	User preferences
	Personal user information
Environment context	Current timestamp
	Weather condition
	Number of people present in involved buildings/rooms
	Status of involved buildings/rooms (e.g., temperature)
	Other static information (e.g., information about the involved building)
Available IoT devices informa-tion	Owner
	Current location
	Current status (e.g., on, off, standby)
	Available modes

Table 2.5 Context information

Information about incoming notification	Sender
	Receiver
	Type of notification
	Timestamp of receipt
	Contained message/information
Assigned labels to outgoing notifications	Target devices
	Chosen mode
	Delivery timestamp

Table 2.6 “Converted Notifications” information

As a running example, the following paragraph will report the information that the dataset would contain in the situation described in the scenario presented at the beginning of the section. Only key information are reported to let readers better understand the behavior of the *Decision maker*:

- User context information
  - user: Maria
  - current activity: having a shower
  - current location: house - bathroom
  - user habit: activities performed in the bathroom usually last 15 minutes
- Environment context information
  - current timestamp: 2015-06-11T10:00:30
  - involved building: user house
  - involved rooms: bedroom, kitchen, bathroom
  - people in the bedroom: 0
  - people in the kitchen: 0
  - people in the bathroom: 1
- Available IoT devices information
  - Smartphone
    - current location: bedroom (home)
    - current status: on
  - Smart TV
    - current location: kitchen (home)
    - current status: off
- Information about incoming notification
  - Sender: washing machine
  - Receiver: every family member
  - Type of notification: service message

  - Timestamp of receipt: 2015-06-11T10:00:15
  - Message: washing cycle ended.

The *Decision maker* uses this dataset to make the following decision (the following information will be contained in the outgoing notification):

- target devices: smartphone
- chosen mode: sound + message
- delivery timestamp: 2015-06-11T10:15:30.

Moreover, as depicted in Figure 2.3, the *Decision maker* is supported by other modules. The first one is the ***Notification Collector*** that is responsible for retrieving any kind of notification generated or arrived from every external service and IoT device. It is also responsible for the conversion of the incoming notifications into a common JSON format containing the information reported in Table 2.5 and Table 2.6.

Secondly, the ***Dispatcher*** module is responsible for the distribution of the notifications: it adapts the notifications generated by the *Decision maker* ("converted notifications + labels) according to the output methods supported by each device and to the method selected by the *Decision maker*. Then, it sends them to the right device.

2.6 Prototypes: design, implementation and preliminary deployment

The modular nature of the SNS system architecture inspired the design and the prototyping of three main groups of contributions provided as first outcome of the present dissertation:

- The *Decision Maker* contribution, that presents all the experiments performed with the objective of validating the role of the ***Decision maker*** module and to test its feasibility;

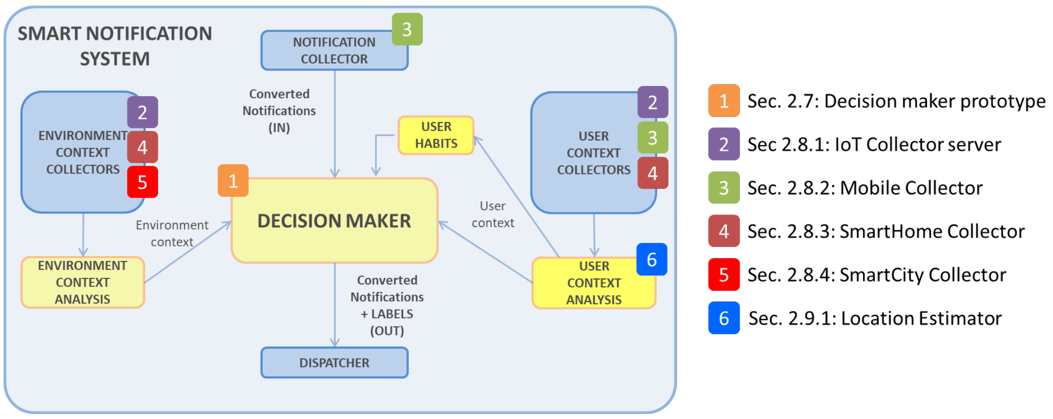

SMART NOTIFICATION SYSTEM
NOTIFICATION COLLECTOR
3
Converted Notifications (IN)
ENVIRONMENT CONTEXT COLLECTORS
2
4
5
USER HABITS
USER CONTEXT COLLECTORS
2
3
4
1
DECISION MAKER
Environment context
User context
ENVIRONMENT CONTEXT ANALYSIS
USER CONTEXT ANALYSIS
6
Converted Notifications + LABELS (OUT)
DISPATCHER
1 Sec. 2.7: Decision maker prototype
2 Sec 2.8.1: IoT Collector server
3 Sec. 2.8.2: Mobile Collector
4 Sec. 2.8.3: SmartHome Collector
5 Sec. 2.8.4: SmartCity Collector
6 Sec. 2.9.1: Location Estimator

Fig. 2.4 Simplified version of the SNS Architecture with additional information about the developed contribution

- The *Collectors* group of contributions, that gathers all the work performed to validate and test a) the ***Environment Context Collectors*** and the ***User Context Collectors*** modules responsible for collecting real context data, and b) the ***Notification Collector*** module responsible for retrieving the notifications generated by or arrived from external service and IoT device.

- The *Context Analysis* group of contributions, that gathers all the work performed to validate and test the ***Context Analysis*** module responsible for providing additional information, i.e., information extracted from the raw data received from the collectors.

Each contribution may have multiple roles with respect to the modules of the architecture: Figure 2.4 reports a simplified version of the already presented SNS architecture on which some graphical signs help to identify the role of each contribution. Specifically, a colored and numbered circle is assigned to each contribution and, as a consequence, each circle is replicated near the architectural modules at which it actually contributes. As can be depicted in the Figure, each contribution can be involved in more than one architectural module: for instance, the IoT Collector server acts in both the Environment and User Context Collectors modules because it is used to collect data from both contexts. In addition, each architectural module can be implemented in different contributions: for instance the User Context information (gathered by the User Context Collectors module) can be gathered through different contributions (e.g., the IoT Collector server and the Mobile Collector).

The details of such prototypal contributions are discussed in the following sections together with the data and results obtained from the preliminary deployment sessions already performed.

2.7 Decision Maker

The first contribution provided as part of the SNS system regards the development of a prototype of the *Decision maker* module with the aim of obtaining some preliminary results and feedbacks about the feasibility of the proposed approach.

As already depicted in the previous sections, the *Decision maker* module is responsible for all the decisions regarding a) who should receive the incoming notification (one or more users), b) what is the best moment to show the notification to the chosen user(s), c) on which device(s) the chosen user(s) should receive the notification, and d) which is the best way to notify the incoming notification.

The most important characteristic of the *Decision maker* module is the way used to make decisions: a supervised learning classification algorithm is responsible for all the described choices.

2.7.1 Preliminary experiments and results

Figure 2.5 shows all the steps followed in developing and testing a prototype of the *Decision maker* module. In this preliminary prototype implementation the work of the *Decision maker* was simplified with respect to the one presented in the previous Section by:

- choosing only one device as receiver of the notification instead of more than one;
- assuming that only one user is involved in the system;
- assuming that each device has only one available mode to show the notification;
- ignoring the decision related to the best time to deliver the notification.

Therefore, due to this simplification, the first step in developing such prototype regards the identification of a reduced set of information needed by the *Decision*

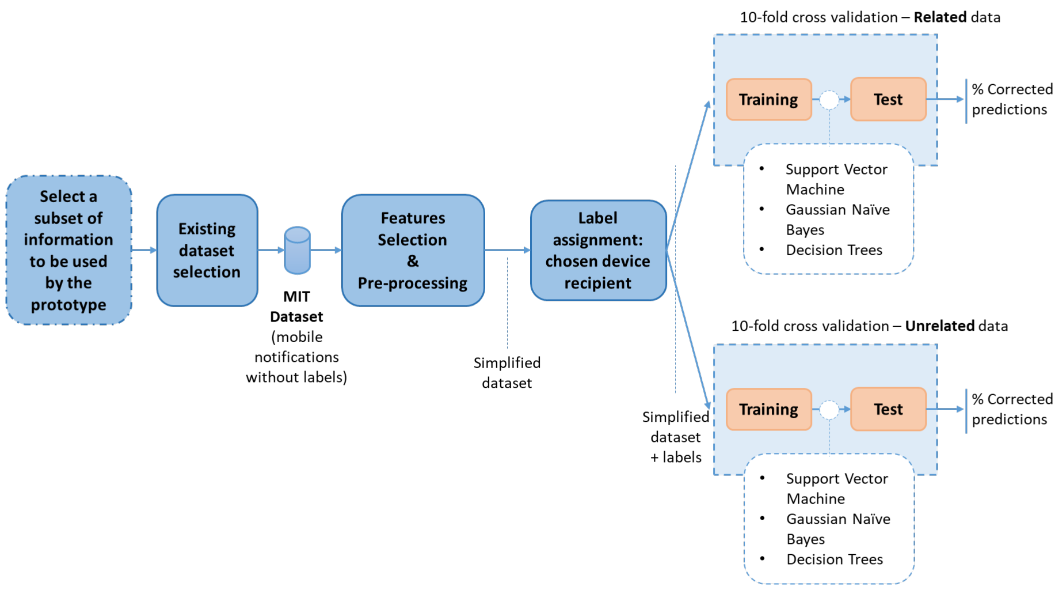

10-fold cross validation – **Related** data
Select a subset of information to be used by the prototype
Existing dataset selection
MIT Dataset (mobile notifications without labels)
Features Selection & Pre-processing
Simplified dataset
Label assignment: chosen device recipient
Simplified dataset + labels
Training
Test
% Corrected predictions
Support Vector Machine
Gaussian Naïve Bayes
Decision Trees
10-fold cross validation – **Unrelated** data

Fig. 2.5 Steps followed in test the first prototype of the Dispatcher module

maker module to make predictions in such simplified version. In fact, starting from the discussion conducted in Section 2.5, a simplified list of needed information was extracted from Table 2.5 and Table 2.6. The result is reported in Table 2.7 and Table 2.8 and represents the data that the prototype of the Decision Maker module will use in this preliminary evaluation.

Then, the second step in developing such prototype regards the identification of an existing dataset to be used by the Machine Learning algorithm(s) that will be then tested. The main requirement of the searched dataset regards the contained data: the

User context	Current activity
	Current location
Environment context	Current timestamp
Available IoT devices information	Owner
	Current status (e.g., on, off, standby)

Table 2.7 Simplified version of Context information: for the prototype

Information about incoming notification	Sender
	Receiver
	Type of notification
	Timestamp of receipt
Assigned labels to outgoing notifications	Target devices

Table 2.8 Simplified version of Notifications information: for the prototype

chosen dataset should contain as much information as possible with respect to the just reported tables (Table 2.7 and Table 2.8).

Therefore, the dataset collected within the MIT Media Laboratory Reality Mining project [34] was chosen as the basis dataset for our experiments: even though only mobile notifications are collected in such a dataset, it contains most of the information to be used (in some cases after some adaptation/elaboration) by the Decision Maker module to make decisions. Specifically, MIT researchers used mobile phones to collect data about call logs, Bluetooth devices in proximity, cell tower IDs, application usage, and phone status. 94 people over 9 months were monitored and the collected data were, then, used to infer different user information including location.

As a third step, with the aim of adapting the MIT dataset to our needs, a simplified version of the MIT dataset was created by using a Python script that removed all the useless information and added the absent information as synthetic information.

The following list summarizes the data contained in the resulting dataset with the details on the origin of each data:

- Sender; it was randomly generated among 4 possible senders.
- Receiver; it was extracted from the MIT dataset: it is the owner of the monitored device.
- Type of notification; it was randomly generated among 5 possible types.
- Timestamp of receipt; it was generated based on the available information contained in the MIT dataset.
- User current activity; it was randomly generated among 7 possible activities.

Information	# of elements	Used values
senders	4	/
users (possible receivers)	94	/
personal devices per user	2	/
overlapping activities	7	working, sleeping, cooking, break from work, having a shower, programming, surfing the Internet
notification types	5	social network, security alarm, personal health, temperature notification, wearable fitness tracker notification
available user locations	3	work, house, elsewhere

Table 2.9 Dimension of used dataset

- User current location; it was generated by taking into account the other already generated or extracted information (e.g., if the user was having a shower it is at home).
- Available devices for the user; there are only 2 devices per user: the first one's status was extracted from the MIT dataset, while the second one was randomly generated.

In addition, for the purpose of training, a label was programmatically assigned to each sample representing the chosen device: considering that there were 2 devices per user, if the first one was available it was always selected as target device, otherwise, the second one was chosen. If also the second device was not available, no device was chosen. The case in which both devices were selected was not considered as a possibility. The label is a text label assuming multiple values: 3 for each user.

Table 2.9 summarizes the dimensional details of the used dataset containing 165,289 samples.

As already discussed, machine learning algorithms differently handle labeled and unlabeled data, however another distinction between input data is needed. Input data can be divided into two main groups:

- *related data*, that is represented by an ordinal value [35], possessing the properties of ordering and proximity. An example of related data is the receipt time of a notification: the proximity of one hour to another one (1 o'clock is nearer to 2 o'clock than to 4 o'clock) represents a useful information for every algorithm dealing with such information. In fact, for instance, if an algorithm has to predict an action based on the current date and time it could be important to know if the current time is nearer or not to a specific decision already taken in the past;

- *unrelated data*, that is data in nominal scale, i.e., data with equality property and without the ordinal and proximity ones [35]. An example of unrelated data could be the user name: if we assume that an identification number is assigned to each user, the proximity of one user to another one could not be as useful as it is with related data. Thus, for instance, a machine learning algorithm that makes a prediction based on user name, would not actually take advantage from knowing the proximity of the present user identification number to another one associated to an information acquired in the past.

In our situation all the information contained in the simplified dataset can be considered as unrelated data except the receipt timestamp.

Each machine learning algorithm deals differently with related and unrelated data, so we decided to compare three different machine learning algorithms for this preliminary implementation: Support Vector Machine (SVM) [36], Gaussian Naive Bayes (GNB) [37] and Decision Trees (DT) [38]. The first two algorithms try to find a correlation between inserted data in order to classify them and so they work better with related data. Instead, the DT algorithm tries to create a flowchart-like structure in which each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a label. The paths from root to leaf represents classification rules and, as can be understood, it does not look for relations between data.

Two different experiments were conducted with each algorithm, one with only unrelated data and another one with the addition of related data (the receipt timestamp): we expected that SVM and GNB would work worse with unrelated data than with related one and that DT would work better with unrelated than with related.

ML Algo-rithm	**Percentage of correct predictions with unrelated data**			**Percentage of correct predictions with related data**		
	Accuracy %	**Precision %**	**Recall %**	**Accuracy %**	**Precision %**	**Recall %**
Support Vector Machine	81.60	**99.89**	82.40	**96.10**	84.32	**96.90**
Gaussian Naive Bayes	51.30	99.80	51.30	83.40	**95.25**	83.40
Decision Trees	**99.90**	97.06	**99.90**	93.90	92.76	93.90

Table 2.10 Percentage of correct predictions obtained with used algorithms

The three algorithms were implemented using the Python programming language through the Anaconda distribution[4] and the Scikit-learn tool [39]. We used the 10-fold cross-validation technique to build the model. Specifically, 80% of the data was used as training dataset and the other 20% for tests over 165,289 samples.

A notebook with the following characteristics was used for running all the experiments:

- CPU: Intel Core i7-4800MQ
- RAM: 16GB PC3-12800 (800MHz)
- OS: Windows 7 Student Edition
- Python version 2.7.9
- Anaconda version 3.7.3

Table 2.10 shows the prediction results obtained by testing the three algorithms with both related and unrelated data in terms of accuracy, precision and recall. As

[4]https://store.continuum.io/cshop/anaconda/, last visited on March 22, 2017

ML Algorithm	**CPU time with unrelated data**	**CPU time with related data**
Support Vector Machine	5296.1 s	5801.1 s
Gaussian Naive Bayes	1.2 s	12.9 s
Decision Trees	15.5 s	13.9 s

Table 2.11 CPU time for a training phase with 33058 samples

ML Algorithm	**CPU time with unrelated data**	**CPU time with related data**
Support Vector Machine	40.19 ms	40.22 ms
Gaussian Naive Bayes	0.29 ms	0.31 ms
Decision Trees	0.001 ms	0.001 ms

Table 2.12 Average CPU time for each notification classification

expected, SVM and GNB seem to work worse with unrelated data than with related one, while the DT algorithm works exactly in the opposite way, reporting better values with unrelated data. However, it can be observed that the DT algorithm is the only one that obtains a high value for all the reported metrics: while in both experiments (with unrelated and related data) an high value (higher than 90%) of the accuracy and recall falls in a lower value (lower than 90%) of the precision and viceversa, with the DT algorithm the values obtained for all the three measures is always high (higher that 90%). Moreover, through the analysis of each wrong decision (the detailed list of all the made decision is not reported in this dissertation for shortness needs), it can be noticed that all the errors made by DT in the second experiment were related to combination of attributes that were not present in the training set, implying that DT do not work very well with unknown notifications contained in related data.

The CPU time was collected and the obtained values are reported in Table 2.11 and Table 2.12. Table 2.11 reports the CPU time needed to train the machine learning algorithms with the selected dataset. While, Table 2.12 reports the CPU time needed to make prediction about the best device on which the notification would have been sent (the time reported in the table is a mean value: it is computed dividing the execution time for making a prediction on all the notifications, for the total number of notifications). Confidence intervals are not presented in the tables since all the calculations occur for a single run of each ML algorithm. As can be seen, the SVM algorithm is the slowest algorithm between the analyzed ones, however, the classification time is acceptable for real situations. Consequently, considering the accuracy percentage obtained with related data (that are the most complete) and the obtained CPU time, DT is the most promising machine learning algorithm for notification classification and so it will be at the core of our future work.

2.7.2 Preliminary Evaluation and Conclusion

The previous paragraphs discussed preliminary experiments conducted with the aim of evaluating the best machine learning approach applicable to manage overwhelming notifications. The implemented prototype was tested using a dataset derived from real data provided by the MIT Media Laboratory Reality Mining project, enriched with additional synthetic information. Three different machine learning algorithms were used and results show that the DT algorithm is the most promising algorithm for our purposes in terms of balanced prediction accuracy, precision and recall and, so, it will be at the core of our future efforts. Future work would extend the dataset to include all the information discussed in the Architecture and the real data collected using the modules presented in the next chapters will be used. Furthermore, a careful evaluation of the machine learning algorithms would be performed and the possibility of using different algorithms after other ones would be evaluated. Figure 2.6 shows a possible future scenario in this direction: it exploits a subset of input features to predict an intermediate value through one algorithm (e.g., the SVM one) while it exploits another algorithm (e.g., the DT one) to predict the final expected output by using both the intermediate prediction and the remaining input values.

2.8 Collectors

The second group of contributions provided as part of the Smart Notification System is gathered in the *Collectors* group.

All the contributions discussed in this section are mainly implementation of the following list of abstract architectural modules reported in the architecture presented in Section 2.5):

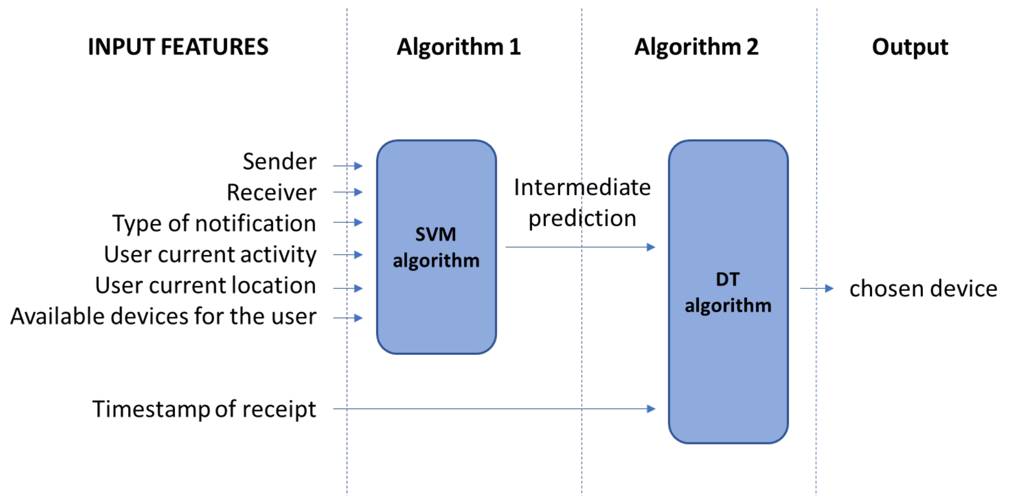


Fig. 2.6 Possible future scenario

- the ***Environment Context Collectors*** and the ***User Context Collectors*** modules responsible for collecting real context data,
- the ***Notification Collector*** module responsible for retrieving the notifications coming from external services and/or IoT devices.

The main objective of these contributions is the design and the development of a *Collectors* subsystem able to: a) collect and expose real user and environment context data to be also used in experiments with the other blocks of the Smart Notification System Architecture, and b) collect and expose real IoT and mobile notifications, with the corresponding user reaction, to be used by the Machine Learning approach proposed in the previous section.

Aiming at satisfying such requirements and looking at the information needed by the Smart Notification System to enhance the experience of Maria, the stakeholder of the running example presented in the 2.5 Section, three main source of information were identified as main inputs for our architecture:

- the city source, that gathers all the information that can be collected in a town related to the environment status, e.g., air quality, level of pollution, or level of traffic;
- the home source, that gathers all the information that can be collected in a house, e.g., house temperature, house humidity, the light, the noise, or the number of people that are currently present in a room;

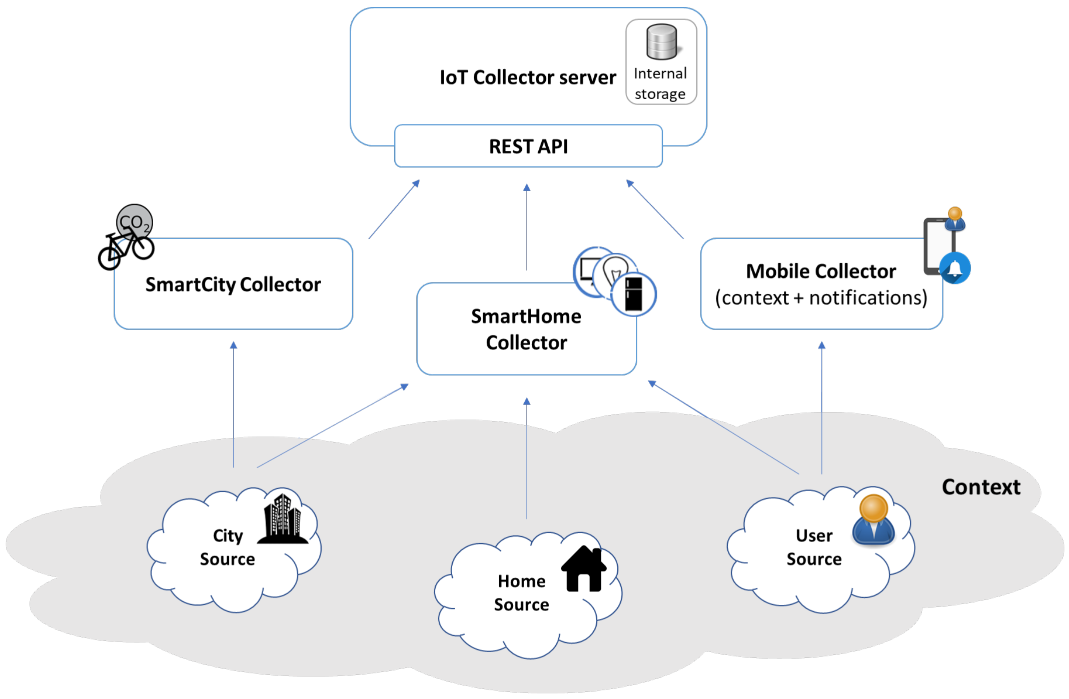


Fig. 2.7 Collector Architecture design

- the user source, that groups all the status information that can be gathered from a person, e.g., her position, her current activity, or the incoming notifications.

Four different independent prototypes were designed with the aim of fostering the collection of an initial dataset able to cover all the three identified source of information.

Figure 2.7 summarizes the four designed prototypes. Except from the *IoT Collector Server* that acts as an aggregator, each of the other prototypes is responsible for collecting data from at least one of the three presented source of information, as shown by the connection among the contexts and the three prototypes.

The names are representative of the target infrastructure on which they were supposed to be installed.

The *IoT Collector server* is the core of the Collectors subsystem. It acts as a data collector that receives new acquired data through the exposed API and stores them in an internal database. The *Mobile Collector* represents a mobile component that should be installed on the user smartphone(s) to collect a) user context information, b) every received mobile and IoT notification, and c) the user reaction to new

Source	**Information**	**Collectors**
City	CO2	SmartCity Collector
	Temperature Humidity Pressure	SmartHome Collector
Home	Temperature Humidity CO2 Pressure Noise luminescence User presence	SmartHome Collector
User	Activity Location Personal information Notification information (reported in Table 2.5 and in Table 2.6)	Mobile Collector

Table 2.13 Information identified for creating pilot dataset

received notifications. The *SmartHome Collector* represents all the modules that are physically installed in the user house and that are responsible for collecting home context information, notifications generated by the available IoT devices, and, in addition, the outdoor information (temperature, humidity, pressure) acquirable through the home IoT devices (e.g., IoT thermostat with an outdoor submodule). Finally, the *SmartCity Collector* groups all the modules that are physically placed in the city and are responsible for collecting city related environment information.

As detailed in the following subsections, the first implementation of the prototypes are mainly dedicated to the collection of a reduced set of information for each source.

Table 2.13 summarizes the list of the data that are collected in this initial implementation of the prototypes: the first column reports the source from which each information (reported in the second column) are extracted. In addition, the third column provides the name of the prototypes that were designed and implemented to collect the related information.

The following subsections discuss all the details of the four implemented prototypes and present the experiments performed with each prototype and the corresponding results.

2.8.1 IoT Collector server

The *IoT Collector server* is the core of the Collectors subsystem that acts as a data storage. Its main objectives are summarized in the following four high level requirements:

- accept registration of new devices/systems able to send data to be stored;
- accept registration of new users as owner of one or more devices and as receiver of the incoming notifications;
- store all the incoming information in a database to be reused in future experiments;
- allow registered devices/systems to access stored data.

To satisfy the exposed requirements, a data model and some APIs were defined.

2.8.1.1 Data structure: ER model

An entity relationship diagram, also called an entity-relationship (ER) model, is a graphical representation of entities and their relationships, typically used to represent the organization of data within databases or information systems. In this section it is used to represent the data that the *IoT Collector server* will store and their relationships.

Figure 2.8 reports the ER diagram that was designed to collect all the data coming from the envisaged prototypes and other future *Collectors* modules.

The main entities are described in the following list.

- The *Device* entity represents each device registered to the system as a data collector. It can be associated to one or more user(s) (e.g., a smart TV can be associated to all the people that usually live in the house in which it is installed)

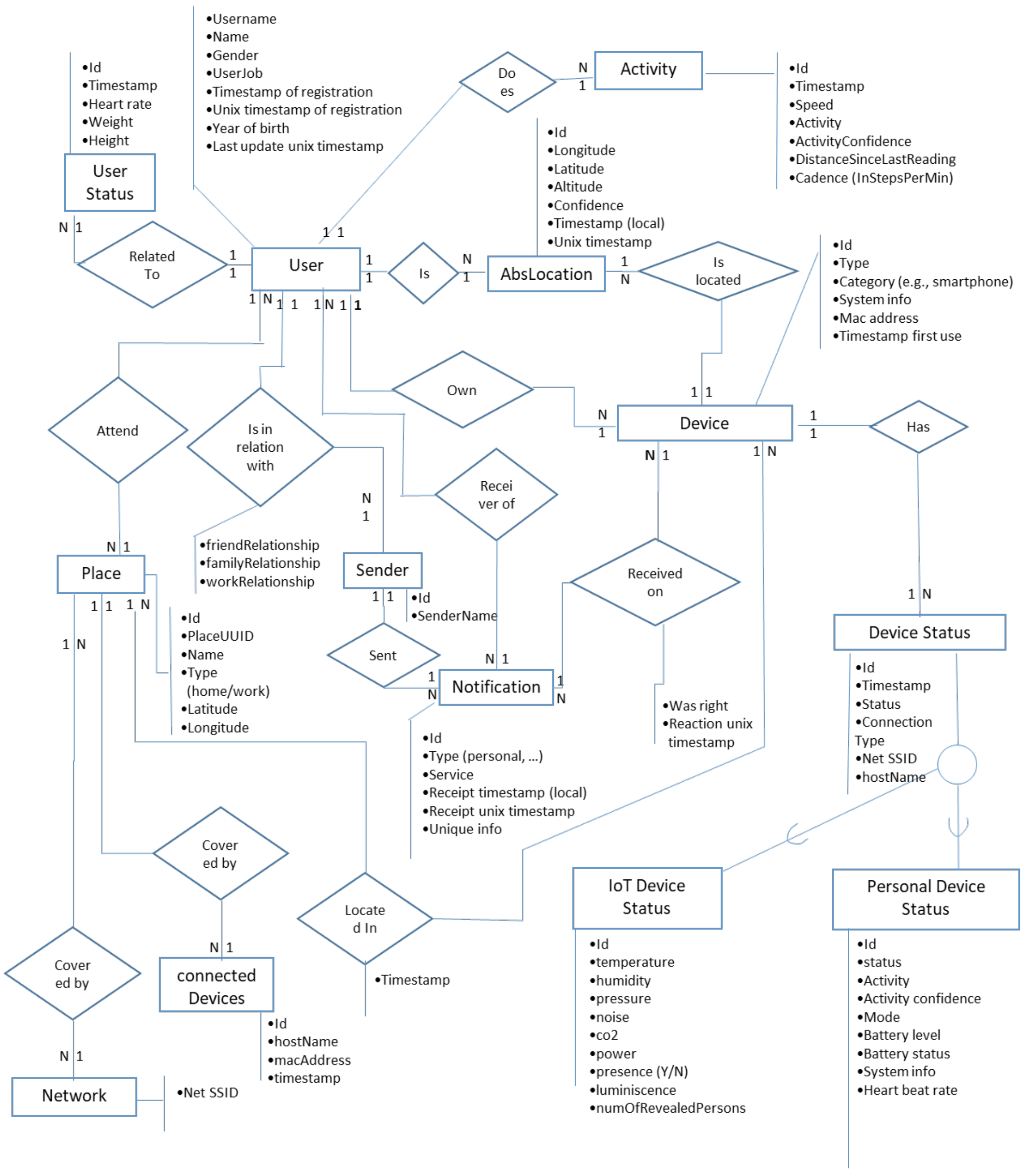


Fig. 2.8 ER diagram representing the data structure implemented on the IoT Collector server

and contains the static properties related to the Device (e.g., type, category, MAC address). Its dynamic properties (e.g., current power consumption) are stored in the *Device Status* entity.

- The *Device Status* entity stores all the dynamic properties of a *Device*, e.g., the network SSID at which it is currently connected. Its properties are split into 2 different groups: the *IoT Device Status* group that gathers all the information coming from IoT devices (e.g., the smart fridge) and the *Personal Device Status* group that gathers all the information arriving from personal devices (e.g., the user smartwatch).

- The *User* entity represents each user registered to the system as owner of a device or as a recipient of a notification. It stores all the static information related to a user and is associated to one or more *User Status* that stores user dynamic properties (e.g., user current heart rate). In addition, it can be associated to one or more *Device* actually owned by the user.
- The *Activity* entity stores the temporal sequence of user activities.
- The *AbsLocation* entity stores the temporal sequence of user locations in terms of raw latitude and longitude.
- The *Place* entity represents a logical place, e.g., user house, in which the user would be in a specific moment. Through the *Attend* relationship, in fact, it allows to know, when it is available, where the user is in a specific moment.
- The *Network* entity represents the known networks at which a user or a device could be connected. It can be associated to a place in which the specified network is available.
- The *Connected Devices* entity stores the list of the devices that are connected to a router/gateway even though they were not yet registered to the system. In fact, some IoT systems allow to know how many devices are connected to the network with some other possible additional information.
- The *Notification* entity stores all the incoming notifications. It can be associated to a) one or more *User* as receiver, b) one or more *Device* at which the notification was directly sent (e.g., in the case in which a service is already using a notification strategy to send the notifications to specific devices), c) a *Sender* with the additional information about its relationship with the receiver of the notification.
- The *Sender* entity represents a sender of a notification. Each sender is uniquely identified by a *SenderName* that is unique for each receiver: the real name of the sender (present in the mobile Contact list) is cyphered so that the same sender has always the same senderName, but her real name is not anymore

Verb	Description
GET	Used to retrieve resources (i.e., devices used as collectors) and their status or description or more other information
POST	Used to create resources, or performing custom actions
PUT	Used to update or insert resources
DELETE	Used to delete resources

Table 2.14 HTTP Verbs available in the IoT Collector Server

accessible. Consequently, if the same person sends a notification to two different users registered to the system, two different senders will be stored, while, if the same person sends more notifications to the same user, only one sender is registered and the sent notifications are registered as sent by him.

2.8.1.2 REST API

As the name suggests, the IoT Collector Server REST API is an implementation of the REpresentational State Transfer (REST) architectural style [24] (presented in Section 2.2.2).

Thus, all the consumers of IoT Collector Server API are able to use GET, POST, PUT, and DELETE verbs. These verbs greatly enhance the clarity of what a given request should do, as described in Table 2.14.

The IoT Collector Server RESTful API allows to:

- register new devices/systems able to send data to be stored;
- register new users as owner of one or more devices and as receiver of the incoming notifications;
- accept incoming information from registered users and/or devices;
- query the server to obtain stored information

To summarize, Table 2.15 and Table 2.16 report a function summary that shows the correspondence between URL and resources exposed by the system. The terms in "{...}" are parameters needed by the specific function associated to the corresponding resource. A parameter is usually a unique identifier for a device, a user, or a place.

Resource path	**Available methods**
/users/{username}	POST: Add a new user PUT: update user information
/users/{username}/status	GET: Get last user status PUT: Update user status
/users/{username}/location	GET: Get last user location PUT: Update user location
/users/{username}/activity	GET: Get last user activity PUT: Update user activity
/devices	GET: List all available devices related to the authenticated user POST: Add a new device
/devices/{device-id}	GET: Retrieve description of the specified device PUT: Update device information
/devices/{device-id}/deviceStatus	POST: Add a list of device statuses to the device specified by {device-id}
/devices/{device-id}/absLocation?start={start-Timestamp}&end={end-Timestamp}	GET: Return the list of absolute location related to the specified device received within the specified interval (if not specified only the last inserted absLocation is returned)
/devices/{device-id}/absLocation	POST: Add absolute locations to the device specified by {device-id}
/devices/{device-id}/deviceStatus?start={start-Timestamp}&end={end-Timestamp}	GET: Return the list of device statuses related to the specified device within the specified interval (if not specified only the last inserted deviceStatus is returned)

Table 2.15 IoT Collector Server: API Specification - part 1

All the resources that expose a POST or a PUT method accept as input a list of parameters formatted in JSON format. The details of such list are not reported in this thesis for brevity reasons, however the required parameters mainly reflect the list of attributes of the corresponding entities reported in Figure 2.8 (e.g., the parameters required for the place resource are the attributes of the *Place* entity).

Resource path	**Available methods**
/devices/{device-id}/notification?start={start-Timestamp}&end={end-Timestamp}	GET: Return the list of notifications related to the specified device within the specified interval (if not specified only the last inserted notification is returned)
/devices/{device-id}/notification	POST: Add some notifications (attached to the request in JSON format) to the list of notifications received on the device specified by {device-id}
/places	POST: Add a new place
/places/{place-id}	PUT: Update place information
/places/{place-id}/networks	POST: Add the list of Wi-Fi networks achievable in the specified place
/places/{place-id}/users	POST: Add an “inhabitant” to the specified place
/places/{place-id}/devices	POST: Add the provided list of devices as present in the specified place

Table 2.16 IoT Collector Server: API Specification - part 2

2.8.1.3 Prototype Implementation

A prototypal implementation of the IoT Collector server was developed using the Java programming language. Specifically, a RESTful web service was implemented to expose the APIs using the Jersey framework[5]. It is a framework for developing RESTful web services in Java and provides support for JAX-RS[6] (Java API for RESTful Services). In addition, the web service interacts with a MySQL database that reflects the data structure reported in Figure 2.8.

2.8.1.4 Preliminary Deployment Session

From the 1st of January 2016 and the 30th of June 2016, an instance of the IoT Collector server was run on a desktop PC with the following characteristics:

- CPU: Intel Core i5-750
- RAM: 8GB PC3-12800 (800MHz)

[5]https://jersey.java.net/, last visited on March 01, 2018
[6]https://github.com/jax-rs, last visited on March 01, 2018

- OS: Windows 7 Student edition
- Java version 7

It was deployed to support all the experiments conducted with the other prototypes presented in the following sections.

2.8.2 Mobile Collector

The *Mobile Collector* represents a mobile application to be installed on the user smartphone(s) with three main objectives:

- collect user context information (e.g., location and activity);
- collect all the mobile and IoT notifications received on user smartphone;
- collect the user reaction to the received notifications.

The development of the proposed application was divided into three main steps. At first, an experiment was conducted to understand which is the best way to collect user reaction to incoming notifications. Secondly, a set of information to be collected through the *Mobile Collector* was defined and, finally, the application was actually designed and developed.

2.8.2.1 How to collect user reactions to notifications

As the first step in developing the *Mobile Collector*, an experiment was conducted to investigate which is the best method to collect user reaction to notifications among two identified alternatives:

- ask users to fill in the survey shown in Figure 2.9 every 4 hours asking if the notifications received after the last survey were a) received at the right moment and b) in the right location;
- register the user touch gesture performed as reaction to the received notification. It is possible through the technique already used by Mehrotra et al [31]: if the user clicks on the notification, it means that the notification was appreciated, otherwise, if it was swiped away, it means that it was not.

Both methods have some limitations. The technical problems are discussed with respect to the Android operating system, but can be replicated for other existing mobile operating system.

- The "survey" approach is mainly affected by the following limitation:
  - the amount of time that lasts since the actual receipt of the notification. In fact, considering that the user is asked to fill the survey after a period of 4 hours it may happen that she does not remember the feeling felt when a 4 hour before notification arrived.
- The "user touch gesture" approach is affected by the following limitations:
  - the human errors. If, in fact, a user accidentally swipes away a notification instead of clicking on it, a non adequate reaction is registered;
  - the lack of a support for registering user reactions to notifications provided by the operating system. For example, Android API does not provide specific methods to understand the user reaction to notification (swipe/click). As a consequence it is necessary to adopt a "trick" for such an observation: Android API expose the *onNotificationRemoved* method[7] that is invoked every time a notification is removed from the Notification bar of a smartphone that runs Android. Then, to acquire user reaction it is necessary to monitor which is the foreground application in the moment in which it is invoked: if the user clicked on the notification, the application that generated the notification is on the foreground, otherwise she swiped it away. Such technique is only a "trick" that is not documented in any official documentation, so, some reliability evaluations may be needed, also to understand what happens in stressed conditions;
  - the possibility that the smartphone operating system has to group notifications of the same type. For example, in some version of Android, the operating system allows to group notifications of the same type and in that case, the *onNotificationRemoved* method (discussed in the previous point) should be invoked a number of times corresponding to the number of condensed notifications.

[7]https://developer.android.com/reference/android/service/notification/NotificationListenerService.html#onNotificationRemoved(android.service.notification.StatusBarNotification), last visited on March 01, 2018

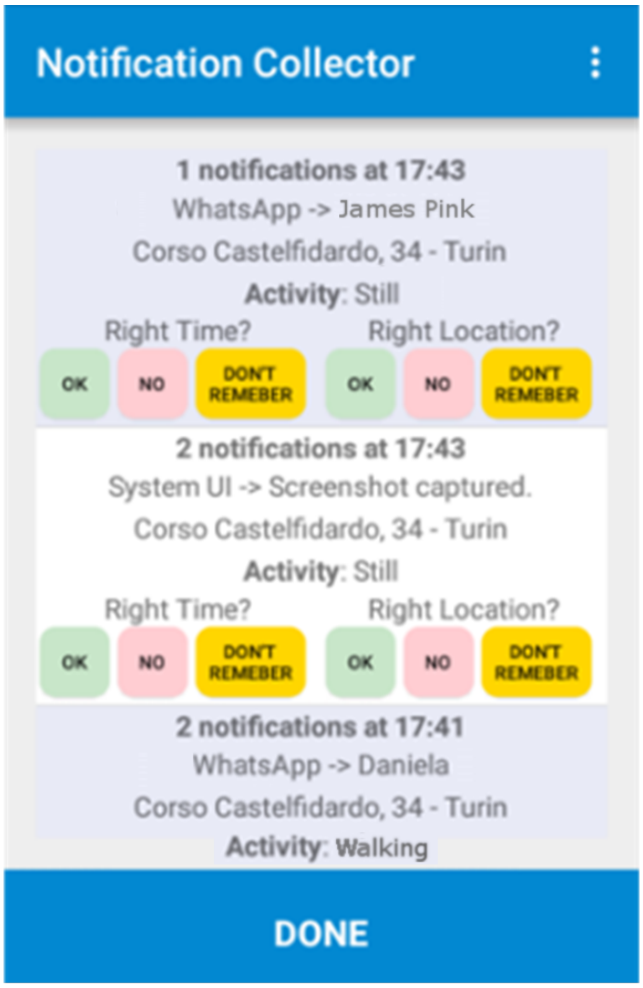


Fig. 2.9 Temporary App1: Survey to acquire User Reaction to Notifications

Within this experiment, the first exposed limitations was managed by adopting the following measures: in the “Survey” approach, only a subset of the collected notifications were shown to the user. Specifically, the notifications that arrived from the same person, while she was performing the same activity, within a minute were grouped in a single question showing the information related to the first of the grouped notifications.

Instead, for all the other limitations a more detailed investigation was needed to estimate the feasibility of the approaches and the reliability of the used techniques.

Consequently, the aims of the experiment presented in this section were two: a) the collection of user opinion about the two alternatives and b) the analysis of the feasibility and reliability of both approaches.

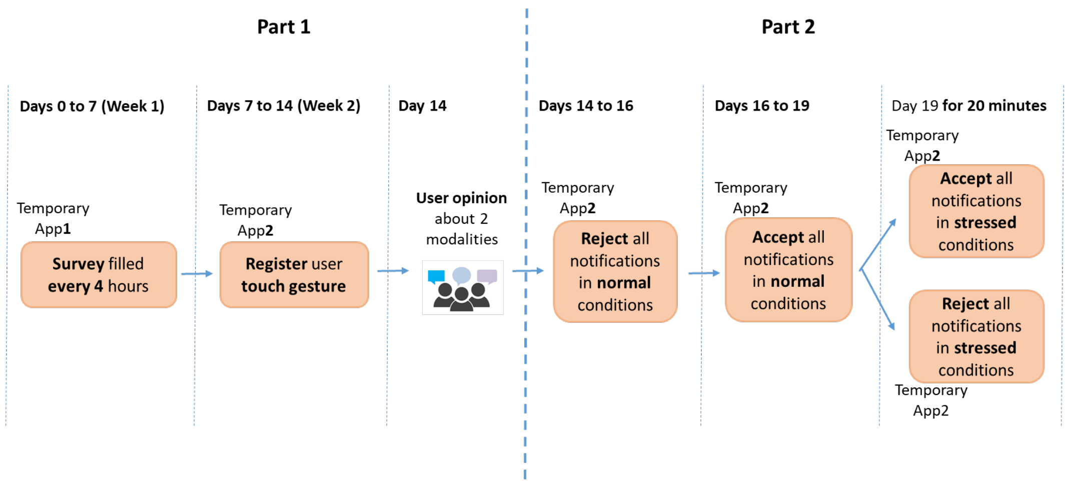

Part 1
Part 2
Days 0 to 7 (Week 1)
Days 7 to 14 (Week 2)
Day 14
Days 14 to 16
Days 16 to 19
Day 19 for 20 minutes
Temporary App1
Survey filled every 4 hours
Temporary App2
Register user touch gesture
User opinion about 2 modalities
Temporary App2
Reject all notifications in normal conditions
Temporary App2
Accept all notifications in normal conditions
Temporary App2
Accept all notifications in stressed conditions
Reject all notifications in stressed conditions
Temporary App2

Fig. 2.10 Experiment to investigate on the best way of acquiring user reaction to notifications

Two ad-hoc Android "temporary" applications were developed for this experiment:

- the first one ("temporary App1") a) intercepted every incoming notification and b) every 4 hours showed the survey presented in Figure 2.9 to collect reaction to notifications;
- the second one ("temporary App2") a) intercepted notifications and, b) immediately, registered the gestures performed as a reaction to them.

Figure 2.10 shows all the steps that compose the performed experiments with such applications (in each block there is an indication about the application used within each step):

- In the first part, five volunteers were asked to install separately (in different moments) the two ad-hoc Android applications to collect user feedbacks about the two modalities;
- In the second part, due to the novelty of the approach used in the second application (i.e., the one that registers user gestures), the same volunteers were asked to test the feasibility of the proposed approach through some ad-hoc tests.

Specifically, in the first part of the experiment the five volunteers were asked to perform the following actions:

- for one initial week, install the first Android application and fill the provided survey every 4 hours. During the night the survey was automatically discarded by the system and the user reaction to notifications was, so, not collected. In this first phase, almost 10000 reactions were collected among all the users;
- for another week, install the second Android application and behave as usual (i.e., swipe useless notifications and click on interesting ones) every time a new notification arrived.

At the end of this first part of the experiment each user was asked to express her opinion about the two alternatives, and, as expected, the second approach (i.e., register user gesture) was the most appreciated mainly because it did not require any explicit user intervention.

In the second part of the experiment, instead, the five volunteers were asked to participate to the following activities with the aim of estimating the reliability of the trick used to collect user reactions through gestures:

- **reject** all incoming notifications in **normal** conditions. In this phase, users were asked to swipe (to indicate the inutility) on all incoming notifications for 3 days;
- **accept** all incoming notifications in **normal** conditions. In this phase, users were asked to click (to indicate the utility) on all incoming notifications for 3 days;
- **reject** all incoming notifications in **stressed** conditions. In this phase, within an interval of 20 minutes, 1000 notifications were programmatically sent to users and she was asked to reject all of them through a swipe gesture;
- **accept** all incoming notifications in **stressed** conditions. In this phase, within an interval of 20 minutes, 1000 notifications were programmatically sent to users and she was asked to accept all of them through a swipe gesture;

The first two activities aimed at estimating the reliability in normal conditions (i.e., when a user receives a mean of 1 notification per minute), while the last two aimed at evaluating it in stressed conditions (i.e., when a user receives a mean of 50 notifications in a minute).

The most problematic situation revealed by the experiments was the one in which users were asked to accept all the notifications in stressed conditions: the accuracy of the collected feedback was 90%. The errors were mainly caused by two different problems.

- The time between a click and another: most of errors were generated because of the process of storing feedbacks and its slowness with respect to the user reaction. The user is, in fact, faster than the operating system, so the app is not always able to register the right performed action. The following scenario will clarify the problem: in the notification bar there are a) a Telegram condensed notification that groups 3 Telegram notifications (from 3 different senders), b) a Whatsapp condensed notification that groups 2 Whatsapp notifications (from 3 different senders) and c) a single notification from Hangouts. The user clicks on the Telegram condensed notification, so the *onNotificationRemoved* method is invoked 3 times, one for each of the 3 grouped notifications. In the meantime, while only the first instance of the method was completed, the user clicks on the Whatapp condensed notification and, even if the storing process was not yet completed, Whatsapp is brought to the foreground. So, the next 2 instances of the method still related to Telegram notifications will not find Telegram as foreground process, but Whatsapp, so they will assume that the user swiped the notifications.
- Accidental unwanted swipes on notifications.

In all the other circumstances the percentage of accuracy was between 99% and 100%.

After the analysis of strengths and weaknesses of both presented methods, even though the “gestures” method can register wrong feedbacks it was selected to acquire user reaction to incoming notifications for two main reasons: a) due to the absence of any user effort, it was chosen as the most appreciated by involved users, and b) in any case, the reported problems are mostly relegated to stressed conditions that rarely occur in real life.

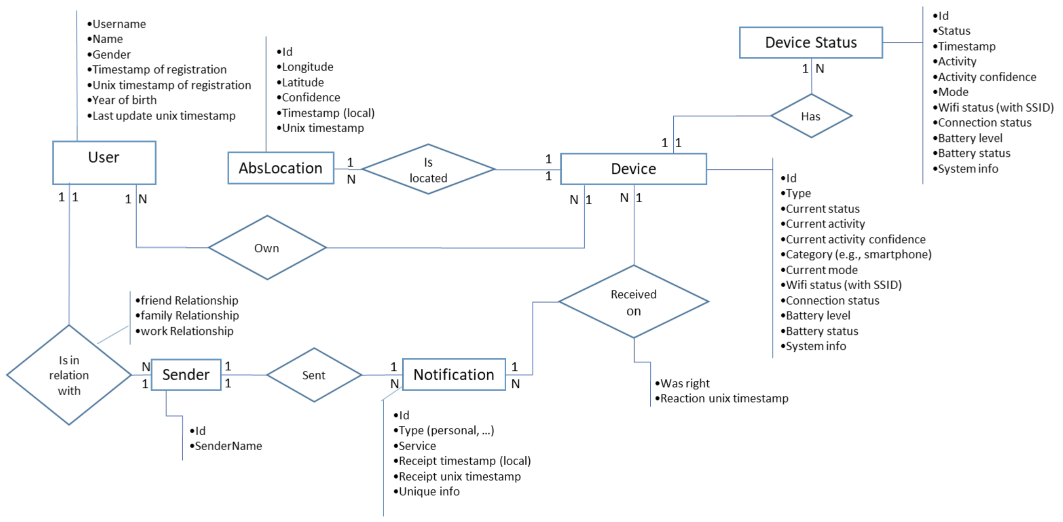


Fig. 2.11 ER diagram representing the data structure implemented on the Mobile Collector

2.8.2.2 Dataset definition

Starting from the data structure defined for the IoT Collector Server in Paragraph 2.8.1.1, a similar data model was designed for the Mobile Collector to store all the information that the Mobile Collector would collect.

The ER model reported in Figure 2.11 represents the data model designed for the Mobile Collector. The main entities are:

- the *User* entity that represents the user with her static information (e.g., name, gender);
- the *Sender* entity that represents the sender of a notification. It is related to the User with a relation that contains information about the relationship among the receiver (User) and the sender, i.e., friend, work or family relationship are the three available possibilities; Each sender is uniquely identified by a *SenderName* that is unique for each receiver: the real name of the sender (present in the mobile Contact list) is cyphered so that the same sender has always the same *senderName*, but her real name is not anymore accessible;
- the *Notification* entity that stores all the incoming notifications. It is linked to the *Device* entity with a multi-to-multi association: even though on each single

device there is only information about that specific device, it was mapped with such an association to reflect the same structure used on the IoT Collector Server. In addition, the Notification entity is also linked to the *Sender* entity: the relation stores, also, the information about user reaction;

- the *Device* entity that represents the up-to-date information related to the device (e.g., current status and current battery level). As can be observed in the diagram, heterogeneous data can be stored in the Device entity: considering that data comes from the device, we preferred to store data as related to the device that generated it instead of associate them to the user or to the location or anything else. The *currentActivity* property was designed to store the information that the modern mobile operating systems provide about the device current activity. For instance, Android is able to distinguish 8 activities: IN_VEHICLE, ON_BICYCLE, ON_FOOT, RUNNING, STILL, TILTING, UNKNOWN, and WALKING. The *Device* can also be connected to one or more users: if, for instance, a person borrow her smartphone to a friend it is possible to store the data separately for the two users.
- the *Device Status* entity that stores all the dynamic properties of a *Device*, e.g., the network SSID at which it is connected. This entity mainly stores the history of the current status of a device;
- the *AbsLocation* entity that stores the temporal sequence of user locations in terms of raw latitude and longitude.

2.8.2.3 Prototype implementation

A prototypal implementation of the Mobile Collector was developed for the Android operating system. The main requirements of the developed mobile application are reported in the following list:

- collect all the information reported in the ER diagram presented in the previous paragraph;
- send collected data to the IoT Collector Server at least one time a day and, in any case, every time a new Wi-Fi connection is available and the battery is not in critical status (i.e., under 15%);

- provide statistical information about the Notification received within the day, week or selected date interval;
- do not consume too much battery.

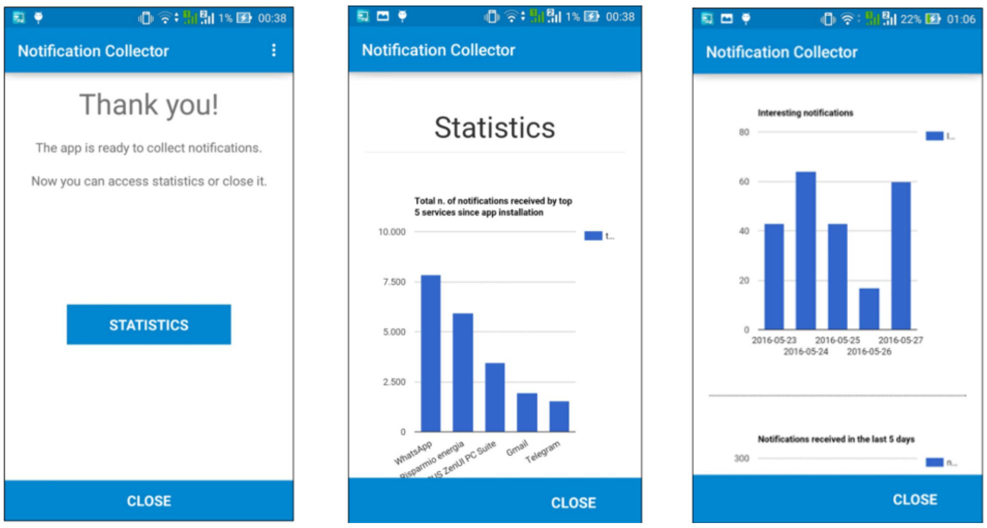
With the aim of satisfying all the requirements, an Android application was developed.

Among existing mobile operating systems Android was selected due to its diffusion and also for the low required costs for developers: as declared by IDC (International Data Corporation[8]) in 2015 Android was the most spread mobile operating system all around the world. Moreover, we concentrated our effort only on devices with Android 5.0 (Lollipop) or higher, due to new features introduced to manage notifications.

The developed application implements different background services registered as listeners of operating system messages and/or signals to collect needed data. Furthermore, every time the user enables GPS and/or network positioning services (for example, when she uses mapping services) the application registers available absolute location that will be used as features in the training phase of the Machine Learning algorithm.

The application was designed to work silently in the background, however, after the installation it asks people to insert a) some statistical information (age, gender, employment) and b) the relationship with the most important contacts present in their contact list. Such a second request was asked to understand the correlation among the behavior of the user with respect to notifications and the sender of a message/notification. After inserting such information, users were not asked to perform any other particular action: the first screenshot reported in Figure 2.12, in fact, present the message shown to the user after the application installation. It is representative of the absence of disruption caused to users. Nonetheless, as shown by the other two screenshots, after a few days of usage, the app was able to show statistics about the received notifications, but only after the user request about such functionality. Finally, it is possible to change settings and relationship with contacts present in the contact list.

[8]http://www.idc.com/prodserv/smartphone-os-market-share.jsp, last visited on March 22, 2017


Thank you!

The app is ready to collect notifications.

Now you can access statistics or close it.

Fig. 2.12 Screenshots of the implemented Mobile Collector application

As already depicted in the first part of this subsection, the reaction of users to notification was collected through the *onNotificationRemoved* method[9] provided by the Android API. It was invoked every time a notification was removed from the Notification bar of the Smartphone.

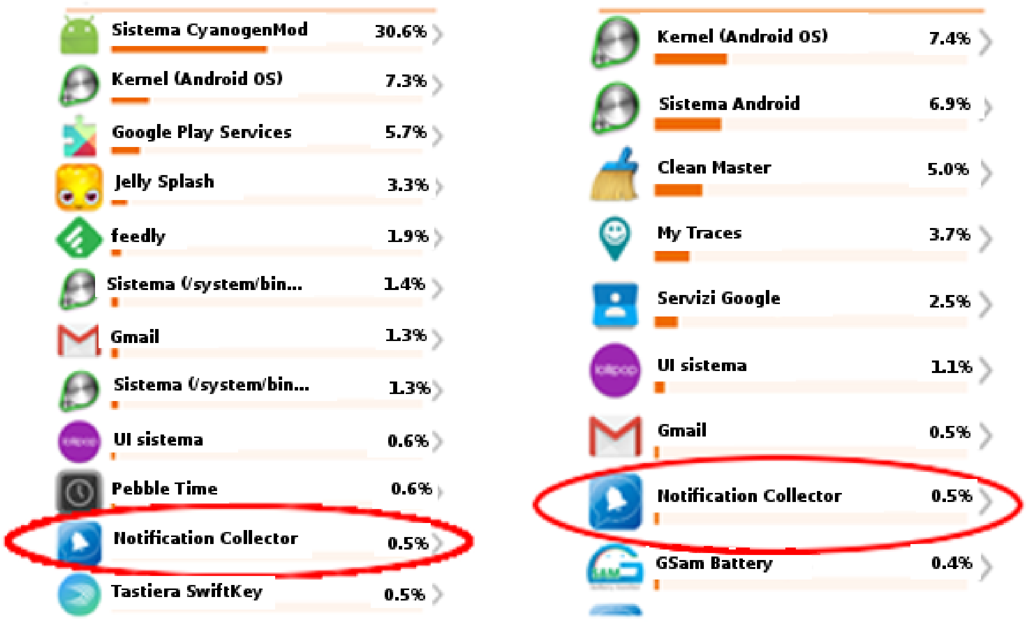
Furthermore, with the aim of reducing battery consumption, only the "battery saving" mode[10] was used to acquire GPS locations: it is a modality that uses only network information (i.e., Wi-Fi, Bluetooth or mobile network") to determine the location; even though its accuracy is lower than the one obtained with the "high accuracy" mode, it is enough accurate for our experiments (our experiments revealed that the location is provided within a radius of 100 meters).

Finally, to obtain detailed information about the battery consumption of our application, 2 users were asked to test the application consumption for one week through the "GSam Battery Monitor" application[11]. The test was performed on one

[9] https://developer.android.com/reference/android/service/notification/NotificationListenerService.html#onNotificationRemoved(android.service.notification.StatusBarNotification), last visited on March 01, 2018

[10] https://support.google.com/pixelphone/answer/6179507, last visited on March 14, 2018

[11] https://play.google.com/store/apps/details?id=com.gsamlabs.bbm, last visited on March 26, 2018



Sistema CyanogenMod	30.6%
Kernel (Android OS)	7.3%
Google Play Services	5.7%
Jelly Splash	3.3%
feedly	1.9%
Sistema (/system/bin...	1.4%
Gmail	1.3%
Sistema (/system/bin...	1.3%
UI sistema	0.6%
Pebble Time	0.6%
Notification Collector	0.5%
Tastiera SwiftKey	0.5%

a) consumption on OnePlus One

Kernel (Android OS)	7.4%
Sistema Android	6.9%
Clean Master	5.0%
My Traces	3.7%
Servizi Google	2.5%
UI sistema	1.1%
Gmail	0.5%
Notification Collector	0.5%
GSam Battery	0.4%

b) consumption on Karbonn Sparkle V

Fig. 2.13 Power consumption of the Notification Collector app on the two involved smartphones

"OnePlus One"[12] and one "Karbonn Sparkle V"[13] smartphones. As shown in Figure 2.13, on both smartphones, our application consumed only from 0.5% to 1% of battery a day with the "battery saving" mode always enabled for the whole day.

2.8.2.4 Preliminary Deployment Session and Results

With the aim of collecting an initial dataset containing a) real user context data and b) real IoT and mobile notifications, with the corresponding user reaction, a preliminary deployment session was organized. Specifically, from the 1st of January 2016 and the 30th of June 2016, the "Notification Collector" Android application was published on the Google Play Store[14] and 37 users installed it to help us in collecting data. During the preliminary deployment session, 37 people installed the "Notification Collector" Android application on their smartphones, with 6 females

[12]https://oneplus.net/it/one, last visited on March 26, 2018
[13]https://www.gsmarena.com/karbonn_sparkle_v-6687.php, last visited on March 26, 2018
[14]https://play.google.com/store, last visited on March 14, 2018

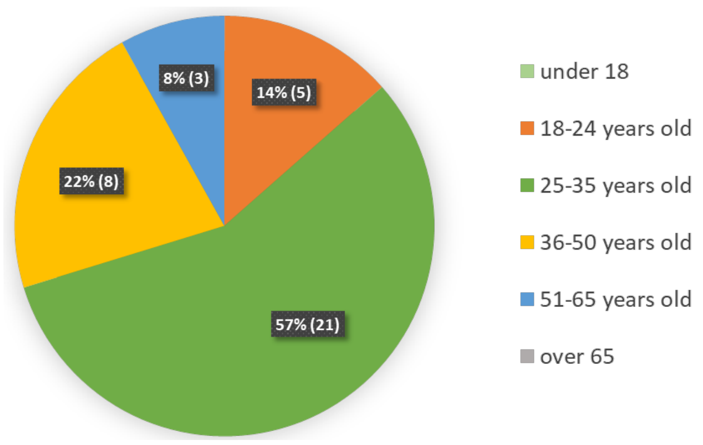


Fig. 2.14 Age of participants

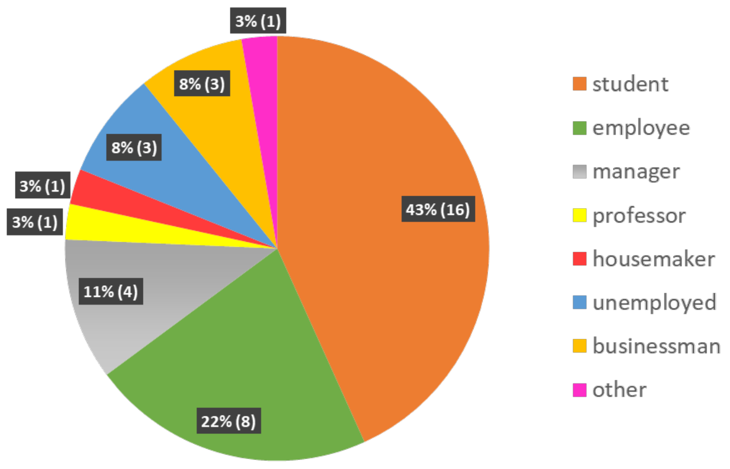


Fig. 2.15 Employment of respondents

and 31 males. 8 of them (1 female and 7 males) used the application for less than 5 days, while, the others, allowed us to collect data for a mean of 78 days (with a standard devation of 48.38). As already depicted, after the installation, the Android application asked users to insert some statistical information to identify their age and employment.

As shown in Figure 2.14, most of the participants were aged in the interval "25 - 35" and, as shown in Figure 2.15 they were mainly employed in University as students.

In addition, collected data reveals that:

- users receives an average of 247 notifications a day (with a standard deviation of 185);
- users are almost always in the same 3 or 4 places; even though we did not collect information about the meaning of each place, we suppose they are workplace, home, and some other places in which they usually go in their week routine;
- users receive most of the notifications from non-important contacts than from important ones. Unfortunately, considering that the name of the senders were crypted, we considered as “non-important person” all the senders that the users did not mark as friends, family or work related people. Consequently, all the social groups (e.g., Whatsapp/Telegram groups) that are, obviously, not saved in the contact list and people that users did not select as “important” were considered “non-important persons”;
- users often disable positioning features even though, sometimes, they activate it again in a few minutes;
- users are usually not aware of the existence of different modes to acquire their current location (Android, for example, has three different modalities: “high accuracy”, “battery saving”, and “device only”[15]).

2.8.3 SmartHome Collector

The *SmartHome Collector* represents all the modules that are physically installed in the user house with the aim of collecting home context information, notifications generated by the available IoT devices, and, in addition, the outdoor environment information (temperature, humidity, pressure) acquirable through the home IoT devices (e.g., IoT thermostat with an outdoor submodule).

The first step in designing the *SmartHome Collector* prototype regarded the identification of all the information that are both valuable for the experimentations with the Smart Notification System and easily acquirable through common IoT devices. The following list presents all of them. The *SmartHome Collector* should be able to collect data related to:

[15] https://support.google.com/pixelphone/answer/6179507, last visited on March 14, 2018

- user presence in the monitored house;
- presence of other persons in the monitored house;
- turned on devices in the monitored house;
- home environment information (i.e., temperature, humidity, noise and CO2);
- outdoor conditions information (i.e., external temperature and humidity);
- all the generated IoT notifications.

As second step in designing the *SmartHome Collector* prototype, the most common strategies and technologies usually adopted to acquire the information described in the previous paragraph were evaluated. As a result of such analysis, a list of different possible strategies were identified to collect each reported information. To collect **user presence in the monitored house** and **presence of other persons in user house**, the following alternatives were identified:

- an IoT presence detector (e.g., a Z-Wave presence detector) could be placed in the most attended places;
- a single-board computer (e.g., a Raspberry Pi[16]) could be configured to monitor the devices connected to the house Wi-Fi connection in specific moments of the day;
- a smartphone application could be installed on each user smartphone to collect both user position and current used Wi-Fi network;
- an IoT indoor weather station could monitor the temperature, humidity, CO2, level of noise and pressure of the house/room to understand if anyone is present in a room at specific moment of the day;
- some power outlets connected to the most used devices (e.g., TV, hifi, and microwave) could monitor appliances' usage and, consequently, the users presence in a room.

To collect information about **turned on devices in monitored house**, the following alternatives were identified:

[16]https://www.raspberrypi.org/, last visited on March 22, 2017

- a single-board computer (e.g., a Raspberry Pi) could be configured to monitor the devices connected to the house Wi-Fi connection in specific moments of the day;
- some power outlets connected to the most used devices (e.g., TV, hifi, and microwave) could monitor appliances' usage and, consequently, their status (turned on or off).

To collect information about **home environment information (i.e., temperature, humidity, noise and CO2)**, only one feasible alternative was identified:

- an IoT indoor weather station could monitor the temperature, humidity, CO2, level of noise and pressure of the house/room;

To collect information about **outdoor conditions information (i.e., external temperature and humidity)**, only one feasible alternative was identified:

- an IoT weather station with an external module could monitor the external temperature and humidity;

To collect **all the generated IoT notifications**, only one feasible alternative was identified:

- considering that, nowadays, all the IoT notifications mainly arrive on the user smartphone, a smartphone application could be installed on each user smartphone to collect both user position and current used Wi-Fi network;

2.8.3.1 Prototype implementation

With the aim of providing a reusable component to be used in multiple houses simultaneously, the most feasible alternatives presented in the previous section were selected. Consequently, the following list of devices, also shown in Figure 2.16, were identified to monitor all the presented activities.

- One Netatmo personal weather station[17] with an additional indoor module was chosen to monitor a) the ambient external temperature, humidity and CO2,

[17]https://www.netatmo.com/en-US/product/weather/, last visited on March 11, 2018

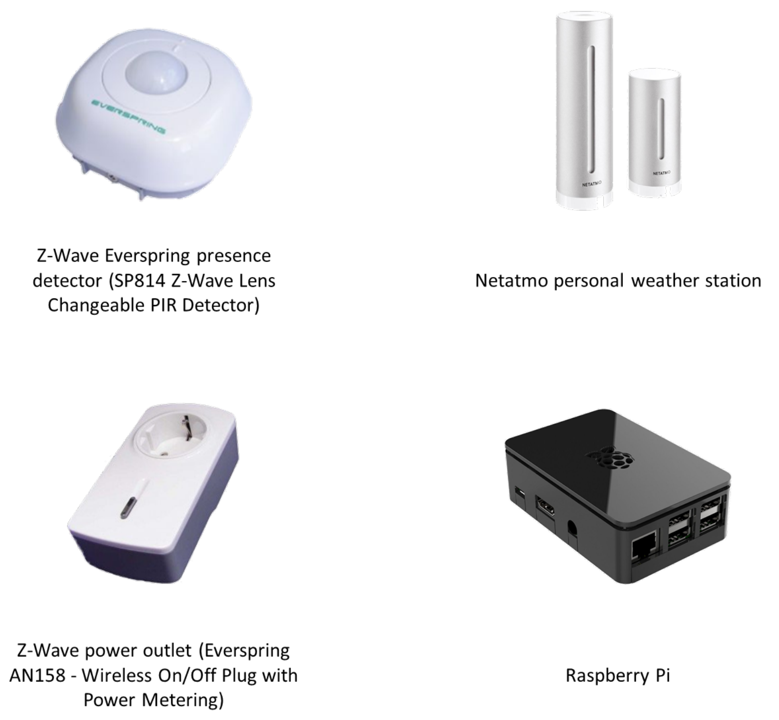

Z-Wave Everspring presence detector (SP814 Z-Wave Lens Changeable PIR Detector)
Netatmo personal weather station
Z-Wave power outlet (Everspring AN158 - Wireless On/Off Plug with Power Metering)
Raspberry Pi

Fig. 2.16 Devices chosen for developing the SmartHome Collector prototype

and b) the room temperature, humidity, noise, and pressure and CO2. One Netarmo personal weather station is supposed to be installed in each room of the involved houses.

- One Z-Wave Everspring presence detector (SP814 Z-Wave Lens Changeable PIR Detector)[18] was chosen to monitor the presence of people in a room. One presence detector is supposed to be installed in each room of the involved houses.
- One Z-Wave power outlet (Everspring AN158 - Wireless On/Off Plug with Power Metering)[19] was chosen to monitor the status of each appliance installed in the house. One power outlet is supposed to be connected to each appliance present in the involved houses.
- One Raspberry Pi was chosen to monitor the devices connected to the house Wi-Fi network. In addition, every Raspberry Pi is equipped with a RaZberry

[18] http://www.everspringindustry.com/SP814.aspx, last visited on March 11, 2018
[19] http://www.everspring.com/portfolio-item/an158-onoff-and-metering-plug/, last visited on March 11, 2018

card[20] to interact with all the other devices that uses the Z-Wave protocol to exchange information.

With the exception of the Raspberry Pi, this group of devices constitute a set of appliances that should be installed in every room of the house to be monitored.

The software running on the Raspberry Pi to monitor the devices connected to the Wi-Fi network was developed in the Python[21] programming language and the SQLite[22] SQL database engine was used to store data locally: the collected information are, in fact, sent to the server only once a day. Furthermore, the interval among each single acquisition was set to 5 minutes.

2.8.3.2 Preliminary Deployment Session and Results

Due to the already discussed role of context information in the SNS architecture, with the aim of collecting an initial dataset containing real user and environment context data, a preliminary deployment session was organized for the SmartHome Collector. It was performed during the last month of the deployment session organized for the Mobile Collector (June 2016) to obtain a complete dataset (containing almost all the information required by the SNS to make decisions) for future experiments. The developed prototype was installed in the kitchen of a house located in Torino, Italy and monitored the occupant behaviors for 30 days. In the house lived 2 housemates and the only shared ambient was the kitchen, in which all the shared activities (e.g., watch TV) were performed. Unfortunately, only one of the house occupants, was also involved in the "Mobile Collector" experiment. The information about such persons are listed below:

- the person also involved in the Mobile Collector experiment was 28 years old, male and employed as a Ph.D. student;
- the other person was 23 years old, male, and engaged as university student.

Collected data demonstrates that:

[20] https://z-wave.me/products/razberry/, last visited on March 11, 2018
[21] https://www.python.org/, last visited on March 11, 2018
[22] https://www.sqlite.org/index.html, last visited on March 11, 2018

- depending on the person present in the house, the temperature of the kitchen was different: maybe one of them preferred lower temperatures than the other; this information could be used by the SNS to predict the present of one of the house occupants in the house;
- when both people were present in the house, more noise was revealed especially during lunch and dinner time; by detecting the presence of both the housemates, for example, the SNS could establish which is the best moment to read loudly notifications that can be useful for both of them (for instance, the absence of gas during the next weekend due to a scheduled maintanance intervantion);
- the TV was mainly used when only one person was in the house and was off every time more than 3 people were present in the house (maybe for parties); this information could be used by the SNS to avoid some kind of notifications that are usually preferred when only the housemates are present in the house (e.g., reading loudly that the toilet paper finished);
- the outdoor weather conditions did not affect the usual behavior of the two housemates.

2.8.4 SmartCity Collector

The *SmartCity Collector* groups all the modules that are physically located or moving in the city and are responsible for collecting city related environment information.

With the aim of providing a first prototype to collect an initial dataset of real city context data, an IoT Crowd Sensing platform that offers a set of services to citizens was designed and prototyped: the SmartBike platform. Due to the extensive use of bicycles observed in literature for monitoring smart cities ([40–44]), the bicycle was selected as a crowd sensing probe for monitoring the city environment and, at the same time, provide services to involved citizens. In fact, based on a survey conducted to identify the most interesting bike-enabled services, the SmartBike platform provides: real time remote geo-location of users' bikes, anti-theft service, information about traveled route, and air pollution monitoring. The proposed SmartBike platform is composed of three main components: the SmartBike mobile sensors for data collection installed on the bicycle; the end-user devices implementing the user interface for geo-location and anti-theft; and the SmartBike central servers for storing and processing detected data, providing a web interface for data visualization and interacting with the IoT Collector server to provide collected data once a day.

The development of the proposed platform was divided into four main steps. At first, even though the main objective of such work was the collection of city context data, a survey was conducted to identify the most interesting bike-enabled features for users and the most important context information valuable for citizens. This step is based on the observation reported by Alam et al. [45] about the usefulness of involving users in designing new services: authors demonstrate that user involvement in designing new services facilitates the development of better and differentiated new services that match exactly customer needs. As a result, in the second phase, the four most preferred features emerging from the survey results were selected to inform the definition of the services provided by the platform. The selected features are the following: a) real time remote geo-location of user bikes, b) anti-theft service, c) information about traveled route (distance, duration, and rise), and d) air pollution monitoring. Except from the anti-theft service, all the features constitute the most valuable context data for citizens and represent the main contribution of the SmartCity Collector to the SNS system. Based on the extracted features, in this phase, the architecture of the SmartBike IoT crowd sensing platform was designed. After that, an initial prototype of the presented platform was implemented in the third phase and used in the forth phase to test the feasibility of the approach and the suitability of the platform.

The SmartBike platform has been designed within the OpenAgorà project[23], one of the proposals selected by the city of Turin for the Torino Living Lab Campidoglio experimentation[24]. Involving different partners (Turin TIM Joint Open Lab, Politecnico di Torino, and two startups, Move Plus[25] and Ponyzero[26]), the Open Agorà project aims at developing and testing solutions for helping people to make their mobility behavior more sustainable. In addition, it aims, at the same time, at providing data and tools that can be used by the city institutions to enhance the overall quality of life of their citizens.

[23]http://openagora.it/index-en.html, last visited on March 22, 2017
[24]http://torinolivinglab.it/bandi/tllcampidoglio/, last visited on March 01, 2018
[25]http://www.moveplus.it/, last visited on March 22, 2017
[26]http://www.ponyzero.com/, last visited on March 22, 2017

2.8.4.1 Architecture

After the first phase dedicated to the identification of the most interesting bike-enabled features for users and the most important context information valuable for citizens (more details are reported in Appendix A), in the second phase of the SmartBike platform development, the four most requested features resulting from the survey were selected. Based on them, two different services were, then, designed and the architecture of the platform was devised. The selected features are: a) real time remote geo-location detection of the users' bikes; b) anti-theft; c) information about traveled route (distance, duration, and rise); d) air pollution monitoring.

Except from the anti-theft service, all the features constitute the most valuable context data for citizens and represent the main contribution of the SmartCity Collector to the SNS system.

2.8.4.1.1 Provided services Looking at the results of the survey, the following two main services were identified as the ones that should be provided by the SmartBike platform:

a) the "city monitoring" service gathers the features (a, c and d of the list reported in the previous paragraph) aimed at collecting and showing air pollution information of areas traveled by involved cyclists. Considering that air pollution can be influenced by temperature, relative humidity and barometric pressure, those data should be acquired through appropriate sensors in addition to the air pollution information. The collected data should be geo-located and periodically sent to SmartBike central servers able to store it. Likewise, the platform should provide a web map showing the traveled routes and the level of pollution of the areas of the city monitored by available bikes. These services should be accessible into two different modes: a "personal" mode showing only user related information to logged users (e.g., the position of the owned bikes) and a "public" mode showing aggregated information obtained by merging data collected by each user;

b) the "anti-theft" service, instead, gathers the anti-theft feature (b of the list reported at the beginning of this section). The SmartBike platform should allow an authenticated user to enable/disable an anti-theft service that, by monitoring movements of the bike, should generate a notification whenever a thief tries to steal the bike. In addition, the platform should provide real time information about the bike location and the device status (e.g., battery level).

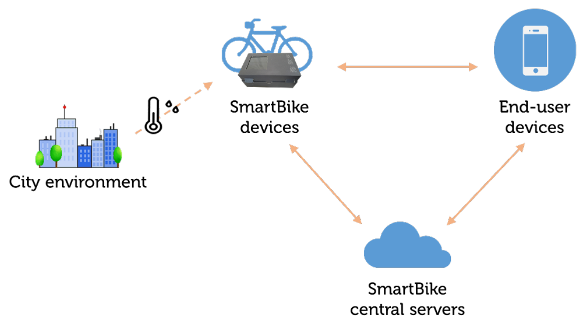


Fig. 2.17 High level architecture design of the SmartBike platform

Figure 2.17 shows the designed logical architecture of the SmartBike platform. It is composed of three main components: the SmartBike devices, the end-user devices (e.g., smartphones and tablets), and the SmartBike central servers.

2.8.4.1.2 Architecture design The **SmartBike devices** block represents the IoT objects that are mounted on the bicycles. They should be autonomous (i.e., always active and connected to the Internet even when the user smartphone/tablet is not close to them) and should provide the following functionalities to support the two services described in Section 2.8.4.1.1:

- environmental monitoring functionality able to collect data about air pollution, temperature, relative humidity and barometric pressure;
- bike status monitoring functionality able to detect any movement of the bike while the anti-theft service is enabled;
- data synchronization functionality able to periodically send acquired data to the SmartBike central servers. When the anti-theft service is enabled the information should be sent in real time to the SmartBike central servers;
- location functionality used to geo-tag acquired information and locate the bike in case of theft or loss;
- communication functionality used to provide interactions with end-user devices.

The **end-user devices** block represents the smartphones or tablets on which a dedicated SmartBike application is installed. This app permits the following functionalities with the SmartBike devices and the SmartBike central servers:

- authenticate the user and enable interactions with her own SmartBike devices;
- enable/disable anti-theft service;
- generate theft alert notifications whenever it is informed about a theft by the SmartBike central servers;
- present the current status information (i.e., gas concentration, temperature, relative humidity, barometric pressure and battery level) about all SmartBike devices owned by the user;
- visualize the location of the device on a map;
- show information about traveled route.

Finally, the **SmartBike central servers** represent one or more back-end servers that provide four different kinds of functionalities as parts of both the two described services:

- a data collection functionality, able to periodically receive data sent by SmartBike devices owned by different users. These data are useful for both final users and city institutions;
- a web interface similar to the one provided by the app run on end-user devices to supply a) a map to geo-locate in real time the user bikes, b) a map based on historical information to show the air pollution conditions of the areas traveled by involved cyclists, c) information about personal traveled route. This interface, and specifically, the functionalities described in b) and c) will be interesting for both cyclists and city institutions;
- an anti-theft functionality that a) redirects theft notifications sent by SmartBike devices to the right end-user devices and b) sends an anti-theft notification if no data are received from a SmartBike device for a certain amount of time;
- a synchronization functionality that redirects all the received data to the IoT Collector Server.

2.8.4.2 Prototype

In order to evaluate the suitability of the platform, an initial prototype of the presented architecture has been implemented. As an initial prototype, the main assumption considered in the following description is that a user owns only one bike and one end-user device (e.g., smartphone and tablet). However, the prototype can be easily extended to cases in which a single user owns more than one bicycle and/or more end-user devices. In addition, in this prototype, the air pollution was monitored through the carbon monoxide concentration, only.

The following subsections illustrate the details of the components that compose the presented platform.

2.8.4.2.1 SmartBike devices Figure 2.18 shows the implemented SmartBike device prototype.

It was implemented using the STM32 Nucleo L476RG board [27] equipped with the ARM mbed 3.0 Operating System [28] as control board. The board contains the basic components (i.e., the CPU, memory and some ports) but it can be easily extended with a large number of specialized application hardware add-ons and shields. In this work it was expanded by adding the following shields:

- Adafruit FONA808 [29], as GSM/GPRS and GPS module for transmitting data to the SmartBike central servers and provide the location of the bikes;
- ST X-Nucleo-IDB05A1 [30], as Bluetooth Low Energy 4.1 shield for providing connections with near end-user devices;
- ST XNucleo-IKS01A1 [31], as environmental and motion sensor shield for acquiring temperature, humidity, pressure and motion (acquired through the accelerometer);

[27] http://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-nucleo/nucleo-l476rg.html, last visited on March 01, 2018
[28] https://www.mbed.com/en/platform/mbed-os/, last visited on March 01, 2018
[29] https://www.adafruit.com/product/2542, last visited on March 01, 2018
[30] http://www.st.com/en/ecosystems/x-nucleo-idb05a1.html, last visited on March 01, 2018
[31] http://www.st.com/en/ecosystems/x-nucleo-iks01a1.html, last visited on March 01, 2018

- Nemoto NAP-505 [32] with Texas Instruments LMP91002 Analog Front End (AFE) [33], as electrochemical CO sensor chosen for its low power consumption and good reading accuracy at a reasonable cost.

The device is autonomous: a battery guarantees its power supply and the GSM/GPRS module guarantees its connectivity functions even when it is not connected to a smartphone/tablet. As a requirement of this prototype, a 10 Ah LiPo battery was chosen to let the device be active for 48 hours while collecting data with an interval of 12 seconds.

2.8.4.2.2 End-user devices Figure 2.19 shows two screenshots of the designed end-user prototype application. The application was developed for Android devices running at least Android 5.0 and was developed using the Android Studio IDE. The implemented user interface is minimal and is composed of two activities providing different functionalities. The main activity is shown in Figure 2.19a. It implements the following functionalities:

- it provides a button to connect the end-user device to the hardware device via a Bluetooth Low Energy connection;
- if connected to the hardware device, it provides the current status of the hardware device showing the revealed carbon monoxide, temperature, relative humidity and pressure;
- if connected to the hardware device, it provides a button to activate or deactivate the anti-theft service.

The second activity, i.e., the map activity, is shown in Figure 2.19b and provides a map showing the current location of the bike.

In addition, two background services were implemented: the first one provides an interface for receiving notifications from the SmartBike central servers in case of theft, while the second one manages the connections needed to periodically send collected data to the SmartBike central servers.

[32] http://www.nemoto.eu/nap-505.html, last visited on March 01, 2018
[33] http://www.ti.com/product/LMP91002, last visited on March 01, 2018

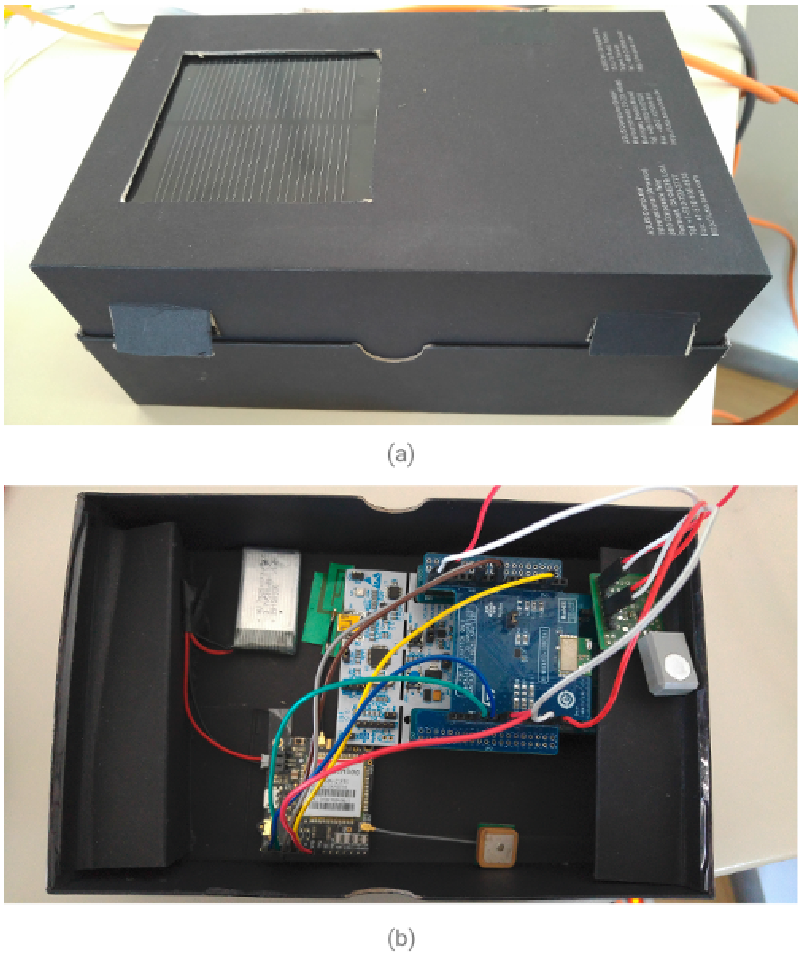
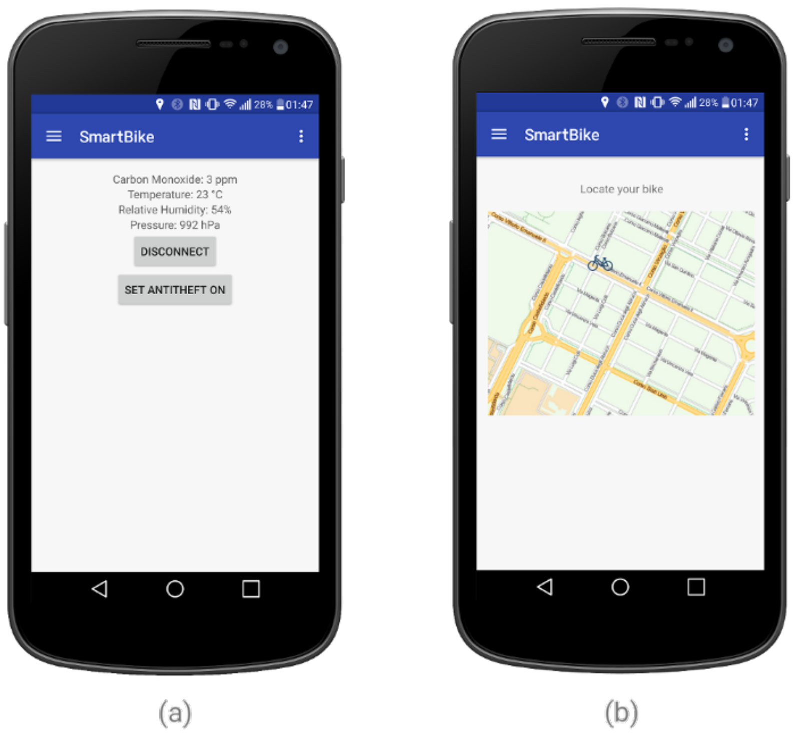


Fig. 2.18 The SmartBike device prototype

2.8.4.2.3 SmartBike central servers To provide the designed services, a solution integrating a back-end and a front-end platform was adopted in implementing the SmartBike central servers block. SiteWhere[34], an open source IoT server platform, was used as back-end to collect and store all the data sent by both SmartBike devices and end-user devices and to provide this data to the mobile and the web applications. It collected: a) air pollution information, b) location information, c) bike motion (acquired through accelerometer), d) SmartBike devices' statuses.

The web application, acting as front-end platform, was implemented based on Meteor [35], a full-stack JavaScript platform for developing modern web and mobile applications. The implemented Meteor application grants a) access to a map showing

[34]http://www.sitewhere.org/, last visited on March 01, 2018
[35]https://www.meteor.com/, last visited on March 01, 2018


SmartBike

Carbon Monoxide: 3 ppm
Temperature: 23 °C
Relative Humidity: 54%
Pressure: 992 hPa

SmartBike

Locate your bike

Fig. 2.19 Screenshots of the designed prototypal application

traveled routes and the level of pollution of the areas of the city monitored by available bikes, and b) the generation of all the notifications sent to users to report theft attempts. The solution was deployed on a free Amazon EC2 [36] instance.

Figure 2.20 shows the map reported in the web interface to let the user know her bike current location and the recently traveled route.

2.8.4.3 Preliminary Results

To evaluate the feasibility of the SmartBike platform, its suitability and the accuracy of the implemented prototype, the whole prototype has been subjected to a test phase. A volunteer cyclist was invited to bike for 30 minutes: the implemented prototype of the SmartBike device was placed inside the basket of his bike and the Android application was installed on his smartphone running Android 6.0.1 (a Karbonn Sparkle V smartphone).

[36]https://aws.amazon.com/ec2/?nc1=h_ls, last visited on March 01, 2018

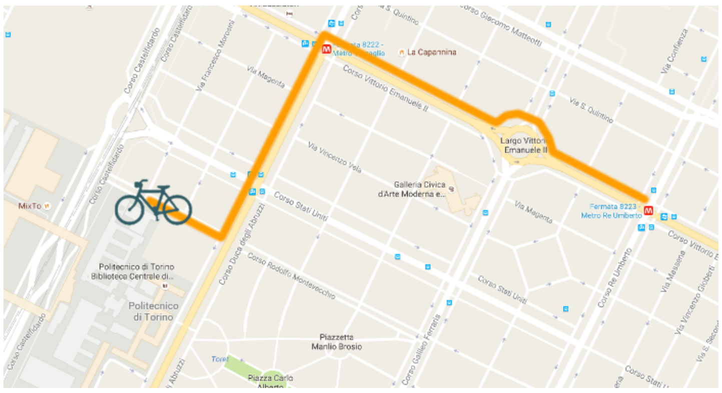


Fig. 2.20 The position of the bike can be remotely visualized on a map with the traveled route

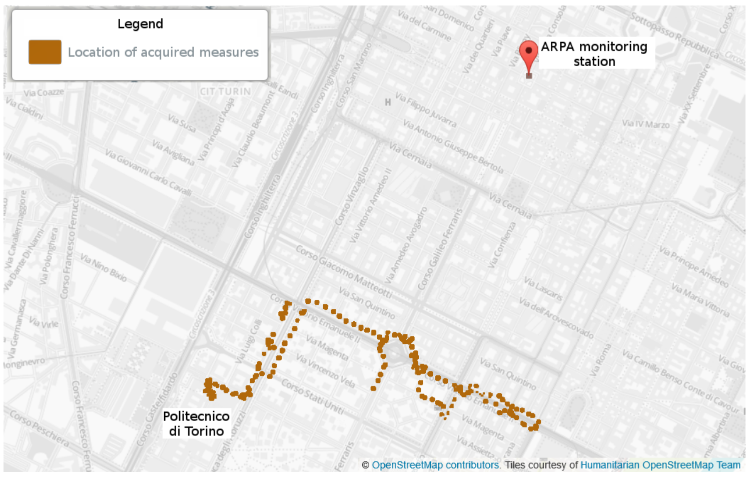

Legend
Location of acquired measures
ARPA monitoring station
Politecnico di Torino
© OpenStreetMap contributors. Tiles courtesy of Humanitarian OpenStreetMap Team

Fig. 2.21 Map of the area of Turin (Italy) monitored within the test

The experiment was conducted on the 26th of May 2016 for 30 minutes between 14:00 and 14:30. The user went through the area of Turin (Italy) shown in Figure 2.21 and located near Politecnico di Torino.

Air pollution information were collected simultaneously using a) the implemented platform prototype and b) the closest ARPA (Italian acronym of “Regional Environmental Protection Agency”) city monitoring station situated about 3 km far from the area of experiment (it is precisely located in Via della Consolata in Turin, Italy).

Moreover, at the end of the experiment, the anti-theft service was tested by trying to move the parked bike to simulate a theft attempt without the presence of the bike owner.

The analysis of collected data actually demonstrated that the implemented prototype was able to provide all the services described in the previous sections:

- the CO value measured by the SmartBike device was compatible with the one measured by the ARPA station. The average value of carbon monoxide acquired by the SmartBike device and calculated over 180 samples (1 every 12 seconds) is 1.2 ± 0.5 ppm (part per million). Instead the one monitored by the ARPA station[37], calculated as the average value of the two measures acquired at 14:00 and 15:00, is 0.892 ppm (1.1 $\mathrm{mg\,m^{-3}}$) and the two measurements are compatible;

- a notification arrived on the user smartphone after a few seconds from the beginning of the theft attempt.

2.8.5 Final Performed Experiments

With the aim of validating the Machine Learning approach described in the 2.7 Section, the same experiment was repeated with the real data collected in the parallel Collectors' deployment sessions described in the 2.8.2.4 and 2.8.3.2 subsections. Specifically, the experiments aimed at testing the effectiveness of the model (in terms of accuracy, precision and recall) and comparing them with the results obtained with the MIT dataset to understand if the promising adoption of machine learning approach in such domain is actually feasible and, eventually, continue experiments by involving real users in the evaluation.

Considering that the software responsible for interacting with the dataset was implemented through the Java programming language and with the aim of reusing existing code, the present experiments were performed using the Weka [46] workbench for Machine Learning for Java. As already done in the evaluation phase of the 2.7 Section, the k-fold cross validation technique was used to train the machine learning algorithms and then test them.

As already explained, the k-fold cross validation method is a technique used to validate a built machine learning model: the dataset is divided into k subsets and, then, the training and test phases are repeated k times, such that each time, one of

[37]Actual values acquired through the ARPA monitoring station were taken from the ARPA official website[38].

the k subsets is used as the test set and the other k-1 subsets are put together to form a training set.[39].

Within this experiment the k value was set to 10.

Unfortunately, at the time of the experiment, the “Support Vector Machine” (SVM) algorithm was not working properly in the Weka workbench: it crashed after a few seconds of running reporting generic Java exceptions that, after several tests, the author of this thesis reported to the Weka devolepers without receiving any feedback. Consequently, experiments were only performed with the “Gaussian Naive Bayes” (GNB) and the “Decision Trees” (DT) algorithms.

The following list summarizes the information used as machine learning features. Each record in the dataset corresponds to an incoming notification and contains all the listed information.

- Notification type: it distinguishes mobile notifications from IoT ones.
- Generating service: it represents the service that generated the notification (e.g., Telegram);
- Ringtone mode: it represents the mode that was set at the moment in which the notification arrived (possible value: Silence, Vibration, Sound).
- Notification sender: it represents the sender of the notification.
- Sender-Receiver FAMILY relationship: it is set to true if the sender and the receiver of the notification are family related persons.
- Sender-Receiver FRIEND relationship: it is set to true if the sender and the receiver of the notification are friends.
- Sender-Receiver WORK relationship: it is set to true if the sender and the receiver of the notification are colleagues.
- Day of the week in which the notification was received; Considering that Monday is near Sunday and we wanted to infer such an information to the machine learning algorithm, it was splitted into 2 different values:

[39]https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f, last visited on March 28, 2018

  - Sine of day of week; it is calculated with the following formula:

$$\sin(\frac{2*\pi*day}{7}) \tag{2.4}$$

  - Cosine of day of week; it is calculated with the following formula:

$$\cos(\frac{2*\pi*day}{7}) \tag{2.5}$$

- Day of the month in which the notification was received. Considering that the 1st day of the month is near the last day of the previous month and that we wanted to infer such an information to the machine learning algorithm, it was splitted into 2 different values:
  - Sine of day of the month;
  - Cosine of day of the month;
- Month in which the notification was received; Considering that January is near December, it was splitted into 2 different values:
  - Sine of month;
  - Cosine of month;
- Time at which the notification was received. Considering that the time is a periodic function, it was splitted into 2 different values:
  - Sine of time; it is calculated with the following formula:

$$\sin(\frac{2*\pi*time}{24*60*60}) \tag{2.6}$$

  - Cosine of time; it is calculated with the following formula:

$$\cos(\frac{2*\pi*time}{24*60*60}) \tag{2.7}$$

- User location expressed in Longitude and Latitude.
- Activity performed at the time of receipt; Possible values are IN_VEHICLE, ON_BICYCLE, ON_FOOT, RUNNING, STILL, TILTING, UNKNOWN, WALKING.

- Battery level at the time of receipt.
- Battery status (charging or not charging).
- Connection type; possible values are Wifi, network, NoConn.
- Wifi SSID.

Maybe due to the lack of services directly provided to participants and the users' concern on user privacy (even though we provided all the needed information to reassure users about our effort in making anonymous all collected data, some users declared that they were yet worried about their privacy), only 14 users allowed to collect all the reported information for a meaningful number of days (at least 15 days): Android allows to disable specific services or access to specific data at runtime, so, even though the app was installed on the users' smartphones, some information were not collected for all the day in which the application was installed. Furthermore, even though experiments were independently performed for each involved user, to obtain reliable results, the same period was considered for all the users: considering that some users uninstalled the application after a few weeks of usage, only 20 days (the ones in which all the 14 users were active: from 1st of February 2016 to 20th of February 2016) were used fot the experiments.

Table 2.17 reports the results obtained by using the machine learning algorithm to estimate if the arriving notification would be appreciated or not by the user.

The table shows prediction accuracy results obtained with both GNB and DT algorithms. Even though the mean values would suggest to trust both algorithms, some observations are required. Looking at the percentage of the appreciated notifications reported in the second column of the table, it is clear that the highest accuracy is obtained with users that have a highest percentage of accepted notifications: it is evident that for the users with a high percentage of accepted notification (higher than 70%) the algorithm fell in overfitting case. Consequently, only users with a percentage of appreciated notifications that is included in the interval 50%-70% can be considered in the evaluation of the technique. However, excluding the 4 users that brought the algorithm in overfitting (i.e., Users 3, 5, 10, and 14) the accuracy remains high (mean: 73%) and the two algorithms seems to be equiparable (the number of users that obtain an higher accuracy with the first algorithm is almost the same of users that obtain an higher accuracy with the second one). Future development will

User	% of appreciated notifications over all	Naïve Bayes			J48 (Decision Trees)		
		accuracy	precision	recall	accuracy	precision	recall
User 1	62%	75%	75%	75%	69%	69%	69%
User 2	70%	94%	94%	94%	99%	99%	99%
User 3	96%	92%	91%	92%	95%	91%	95%
User 4	63%	60%	63%	60%	68%	68%	68%
User 5	81%	92%	94%	92%	92%	91%	92%
User 6	65%	58%	56%	58%	58%	53%	58%
User 7	54%	78%	78%	78%	83%	83%	83%
User 8	72%	72%	78%	72%	70%	65%	70%
User 9	57%	67%	68%	67%	60%	59%	60%
User 10	77%	72%	75%	72%	70%	64%	70%
User 11	66%	70%	70%	70%	74%	81%	74%
User 12	61%	82%	82%	82%	91%	92%	91%
User 13	53%	78%	82%	78%	88%	88%	88%
User 14	87%	74%	76%	74%	89%	80%	89%
Mean		76%	77%	76%	79%	77%	79%
Standard deviation		11%	11%	11%	13%	13%	13%

Table 2.17 Preliminary results with real data

test the algorithms with more extended dataset and the SVM algorithm will be also included as third alternative.

2.9 Context Analysis

The data collected through the Collectors and exposed in the previous section, highlighted some significant habits usually maintained by users and already discussed in the previous section. One of them was particularly interesting from our point of view: the one related to the location estimation and its battery consumption. Specifically, even though we asked people to bring location features always activated and in "battery saving" mode, they usually disabled it due to their fear of consuming too much battery power. Such an observation together with the fact the users usually stay in no more than 3 or 4 significant places during their weeks, inspired the design of the following work. The proposed method is able to estimate user presence in meaningful places without energy expensive methods, like GPS or network positioning techniques and could be, also, used by the SNS to infer user current location when needed to make decisions on the distribution of notifications (e.g., a notification could be shown only when the user enters her house).

The term "meaningful place" used within this section represents a place in which a user recurrently stays for a certain period of time [47]: it is a group of near locations that can be considered as a unique place, such as home, school or the workplace.

2.9.1 Location Estimation

The remainder of the section is organized as follows: Section 2.9.1.1 explains the proposed method, Section 2.9.1.2 describes the data selection and collection process, Section 2.9.1.3 describes the method used to identify the 2 most attended meaningful places, and Section 2.9.1.4 shows the operations performed on data to obtain useful features for the cross validation phase presented in Section 2.9.1.5. Finally, Section 2.9.1.6 evaluates results obtained by using a Machine Learning classification algorithm based on Decision Trees, Section 2.9.1.7 reports the related works and Section 2.9.1.8 presents some concluding considerations and discusses possible future works.

2.9.1.1 Method

As declared by Chon et al. [48], people usually spend 85 $\pm$ 3% of their time staying in a place, while they spend 13 $\pm$ 3% of their time on the move. Based on this knowledge, in this section a method that is able to establish user presence in the 2 most attended meaningful places is evaluated. Specifically, it is demonstrated that it is possible to estimate where the user is in a certain moment of the day with high accuracy and without using energy expensive methods. In order to demonstrate such an assumption, a method that uses Machine Learning supervised classification algorithms based on Decision Trees is proposed for predicting user presence in a meaningful place.

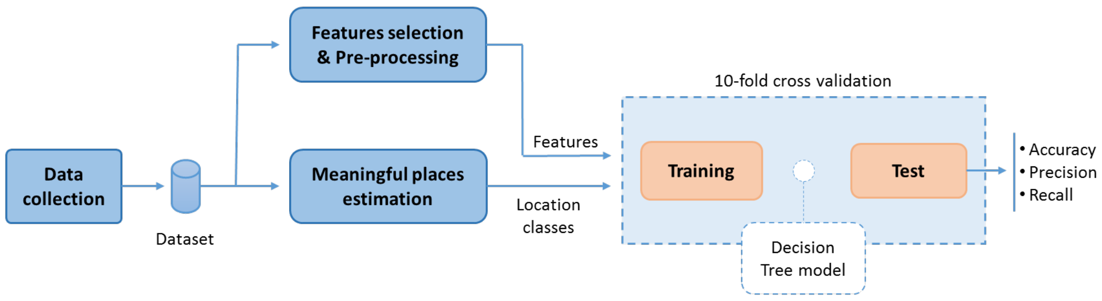


Fig. 2.22 Model that describes the estimation process performed for each user

The Decision Tree algorithm has been evaluated according to the work-flow shown in Figure 2.22. After an initial phase in which data are collected through the Android application presented in Section 2.8, the 2 most attended meaningful places are identified in the "Meaningful places estimation" phase and labels are assigned to collected data. In addition, collected items are filtered in the "Features selection & Pre-processing" phase: it is dedicated to a) remove useless information from the collected data and b) encode data for better performance of the classification algorithm. The dataset resulting from these three phases is used as input for the 10-fold cross validation process that produces accuracy, precision and recall measures as method estimation results.

Due to the different daily routine of each user, the shown steps are replicated for every user involved in the study and, at the end, a mean value is calculated for each computed measure. The following subsections describe each phase with more details explaining the contribution of each step to the whole accuracy estimation.

2.9.1.2 Data collection

As presented in the previous section, the first step in evaluating the proposed method is the "Data collection" phase. For our purpose, the data collected by the Android application developed as part of the Collectors was used.

Table 2.18 shows the information chosen among all the collected one that do not require extra energy to be retrieved and that is collected through the Android application. The table shows 4 main categories of information: the "Time information" represents the moment in which the estimation is performed. It is expressed by three different fields: the time of day, in seconds, the date (that contains the day of the month (1-31), the month and the year), and the day of the week. These

Time information	Time
	Date
	Day of the week
Notification information	Type
	Generating service
	Sender-receiver relationship
Device state	Battery level
	Charging state
	Ringtone mode
User information	Current activity
	Absolute location

Table 2.18 Collected data

values are always available on all smartphones and do not require any extra energy consumption to be retrieved. Moreover, "Notification information" represents the data contained in a notification, such as a) the type of the notification (e.g., message, email, ... etc), b) the generating service (e.g., Telegram, Whatsapp, Snapchat, ...), and c) the relationship between the sender and the receiver of the notification. The sender-receiver relationship is expressed by one of the following values: "family", when the sender and receiver are relatives, "friend", when the sender is a friend of the receiver, and "work", when the sender works with the receiver. This information is asked to the user at the first application installation for only the most important persons present in her contact list. On Android smartphones, every time a notification is received, a broadcast message with all available information related to the just received notification is sent to all registered apps. Consequently, the exposed means of information are acquired without consuming extra energy. Furthermore, "Device state" indicates the values related to battery level, charging status (i.e., charging/non charging) and selected ringtone mode (i.e., silence, vibration, and sound). As declared in the Google documentation[40] these values do not require a lot of energy if they are acquired with the right frequency. In addition, 2 more pieces of information about user are collected: user current activity and user current absolute location.

[40]http://developer.android.com/training/monitoring-device-state/battery-monitoring.html, last visited on March 22, 2017

User current activity is retrieved using the Google Activity Recognition service: the activities are detected by periodically waking up the device and reading short bursts of sensor data. The Google documentation[41] reports that the activity recognition process “only makes use of low power sensors in order to keep the power usage to a minimum”.

Apart from these features, for meaningful places estimation, user current absolute location is collected. It can be acquired through 2 different methods: using the GPS module or using network positioning techniques. In both cases extra energy is needed, but within this study the absolute location is needed only to establish label (meaningful places) for the training of the Machine Learning algorithm, so, it is acquired only whenever the user enables GPS and/or network positioning services for her purposes (for example, when she uses mapping services).

Only one portion of data acquired through the Android application presented in the Collectors Section were useful for the purpose of estimating user presence in meaningful location: only the data collected by the 14 most collaborative users (that turned on the GPS for more than 15 days and did not deactivate any service used by the application) was used. Consequently, data collected by 14 users were used and 20 days (the ones in which all the 14 users were active) were considered.

To preserve user privacy, all collected information were anonymous: all sensible pieces of information present in the shown list were anonymized using an hash function.

2.9.1.3 Meaningful places estimation

The second phase in evaluating the proposed method is related to “Meaningful places estimation” step: the 2 most attended meaningful places for each user were identified using the unsupervised machine learning algorithm known as K-means algorithm: as declared by Zhou et al. [49] it is one of the most known and used partitioning clustering algorithm for detecting user meaningful places.

As samples of collected data, Figure 2.23 and Figure 2.24 show the absolute locations stored for 2 different users. The round points represent all the unique absolute locations stored for the single user, instead the cross points are the centers of

[41] https://developers.google.com/android/reference/com/google/android/gms/location/ActivityRecognitionApi#public-methods, last visited on March 22, 2017

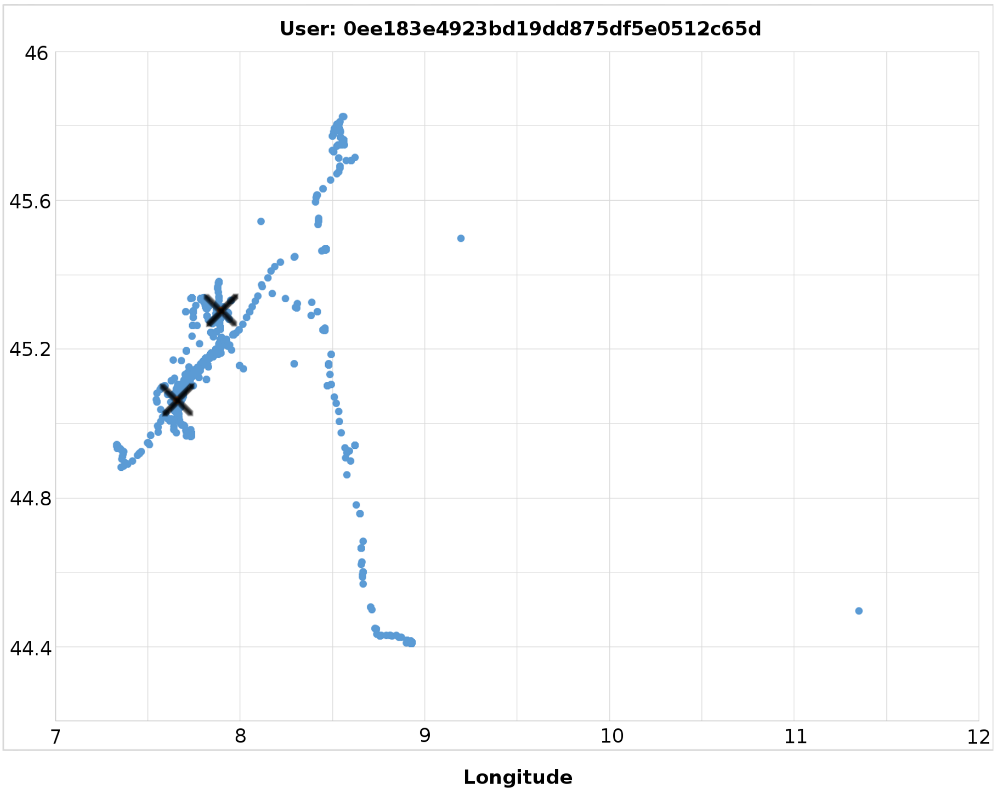


Fig. 2.23 Absolute locations recorded for user 5

the 2 meaningful places identified by the K-means algorithm. As can be seen, in both cases the 2 meaningful places identified by the algorithm are actually representative of most of the recorded absolute locations. The user location was within 1 km from either cluster centers in 64.26% of the samples.

2.9.1.4 Features selection and Pre-processing

Another phase in evaluating the proposed method is related to the “Features selection & Pre-processing” step. It refers to every operation performed on the data in order to remove useless information from the collected items and encode data for better performance of the classification algorithm. In this work, three main operations were performed during this phase:

- date and time information were further splitted in 4 main items: time (seconds since midnight), day (number of days since the 1st day of the month), month, year;

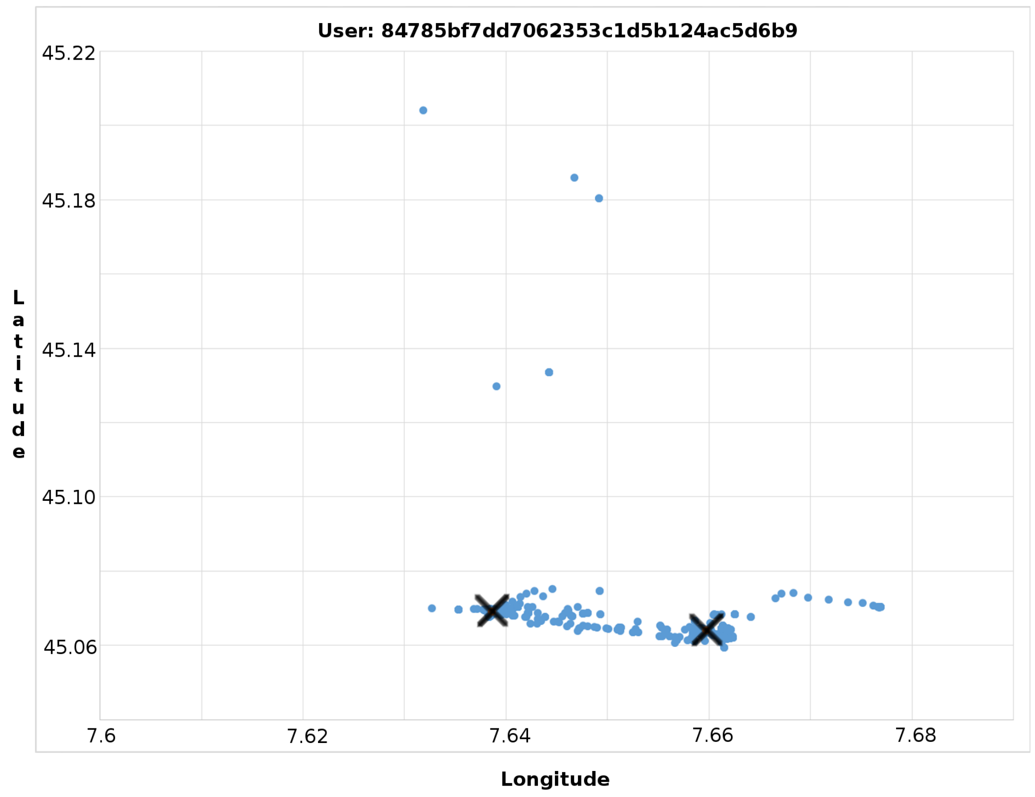


Fig. 2.24 Absolute locations recorded for user 11

- time, day, month, and day of the week fields were splitted in aggregated features; in order to consider the periodicity of time (i.e., 23.59 is near 00.01), in fact, we exctracted 2 different values for time instead of only one: $sin(\frac{2 \cdot \pi \cdot time}{24 \cdot 60 \cdot 60})$ and $cos(\frac{2 \cdot \pi \cdot time}{24 \cdot 60 \cdot 60})$, where the denominator represents the seconds contained in a day and "time" represents the number of seconds since midnight. The same split was performed for the month, the day and the day of the week but substituting the components of the formula with the right values.
- considering that experiments were performed in the same year, the "year" information was removed from the selected features.

2.9.1.5 Cross validation

After filtering data collected trough the Android application presented in Collectors Section (with Features selection, Pre-processing and Meaningful places estimation), an off-line evaluation of our method was performed through a 10-fold cross validation process over the collected dataset.

In the presented work, the Weka [46] workbench for Machine Learning was used for all the experiments and all the evaluations were done using the k-fold cross validation method with k set to 10.

The dataset used for experiments performed in this study is composed of 27142 samples (a mean of 1507 ± 970 samples per user) labeled with user meaningful places; user presence in a meaningful place was estimated every time a new notification is received, consequently each dataset sample represents data stored at the moment of a notification reception.

For each experiment 3 measures are reported:

- Accuracy, that represents the percentage of correct estimations.
- Precision, that is a measure of result relevancy.
- Recall, that is a measure of how many truly relevant results are returned.

The precision and recall values reported in this section are calculated as weighted values of repeated single class classification. In our work, in fact, we have 2 classes (location 1 and location 2) data label, so Weka repeats the prediction 2 times: the first time it estimates precision and recall predicting class 1, instead at the second one it estimates precision and recall predicting class 2. Then, it calculates a weighted value considering the class size.

14 users were involved in the study and, considering that each user has her own habits and attends different meaningful places, the model was evaluated separately for each user. Then, a mean value is calculated for each measure over all 14 users.

2.9.1.6 Results evaluation

Collected data (Table 2.18) is converted into features suitable for the classification algorithm, as shown in Table 2.19, that contains further details on feature representation.

The first considered feature category (A) is related to time information. As already discussed, all the data were collected at each notification reception, con-

Feature ID	**Feature**	**Type**
A	Time	INTEGER
	Month	INTEGER (1-12)
	Day	INTEGER (1-31)
	Day of the week	Class(Monday, ...)
B	Type	Class(msg, email, ...)
	Generating service	STRING (e.g., Telegram)
	Sender-receiver relationship	Class(family, friend, work)
C	Battery level	INTEGER
	Charging state	Class(charging/ not charging)
D	Ringtone mode	Class(silence, vibration, sound)
E	Current activity	Class(in vehicle, on bicycle, on foot, running, still, tilting, unknown, and walking)

Table 2.19 Legend of considered features

sequently, this data represents the time, date and day of the week at which the notification was received.

The second feature category (B) is related to notification information: it represents the type of the notification, the service that generated it and the relationship among the owner of the smartphone and the sender of the message.

Furthermore, features related to device status (battery information (C) and selected ringtone mode (D)) were considered in addition to the revealed user activity. The activity (E) is acquired through the Google Activity Recognition API and can assume one of the following values: a) in vehicle, b) on bicycle, c) on foot, d) running, e) still, f) tilting, g) unknown, and h) walking.

Exp.	**A**	**B**	**C**	**D**	**E**	**Accuracy**	**Accuracy St. Dev.**	**Precision**	**Precision St. Dev.**	**Recall**	**Recall St. Dev.**
1	Y	Y	Y	Y	Y	89.40	8.27	89.04	8.63	89.40	8.27
2	Y	Y	Y	Y	-	89.31	8.50	88.83	9.13	89.31	8.50
3	Y	Y	Y	-	Y	89.01	8.46	88.59	8.95	89.01	8.46
4	Y	Y	Y	-	-	89.12	8.51	88.64	9.13	89.12	8.51
5	Y	Y	-	Y	Y	89.17	8.41	88.66	8.85	89.17	8.41
6	Y	Y	-	Y	-	89.27	8.23	88.66	8.91	89.27	8.23
7	Y	Y	-	-	Y	88.73	8.39	88.15	8.85	88.73	8.39
8	Y	Y	-	-	-	88.96	8.39	88.35	9.03	88.96	8.39
9	Y	-	Y	Y	Y	**90.19**	7.81	**89.96**	7.90	**90.19**	7.81
10	Y	-	Y	Y	-	90.11	7.94	89.83	8.13	90.11	7.94
11	Y	-	Y	-	Y	89.52	8.11	89.18	8.38	89.52	8.11
12	Y	-	Y	-	-	89.61	8.01	89.23	8.40	89.61	8.02
13	Y	-	-	Y	Y	89.82	8.39	89.34	8.76	89.82	8.39
14	Y	-	-	Y	-	90.08	7.83	89.72	8.05	90.09	7.83
15	Y	-	-	-	Y	89.38	8.37	88.67	9.06	89.38	8.37
16	Y	-	-	-	-	89.90	7.78	89.51	8.01	89.90	7.78

Table 2.20 First part of experimental results (Accuracy, Precision and Recall) - all values are percentages

Finally, the target class used as label for the Machine Learning algorithm training is the "Meaningful Place" that can assume one of the 2 values "locationClass1" and "locationClass2".

Aiming at studying the importance of each selected features 31 different experiments were performed: the Decision Tree model was cross-validated on the collected dataset with different combinations of the presented features.

Table 2.20 and Table 2.21 show the list of performed experiments: in every row we report the features selected for each experiment and the obtained accuracy, prediction, and recall mean values with corresponding standard deviation. The legend of feature IDs is contained in Table 2.19.

Exp.	**A**	**B**	**C**	**D**	**E**	**Accura-cy**	**Accura-cy St. Dev.**	**Precisi-on**	**Precisi-on St. Dev.**	**Recall**	**Recall St. Dev.**
17	-	Y	Y	Y	Y	86.92	8.56	86.65	8.76	86.92	8.56
18	-	Y	Y	Y	-	86.94	8.02	86.71	8.20	86.94	8.02
19	-	Y	Y	-	Y	86.03	8.69	85.80	8.85	86.03	8.69
20	-	Y	Y	-	-	86.01	8.29	85.81	8.45	86.01	8.29
21	-	Y	-	Y	Y	78.89	10.91	78.76	10.98	78.89	10.91
22	-	Y	-	Y	-	77.89	11.10	77.75	11.12	77.89	11.10
23	-	Y	-	-	Y	76.67	11.89	76.56	11.99	76.67	11.89
24	-	Y	-	-	-	75.50	12.36	75.59	12.20	75.50	12.36
25	-	-	Y	Y	Y	87.57	8.01	87.31	8.22	87.57	8.01
26	-	-	Y	Y	-	**87.68**	7.48	**87.49**	7.64	**87.68**	7.48
27	-	-	Y	-	Y	87.26	8.23	86.98	8.51	87.26	8.23
28	-	-	Y	-	-	86.98	8.08	86.74	8.36	86.98	8.08
29	-	-	-	Y	Y	75.59	11.69	75.56	11.62	75.59	11.69
30	-	-	-	Y	-	73.44	11.88	73.10	12.29	73.44	11.88
31	-	-	-	-	Y	72.92	12.49	72.24	12.95	72.93	12.49

Table 2.21 Second part of experimental results (Accuracy, Precision and Recall) - all values are percentages

The first 16 experiments reported in Table 2.20 show the importance of timing information in predicting presence in meaningful places, especially when coupled with other device or user-related features. The importance of this feature was expected to be strictly related to user habits and results demonstrate an obtained accuracy between 88% and 91%.

Moreover, results show that the “Current activity” (E), the only feature that consumes extra energy, is not necessary: when it is considered, the accuracy changes only of less than 1%. Consequently, it can be removed from the used features bringing the proposed method to a zero-energy method.

2.9.1.7 Related works

User location estimation has been subject of extensive studies in the literature, nonetheless, location information is usually retrieved through energy expensive methods like GPS or network positioning techniques. Therefore, considering that the utility of smartphones is limited by their battery life (as declared by Metri et al. [50]), there is a growing interest in the problem of reducing power consumption without giving up the benefits of using location information. Most of related works focus on how to reduce the frequency or decrease the duration of GPS usage: Kjaergaard et al. [51], for example, propose a system that, based on the estimation and prediction of system conditions and mobility, schedules position updates to both minimize energy consumption and optimize robustness. The developed system calculates the optimal plan for power-on and power-off times of device sensors and peripherals, such as the GPS module. Furthermore, Xu et al. [52] proposed a hybrid method for semantic location recognition, which combines k-NN (the k-nearest neighbors algorithm is one of the most used machine learning algorithms for pattern recognition) and multiple decision tree models to effectively recognize the location both in outdoor and indoor environments. To reduce battery consumption, they use a decision tree model to check if the user is moving and only when the user stays in a place for a significant time period, the recognition procedure is performed. A similar approach was used by Ryoo et al. [53]: they designed a geo-fencing framework able to determine when users check in or out of a specific area in an energy efficient way, so that appropriate Location Based Services (LBS) can be triggered. In addition, their study is based on the observation that users usually move from one place to another and then stay at that place for a while. This observation is supported by Klepeis et al. [54] and is the basis of several other works related to energy consumption reduction in smartphone location prediction. As demonstrated by Klepeis, in fact, people spend approximately 87% of their daily time in enclosed buildings, so it is possible to identify some user meaningful places in which user spends most of their daily time. This property is also used by Chon et al. [47], that proposed a system able to reduce smartphone battery consumption switching from a high-level to a low-level sensing mode; the two different modes differ in the sensors used to evaluate location. The system assumes that, when a user stays at a place for a certain period of time, she is in a "meaningful place". Consequently, when these places are recognized, the system saves a location signature (i.e., internet connectivity, visiting time, residence-time, and Wi-Fi signature) for future prediction. Therefore, whenever a user enters a place, at first the system tries to identify the location using location signature (this step is called low-level sensing step) and then, only if it is not able to identify the place, it activates the more energy hungry high-level sensing.

The same property is used in the work described in this section of the thesis, but aiming at using new methods and sources of information to estimate users' meaningful places: we present a technique for applying Machine Learning classification algorithms based on Decision Trees on data available on the smartphone that do not require extra energy to be retrieved.

New methods for obtaining a smartphone location estimation have already been presented in some works found in the literature. Paek et al. [55], for example, propose a system that leverages Cell-ID transitions and a history of GPS readings obtained within a cell to provide an accurate estimation of user current location. The paper demonstrates that the proposed system achieves reasonable accuracy while keeping a low energy overhead. Furthermore, Garbe et al. [56] present a system for mobile devices able to determine user's location when neither GPS nor network positioning information is available. The proposed system uses information coming from 3 different sensors: a gyroscope, a magnetometer that measures the Earth's magnetic field and a barometer that estimates the user's altitude based on air-pressure readings.

Moreover, another useful work is the one presented by Qin et al [57]: the main issue that authors address is how predictable individuals are in their mobility. They use the raw cell ID timestamps combined with the smartphone location (i.e., its coordinates) to calculate and estimate patterns (i.e., the days are classified for each person into regular, personal patterns) and "life" entropy, used then for meaningful places (that they call "persons important locations") estimations.

2.9.1.8 Conclusions and future work

This section proposed a new energy efficient method to estimate user presence in a meaningful place. In this study we presented results obtained from the analysis of data acquired through an Android application installed on 14 user smartphones for 20 days. We demonstrated that it is possible to use a method that applies a Machine Learning algorithm based on Decision Trees, to predict the user presence in a meaningful place by collecting and analyzing: a) user activity, b) information from received notifications (receipt time, generating service, sender-receiver relationship), and c) device status (battery level and ringtone mode). A 10-fold cross-validation process was used to evaluate the method estimating user presence in a meaningful place every time a notification is received.

In order to identify the best combination of features for our purposes 31 experiments were performed. Results demonstrate that the most important features among the considered ones are related to time information. In fact, when timing features are considered, the best obtained accuracy value (percentage of correct predictions) is 89.40% (standard deviation: 8.27%) with a precision of 89.04% and a recall of 89.40%.

In the future we plan to repeat experiments with a larger dataset: more users will be involved in the study for a larger observation period. Moreover, more Machine Learning algorithms will be tested for estimation in order to determine if it is possible to obtain better results with different algorithms.

2.10 Discussion and Conclusion

In this chapter, we presented the SNS system, a modular architecture that uses machine learning algorithms to manage incoming notifications. According to context awareness and user habits, the presented architectural system is able to decide who should receive the incoming notification, on which device(s), in which moment and with which mode(s) (e.g., vibration, sound, light signal). A simplified version of the different modules that constitute the architecture has been prototyped and a preliminary deployment session has been performed with the aim of collecting real data and perform some validating experiments.

The data obtained from the preliminary deployment sessions and the results obtained from the evaluation of the machine learning approach applied to notifications can foster future development in such fields, as already done in the proposed inspired extra experiment that mainly present methods to estimate user presence in meaningful locations.

Future works will extend the implemented prototypes to allow the collection of more accurate and complete set of real data. In addition, the Decision maker prototype will be enhanced and other machine learning algorithms will be evaluated for letting it make all the four designed decisions: a) who should receive an incoming notification; b) what is the best moment to show the notification to the chosen user(s); c) on which device(s) the chosen user(s) should receive the notification; d) which is the best way to notify the incoming notification (e.g., vibration, light, sound).

Furthermore, the possibility of using different algorithms after other ones will be evaluated. Finally, a more complete implementation of the whole SNS system will be implemented and will be evaluated by an adequate number of people aiming at obtaining structured information about the accuracy of predictions made by it and a more completed dataset for the system training.

Chapter 3

The XDN (Cross-device Notification) Framework[1]

As already depicted in the Introduction Section of this thesis, during the last decade, the presence of notifications in people's routines has grown. Although people are becoming accustomed to them, the usefulness and the importance of each notification often depends on various factors that can influence the reaction and the disruption of recipients. The information contained in notifications, or the device(s) on which they are presented, are only a few examples of factors that can influence the user reaction and disruption caused by notifications.

As already discussed, two main approaches were identified in this thesis to improve user experience with notifications. In this chapter, the approach that acts at the design level is presented. Specifically, while the previous chapter presents a work that provides a solution that directly influences users' behaviors with respect to notifications, the present chapter proposes a framework that allows developers to define their strategies to let their software, then, influence users' behaviors with respect to notifications. In addition, developers are encouraged to exploit the advantages provided by the cross-device approach to design notifications and their distribution strategies.

[1]Part of the work described in this chapter has been previously published in [15]. Specifically, the architecture of the proposed solution was revised to introduce new, more detailed, components. In addition, as an original contribution of this thesis, with the aim of evaluating the feasibility and the effectiveness of the framework, the preliminary prototype of the XDN framework was tested by 12 volunteers and the results are presented in the last section of this chapter.

The chapter presents the architecture of the proposed solution together with the preliminary test session performed with 12 volunteers (developers) and the results obtained from such tests.

3.1 Introduction

As declared by Seshadri et al. [58] about notifications, even though providing individuals with relevant information is an essential element in facilitating their activities, the challenge for developers is “to provide information in a desired manner notwithstanding vast differences in individuals’ information and delivery preferences”. Consequently, developers should create applications that “deliver timely and personalized information on whatever suitable device is available and accessible” [58].

In addition, due to the growing importance of the Internet of Things (IoT) in the notification context and the increasing spread of IoT devices, as declared by Weber [8], “the ongoing wave of smart devices makes it possible to reach the user through multiple devices at once, amplifying the effects of notifications”. Nonetheless, in recent years, this possibility has not been fully exploited by developers: most existing applications mainly duplicate the same notification on all the available supported devices.

While it is necessary to personalize notifications according to their importance, the developed notification strategies (i.e., algorithms for distributing notifications) should exploit the possibility of reaching the same user through different devices. As already proposed [59, 60], a possible contribution entails the adoption of a cross-device approach [61] to notifications, a growing trend of the last decades that consists of extending an application user experience across multiple devices. By applying the cross-device approach to notifications, in fact, developers could distribute different “signals” related to the same notification on different devices. For instance, a warning sound could be sent to the smart Hi-Fi, while vibration could be activated on the personal smartphone and the notification content could be shown on the smart TV.

Developers should therefore focus on both personalizing notifications to differentiate the presentation of important and unimportant information, and designing cross-device notifications strategies responsible for informing users without causing too much disruption and involving mobile and IoT devices. Nonetheless, as highlighted by related works, developers are not yet supported in implementing solutions that respect both needs in all their aspects.

This chapter presents XDN (Cross Device Notification), a framework that allows developers to create, by scripting, cross-device notifications. Inspired by the Chord framework [62][2], a framework for cross-device interactions between mobile devices, and with the aim of contributing to its future development (if the project will be revived), XDN assists developers in a) designing personalized notifications, and b) designing, implementing and testing notification strategies able to distribute notifications among mobile and IoT devices using a cross-device approach. The framework architecture is composed of four main parts:

- the XDN library, that implements a set of high-level APIs to let developers a) handle incoming notifications and their properties (i.e., content, receipt date/-time, generator, and icon), b) select devices to be involved in the notification distribution, and c) separately perform different actions on selected devices (e.g., play a sound, activate the vibration, or show the notification content);
- the XDN runtime environment, that is a service supposed to be run on a server and that is able to a) accept notifications from IoT/mobile devices, b) process them by running the deployed notification strategy(ies) and c) distribute processed notifications to available devices through its dispatcher module. It is also responsible for d) registering new devices and e) storing device status updates;
- the XDN GUI that provides a) an editor for allowing developers implement and debug their notification strategies and b) a simulator that shows how the predefined devices will behave when a new notification arrives by simulating the runtime environment;
- the XDN IoT/mobile library that will be imported in every application/service that uses XDN to generate and distribute notifications. It allows developers to a) generate notifications compatible with the format supported by XDN, and b) send the generated notifications to the XDN runtime environment. In addition, it is also able to autonomously c) receive commands from the XDN runtime environment, and d) execute them.

[2]https://github.com/google/chord, last visited on July 10, 2017, last updated on December 5, 2015

A first prototype of the XDN framework, that implements only the XDN library and the XDN GUI was developed in the Node.js[3] framework. Consequently, the methods, classes and objects provided by the XDN library are provided for JavaScript applications. Furthermore, the feasibility of the framework and the fulfillment of all the requirements presented in Section 3.3 were verified through a test session in which 10 real users were asked to design and develop two different scenarios in a limited time.

The main contribution of this chapter is a) a new cross-device proposal for customizing and distributing notifications among ad-hoc networks of end-user mobile and IoT devices; b) the XDN framework, that provides the APIs, the GUI, the runtime enviroment, and the IoT/mobile library for developers that would develop their algorithms respecting the cross-device approach; c) two different scenarios used within a test session in which 10 users to verify the feasibility of the framework and the fulfillment of all the requirements.

3.2 Background and Motivation

To better appreciate the contribution of our work with respect to the literature, the following Section 3.2.1 presents a motivational example inspired by an existing one.

3.2.1 Scenario: Messaging Application

With the aim of helping readers in understanding the situations in which the XDN framework would help developers, this section presents a sample notification strategy that will be also used as a running example through the remainder of the chapter. Three different specific scenarios, inspired by the motivating scenarios presented by Campbell et al. [60] will be used to extract the final general strategy.

Ashley is a developer, working on a messaging application that should differently disrupt users depending on their current activity and location. She identified three sample situations that should be handled by her algorithms, considering some hypothetical users of her application.

[3]https://nodejs.org/, last visited on January 15, 2017

- The user is at his business location, is wearing a smartwatch and is using his desktop computer. Unfortunately, his smartphone went out due to low battery and, suddenly, an important personal message arrives. The user should be warned about that message as soon as possible by showing it on the desktop computer that he is using and by making the smartwatch vibrate. In addition, no other persons should be disrupted (e.g., by playing a warning sound for more than one time on other devices).
- The user is at home and his smart TV is playing a movie. The user owns one smartphone. It is supposed that the user is relaxing, so the system should not cause any disruption to him. In this situation, a notification should be shown on the smart TV, while the smartphone vibration should be also activated just because the user could not be actually in front of her TV.
- The user is driving. The car is equipped with an IoT Hi-fi system, i.e., an Hi-fi that is connected to the Internet to provide different services. One of these services consists of loudly reading incoming notifications. While the user is driving, he receives two notifications on his smartphone: one is from his daughter who is waiting her in front of the school, while the other one is an advertisement. The system should a) generate a textual notification on user smartphone for both received messages, b) activate the vibration on the smartphone to inform the user about the daughter's message, only, and c) read the daughter's message on the car Hi-fi.

The above situations are not exhaustive, for this reason, in all the situations that are not covered by the described ones, the smartphone will be used as the preferred device: the notification content will be shown on the smartphone display, the vibration will be activated and the smartphone led will be made blink.

Ashley identified the following notification strategy expressed in term of general rules that she will implement. In the following list the adjective "available" will be used to indicate a device that is turned on and connected to the internet.

- If a message is received from a service that inserted the words "@important" and "@personal" in the notification metadata[4] and the user is working (i.e., no

[4]ït is supposed that the documentation of the used operating system reports that these words are usually inserted in the notification metadata to distinguish incoming notifications

smart Tv and/or Hi-Fi and/or car Hi-Fi is available, while a PC is available), the message should be notified by showing the notification content on all the available PCs and tablets and the vibration should be activated only once and on all the available devices that support it;

- If a message is received when the user is relaxing in her house (i.e., one smart TV or one Hi-Fi system is available and is playing music/video), the notification content should be shown on all the available TVs, the vibration should be activated on all the available devices that support it and the notification content will be also shown on all the available smartphones (in case the user cannot see it when it is shown on the TV);
- If a message is received when the user is driving (i.e., a car Hi-Fi system is available and playing music), the system should distinguish useful and useless messages. It is supposed that the app generating the message inserts the word "@unimportant" in the notification metadata, when needed. If the message is not important, only a textual notification will be shown on the user smartphone. Otherwise, the following actions will be performed: a) a textual notification will be generated on all the available smartphones, b) the vibration will be activated on all the available smartphones, c) the notification content will be read on the car Hi-fi system.
- In all the cases that are not covered by the previous ones, a) a textual notification will be shown on all the available smartphones, b) the vibration will be activated on all the available smartphones, c) the smartphones led will be made blink.

In addition, Ashley must be able to develop the just described rules without moving from her workplace (e.g., to test the situation in which the user is driving) and to test them without actually owning the involved devices. Consequently, she needs a tool to: a) design and develop the notification strategies able to manage the described situations, b) simulate the behavior of devices that she does not actually own by simulating the arrival of notifications in all the presented realistic situations.

The following subsections will discuss the drawbacks and problems of the related works with respect to the presented notification strategy.

3.2.2 Frameworks for customizing notifications and/or designing notification strategies

As already discussed in previous sections, due to the increasing presence of notifications in people's lives, user frustration and/or disruption caused by notifications have been examined in the literature [10–13]. Some works also proposed solutions to mitigate the negative effects that notifications cause on user attention [63, 32, 31, 30, 62]. Almost all these solutions rely on the customization of notification strategies at the distribution level (i.e., notifications are intercepted and then systems decide if, when, and how showing them). Instead, in our opinion, another approach is also feasible: the customization of notification strategies at the design level (i.e., notification strategies are designed with the aim of reducing user disruption).

Specifically, nowadays, developers misuse notifications and generate/show them on almost every available device and in every moment of the day, with no uniform mechanism for considering the importance of the notification and/or the user availability. For this reason, we believe that developers should design their algorithms so that notifications could be distributed according to a well-designed and accurately tested strategy. Different works [32, 62, 64], for example, as a result of their experiments report some issues about user attention and/or preferences that developers could consider in implementing their notification strategies with the aim of reducing the overall user disruption. Likewise, some commercial systems have already improved notification distribution strategies at the design level: Slack[5], a cloud-based team collaboration tool that generates a huge number of notifications a day, for instance, already implemented a notification strategy[6], shown in Figure 3.1, that "smartly" distributes notifications to a cleverly chosen subset of available devices.

However, the complexity of such techniques could cause different problems in managing and testing them without a dedicated tool.

The community of developers needs a framework that allows developers to design, develop and finally test their own algorithms to generate customized notifications and distribute them among available mobile and IoT devices using a cross-device approach.

[5]https://slack.com/, last visited on July 10, 2017

[6]https://slack.engineering/reducing-slacks-memory-footprint-4480fec7e8eb#8c3c, last visited on July 10, 2017

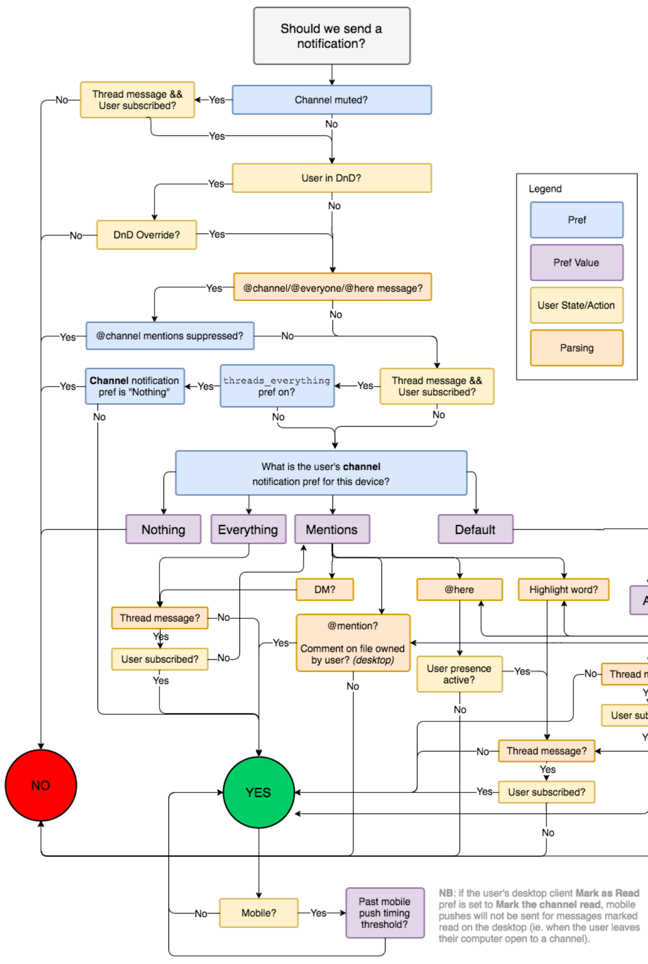

Should we send a notification?
Channel muted?
Thread message && User subscribed?
User in DnD?
DnD Override?
@channel/@everyone/@here message?
@channel mentions suppressed?
Channel notification pref is "Nothing"
threads_everything pref on?
Thread message && User subscribed?
What is the user's channel notification pref for this device?
Nothing
Everything
Mentions
Default
What is the user's global notification pref for this device?
DM?
@here
Highlight word?
All
Mentions
DMs (mobile)
Highlight Words (mobile)
Never
Thread message?
User subscribed?
@mention?
Comment on file owned by user? (desktop)
User presence active?
Thread message?
User subscribed?
DM?
@mention?
Highlight word?
Thread message?
User subscribed?
NO
YES
Mobile?
Past mobile push timing threshold?
Legend
Pref
Pref Value
User State/Action
Parsing
Yes
No
NB: if the user's desktop client Mark as Read pref is set to Mark the channel read, mobile pushes will not be sent for messages marked read on the desktop (ie. when the user leaves their computer open to a channel).

Fig. 3.1 Notification strategy implemented by the Slack team

To the best of our knowledge, no work provides such a similar support, consequently, the analysis of related works is split in two different topics: Section 3.2.2.1 treats the development of applications, services, or systems able to generate and/or distribute cross-device notifications among available mobile and IoT devices, while Section 3.2.2.2 treats the development of a framework for developers for customizing notifications and/or their distribution.

Both subsections discuss the drawbacks and problems of the related works with respect to the notification strategy presented in the previous Section 3.2.1. In addition, Table 3.2 summarizes all the presented related works with respect to the exposed features and services reported in Table 3.1.

Feature	Description
F1	Design and develop notification strategies
F2	Support IoT devices as receipt of notifications
F3	Simulate selected device(s) (not owned)
F4	Simulate more than one device at a time
F5	Multi-device
F6	Multi-platform
F7	GUI for developers
F8	Support cross-device approach

Table 3.1 Summary of features provided by related works

Related work	F1	F2	F3	F4	F5	F6	F7	F8
meSchHub [65]	Yes	Yes	No	No	Yes	Yes	No	No
Seshadri et al. solution [58]	Yes	No	No	No	No	No	Yes	No
Apple Framework	Yes	Partially	Yes	No	No	No	Yes	No
Google Framework	Yes	Partially	Yes	No	No	No	Yes	No
Apache Cordova Framework	Yes	Partially	Yes	No	No	Yes	Yes	No
Panelrama [66]	No	Partially	No	Yes	Yes	Yes	No	Yes
XDStudio [67]	No	Partially	Yes	Yes	Yes	Yes	Yes	Yes
Connichiwa [68]	No	Partially	Yes	Yes	Yes	Yes	No	Yes
Notification Platform [59]	No	No	No	No	Yes	Yes	No	No
Campbell et al. solution [60]	No	No	No	No	Yes	Yes	No	No
Chord [62]	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
XD-Testing [69]	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 3.2 Summary of related works' features

3.2.2.1 Frameworks and tools for developers for managing notifications

As already mentioned, notifications have been extensively examined in the literature, but only a few works provide developers with aids for customizing notifications and/or their distribution. The main characteristics that distinguish almost every presented work from XDN are related to the absence of a simulator or the impossibility of supporting cross-device interactions.

An interesting work specifically proposed for designing and deploying notification strategies is presented by Kubitza et al. [65]. They describe an infrastructure for homes and offices that allows designers and web developers to design and deploy context sensitive notification strategies using arbitrary things and smart home products connected to their meSchHub gateway, such as TVs, tablets, projections, lamps, speakers and many more. In the proposed infrastructure, the notifications received on the user smartphone are sent to the meSchHub gateway that forwards them to available IoT devices according to interaction scripts pre-defined by developers. One gateway can be set up per each smart space (e.g., office, flat, house) and, in the proposed architecture only a single smartphone can be connected to a single gateway. When a smartphone leaves the range of a certain smart space (e.g., out of home WiFi) and comes into the range of another known smart space (e.g., office WiFi) the app automatically discovers the local meSchHub gateway and starts working with the interaction scripts that are defined for that space. According to the description of the system, the meSchHub system lacks a simulator: Ashley, the developer of the "Messaging Application" scenario, in fact, would not be allowed to test her notification strategies on devices that she does not actually own. In addition, the meSchHub system is limited to the environment in which the gateway is installed and in which the user is currently present.

Another interesting related work is the patent proposed by Seshadri et al. [58] that presents a system and a methodology to facilitate the development, debug, and deployment of a notification platform application. The whole system is based on an Application Definition File (ADF) which describes all the components that interact to perform notification services, wherein the components are often in various languages and formats. In their proposal, a visual user interface is provided to facilitate efficient design, debug, management and deployment of an ADF and related configuration file (and other related files) when developing notification applications. Developers are, in fact, directed through visual diagrams and processes leading to the development and ultimately deployment of a notification application. However, the proposed system does not provide any simulator to test the designed algorithms. In addition, according to the description of the system, it does not support any cross-device interaction (e.g., activate the vibration on a device and make the led blink on another device at the same time).

Moreover, in addition to the solutions found in the literature, developers are supported by all the existing commercial frameworks and/or APIs used in the development of mobile and/or IoT applications. The following analysis will focus on the three most used frameworks/tools for mobile devices development, but the reported observations also apply to other existing solutions that provide the same or similar features: a) the frameworks proposed by Apple for managing notifications within iOS, tvOS, watchOS or macOS applications, b) the framework proposed by Google for the customization and distribution of notifications within Android applications, c) the framework proposed by Apache Cordova (formerly PhoneGap) for the distribution of notifications accross multiple platforms (i.e., Amazon Fire OS, Android, BlackBerry 10, Browser, Firefox OS, iOS, Tizen, Windows Phone 7 and 8, Windows). All of them provide support for managing a single device at a time, thus it is not possible to develop cross-device strategies nor to simulate more than one device at a time.

Apple provides two different frameworks for customizing notification in the development of mobile applications: the User Notifications framework that mainly handles the content of the notifications, and the User Notifications UI framework, available only in the iOS SDK (i.e., currently it is not usable for the development of tvOS, watchOS, and macOS apps), that mainly handles the appearance of the notifications. Even though, in some cases, the notification appearance can be customized, developers can only choose how the user should be notified by selecting one of the following options for delivering the notification: a) an onscreen alert or banner, b) a badge on the app's icon, c) a sound that accompanies an alert, banner, or badge. On the other hand, Google provides a similar solution for Android devices but allows more customization[7]. In the development of Android applications, in fact, it is possible to personalize: a) the notification content, b) the notification icon, and c) the notification priority. In addition, in the upcoming "O" release of Android, the following customizations will be added: a) specify the notification channel at which the notification belongs, b) remove or update a snoozed notification, c) set a timeout for creating a notification after a specified time, d) set and enable a background color for a notification, e) add some style to the notification, and f) know the user reaction to a notification (i.e., dismissed or not).

Finally, Apache Cordova (formerly PhoneGap)[8], one of the most used frameworks for building cross-platform applications, provides the "cordova-plugin-dialogs" plugin[9] for customizing notifications. The supported methods mainly provide the following customizations: a) specify the message of the notification, b) specify the title of the notification, c) specify the function that should be invoked when the user

[7]https://developer.android.com/guide/topics/ui/notifiers/notifications.html, last visited on March 22, 2017

[8]https://cordova.apache.org/, last visited on May 02, 2017

[9]https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-dialogs/, last visited on May 02, 2017

presses on the notification, d) specify the function that should be invoked when the user presses on a button present within the notification. In addition, it is possible to play a sound when the notification arrives on all the supported platforms except Firefox OS and Windows (it is possible only on Windows 8).

The provided description of such solutions highlights that it is already possible to personalize notifications, even though some limitations can be revealed with respect to the customization we are proposing (e.g., it is not possible to activate only the vibration). However, the main disadvantage of such solutions is that it is not possible to develop notification strategies implementing the cross-device approach. Likewise, some other drawbacks emerge from this brief analysis. First of all, the first two presented commercial frameworks are strictly limited to the development of applications for specific devices and/or platforms. Thus, if a notification strategy is developed for a platform (e.g., Android) it is not easy, then, to export and use it in other platforms (e.g., iOS). In addition, the available developer tools (e.g., Android Studio[10] for the development of Android applications, XCode[11] for the development of iOS, tvOS, watchOS and macOS applications) and the PhoneGap Developer App do not allow the simultaneous simulation of different devices to test the designed notification strategy on all target devices.

3.2.2.2 Cross-device interactions

A growing trend in ICT is the use of the cross-device approach in the development of applications [61]. It consists of extending an application user experience across multiple devices. The development of cross-device applications has already been applied in different domains to solve different problems, but only a limited number of works are devoted to the generation and/or the cross-device distribution of notifications among different devices.

Three of the most popular frameworks that facilitate the creation or the conversion of cross-device applications do not, in fact, support notifications at all. One of them is presented by Yang et al. [66]: Panelrama. It is a framework that facilitates the creation and the easy conversion of existing web applications to enable cross-device interaction. In brief, an application is decomposed in a set of panels, that are distributed among all available devices and properties and statuses are synchronized.

[10]https://developer.android.com/studio/index.html, last visited on March 22, 2017
[11]https://developer.apple.com/xcode/, last visited on March 22, 2017

The main advantage of Panelrama is that applications can be tested, and also deployed on every available device that is equipped with a browser. However, a) it is not possible to test the designed application on not owned devices, b) until the moment of writing, neither the simulation of IoT devices nor the management of IoT-dedicated properties and/or hardware accessories (e.g., the vibrator) are supported by existing browsers and, consequently, by Panelrama. Moreover, as already mentioned, according to the provided description, Panelrama is not designed to manage or customize notifications. A possible solution to the absence of a support for notifications could resort to the use of Web Push Notifications inside the browsers by using standard Web Notifications[12], proprietary APIs (like Mozilla Notification API[13]) or public libraries (like Roost [14] or Push.js[15]), but, in this way the notifications could not be received when the browser is closed.

The second and the third frameworks able to facilitate the development of cross-device applications are XDStudio [67] and Connichiwa [68], but, unfortunately, they lack support for notifications. XDStudio provides a GUI builder designed to support interactive development of cross-device web interfaces. The most important advantage with respect to Panelrama is the presence of a simulation tool that allows developers to design algorithms for a multi-device environment and test them on devices that are not owned by the developer. Connichiwa [68], instead, is a versatile framework for creating web applications across multiple devices. It is based on an event-based mechanism to imperatively show and hide content on devices upon connection or disconnection of other devices [69]. Both of them are designed to mainly support the development of user interfaces and do not handle notifications, too.

Consequently, the considerations reported for Panelrama could be extended for XDStudio and Connichiwa, too: a) notifications could not be used when the browser is closed, and b) XDStudio and Connichiwa are not able to manage IoT-dedicated properties and/or hardware accessories (e.g., a led or a vibrator).

Two works that are specifically designed for notifications are described in the following. Although the authors declare that their proposals are designed to support

[12]https://www.w3.org/TR/notifications/, last visited on March 22, 2017
[13]https://developer.mozilla.org/en-US/docs/Web/API/notification, last visited on March 22, 2017
[14]https://goroost.com/, last visited on March 22, 2017
[15]https://nickersoft.github.io/push.js/, last visited on March 22, 2017

cross-device notifications, the description does not provide any evidence of it. In fact, to the best of our knowledge, both works provide only support for multi-device notifications, but do not support developers in distributing different “signals” related to the same notification on different devices.

First, Horvitz et al. [59] present the Notification Platform, a cross-device messaging system that modulates the flow of messages from multiple sources to other devices by performing ongoing decision analysis. Specifically, it balances the costs of disruption with the value of information from multiple message sources. The system employs a probabilistic model of attention and executes ongoing decision analyses about ideal alerting, fidelity, and routing. The main drawback of this work is that the developer would not be allowed to modify in any way the predicted model. In fact, it is not possible to customize the notification distribution strategies. In addition, according to the provided description, they apply the cross-device approach to mobile devices only, without considering IoT devices.

Second, Campbell et al. [60] present some techniques for cross-device notifications. They start from the consideration that a notification could be missed for any reason (e.g., because the smartphone is in a bag) and, even though other devices are in use, the user remains unaware about it. Consequently, they propose a solution that involves available devices to allow user to be warned about arriving notifications. They provide a solution to handle the receipt of a notification from a device different from the one that received it but does not allow to customize the notification or its distribution. Consequently, with respect to the “Messaging Application” scenario, in which Ashley would like to personally design the strategy for distributing the notification and/or its properties, this solution lack of the possibility to do both actions. In addition the proposed technique does not support IoT devices.

More interestingly, Chi et al. [62] present Chord (previously known as “Weave”), a framework for developers to create cross-device wearable interaction by scripting. According to the supplied description, it provides a set of high-level APIs, based on JavaScript, for developers to easily distribute UI output and combine sensing events and user input across mobile and wearable devices. It also contributes an integrated authoring environment for developers to program and test cross-device behaviors, and when ready, deploy these behaviors to its runtime environment on users’ ad-hoc network of mobile devices. The main characteristic of Chord is that it is designed to assist the implementation of cross-device interactions. Thus, it does not treat at all notifications and/or features that are essential to adequately reduce user disruption caused by them. First of all, in fact, in Chord only two characteristics are provided as output of the available devices: the display and the speaker (with a corresponding show and play method), while some other features like vibration and light and the corresponding methods (i.e., vibrate, on, off, blink, ... etc) are not provided. In addition, it is not possible to manage notification properties like the notification content, the notification arrival date, the notification generator.

Finally, Husmann et al. [69] present XD-Testing, a library for testing web-based cross-device applications quite similar to Chord. In facts, it provides a similar data structure and similar related methods, but, in addition, provides a mechanism for specifying formal tests and automating their execution. Furthermore, XD-Testing introduces concepts for implicitly and explicitly selecting devices that are needed, respectively to address specific devices or to dynamically choose the appropriate devices for each command that is executed on them. The absence of an explicit support for notifications is the main drawback of XD-Testing.

In this work we decided to develop XDN to be a) inspired by Chord, mainly due to its cross-device nature, its ease of use and its linear data structure, b) compatible with Chord to be integrated in its future extensions.

3.3 Requirements

This section presents the design-time requirements devised for the XDN framework: they were designed by analyzing both the shortcomings and the successes of existing solutions discussed in the *Background and Motivation* section of this chapter.

(R1) API for selecting available devices and executing specific actions

The two most complete cross-device solutions found in literature are the Chord framework and the XD-Testing library. Even though they do not provide support to manage notifications, they introduce some useful methods and functions that allow developers to easily select available devices and then execute specific actions (e.g., activate the screen). This model was really appreciated by developers that tested their solutions, consequently it will be used as inspiration for providing similar methods and classes in XDN. Specifically, XDN will provide APIs to:

- select available mobile/IoT devices based on their specific properties and status;
- perform specific actions on selected devices (e.g., turn on a LED or make it blink);
- manage notification properties (e.g., arrival time, notification content).

As already done in Chord [62], XDN will be implemented following the "object-oriented event-driven paradigm". Specifically, inspired by Chord and popular JavaScript libraries such as jQuery[16], XDN will provide a high-level abstraction for programmers to manipulate notifications and device's properties and actions.

(R2) GUI

One of the most important features provided within existing works is a Graphic User Interface (GUI) that support developers in a) implementing their algorithms/code, b) visualize the current status of available devices, c) visualize/simulate the behavior of the designed algorithms/strategies. Its presence in almost all the discussed related works, in fact, confirms that it is essential in development tools like the one we are going to develop. Therefore, XDN should provide a GUI to a) develop notification strategies and, then, b) monitor the behavior of the simulated devices performed when one or more new notification is delivered to users.

(R3) Notification strategies simulation on not owned devices

A limitation of some existing works (e.g., [65]) able to simultaneously distribute notifications across multiple devices is the absence of a simulator for testing notification strategies on not-owned devices. Considering that the number of IoT and mobile devices is growing day by day, in fact, developers could not be asked to own every existing device to test their algorithms on them. Therefore, XDN should support the simulation of developed notification strategies on devices that are not actually owned by developers.

(R4) Simulation of the designed notification strategies

The example reported in the *Background and Motivation* section, i.e., the notification strategy developed by the Slack platform (that is able to distribute notifications across multiple devices) demonstrates that the complexity of notification strategies

[16]https://jquery.com/, last visited on January 15, 2017

is growing. In addition, by introducing the cross-device approach, they will become more complex in the next years and a graphic simulator able to simultaneously show the behavior of more than one device could be really appreciated by developers.

Unfortunately, the existing solutions that allow developers to define notification strategies (i.e., the work of Seshadri et al. [58], or commercial solutions like Android Studio or Apple XCode) do not provide any graphic simulator for visualizing such behaviors. Consequently, XDN should provide a simulator able to show the behavior of more than one device performed when a new notification arrives.

(R5) Multi-platform

Although different existing works propose some solutions to let developers design cross-device applications and algorithms, they have two main drawbacks related to the distribution of such applications/algorithms. Specifically, at first, most of the discussed solutions propose web-based applications able to distribute notifications within a browser. However, this solution has a main shortcoming: users cannot be reached when the device is not in use and when a browser is not running. Second, the solutions that support the distribution of notifications outside the browser (e.g., development of Android app through Android Studio) mainly allow the creation of software that is strictly limited to some specific devices (e.g., Android devices). Instead, XDN should allow the development of notification strategies that could be easily exported on different platforms and run without using additional tools like a browser.

(R6) Support for IoT devices

Nowadays, with the increasing spread of the IoT, new smart devices and appliances are developed everyday with the ability not only to generate but also to show notifications. Consequently, in addition to existing mobile devices, the XDN framework should support existing IoT devices to let developers test their notification strategies.

(R7) JavaScript

As already discussed, the two most complete cross-device existing solutions found in literature are the Chord framework and the XD-Testing library. Due to its ease of use and its linear data structure, and considering that we are going to use the same design pattern (as declared in R1), XDN will be designed to be compatible with Chord. Consequently, XDN will provide JavaScript methods and classes to let

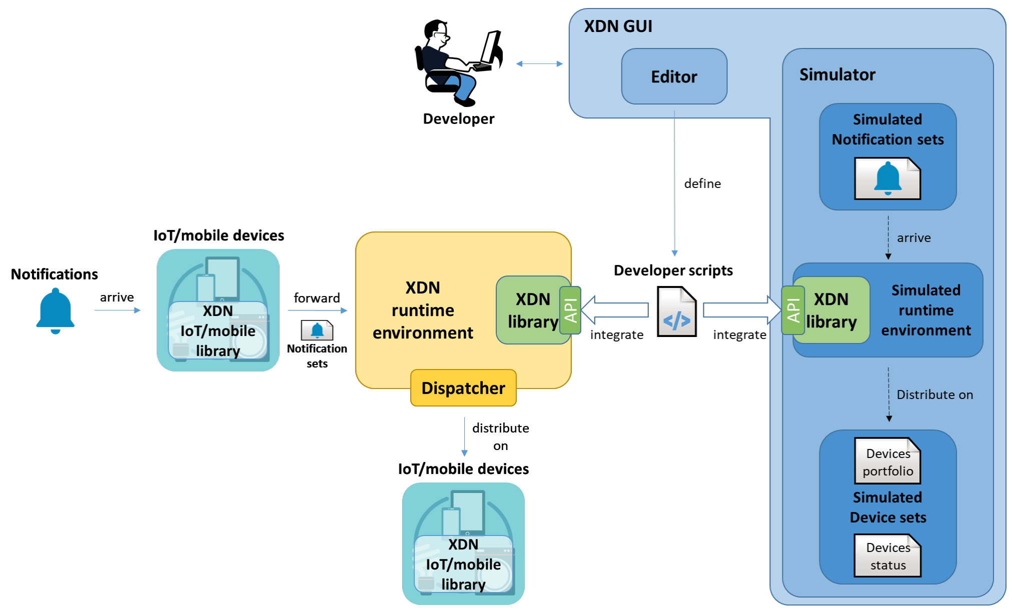


Fig. 3.2 Architecture of the XDN framework

developers implement their strategies using the JavaScript programming language and let them reuse the implemented code in future version of Chord.

3.4 Framework

The architecture of the XDN framework was designed to satisfy all the reported requirements. XDN was designed to be compatible with the architecture of the Chord framework [62], therefore the syntax of the Chord APIs has guided the design of the XDN APIs.

As shown in Figure 3.2, the architecture of the framework is composed of four main blocks: the **XDN library**, the **XDN GUI**, the **XDN runtime environment** and the **XDN IoT/mobile library**.

The **XDN library** implements a set of high-level APIs to perform the following actions: a) handle incoming notifications and their properties (i.e., content, receipt date/time, generator, and icon), b) select devices to be involved in the notification distribution, and c) separately perform different actions on selected devices (e.g., play a sound, activate the vibration, or show the notification content). It is mainly designed to be integrated in the XDN runtime environment module and its simulator.

The details of the XDN library will be discussed in the following *XDN library* sub-section.

The **XDN runtime environment** is a service that is run on a server and is always available (i.e., turned on and connected). In fact, it is able to a) accept real notifications, b) process them by running the deployed script and c) distribute processed notifications to available devices through its internal dispatcher module. In addition, it is responsible for d) registering new devices and e) storing device status updates. The details of the XDN runtime environment will be discussed in the following *XDN runtime environment* sub-section.

Moreover, developers will mainly interact with the **XDN GUI** module. It provides a) an editor to implement and test notification strategies and b) a simulator to test them. Specifically, the editor provides support to write JavaScript notification strategies or load them from an existing developer script. Meanwhile, the simulator is able to simulate the arrival of notifications by permitting to:

- define or load a device set (better explained in the following description) to be used during the simulations;
- define or load a list of notifications used during the simulation as arriving notifications;
- run the simulation, to actually simulate the arrival of notifications and visualize the behavior (only related to the arrival of a notification) of all the loaded devices.

As can be observed in the architecture shown in Figure 3.2, the simulator relies on *simulated notification sets* and *simulated device sets*. The *simulated notification sets* block represents the list of notifications that will be "sent" during the simulation. While, the *simulated device sets* block represents the group of devices that will be simulated: their behavior corresponding to the arrival of a new notification will be shown during the simulation. A single simulated device set is composed of a) some devices' properties that are static and represent the device capabilities, and b) the corresponding current devices' statuses, that represent the current values for each device's property. The details of the **XDN GUI** module will be discussed in the following *XDN GUI* sub-section.

Finally, the **XDN IoT/mobile library** is the module that is supposed to be imported in every application/service that uses XDN to generate and distribute notifications. It allows developers to a) generate notifications compatible with the JSON format supported by XDN, b) send the generated notifications to the XDN runtime environment. In addition, it is also able to autonomously c) receive commands from the XDN runtime environment, and d) execute them.

The details of the presented modules will be discussed in the following subsections, where the "Messaging Application" scenario will be used as a running example.

3.4.1 XDN GUI

Figure 3.3 shows a screenshot of the XDN GUI. It is composed of four different main parts:

- the script editor (letter A), that allows developers to develop their notification strategies in JavaScript. It integrates graphic warning signals to inform the developer of any syntax error in the implemented code;
- the device set column (letter B), that shows the behavior of each loaded device during the simulation of a notification arrival. It also allows to save and load sets of devices to use during the simulation;
- the notification set column (letter C), that shows the list of notifications that will be "sent" during the simulation. It also provides two buttons to respectively save and load a list of notifications;
- the log (letter D), that shows developers any error and warning generated during the simulation and, also, shows the list of all the actions performed on the available devices. Thus, it actually acts as a storyboard of the behaviors of the devices loaded in the device set.

To better clarify the role of each presented component, the "Messaging Application" will be used as a running example to explain the actions that a developer should perform to use the XDN GUI to design, implement and test her notification strategies.

As already presented in the "Messaging Application" example, Ashley is developing a messaging application that should differently notify users depending on their current activity and location (acquired by analyzing the available devices and their statuses). As a first step, Ashley connects to the web-based XDN GUI and starts from the selection of the devices a user will supposedly own and use. She can choose between two options: select one of the existing set of devices provided in the XDN GUI or create a new custom set of devices. She decides to choose the first option and she selects and loads the predefined set of devices (i.e., a smartphone, a smartwatch, a smart TV, a PC, and a car Hi-Fi). Loaded devices appear in column "Device Set" (B).

Now it is time to write the code she will load on the runtime environment. She can write it by using the editor present in the GUI, located in the left column (A).

The details of the algorithm will be discussed in the following Section 3.4.2.

After creating the algorithm, Ashley tests it using the simulator. To do it she has to perform three actions: a) load the initial status of the available devices by using the buttons present at the top of column B, b) load the notifications set that will be sent during the simulation, by using the buttons present at the top of column C and, c) run the simulation using the *Run code* button located at the top of the editor (A).

When all these actions are performed, Ashley will see all the updated statuses in column B and will be able to analyze the list of all the performed actions in the log section (D).

3.4.2 XDN library

The XDN library provides a set of APIs that clearly define methods to customize and distribute cross-device notifications by reducing repetitive code. They are based on two main objects: **xdn.notification** and **xdn.device**.

The **xdn.notification** object is responsible for all the interactions with the notifications and their properties. The class diagram reported in Figure 3.4 shows its structure. As shown in the diagram, the **xdn.notification** object provides only the *onNotification* function. It allows developers to attach an event handler to the arrival of a notification. In fact, whenever a new notification arrives, the handler function is called. Thus, developers can implement their notification strategies inside a function

Property	Type
dateTime	date and time at which the notification was received
content	the content of the notification
generator	meta-data associated with the notification, inserted by the generating app
icon	the icon associated to the notification, if available

Table 3.3 Notification properties

that should be, then, passed as input parameter of the *onNotification* method. Notification properties can be accessed by the *NotificationObject* object accepted as input parameter of the handler function. It represents a single notification and contains the properties reported in Table 3.3. As an example, the code for logging the content of the received notification is:

```
xdn.notification.onNotification(function(myNotification) {
  var content = myNotification.content;
  xdn.log(content);
})
```

The **xdn.device** object implements all the classes, sub-objects and methods needed by developers to interact with available devices. The class diagram reported in Figure 3.5 shows its structure. It is only an abstract object that exposes the methods implemented by the *DeviceSelection* sub-object. The provided methods can be used to a) search and filter across a set of devices, and b) perform actions on one or more selected devices. The list of all the registered devices is stored inside the private *devices* object with their properties, sub-properties and statuses. Each property is assigned to each device depending on its nature: every time a new device is registered to the system (through the XDN runtime environment), only the properties that are supported are specified. As an example, if Ashley is using a smartphone, it has for sure a display and a speaker, consequently the properties *display* and *speaker* will be available.

All possible properties and sub-properties of a device are listed in Table 3.4 in the *Property* and *Sub-Property* columns. The lower part of the table lists the status properties and sub-properties.

Before interacting with devices it is necessary to select them by using one of the methods listed below. According to the specified criteria they return a *DeviceSelection* object containing the selected *devices*.

Property	SubProperty/[value]	Note
name	-	unique identifier
deviceType	[smartphone, smartwatch, bracelet, smartLight, tablet, PC, fridge, hi-fi, smartTv, carHi-fi, smartToothbrush]	-
display	size, privacy: [high, normal, low], touch: [true, false],	privacy = indicate if the message could be read from only the recipient or also by others
speaker	privacy: [high, normal, low]	privacy = indicate if the message could be read from only the recipient or also by others
light	colors, intensity: [true, false], frequency: [true, false], blink: [true, false]	it indicates both the light of a bulb or the light of a led (e.g., the status led of a smartphone)
vibration	[true, false]	-
os	-	-
isAvailable	[true, false]	it is set to true if the device is turned on and connected
display	currentStatus: [on, locked, off]	-
speaker	currentVolume, currentStatus: [playing, off]	-
light	currentStatus: [on, off, blinking], currentIntensity, currentColor	-
vibration	currentStatus	-

Table 3.4 Device properties (above) and statuses (below)

- *xdn.device.select*: selects the devices that satisfy the specified criteria (e.g., the code `xdn.device.select('deviceType=="smartwatch"')` returns a list of all the smartwatches, that are all the devices with the “deviceType” property set to “smartwatch”);
- *xdn.device.selectWith*: selects the devices that has the specified property (the property is set as input parameter);
- *xdn.device.selectAll*: selects all the devices registered to the system;

Enabling property	**Action**
display	.show
speaker	.play, .ring
light	.on, .off, .changeColor, .changeIntensity, .blink
vibration	.vibrate

Table 3.5 Device actions

- *xdn.device.getDeviceByName*: selects a single device, the one with the specified name (*deviceName* property set to the specified name)
- *xdn.device.not*: selects the all the devices that do not satisfy the specified criteria. This method is used to exclude one or more devices, e.g., if Ashley wants to exclude all the smartwatches: `xdn.device.not('deviceType == "smartwatch"')`.

These methods can be concatenated with a "fluent" programming pattern, so that only the *Device* objects satisfying all the specified criteria are selected. Thus, if Ashley wants to select all the available smartphones, she can use the following code:

```
xdn.device.select('deviceType=="smartphone"').select('
    isAvailable==true')
```

Table 3.5 summarizes all the actions that it is possible to perform on each selected device. It is important to note that the available methods can be used only if the corresponding property (reported in the column "Enabling property") is defined for the specific device. For example, it is possible to turn the light on with the action *.on* only if a light property is specified for the device. Otherwise, a warning message is shown in the log section.

Finally, Listing 3.1 shows the final complete algorithm written by Ashley to implement the designed behavior of her "Messaging Application" system.

```
xdn.notification.onNotification(function(myNotification) {
  var content = myNotification.content;
  var generator = myNotification.generator;
  var selectedSmartTVPlaying = xdn.device.select('deviceType=="
      smartTv"').select('isAvailable==true').selectWith('speaker
      ').select('speaker.currentStatus=="playing"');
  var selectedHiFiPlaying = xdn.device.select('deviceType=="hi-
      fi"').select('isAvailable==true').selectWith('speaker').
      select('speaker.currentStatus=="playing"');
```

```
var selectedCarHiFiPlaying = xdn.device.select('deviceType=="
    carHi-fi"').select('isAvailable==true').selectWith('
    speaker').select('speaker.currentStatus=="playing"');
if (generator.toLowerCase().includes('@important') &&
    generator.toLowerCase().includes('@personal')) {
  var selectedSmartTV = xdn.device.select('deviceType=="
      smartTv"').select('isAvailable==true');
  var selectedHiFi = xdn.device.select('deviceType=="hi-fi"')
      .select('isAvailable==true');
  var selectedCarHiFi = xdn.device.select('deviceType=="carHi
      -fi"').select('isAvailable==true');
  var selectedPC = xdn.device.select('deviceType=="PC"').
      select('isAvailable==true');
    xdn.device.select('deviceType=="smartphone"').show(
        content).vibrate();
    xdn.device.select('deviceType=="carHi-fi"').play(content,
         '50');
  if (selectedSmartTV.length() < 1 && selectedHiFi.length() <
       1 && selectedCarHiFi.length() < 1 && selectedPC.length
      () >= 1) {
       selectedPC.show(content);
       var selectedTablet = xdn.device.select('deviceType=="
           tablet"').select('isAvailable==true');
       selectedTablet.show(content);
  }
}
else if (selectedSmartTVPlaying.length() >= 1 ||
    selectedHiFiPlaying.length() >= 1 ) {
  selectedSmartTVPlaying.show(content);
  xdn.device.select('deviceType=="smartphone"').show(content)
      ;
  xdn.device.selectAll().selectWith('vibration').vibrate();
}
else if (selectedCarHiFiPlaying.length() >= 1) {
  if (generator.toLowerCase().includes('@unimportant')) {
    xdn.device.select('deviceType=="smartphone"').show(
        content);
  }
  else
  {
    xdn.device.select('deviceType=="smartphone"').show(
        content).vibrate();
      xdn.device.select('deviceType=="carHi-fi"').play(content,
          '50');
    }
  }
  else
  {
      xdn.device.select('deviceType=="smartphone"').show(
         content).vibrate().blink(1/60);
  }
});
```

Listing 3.1 Messaging Application system script

3.4.3 XDN runtime environment

The **XDN runtime environment** is a service that is run on a server. Developers can configure its behavior through the notification strategies' scripts, so that every arriving notification will be treated respecting the defining strategies. The XDN runtime environment interacts with the XDN library to:

- accept *registration* requests from a device. Every time a new device should be registered to the system, it has to contact the runtime environment providing information about its properties and statuses (Table 3.4);
- accept *update* requests arriving from registered devices to inform the system about the changes in their statuses;
- accept new notifications generated by the registered devices. The notification will be accepted in the JSON format, with the fields listed in Table 3.3;
- customize and dispatch the notifications according to the notification strategy defined by the developer.

3.4.4 XDN IoT/mobile library

The **XDN IoT/mobile library** is the module that developers should integrate in their applications and services for IoT/mobile device to:

- generate notifications as presented in the previous sub-section;

- send the generated notifications to the XDN runtime environment;
- receive commands from the XDN runtime environment, in JSON format containing the properties of Table 3.5;
- execute the received commands.

Considering that this module should be integrated in almost every existing IoT/mobile application, it is supposed to be developed in different programming languages and for different platforms.

3.5 Implementation

To demonstrate the feasibility of the proposed XDN framework and the fulfillment of all the requirements (Section 3.3), a prototypical version of the XDN framework was implemented. The implemented XDN prototype, currently, integrates the XDN GUI and the XDN library, only.

3.5.1 Implementation details

The implemented prototype consists of a backend server and a frontend user interface. The backend server is a web application based on Node.js and jQuery[17] and was packaged by NW.js[18] to become a native application. It serves different purposes: a) it maintains and exposes the XDN library with its methods and classes, b) it hosts the predefined *simulated notification sets* and the *predefined simulated device sets*, c) it provides the methods needed to load and/or store developer-defined scripts, device sets and notifications, and d) it provides the methods used by the GUI to simulate the arrival of a notification.

A developer interacts with the frontend application which implements the XDN GUI. The GUI is composed of an editor, and a simulator which includes a log and two modules for loading the device sets and the notifications. The frontend application was built upon ace[19], an embeddable code editor. When a developer edits the script

[17]https://jquery.com/, last visited on January 15, 2017
[18]https://nwjs.io/, last visited on January 15, 2017
[19]https://ace.c9.io/, last visited on January 15, 2017

algorithm, after pressing the “Run code” button, the frontend app automatically stores the last version of the code and then interprets the developer’s code executing the specified operations. In the current prototypical implementation, the simulated runtime environment adopts JavaScript *eval()* function to interpret developer’s code.

Finally, the *xdn* class was implemented as a JavaScript object literal providing the notification, device and log objects.

3.6 Evaluation

XDN has been evaluated with a small group of volunteers to demonstrate the fulfillment of all the requirements (Section 3.3) and collect a feedback about both the XDN framework APIs and the XDN GUI. Twelve participants used XDN for one hour, performing two tasks each. The first task regarded the alteration of a notification strategy already implemented by the author of this work implementing a simple scenario reported in the following *Scenario 1: Smart Fridge* subsection. While the second task regarded the design and development of the notification strategy implementing the simple scenario reported in the following *Scenario 2: messaging application* subsection.

The feedback received by participants allowed qualitative analysis and helped the authors in identifying strong and weak points of the framework.

3.6.1 Tests’ goal

The main objective of this test session is to demonstrate that the XDN framework is actually useful for developers in developing new applications able to generate notifications. Specifically, the tests aimed at answering the following research questions:

- Demonstrate the perceived usefulness of XDN:
  - Do developers find XDN useful in developing algorithms that implement the proposed use cases?
  - Would developers use XDN in developing algorithms that can customize and/or manage notifications?

  - Would developers use XDN in developing the applications/services they usually develop?

- Demonstrate that the XDN APIs are easy to use from developers:
  - Is it easy to use the methods, classes and objects provided by XDN to implement the proposed use cases?

- Verify that the XDN GUI is easily interpretable and usable:
  - Is it easy to understand how to use the various components available in the XDN GUI to implement the proposed use cases?
  - Is it easy to implement the proposed use cases? Is it possible to implement them in the required time?
  - Are the status log and the simulation useful for identifying problems/errors/incompatibilities? And to correct them?

3.6.2 Scenarios for tests

As already discussed, during the tests, participants were asked to perform two different tasks. The first one regarded the alteration of a notification strategy already implemented by the author of this work and implementing the following Scenario 1: Smart Fridge simple scenario. While the second task regarded the design and development of the notification strategy implementing the following Scenario 2: messaging application simple scenario.

The following subsections report all the details of the designed Scenarios.

3.6.2.1 Scenario 1: Smart Fridge

This scenario presents the details of the notification strategy that the involved participants were asked to modify in the first task.

Katie is implementing a smart fridge that can send a notification every time the expiration date of a contained product is reached. Consequently, the system should be able to warn the user in two situations:

- when 3 days are left to the expiration date;

- when 1 day is left to the expiration date.

In the first circumstance, the notification should be shown on the screen of all the fridge that has a display; while in the second case, the notification should be shown on all the available smartphones. In addition, in both cases, if no fridge has a display, the screens of all the available smartphones will be used instead.

3.6.2.2 Scenario 2: messaging application

While the first scenario was designed as simple as possible to allow developers understand how the XDN framework works and what it can offer, this second scenario was designed as representative of a real application that developers could develop.

Ashley is a software developer who is working on a messaging application. The application should be able to generate notifications that disturb the user in different ways depending on its current position and its current activity. Ashley identified the following notification strategies. In the following list the adjective "available" will be used to indicate a device that is switched on and connected to the internet.

- If an important message [20] is received and the user is working [21], then the message should be notified with the following actions:
  - the notification content should be shown on all available PCs and tablets;
  - the vibration should be activated on all devices available and equipped with a vibrator.
- In all other cases:
  - the notification content should be shown only on the screen of all available smartphones;
  - the vibration should be activated on all available smartphones;
  - the led of all smartphones should be made blink with a blue light.

[20] Important notifications can be identified by the presence of the “@important” strings immediately after the application name in the “generator” field of the notification

[21] The user is supposed to be working if the following conditions are met: a) none of the following devices is available: smart TV and/or Hi-Fi and/or car Hi-Fi; b) at least one PC is available

3.6.3 Test Deployment

A total of 12 volunteers were asked to participate in this evaluation, with ages ranging from 24 to 34 years, 11 males and 1 female. All the participants were selected among the master's and Ph.D. students of the "Politecnico di Torino" studing "Control and Computer Engineering". As a first evaluation of the framework aiming at validating the approach and evaluate the effectiveness of the proposed solution, only people that had already experience in developing using the JavaScript programming language were selected: 5 people over 12 declared that they had among 5 months and 1 year of experience in developing JavaScript applications; 2 people over 12 declared that they had already developed JavaScript application for 2 years; and the other 5 people developed JavaScript application since more than 3 years. This requirement assures that the evaluation is not affected by problems in learning the JavaScript programming language or the concepts on which it is based.

In addition, we asked participants a self evaluation, over a scale that goes from 1 to 5, about their experience with JavaScript. The average value obtained from answers is 3. Specifically, 3 people declared that at least 2 years had passed since their last experience in programming using JavaScript.

Furthermore, we asked participants if they had already developed any application or service able to generate notifications and 10 of the 12 involved people declared that they had.

Finally, we asked if they had never used a JavaScript framework and only 4 people declared that they had: 3 of them had already used AngularJS[22], while only one Node.js.

For each participant, after a short introduction to the test and the collection of demographic data, the XDN framework was presented through the documentation reported in Section 3.4. Then, the first task was presented.

Participants never met during the evaluation: every volunteer independently participated to the test.

[22]https://angularjs.org/, last visited on January 15, 2017

3.6.3.1 Task 1

With the aim of introducing the framework and presenting an example of the typical usage of XDN, the first scenario was introduced to volunteers and, consequently, the first task was presented. It required to modify the notification strategy already developed by the author of this work for the Scenario 1. It is reported in Listing 3.2. The modification should have been done in less than 5 minutes aiming at:

- substituting the fridges' display with the screens of all the available TVs;
- substituting the smartphones with the tablets.

```
xdn.notification.onNotification(function(myNotification) {
    var content = myNotification.content;
    var generator = myNotification.generator;
    if (generator.toLowerCase().includes('@3daysleft')) {
        var fridges = xdn.device.select('deviceType=="fridge"')
           .select('isAvailable==true').selectWith('display');
        if (fridges.length() >= 1)
        {
            fridges.show(content);
        }
        else
        {
            var smartphones = xdn.device.select('deviceType=="
               smartphone"').select('isAvailable==true').
               selectWith('display');
            smartphones.show(content);
        }
    }
    else
    {
        if (generator.toLowerCase().includes('1daysleft'))
        {
            var fridges = xdn.device.select('deviceType=="
               fridge"').select('isAvailable==true').selectWith
               ('display');
            fridges.show(content);
            var smartphoneS = xdn.device.select('deviceType=="
               smartphone"').select('isAvailable==true').
               selectWith('display');
            smartphoneS.show(content);
            }
            else
            {
                log("do␣nothing");
            }
        }
});
```

Listing 3.2 Smart Fridge

3.6.3.2 Task 2

After presenting the second scenario, then, the second task was also presented to participants. It required to develop the notification strategy that would satisfy the requirements of the second Scenario by starting from an empty editor. However, the code developed during task 1 was provided in a printed version to all participants and a copy of the documentation discussed in Section 3.4 was also provided.

Each participant had at maximum 25 minutes to perform the task and no suggestions were allowed from the organizers.

3.6.4 Results

The results of the experiments revealed that 7 participants over 12 were able to complete all the tasks in the required time: all of them had already developed JavaScript applications for more than 2 years. However, 2 of the remaining people were facing only one last problem in coding when the time finished: one of them was having trouble in understanding that the "blink" command also turned the light on, while the other did not notice that the tablets were selected instead of the smartphones.

Finally, at the end of the experiments, participants were asked to answer to 11 questions to understand what was the general satisfaction in using the XDN framework and an open discussion was also promoted to collect extra feedback.

Table 3.6 summarizes the survey that was proposed to each participant with the corresponding average results. As can be observed in the reported table and as also declared by the participants in the open discussion, all the participants

Question	**Average result (1-5)**
Was it easy to develop the proposed strategies	3.75
How useful the EDITOR was to identify errors?	3.25
How useful the EDITOR was to fix errors?	3.25
How useful the LOG was to identify errors?	4.25
How useful the LOG was to fix errors?	4.00
How useful the whole XDN framework was to develop the 2nd scenario?	4.16
How understandable the XDN API was?	4.08
How complete the XDN API was?	4.00
How useful the EDITOR was?	3.64
How useful the API was?	4.42
How useful the Simulator was?	4.67
How useful the DEBUG function (that was not present) would be?	3.92
Will you use the XDN framework for implementing a notification strategy in the future?	4.25
Will you use the XDN framework in your usual development?	2.00

Table 3.6 XDN: final survey proposed to users

expressed a positive impression in the usage of the XDN framework and in its usefulness. Almost all participants declared that the documentation is clear and the API actually simplifies the selection of devices and their properties. In addition, they appreciated the simulator that allowed to see the behavior of the devices when a new notification arrived and the way by which XDN allows to distribute notifications among heterogeneous devices. However, some issues were also identified and some suggestions were consequently provided. The following list presented all the suggestions/issues provided by the volunteers.

- It would be useful to directly access devices' property (e.g., "device.light.status").
- The "select" instruction could be improved by setting a default behavior that always selects available devices; in fact, the most common usage of the select is to select available devices, thus it would be better to select all the available devices by default and specify other options in other cases.
- It would be necessary to improve the selection options: it would be nice to specify a list of conditions within a single select instead of only one.
- The quotes used to specify conditions are, sometimes, annoying and introduce errors.
- Simulated devices should be organized in a better way: if a lot of devices will be involved, it will be difficult to see the behavior of all of them.
- The DEBUG function is sometimes necessary to identify errors.
- The LOG would allow different level of logging (e.g., WARNING, ERROR, etc.).
- The EDITOR had some problems with suggestions: it is not possible to obtain suggestions when a variable is used to contain all the selected devices.
- The select working with variables (e.g., select("isAvailable == true") should also work without "== true").
- It would be nice to integrate such framework in existing IDE.

3.7 Discussion

This section presents a preliminary analysis of the actual contribution brought by the XDN framework in supporting developers in designing algorithms able to customize and distribute cross-device notifications.

Starting from the observation, supported by related works, about the lack of support for the customization of notifications and the development of cross-device notification strategies, the emphasis of the analysis has been put on the actual advantages and challenges that the XDN framework could provide to developers. This analysis was conducted through the test session discussed in the previous Section.

3.7.1 Successes

In Section 3.3, seven different requirements were presented.

The first one regarded the need of APIs to support developers in: a) selecting available mobile/IoT devices based on their specific properties and status, b) performing specific actions on selected devices (e.g., turn on a LED or make it blink), and c) manage notification properties (e.g., arrival time, notification content). We can claim that the methods and classes provided by the XDN library in conjunction with the services provided by the XDN runtime environment satisfy R1.

In addition, R2 regarded the need of a graphical interface. The designed GUI is able to satisfy all the low-level described requirements: helping developers in implementing the notifications strategies' scripts, b) visualizing the current status of available devices, c) visualizing/simulating the behavior of the designed algorithms/strategies.

Furthermore, the presence of an editor, a log, a module to load devices' portfolios and statuses, a module to load notifications and the possibility of running the simulation satisfy R3 and R4 requirements regarding the necessity of a device simulator that is also able to simulate more than one devices at the same time.

The use of JavaScript, one of the most commonly used programming language, satisfies R7 and makes XDN compatible with Chord. Moreover, the use of Javascript merged with the presence of the XDN runtime environment and the presence of the

XDN IoT/mobile library also satisfies R5 that regarded the possibility of distributing notification strategies among multiple platforms. The spread of such programming language, in fact, guarantees an high compatibility with most of existing mobile and IoT systems and motivate developers in using the XDN framework for their applications.

Finally, R6, that regarded the support for IoT devices, is satisfied by the two modules that allow to load and manage IoT devices and their properties.

3.7.2 Challenges

Thanks to the volunteers' feedbacks reported in Section 3.6.4, some challenges were identified while using this preliminary implementation of the XDN framework for developing the notification strategies described in the scenarios.

The first challenge regards the XDN GUI and specifically, the editor. Even though, in this initial implementation, the editor recognizes JavaScript syntax errors, it is not yet possible to recognize errors in using the XDN API. For example, if the developer writes "DeviceType" instead of "deviceType" the editor does not recognize it as an error. Although the log helps developers identifying such errors, such a feature would be really appreciated by developers.

In addition, another issue regards the XDN APIs: a lot of suggestions reported by testers regarded the way used to select devices. These suggestions reveals that the way proposed by Chord and, then, reproduced in XDN to select devices has some unresolved problems mainly related to the usage of strings to select properties. This behavior has some limitations mainly related to the impossibility of using variables and/or complex conditions defined by developers. Consequently, a better solution should be proposed to address this issues.

Furthermore, another challenge regards the interaction with the available devices. With the current version of the XDN framework it is possible to customize notification strategies and graphically simulate the behavior of the devices when a new notification arrives. However, the current implementation does not treat the possibility of performing specific actions due to the reaction of users to notifications. So, for example, it is not possible to capture the user disruption to notifications and, in consequence, implement some extra code to perform new actions after a predefined time from the one at which, for instance, the user swiped a notification away.

In addition, the absence of a DEBUG was recognized by almost every participant as an important issue to be solved in the next developments.

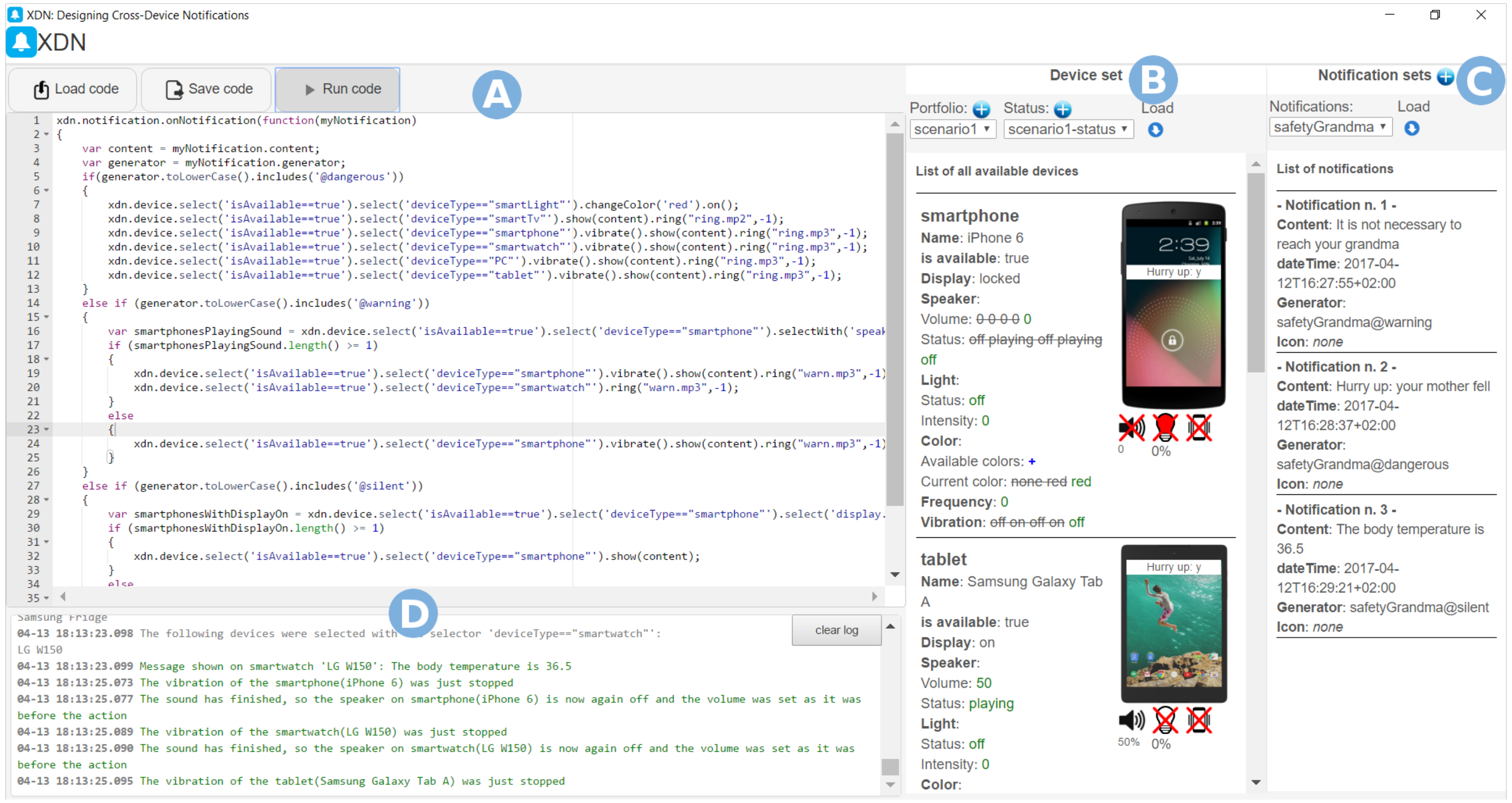
Finally, the last challenge regards the absence of the XDN runtime environment in the developed prototype: even if the simulator helped the test volunteers in testing their notification strategies, the lack of the XDN runtime environment did not allow them to test their strategies in the wild with real devices.

3.8 Conclusions

This chapter proposed a framework for developers to create and distribute cross-device notifications by scripting. The XDN architecture includes a) an XDN library to assist developers in designing personalized notifications to be distributed among ad-hoc networks of mobile and IoT devices, b) an XDN GUI to assist developers in implementing notification strategies and testing them by simulating the arrival of notifications, c) an XDN runtime environment for receiving notifications from IoT/mobile devices, executing the deployed notification strategies, and sending commands to be executed on the devices, and d) an XDN IoT/mobile library to both generate notifications compatible with XDN and execute the commands received by the XDN runtime environment.

Using one simple scenario as a running example, the major components of the framework were presented and explained. To demonstrate the feasibility of the framework and the fulfillment of all the presented requirements, a prototypical version of the XDN framework was implemented. Then, 12 volunteers were asked to implement the notification strategies able to implement two different realistic scenarios: results demonstrate that XDN framework is a promising technology.

In addition, assessment revealed some challenges that will be addressed in future works. As a future work, the prototype will be enhanced by adding other functions that developers suggested during the performed test sessions.



```
xdn.notification.onNotification(function(myNotification)
{
    var content = myNotification.content;
    var generator = myNotification.generator;
    if(generator.toLowerCase().includes('@dangerous'))
    {
        xdn.device.select('isAvailable==true').select('deviceType=="smartLight"').changeColor('red').on();
        xdn.device.select('isAvailable==true').select('deviceType=="smartTv"').show(content).ring("ring.mp2",-1);
        xdn.device.select('isAvailable==true').select('deviceType=="smartphone"').vibrate().show(content).ring("ring.mp3",-1);
        xdn.device.select('isAvailable==true').select('deviceType=="smartwatch"').vibrate().show(content).ring("ring.mp3",-1);
        xdn.device.select('isAvailable==true').select('deviceType=="PC"').vibrate().show(content).ring("ring.mp3",-1);
        xdn.device.select('isAvailable==true').select('deviceType=="tablet"').vibrate().show(content).ring("ring.mp3",-1);
    }
    else if (generator.toLowerCase().includes('@warning'))
    {
        var smartphonesPlayingSound = xdn.device.select('isAvailable==true').select('deviceType=="smartphone"').selectWith('speak
        if (smartphonesPlayingSound.length() >= 1)
        {
            xdn.device.select('isAvailable==true').select('deviceType=="smartphone"').vibrate().show(content).ring("warn.mp3",-1)
            xdn.device.select('isAvailable==true').select('deviceType=="smartwatch"').ring("warn.mp3",-1);
        }
        else
        {
            xdn.device.select('isAvailable==true').select('deviceType=="smartphone"').vibrate().show(content).ring("warn.mp3",-1)
        }
    }
    else if (generator.toLowerCase().includes('@silent'))
    {
        var smartphonesWithDisplayOn = xdn.device.select('isAvailable==true').select('deviceType=="smartphone"').select('display.
        if (smartphonesWithDisplayOn.length() >= 1)
        {
            xdn.device.select('isAvailable==true').select('deviceType=="smartphone"').show(content);
        }
        else
```

```
Samsung Fridge
04-13 18:13:23.098 The following devices were selected with the selector 'deviceType=="smartwatch"':
LG W150
04-13 18:13:23.099 Message shown on smartwatch 'LG W150': The body temperature is 36.5
04-13 18:13:25.073 The vibration of the smartphone(iPhone 6) was just stopped
04-13 18:13:25.077 The sound has finished, so the speaker on smartphone(iPhone 6) is now again off and the volume was set as it was
before the action
04-13 18:13:25.089 The vibration of the smartwatch(LG W150) was just stopped
04-13 18:13:25.090 The sound has finished, so the speaker on smartwatch(LG W150) is now again off and the volume was set as it was
before the action
04-13 18:13:25.095 The vibration of the tablet(Samsung Galaxy Tab A) was just stopped
```

List of all available devices

smartphone
Name: iPhone 6
is available: true
Display: locked
Speaker:
Volume: ~~0 0 0 0~~ 0
Status: ~~off playing off playing~~ off
Light:
Status: off
Intensity: 0
Color:
Available colors: +
Current color: ~~none red~~ red
Frequency: 0
Vibration: ~~off on off on~~ off

tablet
Name: Samsung Galaxy Tab A
is available: true
Display: on
Speaker:
Volume: 50
Status: playing
Light:
Status: off
Intensity: 0
Color:

List of notifications

- Notification n. 1 -
Content: It is not necessary to reach your grandma
dateTime: 2017-04-12T16:27:55+02:00
Generator: safetyGrandma@warning
Icon: *none*

- Notification n. 2 -
Content: Hurry up: your mother fell
dateTime: 2017-04-12T16:28:37+02:00
Generator: safetyGrandma@dangerous
Icon: *none*

- Notification n. 3 -
Content: The body temperature is 36.5
dateTime: 2017-04-12T16:29:21+02:00
Generator: safetyGrandma@silent
Icon: *none*

Fig. 3.3 Screenshot of the XDN GUI

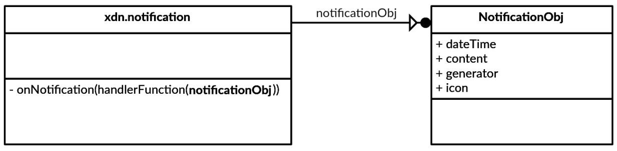


Fig. 3.4 Class diagram that shows the structure of the xdn.notification Object

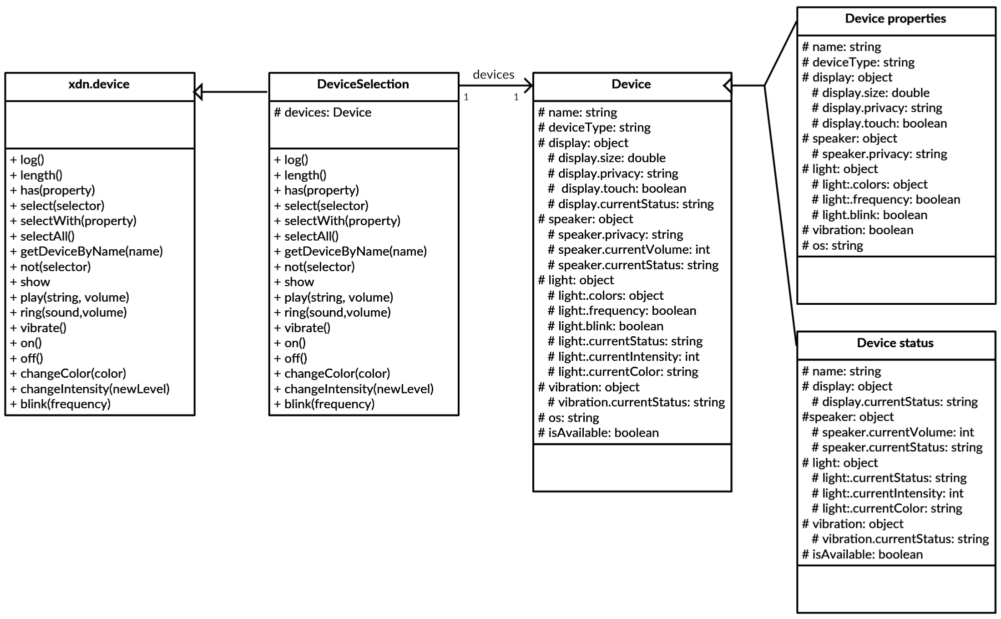


Fig. 3.5 Class diagram that shows the structure of the xdn.device Object

Chapter 4

Conclusions and Future Developments

The research goal of this dissertation regarded the investigation of the intelligence component in Internet of Things (IoT) architectures and applications. The research activity aimed at the study, definition, and prototyping of intelligent distributed architectures, and their main software components that may extract additional value and intelligent behaviors for end users.

Specifically, the distribution and customization of notifications in the IoT domain has been treated as an example of possible future IoT scenarios.

4.1 Summary of contributions

The research has been focused on developing two different architectures that deal with notifications through two different approaches:

- the first one acts at the distribution level, i.e., notifications are intercepted and then the system decides if, when, and how to show them;
- the second one acts at the design level, i.e., developers design notifications and their distribution strategies with the aim of reducing user disruption and fully exploit the advantages brought by the possibility of distributing them among different devices.

An IoT modular architecture is proposed for each approach and each architecture was designed as a group of collaborative modules: the user-centered design methodology was adopted in the design and the prototyping of the most important group of their modules and it allowed the simultaneous design and development of the most important modules of each architecture.

The first architecture regards the **Smart Notification System** that aims at reducing the disruption caused by notifications to end-user through a modular architecture that uses machine learning algorithms to adequately manage incoming notifications. According to context awareness and user habits, the system decides: a) who should receive an incoming notification; b) what is the best moment to show the notification to the chosen user(s); c) on which device(s) the chosen user(s) should receive the notification; d) which is the best way to notify the incoming notification (e.g., vibration, light, sound).

Aiming at fostering the simultaneous independent design and implementation of each part of the architecture and, at the same time, generating initial results and feedbacks, the modular nature of the Smart Notification System architecture inspired the design and the prototyping of three main groups of contributions: the *Decision Maker* contribution, the *Collectors* group of contributions, and the *Context Analysis* group of contributions.

On the other hand, the **XDN (Cross-Device Notification) framework** aimed at assisting developers in creating cross-device notifications by scripting. Inspired by the Chord framework [62][1], and with the aim of contributing to its future development (if the project will be revived), XDN was designed to assist developers in a) designing personalized notifications, and b) designing, implementing and testing notification strategies able to distribute notifications among mobile and IoT devices using a cross-device approach.

The framework architecture is composed of four main parts: the XDN library, the XDN runtime environment, the XDN GUI, and the XDN IoT/mobile library.

Finally, some tests and preliminary deployment sessions were organized for both the proposed architectures with the aim of a) collecting data for future experiments, b) verify the feasibility of the system, c) receive users' feedback about the

[1]https://github.com/google/chord, last visited on July 10, 2017, last updated on December 5, 2015

designed system, and, finally, d) foster the development of future solutions in the IoT notifications' field and related domains.

4.2 Results

The solutions that were presented in the previous chapters and the experiments and tests performed to validate each approach facilitated the identification of two different kind of results.

The first one regards the direct outcome of the presented experiments: they demonstrated the actual feasibility of the proposed approaches and the efficacy of the proposed solutions to enhance both user experience with notifications and developers that want to design, develop and test their own notification strategies. As expected, in fact, the results obtained from the tests of the first approach acting at the distribution level demonstrates the adequacy of the application of machine learning techniques to the notification domain. While the feedbacks obtained from the volunteers involved in the test sessions performed to validate the approach acting at the design level, prove the needs of a tool for developers of notification strategies, and, in addition, highlight the usefulness of the designed system.

Moreover, the second kind of results regards the lesson learned from the performed activities. As already discussed, notifications are overwhelming people lives and they are introducing both advantages and disadvantages. In fact, although, on one hand some activities and tasks have been facilitated and improved, on the other hand some other situations have been complicated by the combination of notifications' and IoT domains (e.g., notifications cause disruption in workers' activity). As a contribution to this scenario, this dissertation proposed two different approaches to transform the disadvantage of receiving the same notification on different devices, with different modalities, and in different moments, into a challenge that could be also transformed into a future opportunity. From our point of view, it is exactly what people expect from research: the proposal of innovative solutions that can allow community to transform the disadvantages get from the introduction of innovative techniques into challenges and, finally, in advantages.

In addition, the good results, in terms of incremental feedbacks obtained by the adoption of the user-centered design methodology, confirm the efficacy of the application of such technique in projects that require, and allow, the simultaneous design and development of independent but cooperative modules.

4.3 Future works

As discussed in the previous chapters, the two approaches proposed within this dissertation to deal with notifications and their related disruptions, have been evaluated through validating experiments and specific user test sessions. The results and feedbacks obtained through such validation revealed some weaknesses and open challenges with respect to both approaches. Thus, for all the two cited approaches, more exploration is possible and desirable.

At first, all the prototypes developed within the Smart Notification System architecture could be enhanced to collect more accurate information. In addition, the performed tests could be repeated by involving more people. Furthermore, performed machine learning experiments could be repeated by introducing new algorithms and/or considering the possibility of using different algorithms after other ones could be evaluated.

Moreover, the XDN framework could be enhanced with the implementation of the XDN runtime environment and considering the suggestions provided by users, like the addition of a debug inside the GUI.

Finally, the two approaches could foster the design and the development of a complete architecture that merges the contribution of the two approaches in a "macro" system. It could, for example, a) allow developers to personalize notifications and design specific notification strategies, b) foster the application of machine learning techniques to smartly manage incoming notification, and c) let developers and/or end-users decide whether apply the notification strategies (designed by developers) before or after the machine learning approach intervention.

References

[1] M. Böhmer, C. Lander, S. Gehring, D. P. Brumby, and A. Krüger. Interrupted by a Phone Call: Exploring Designs for Lowering the Impact of Call Notifications for Smartphone Users. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 3045–3054, New York, NY, USA, 2014. ACM.

[2] B. Poppinga, W. Heuten, and S. Boll. Sensor-Based Identification of Opportune Moments for Triggering Notifications. *Pervasive Computing, IEEE*, 13(1):22–29, Jan 2014.

[3] A. Kamilaris and A. Pitsillides. Mobile Phone Computing and the Internet of Things: A Survey. *IEEE Internet of Things Journal*, 3(6):885–898, Dec 2016.

[4] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[5] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376, Fourthquarter 2015.

[6] H. U. Rehman, M. Asif, and M. Ahmad. Future applications and research challenges of IoT. In *2017 International Conference on Information and Communication Technologies (ICICT)*, pages 68–74, Dec 2017.

[7] F. Corno, L. De Russis, T. Montanaro, and P. Castrogiovanni. IoT Meets Exhibition Areas: A Modular Architecture to Improve Proximity Interactions. In *2015 3rd International Conference on Future Internet of Things and Cloud*, pages 293–300, Aug 2015.

[8] D. Weber, A. S. Shirazi, and N. Henze. Towards Smart Notifications Using Research in the Large. In *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct*, MobileHCI '15, pages 1117–1122, New York, NY, USA, 2015. ACM.

[9] A. Mehrotra, M. Musolesi, R. Hendley, and V. Pejovic. Designing Content-driven Intelligent Notification Mechanisms for Mobile Applications. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and*

Ubiquitous Computing, UbiComp '15, pages 813–824, New York, NY, USA, 2015. ACM.

[10] B. P. Bailey and J. A. Konstan. On the need for attention-aware systems: Measuring effects of interruption on task performance, error rate, and affective state. *Computers in Human Behavior*, 22(4):685 – 708, 2006. Attention aware systemsSpecial issue: Attention aware systems.

[11] K. Kushlev, J. Proulx, and E. W. Dunn. "Silence Your Phones": Smartphone Notifications Increase Inattention and Hyperactivity Symptoms. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 1011–1020, New York, NY, USA, 2016. ACM.

[12] S. T. Iqbal and E. Horvitz. Notifications and Awareness: A Field Study of Alert Usage and Preferences. In *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work*, CSCW '10, pages 27–30, New York, NY, USA, 2010. ACM.

[13] P. D. Adamczyk and B. P. Bailey. If Not Now, when?: The Effects of Interruption at Different Moments Within Task Execution. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 271–278. ACM, 2004.

[14] F. Corno, L. De Russis, and T. Montanaro. A context and user aware smart notification system. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 645–651, Dec 2015.

[15] F. Corno, L. De Russis, and T. Montanaro. XDN: Cross-device Framework for Custom Notifications Management. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '17, pages 57–62, New York, NY, USA, 2017. ACM.

[16] F. Corno, L. De Russis, and T. Montanaro. Estimate user meaningful places through low-energy mobile sensing. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 003039–003044, Oct 2016.

[17] F. Corno, T. Montanaro, C. Migliore, and P. Castrogiovanni. SmartBike: an IoT Crowd Sensing Platform for Monitoring City Air Pollution. *INTERNATIONAL JOURNAL OF ELECTRICAL AND COMPUTER ENGINEERING*, 7(6):3602–3612, December 2017.

[18] K. Church and R. de Oliveira. What's Up with Whatsapp?: Comparing Mobile Instant Messaging Behaviors with Traditional SMS. In *Proceedings of the 15th International Conference on Human-computer Interaction with Mobile Devices and Services*, MobileHCI '13, pages 352–361, New York, NY, USA, 2013. ACM.

[19] A. Sahami Shirazi, N. Henze, T. Dingler, M. Pielot, D. Weber, and A. Schmidt. Large-scale Assessment of Mobile Notifications. In *Proceedings of the SIGCHI*

Conference on Human Factors in Computing Systems, CHI '14, pages 3055–3064. ACM, 2014.

[20] P. Harrington. *Machine Learning in Action*. Manning Publications Company, 2011.

[21] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[22] J. Makhoul, F. Kubala, R. Schwartz, and R. Weischedel. Performance Measures For Information Extraction. In *In Proceedings of DARPA Broadcast News Workshop*, pages 249–252, 1999.

[23] D. Bužić and J. Dobša. Lyrics classification using Naive Bayes. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1011–1015, May 2018.

[24] Costa, P. F. Pires, F. C. Delicato, and P. Merson. Evaluating a Representational State Transfer (REST) Architecture: What is the Impact of REST in My Architecture? In *2014 IEEE/IFIP Conference on Software Architecture*, pages 105–114, April 2014.

[25] L. Ardissono, G. Bosio, A. Goy, G. Petrone, and M. Segnan. Managing Context-Dependent Workspace Awareness in an e-Collaboration Environment. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 03*, WI-IAT '09, pages 42–45, Washington, DC, USA, 2009. IEEE Computer Society.

[26] C. Roecker, V. Bayon, M. Memisoglu, and N. Streitz. Context-dependent email notification using ambient displays and mobile devices. In *Active Media Technology, 2005. (AMT 2005). Proceedings of the 2005 International Conference on*, pages 137–138, May 2005.

[27] A. Leonidis, G. Baryannis, X. Fafoutis, M. Korozi, N. Gazoni, M. Dimitriou, M. Koutsogiannaki, A. Boutsika, M. Papadakis, H. Papagiannakis, G. Tesseris, E. Voskakis, A. Bikakis, and G. Antoniou. AlertMe: A Semantics-Based Context-Aware Notification System. In *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, volume 2, pages 200–205, July 2009.

[28] S. Banerjee and D. Mukherjee. Towards a Universal Notification System. In *Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2013 IEEE/WIC/ACM International Joint Conferences on*, volume 3, pages 286–287, Nov 2013.

[29] R. Etter, P.D. Costa, and T. Broens. A Rule-Based Approach Towards Context-Aware User Notification Services. In *Pervasive Services, 2006 ACS/IEEE International Conference on*, pages 281–284, June 2006.

[30] R. M. Arlein, S. Betgé-Brezetz, and J. R. Ensor. Adaptive notification framework for converged environments. *Bell Labs Technical Journal*, 13(2):155–159, Summer 2008.

[31] A. Mehrotra, R. Hendley, and M. Musolesi. PrefMiner: Mining User's Preferences for Intelligent Mobile Notification Management. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '16, pages 1223–1234, New York, NY, USA, 2016. ACM.

[32] A. Mehrotra, V. Pejovic, J. Vermeulen, R. Hendley, and M. Musolesi. My Phone and Me: Understanding People's Receptivity to Mobile Notifications. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 1021–1032, New York, NY, USA, 2016. ACM.

[33] V. Pejovic and M. Musolesi. InterruptMe: Designing Intelligent Prompting Mechanisms for Pervasive Applications. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14, pages 897–908, New York, NY, USA, 2014. ACM.

[34] N. Eagle and A. (Sandy) Pentland. Reality Mining: Sensing Complex Social Systems. *Personal Ubiquitous Comput.*, 10(4):255–268, March 2006.

[35] S. S. Stevens. On the Theory of Scales of Measurement. *Science, New Series*, 103(2684):677–680, June 1946.

[36] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[37] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[38] J. R. Quinlan. Induction of Decision Trees. *Mach. Learn.*, 1(1):81–106, March 1986.

[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[40] X. Liu, B. Li, A. Jiang, S. Qi, C. Xiang, and N. Xu. A bicycle-borne sensor for monitoring air pollution near roadways. In *Consumer Electronics - Taiwan (ICCE-TW), 2015 IEEE International Conference on*, pages 166–167, June 2015.

[41] F. Zeiger and M. Huber. Demonstration Abstract: Participatory Sensing Enabled Environmental Monitoring in Smart Cities. In *Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, IPSN '14, pages 337–338. IEEE Press, 2014.

[42] X. Liu, C. Jiang, B. Li, and A. Jiang. Collaborative Bicycle Sensing for Air Pollution on Roadway. In *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, pages 316–319, Aug 2015.

[43] C. Vagnoli, F. Martelli, T. D. Filippis, S. D. Lonardo, B. Gioli, G. Gualtieri, A. Matese, L. Rocchi, P. Toscano, and A. Zaldei. The sensorwebbike for air quality monitoring in a smart city. In *Future Intelligent Cities, IET Conference on*, pages 1–4, Dec 2014.

[44] Y. Taniguchi, K. Nishii, and H. Hisamatsu. Evaluation of a Bicycle-Mounted Ultrasonic Distance Sensor for Monitoring Road Surface Condition. In *Computational Intelligence, Communication Systems and Networks (CICSyN), 2015 7th International Conference on*, pages 31–34, June 2015.

[45] I. Alam. An exploratory investigation of user involvement in new service development. *Journal of the Academy of Marketing Science*, 30(3):250–261, 2002.

[46] G. Holmes, A. Donkin, and I. H. Witten. WEKA: a machine learning workbench. In *Intelligent Information Systems,1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, pages 357–361, Nov 1994.

[47] Y. Chon, E. Talipov, H. Shin, and H. Cha. Mobility Prediction-based Smartphone Energy Optimization for Everyday Location Monitoring. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, pages 82–95. ACM, 2011.

[48] Y. Chon, H. Shin, E. Talipov, and H. Cha. Evaluating mobility models for temporal prediction with high-granularity mobility data. In *Pervasive Computing and Communications (PerCom), 2012 IEEE International Conference on*, pages 206–212, March 2012.

[49] C. Zhou, D. Frankowski, P. Ludford, S. Shekhar, and L. Terveen. Discovering Personally Meaningful Places: An Interactive Clustering Approach. *ACM Trans. Inf. Syst.*, 25(3), July 2007.

[50] G. Metri, A. Agrawal, R. Peri, and W. Shi. What is eating up battery life on my SmartPhone: A case study. In *Energy Aware Computing, 2012 International Conference on*, pages 1–6, Dec 2012.

[51] M. B. Kjærgaard, J. Langdal, T. Godsk, and T. Toftkjær. EnTracked: Energy-efficient Robust Position Tracking for Mobile Devices. In *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services*, MobiSys '09, pages 221–234, New York, NY, USA, 2009. ACM.

[52] H. Xu and S. B. Cho. Recognizing Semantic Locations from Smartphone Log with Combined Machine Learning Techniques. In *Ubiquitous Intelligence and Computing, 2014 IEEE 11th Intl Conf on and IEEE 11th Intl Conf on and Autonomic and Trusted Computing, and IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UTC-ATC-ScalCom)*, pages 66–71, Dec 2014.

[53] J. Ryoo, H. Kim, and S. R. Das. Geo-fencing: Geographical-fencing based energy-aware proactive framework for mobile devices. In *Quality of Service (IWQoS), 2012 IEEE 20th International Workshop on*, pages 1–9, June 2012.

[54] N. E. Klepeis, W. C. Nelson, W. R. Ott, J. P. Robinson, A. M. Tsang, P. Switzer, J. V. Behar, S. C. Hern, and W. H. Engelmann. The National Human Activity Pattern Survey (NHAPS): a resource for assessing exposure to environmental pollutants. *Journal of Exposure and Environmental Epidemiology*, May-Jun 2001.

[55] J. Paek, K.-H. Kim, J. P. Singh, and R. Govindan. Energy-efficient Positioning for Smartphones Using Cell-ID Sequence Matching. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 293–306, New York, NY, USA, 2011. ACM.

[56] L. Garbe. System Identifies User Location without GPS or Wi-Fi. *Computer*, 44(11):15–17, 2011.

[57] S.-M. Qin, H. Verkasalo, M. Mohtaschemi, T. Hartonen, and M. Alava. Patterns, Entropy, and Predictability of Human Mobility and Life. *PLoS ONE*, 7:e51353, December 2012.

[58] P. Seshadri, S. Abileah, N. Nilakantan, H. Knight, S. Pather, R.H. Gerber, C.T. Mensa-Annan, P. Garrett, M.A. Faoro, and D.O. Lavery. User interface system and methods for providing notification(s), april 2008. US Patent 7,360,202.

[59] E. Horvitz, C. Kadie, T. Paek, and D. Hovel. Models of Attention in Computing and Communication: From Principles to Applications. *Commun. ACM*, 46(3):52–59, March 2003.

[60] M. C. Koss, J. Dewitt, K. J. Messerly, and D. Titov. CROSS-DEVICE NOTIFICATIONS, December 2015. Patent US 2015/0373089.

[61] P. Hamilton and D. J. Wigdor. Conductor: Enabling and Understanding Cross-device Interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 2773–2782, New York, NY, USA, 2014. ACM.

[62] P. Chi and Y. Li. Weave: Scripting Cross-Device Wearable Interaction. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 3923–3932. ACM, 2015.

[63] S. T. Iqbal and B. P. Bailey. Effects of Intelligent Notification Management on Users and Their Tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 93–102, New York, NY, USA, 2008. ACM.

[64] D. Weber, A. Voit, P. Kratzer, and N. Henze. In-situ Investigation of Notifications in Multi-device Environments. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '16, pages 1259–1264, New York, NY, USA, 2016. ACM.

[65] T. Kubitza, A. Voit, D. Weber, and A. Schmidt. An IoT Infrastructure for Ubiquitous Notifications in Intelligent Living Environments. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, UbiComp '16, pages 1536–1541, New York, NY, USA, 2016. ACM.

[66] J. Yang and D. Wigdor. Panelrama: Enabling Easy Specification of Cross-device Web Applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 2783–2792, New York, NY, USA, 2014. ACM.

[67] M. Nebeling, M. Husmann, C. Zimmerli, G. Valente, and M. C. Norrie. XD-Session: Integrated Development and Testing of Cross-device Applications. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '15, pages 22–27, New York, NY, USA, 2015. ACM.

[68] M. Schreiner, R. Rädle, H. Jetter, and H. Reiterer. Connichiwa: A Framework for Cross-Device Web Applications. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '15, pages 2163–2168, New York, NY, USA, 2015. ACM.

[69] M. Husmann, M. Spiegel, A. Murolo, and M. C. Norrie. UI Testing Cross-Device Applications. In *Proceedings of the 2016 ACM on Interactive Surfaces and Spaces*, ISS '16, pages 179–188, New York, NY, USA, 2016. ACM.

Appendix A

Detailed SmartBike Survey Results

Based on the observation reported by Alam et al. [45] about the usefulness of involving users in designing new services, as the first step in developing the SmartBike platform presented in Section 2.8.4, an online survey was conducted with the aim of identifying the most interesting bike-enabled features. 288 persons were involved among Politecnico di Torino students and TIM employees that usually move around the city of Turin (Italy) riding their bikes. The survey has been accessible for two months and consisted of 10 questions divided into three main sections. The first section aimed at acquiring user demographic and habits information. The second section, instead, had the objective of analyzing user preferences for the future design of aesthetic characteristics of the proposed solution. The questions asked users to express their preferences about the preferred type of bike (e.g., mountain bike, city bike, etc.) and the most desired bike accessories (e.g., basket, baggage holder, etc.). Finally, the third section was related to the identification of the most interesting services for cyclists in the context of a smart city. The language used within the survey was Italian, results were then translated for the purposes of this thesis.

Demographic information about interviewees

Aiming at recruiting users that usually move around the city of Turin (Italy) riding their bikes, 500 promotional tags with a printed QR code (which pointed to the survey's link) were attached to the bikes parked in the Politecnico di Torino courtyard. Moreover, an email promoting the survey was sent to TIM employees. A population

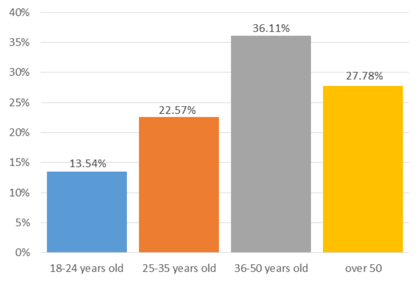


Fig. A.1 User Survey: age distribution

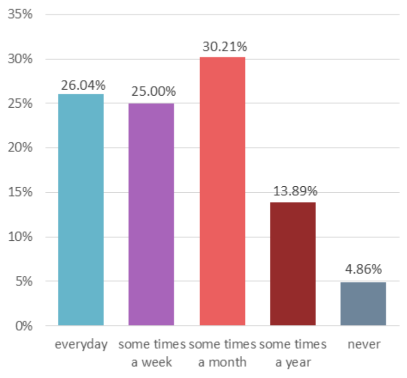


Fig. A.2 User Survey: frequency of bicycle usage

of 288 people replied to the study, with 221 males and 67 females. Most participants were aged in the interval “36 - 50” (Figure A.1).

Furthermore, Figure A.2 shows the frequency with which users declared to use their bicycles: most of the interviewees (81%) commonly use their bikes several times a month, and 50% at least weekly.

Preferred type of bike and most used bike devices

In the second section of the survey, respondents were asked to select the preferred type of bike among the following four types:

- budget bike
- city bike
- mountain bike
- racing bike.

Results reveals that the most preferred type of bike is the city bike (45.8%), followed by the mountain bike (26.7%), the racing bike (22.0%) and, finally the budget bike (5.5%).

Most interesting bike-enabled features

In the final section of the survey, users were asked to select up to six possible future high tech bike improvements out of those listed in Table A.1. The reported values represent the percentage of users that selected each feature.

As can be observed from table A.1, the six most requested features are: real time geo-location detection of the bike in case of loss or theft, anti-theft service which can send notifications to an end-user device, information about traveled route (traveled distance, duration, difference in altitude), air pollution level of traveled roads, GPS navigation device, and information about the speed.

Table A.1 List of possible technological bike improvements with percentage of users that selected each of them

Feature	**Percentage of selections (over 288 users)**
Real time remote geo-location detection of the bike in case of loss or theft	67.72%
Anti-theft feature which can send notification to an end-user device	64.67%
Information about traveled routes (traveled distance, duration, difference in altitude)	47.48%
Air pollution level of traveled roads	42.07%
GPS navigation device	31.49%
Average, minimum, and maximum speed	30.25%
Automatic call for assistance in case of accidents (e-call)	29.70%
Bicycle maintenance status	23.79%
Information about high injury risk roads	22.28%
Burned calories	13.09%
Traffic information	13.05%
Audio instructions about navigation	11.81%
Heartbeat monitoring	9.69%
Reminders based on location on the route	7.98%
Point of interest	7.42%
Information about not accessible roads	6.93%

Appendix B

Publications

- Corno, F and De Russis, L. and Marcelli, A. and Montanaro, T. **An Unsupervised and Non-Invasive Model for Predicting Network Resource Demands.** In IEEE Internet of Things Journal.
- Cagliero, L. and De Russis, L. and Farinetti, L and Montanaro, T. **Improving the effectiveness of SQL learning practice: a data-driven approach.** In 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC).
- Corno, F and Montanaro, T., Migliore, C. and Castrogiovanni, P. **Smartbike: an IoT Crowd Sensing Platform for Monitoring City Air Pollution.** INTERNATIONAL JOURNAL OF ELECTRICAL AND COMPUTER ENGINEERING, 7(6):3602–3612, December 2017.
- Corno, F and De Russis L. and Montanaro, T. **XDN: Cross-Device Framework for Custom Notifications Management.** In Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS '17, pages 57–62, New York, NY, USA, 2017. ACM.
- Corno, F and De Russis L. and Montanaro, T. **Estimate User Meaningful Places Through Low-Energy Mobile Sensing.** In 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pages 003039–003044, Oct 2016.

- Ghajargar, M and Zenezini, G. and Montanaro, T. **Home delivery services: innovations and emerging needs.** In IFAC-PapersOnLine, Volume 49, Issue 12, 2016, Pages 1371-1376.

- Montanaro, T. 2015. **SWARM joint open lab Politecnico Di Torino, Italy.** XRDS 22, 2 (December 2015), 70-71.

- Corno, F and De Russis L. and Montanaro, T. **A Context and User Aware Smart Notification System.** In 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), pages 645–651, Dec 2015.

- Corno, F and De Russis L. and Montanaro, T. and Castrogiovanni, P. **IoT Meets Exhibition Areas: A Modular Architecture to Improve Proximity Interactions.** In 2015 3rd International Conference on Future Internet of Things and Cloud, pages 293–300, Aug 2015.

- Atzeni, A. and Su, T. and Montanaro, T. (2014) **Lightweight Formal Verification in Real World, A Case Study.** In: Iliadis L., Papazoglou M., Pohl K. (eds) Advanced Information Systems Engineering Workshops. CAiSE 2014. Lecture Notes in Business Information Processing, vol 178. Springer, Cham

Submitted - under revision

In addition, the following list reports the paper that was already submitted and is yet under revision.

- Corno, F. and De Russis, L. and Montanaro, T. **XDN: Cross-Device Framework for Custom Notifications Management** In Springer COMPUTING Journal