Enhancing Mobile Devices with Cooperative Proactive Computing

Gilles I. F. Neyens , Remus-Alexandru Dobrican , Denis Zampunieris
University of Luxembourg
Luxembourg, Grand-Duchy of Luxembourg
Email: gilles.neyens.001@student.uni.lu , {remus.dobrican, denis.zampunieris }@uni.lu

***Abstract*—With the increasing popularity of smartphones and with the fact that they are connected to the Internet most of the time, people manage to stay online everywhere they go. They can access online services remotely at any time they want, using their mobile devices. However, in order to make the best out of these circumstances, the users have to use sophisticated mobile applications. These applications do not have to only address key aspects like collaboration and cooperation between various devices but have to deal also with the involvement of the users in order to achieve the desired outcome. The main contribution of this paper is to present a solution, i.e., Proactive Engine for Mobile Devices (PEMD), together with its implementation for Android-based systems, for enhancing mobile devices with proactive properties. The model serves as a basis for developing smart applications that are able to perform complex real-world tasks. Furthermore, it provides a method for achieving cooperation, coordination and collaboration of multiple smart devices. Finally, we provide the performance experiments and we discuss the results and the effects of using PEMD on different devices.**

Keywords–Mobile devices; Collaborative architectures and mechanisms, Proactive Computing.

I. INTRODUCTION

Smartphones and tablets have become more and more popular and more powerful in terms of their computing capacity [1]. This opens numerous possibilities for developing new types of software applications and, for the existing ones, to expand, to incorporate new features and to provide better services to their users. Also, due to the latest advances in computer hardware technology [2] [3], mobile devices are more than ready to be enhanced with proactive features and properties. These properties are essential for enhancing mobile devices with collaborative methods for their applications.

A great challenge of today's mobile world is to provide services and applications that support collaboration and distributed tasks. Middleware systems are seen very often as solutions for supporting the construction of mobile collaborative applications [4]. We investigated how to implement a more specific category of middleware systems, i.e., rule-based middleware systems, because they offer various advantages over other middleware systems such as the possibility of having rules where the developers can easily insert their instructions. These rules, among others, can deal with unforeseen situations, can predict future events and can be programmed to take appropriate actions in most of the situations. The main difficulty consists of implementing this kind of systems on mobile devices because they are very complex and they need many resources, depending on which kind of applications they are executing. Having desktop rule-based systems executing all the tasks of the mobile devices is not a valid solution as the communication between them is not very stable because of the mobility of the mobile devices. The probability of having successful collaborative tasks increases when each mobile device has an integrated middleware architecture, capable of initiating collaborations and performing distributed jobs. The major advantage of our middleware architecture is that it is capable of communicating with other mobile devices and desktop computers without any compatibility issues.

The contribution of this paper is three-fold. First, it proposes a rule-based engine capable of performing complex automated and distributed tasks on mobile devices on behalf of the user. Second, it offers details about the implementation of our model on Android-based mobile devices. And third, it shows how the model was tested on real devices and provides an evaluation of their performance.

The rest of the paper is organized as follows: Section II describes the state of the art related to this study. Section III provides the main characteristics of our model in terms of interaction with the operating system, communication mechanisms, information sharing strategies and data storage. In Section IV, we present the tests that were done to evaluate our model, what methodology was used, then we provide an analysis of these results and, at the end of the section, we discuss the future implications of the results. Finally, Section V concludes the paper and indicates the next research directions that will follow after this work.

II. STATE OF THE ART

Proactive Computing was introduced in [5] and was described as a new computational way in which software systems can operate and can perform different tasks. It was initially seen as a solution for removing humans from the computational loop, for moving from human-centred computing to human-supervised computing. Proactive Systems were characterised as those systems that can work for the user and that can take decisions on their own initiative, without necessarily involving the user in this process [6].

More recent studies [7][8] describe Proactive Systems as being aware of the changes, which occur in their surrounding environment, being able to react to foreseen events and of adapting their behaviour in order to address the increasing needs of their users. Recent empirical investigations [9][10] support these properties and provide additional examples of the advantages that Proactive Systems have to offer in numerous domains of Computer Science. For instance, enhancing Learning Management Systems (LMSs) with Proactive Computing led to an increase of the online participation of the students [9], helped improve the assignment system [11] and raised the chances of student to obtain higher grades at their final exams [10].

Rule 102

Description: This Rule was activated by Rule 101 and started creating a Community of Practice for new cities detected from the student's profile.

```
data acquisition
    String groupName = cityName;
    String [] students = getStudentFromSameCity (cityName);
activation guards
    return groupExists(cityName);
conditions
    return true;
actions
    foreach student in students []
        if(userIsNotPartOfGroup(student.ID, groupName)))
            inscribeUserInGroup(student.ID, groupName);
        end if
    end foreach
rules generation
    if (activationGuard());
        createRule103(groupName());
    end if
    cloneRule(Rule102);
```

Figure 1. A Proactive Rule in pseudo-code

More precisely, using a Proactive Engine (PE) aside the LMS transformed the LMS from a static system into a proactive system. The PE was created as a structure capable or executing Proactive Rules [12], which are simple mechanisms for executing specific actions, like, for instance, sending emails to the users. They were designed for offering developers the possibility of implementing proactive actions in a simple way, without having to have advanced knowledge about proactive systems and their implementation. In Figure 1, an example of a Proactive Rule is given in pseudo-code. The rule was implemented in Java and was used as part of a predefined scenario for automatically creating certain Communities of Practice inside a LMS [13].

Multiple collaborative middleware systems for mobile devices are available on the market, e.g., [14]-[16]. They differ by their type: event-based architectures, e.g., for supporting location aware-mobile applications [17], or publisher-subscriber architectures, e.g., for [4] systems. These frameworks offer communication services for the mobile devices, which use them for performing collaborative tasks. Our framework does not only achieve message passing from one device to another but also permits more complex actions like remote rule activations, parallel commands or complex reasoning algorithms.

Rule-based systems, like Java Expert System Shell (JESS) [18] or C Language Integrated Production System (CLIPS) [19], are powerful desktop-based general-purpose tools, which give the possibility of programming expert systems. With the help of their scripting language facts and rules can be uniformly defined and described. For example, JESS employs the Rete [20] algorithm for compiling and executing forward-chain rules. These rules are simple statements, composed of a left side, i.e., the IF portion, and of a right side, i.e., the THEN portion. Unfortunately, they represent solutions only for the server side and are not suitable for mobile devices as these devices are quite limited in terms of computing power. We do not only propose an engine for mobile devices capable of executing complex rules but we propose a technique, called Global Proactive Scenario (GPaS) [21], for breaking down complex scenarios or applications in multiple sets of Proactive Rules. More about this technique is explained in subsection III-D.

Existing rule-based systems [22][23][24] for mobile devices solve only simple tasks and do not provide methods for achieving more complex tasks like distributed reasoning, task distribution, data sharing, acquiring global context information or/and collaborative filtering. And most of all, these systems do not take advantage of the global information that can be built with information from each particular mobile device. IF THIS THEN THAT [24] is a mobile application that realizes automation for small tasks between Internet-connected services. The user can write simple rules, also called recipes, in order to achieve different goals like adding the photo of a user to the cloud-based archive if the user has been tagged in that particular photo on Facebook [25]. These rules, however, are just simple conditional statements. HeaRT [22], a lightweight rule-based inference engine designed for mobile devices, was used in [26] for providing simple tasks like online reasoning, part of a bigger plan to develop context-aware mobile applications. The rules that are written for this engine can achieve local reasoning only based on the internal sensors of a mobile device and do not explore the possibility of having multiple engines performing global reasoning. Minimal Rule Engine (MiRE) [23], a context-aware processing engine, was implemented in order to obtain an engine capable of processing rules on mobile devices. However, the rules are written in Extensible Markup Language (XML) and, due to their structure, are not capable of integrating more complex logics. The above approaches, [22]-[24], try to address the growing demand of using rule-based tools on mobile platforms and manage to do it but for a very narrow type of applications.

Until now, Proactive Engines were only used on desktop computers [13][27][28], which limited a lot their usability. Analysing the advantages and functionalities of mobile devices makes it clear that mobile devices offer new possibilities for Proactive Systems, as well as the other way around. Proactive Computing can help mobile device become smarter in terms of how they make use of the data coming from all the sensors, of how they exchange information, of how they execute complex tasks and in terms of how they provide services to their users. We therefore developed a PE for mobile devices.

III. THE ARCHITECTURE OF PROACTIVE ENGINES FOR ANDROID-BASED MOBILE DEVICES

In order to better understand the structure of the new engine we will now first explain parts of the already existing server-based engine. The Rules Engine is the core piece of the engine. It consists of two First In First Out (FIFO) lists: the Current Queue, which contains the rules that are currently being executed and the Next Queue, which contains the rules that were created during the current iteration of the Rules Engine and that will be executed at the next iteration. During an iteration, the Rules Engine will execute the rules that are stored in the Current Queue one by one. Proactive Rules can perform different operations like checking for special conditions or constraints, saving events or relevant context information into the local database, or cloning themselves in

order to run at the next iteration. They can also generate other Proactive Rules during the *rules generation phase*, which is one of the five phases that compose a Proactive Rule, as shown in Figure 1. The new rules, created during the *rules generation phase* are stored in the Next Queue. An iteration finishes its execution if there are no more rules in the Current Queue or if there were already N rules executed during the current iteration or if the execution time exceeded the time limit F, an internal parameter of the PE, which represents the frequency of activation periods. At the end of the iteration all rules contained in the Next Queue are added to the Current Queue and the Next Queue gets cleared. The Current Queue is then saved into the database and the Rules Engine will continue with the next iteration. This Rules Engine is the basis for the implementation of our PE for Mobile Devices (PEMD). Other components like the database also had to be adapted in order to fit the requirements of mobile devices.

A. The Rules Engine as background service

The Rules Engine was designed to run constantly in the background in order to allow the user to interact with different applications. On Android this can be achieved with the help of background services. However, these services cannot run constantly as they might be killed or stopped by the Operating System if they use too many resources over an extended period of time. Therefore, the existing solution for desktop computers needed to be adapted in order to still allow the PE to execute Proactive Rules. This was done using an Alarm Manager that activated the background service every *F* seconds and that executed one iteration of the Rules Engine. The Alarm Manager was triggered by the *onBoot* event.

The added essential functionalities are the communication of PEs, explained in detail in Section III-C, and the possibility of notifications to the user from within rules, briefly explained at the end of section III-B.

B. Data Storage

Data storage is an important part of the application. The rules in the Current Queue of the Rules Engine need to be saved at every iteration so that the PEMD can recover in case of failure by using a previous state. This is particularly important as the Android Operating System (OS) may decide to kill the background service that contains the Rule Engine, if the system is low on memory. If the Current Queue is not saved at every iteration, the Rules Engine would recover the state it had when it first started, meaning that every progress in the execution of the Rules is lost. Also, the rules need to be saved when the device shuts down so that the engine can recuperate and continue executing them when the device is turned on again. The saving process of the Current Queue is performed in a transaction, both for performance and failure recovery reasons.

Additionally, the sent and received messages need to be saved to allow the engine to resend lost messages and also the notifications displayed to the user. As there will likely be different types of rules and notifications depending on the purpose of the rules engine, the proposed solution will automatically create the appropriate database table upon installation or update of the application, which is achieved through the Object Relational Mapping Lite (ORMLite) [29] package framework for Android. The ORMLite package is a lightweight package for persisting Java objects to Structured Query Language (SQL) databases. To add new types of rules or notifications to the PEMD, one has only to create the appropriate file with the correct annotations and the PEMD will take care of creating the correct tables and of the saving. Notifications use the internal notification system provided by Android and can be further customised and modified by the applications developers if needed.

C. Communication of Proactive Engines

The most important part of the PEMD architecture is the communication between several devices as this allows the devices to exchange information. There are several technologies available for smartphones to achieve this, each with their own advantages and disadvantages. After analysis of alternatives, we decided that the solution, which suits our needs best and which was used in our implementation of the PEMD is Google Cloud Messaging (GCM). There are another few alternatives on the market like Parse [30], PubNub [31] or UrbanAirship [32]. Nevertheless, while these services make it easier to develop push notifications for iOS and Android, they are still using GCM. One alternative, which does not use GCM at all, is Pushy [33]. However, Pushy's architecture is very similar to GCM, maintaining its own background socket connection, to receive push notifications [34].

1) Google Cloud Messaging: GCM for Android is a service provided by Google, which allows the sending and reception of data between a server and Android-based smartphones. The GCM service handles the delivery process of the messages, meaning that it takes care that the messages are delivered to the correct device. In the case where a message is sent to an offline device, GCM temporarily stores the message until the receiving device comes back online. In order to use GCM for device-to-device communication, we proposed an architecture, illustrated in Figure 2 and Figure 3. The registration process of two devices is shown in Figure 2, while Figure 3 is showing the communication between already registered devices. The devices first have to register at the GCM and get a registration ID. They then communicate this ID along with a username to the server, which stores them in a database. If an already registered device now wants to communicate with another registered device, it sends its message along with the receiving device's username to the server, which retrieves the correct ID and pushes the message along with ID to the GCM, which then takes care of the delivery of the message.

2) Message Structure: The messages exchanged between PEMDs need to follow the same standards. The messages are encoded with JavaScript Object Notation (JSON) and have different attributes depending on the type of message. The only attribute that is common to all message types is the instruction attribute that allows the receiving PEMD to parse incoming messages correctly. There are currently two types of messages, the activate Rule message type and the confirmation message type. The activate Rule message type has the following attributes:*instruction*,*msgID*, *senderID*, *receiver ID*, *ruleName*, *parameterList*.

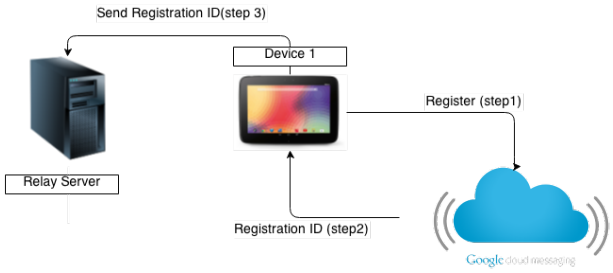

Figure 2. The registration process for one device, passing through Google's Cloud Messaging Server

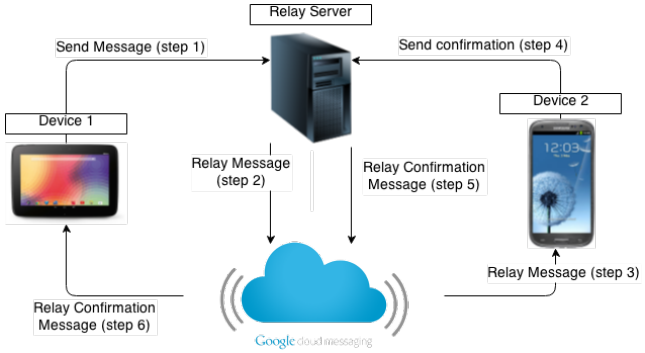

Figure 3. The communication process between 2 registered devices

The *msgID* along with the *senderID* allows the receiving PEMD to keep track of already received messages. This will be explained in more detail in the error handling section. The *ruleName* and *parameterList* attributes are the core of the communication process. They allow the creation of a Rule on another PEMD. The *ruleName* attribute contains the name of the rule that will be created on the receiving device and the *parameterList* contains the parameters necessary to create this rule dynamically. After the rule is created, it will be added to the Next Queue of the Rules Engine. An example of a message exchanged between PEMDs, which contains a command to active a Proactive Rules, can be seen in Figure 4. The confirmation message only contains three attributes: *instruction*, *msgID*, *receiverID*, where *msgID* is the ID of the message whose delivery is confirmed.

3) Error Handling: In a distributed environment, messages are not guaranteed to arrive, as they can be lost along the communication process. In order to prevent the loss of messages, PEMDs keep track of sent and received messages,

```
{ 'instruction':'activate rule',
'msgId':'messageID ',
'senderID':'ID ',
'receiverID':['registrationID '],
'ruleName':'R004 ',
'parameterList ':['param1 ', 'param2 '] }
```

Figure 4. Example of a message exchanged between 2 PEMDs, which contains a command to activate a Proactive Rule

including their senders and receivers, by saving them locally in a database table. After sending a message, the PEMD saves it to the database. If no confirmation message is received within a given time period, the message is sent again. The time period varies depending on the priority level of the message, which can be set when sending it. It is also possible to set no priority level at all so that the message does not have to be confirmed. This is quite useful to do broadcasts in order to find other devices with the same preferences, where it is not important that really every device receives the broadcast. Upon receiving the confirmation for a specific message, this message is deleted from the database.

Similarly after receiving a message, the message is saved in the database and a confirmation message is sent to the sender of the message. In the case the confirmation message does not arrive, the sending device resends the message and as the receiving device has saved the message in its database, it ignores the second message. The stored messages are deleted after a fixed time period. In order to take care of the resending and ignoring of messages, there are two rules constantly running on the engine; one rule that takes care of the messages that were sent and one rule that takes care of the received messages.

4) Limitations of GCM and Workarounds: GCM has two main disadvantages, a size limit on the messages and a limit on the number of devices a message can be simultaneously sent to. The size limit on the messages is of 4 Kb. A lot of messages of our application can be delivered using the direct method as they are smaller than 4Kb. If the size limit is exceeded, our server will store the complete message in its database and just send a small message to the receiving device to notify it that a message is available. The device will then download the message directly from the server and add the rule to the queue.

The second restriction of GCM is that a message can only be sent to 1000 devices simultaneously. In our application this can only happen if a broadcast is sent to all devices. In this case, the server just splits the list of all devices into packets of 1000 and pushes the same message for every packet to the GCM.

D. Information Sharing Strategies between Proactive Engines

As explained in Section I, GPaSs [21] were proposed in order to address complex situations or tasks like the collaboration and collaboration between multiple Proactive Engines. Depending on their complexity, GPaSs contain one or more Proactive Rules from different categories of rules like Adaptation Rules, Cooperation Rules, Coordination Rules, Communication Rules and Notification Rules. For example, an application that would automatically form social groups of people based on common interests or activities would use Cooperation Rules to exchange information between the devices, Context-Awareness Rules to obtain details about the user's context and Notification Rules to keep the users up-to-date with the application's latest actions. The developers have to focus more on how to decide what functionalities they want to include into the application and how to transform it into Proactive Rules then to take care of how the PE executes the rules or how the information is exchanged between multiple PEs.

IV. RESULTS AND DISCUSSIONS

Performance is an important aspect of software development, especially when designing and implementing smart applications for mobile devices. It was thus necessary to investigate how PEMD perform on different devices and what are the factors that influence the overall performance of GPaSs.

In this section, we present the hardware and software specifications of the devices on which PEMD was tested, we explain which type of experiments were performed, why we chose to carry out these experiments and we discuss the results that we collected after the tests.

A. Hardware and Software specifications

Three different computing systems were included in our tests: a smartphone, a tablet and a desktop computer. In Figure 5 we provide, for each device, the most important software and hardware specifications at the moment the tests were performed. These specifications include the versions of the Operating System and of the Kernel, the types of central processing units (CPU) and graphics processor units (GPU) used, the amount of random access memory (RAM), the available sensors and the types of communication protocols that could be used by the devices. The smartphone and the tablet have been available on the market since a couple of years. They were chosen for the tests to show how PEMD behave on devices that are used by the majority of the existing applications on the market and which contain stable version of the OS and of the other frameworks used. The PC was included in our tests to check if there are any specific factors on mobile devices that are influencing the performance of PEMD.

B. Methodology

Our main goal was to see approximatively what is the highest number of Proactive Rules PEMDs are able to run on various mobile devices without affecting the overall performance of these devices. More precisely, we focused on evaluating the how many Proactive Rules can be executed by the PEMD in a reasonable amount of time in order to still be able to provide real-time services to the user.

Our method for analysing the performance of the PEMD involved measuring the time between two consecutive *iterations* of the Rules Engine. An *iteration* is an executing instance composed of a set of Proactive Rules in the Rules Engine and can be measured in terms of duration. Ten different sets of Proactive Rules were considered for the tests: starting from small sets containing 100 rules until large sets with 1000 rules. From a technical point of view, an iteration is composed of two main operations: *the execution phase* and the *saving phase*.

During the execution phase, the Rules Engine runs each instruction, part of each Proactive Rule, which can contain possible actions like acquiring data from the sensors or from the local database, sending notification to the users or even the generation of other rules for the next iteration. The saving phase is mainly used for saving the set of rules that are to be executed during the next iteration. A list of rules, together with their parameters are saved into the local database. This phase was created as a safety measure in case a crash occurs or the Rules Engine is stopped. After a crash, when the Rules Engines restarts, it reads the last list of rules that was saved in the database and it will start executing them.

Multiple rounds of tests were performed, each round of tests containing 10 evaluations for each device. The execution time averages for all the tests were computed for obtaining more accurate values. Other applications and services running on the devices involved in the tests were not explicitly closed because we wanted to analyse the performance of the PEMD in the same circumstances that a common user would be using his/her device.

C. Performance analysis

Table I contains the averages of the total amount of time in milliseconds that one iteration required for running sets of 100, 300, 700 and 1000 Proactive Rules. The total time is divided furthermore into *Saving Time* and *Execution Time*, which are average values in milliseconds that represent the amount of time needed by the Rules Engine for the *Saving Phase* and for the *Execution Phase*. All the sets of rules contained clones of the same basic Proactive Rule. This rule was designed specially to see how much time does the Rules Engine need to save an instance of all the rules that will run at the next iteration. This explains the relatively low values for the *Execution Time*.

For example, running 100 Proactive Rules on the smartphone took, in average, 436 milliseconds for one iteration. The time needed to save the rules for the next iteration took 329 milliseconds, representing 75% of the total amount of time of the iteration. This case is confirmed by the values obtained from running the same 100 Proactive Rules on the tablet, were the time for saving the rules for the next iteration took 83% of the total time of the iteration. The same results were obtained for *saving the rules in the* was also confirmed by the values obtained from running 300, 700 and 1000 rules. If for 100 rules it took 75% of the total iteration time for saving the rules in the database, for 1000 rules the percentage decreased to 60% of the total iteration time and stabilised around that value.

Significant differences for running an instance with 1000 rules, between the smartphone, table and PC, appear in the last column of Table I. If in the PC's case the total time for finishing the execution of one iteration took 220 milliseconds, which is quite fast, the same operation took almost twice more on the tablet and approximately 5 times more on the smartphone.

The results in Table I also meet our expectations in terms of computing capabilities of the involved systems. For instance, executing one iteration with 100 rules required approximately 4 times more time on the tablet than on the PC and more than 8 times more on the smartphone that on the PC. The difference did not change much when executing 300, 700 and 1000 rules. This is mainly due to the particular hardware configuration of each system as illustrated in Figure 5. The smartphone was equipped with a 1 Gigabyte (GB) of RAM and a quad-core 1.4 Gigahertz (GHz) processor, while the tablet, which had better performance results, was equipped with 2 GB of RAM and a dual-core 1.7 GHz processor. The PC had the best results as it had 8 GB of RAM and an i7 processor with 4 physical cores capable of operating at frequencies up to 3.4 GHz.

In conclusion, saving rules on the database took a lot of time in comparison with executing the rules. This may be

Device	Samsung Galaxy S3	Nexus	PC
Model	GT-I9300	10	
OS	Android 4.3	Android 4.4.4	Window7 64-bit
Baseband Compilation	I9300 XXUGNA8	KTU84P	
Kernel	3.0.31-2429075		
RAM	1 GB	2 GB	8 GB
Chipset	Exynos 4412 Quad	Exynos 5250	Intel 6 C200
CPU	Quad-core 1.4 GHz Cortex-A9	Dual-core 1.7 GHz Cortex A15	Intel Core i7-2600 3.4 GHz
GPU	Mali-400MP4	Mali-T604	
Sensors	Accelerometer, gyro, proximity, compass, barometer	Accelerometer, gyro, proximity, compass, barometer	None
WLAN	Wi-Fi 802.11, a/b/g/n, dual-band, Wi-Fi Direct, DLNA, DLNA, Wi-Fi hotspot	Wi-Fi 802.11, a/b/g/n, dual-band, Wi-Fi Direct, DLNA, DLNA, Wi-Fi hotspot	None

Figure 5. Hardware and software specifications of the devices used in the experiments

TABLE I. Iteration average time on different devices

#rules/iteration		100		300		700		1000	
Smartphone	Iteration Time (ms)	436.7		672.5		1099.8		1294.2	
	Saving Time (ms)	329 (75%)	Execution Time (ms) 107.7 (25%)	475 (71%)	197.5 (29%)	671.4 (61%)	428.4 (39%)	772.2 (60%)	522 (40%)
Tablet	Iteration Time (ms)	130.5		233.2		383.9		495.5	
	Saving Time (ms)	108.6 (83%)	Execution Time (ms) 21.9 (17%)	175.4 (75%)	57.8 (25%)	275.4 (71%)	108.5 (29%)	352.4 (71%)	143.1 (29%)
PC	Iteration Time (ms)	42.5		68.5		127.5		220	
	Saving Time (ms)	6.5 (15%)	Execution Time (ms) 36 (85%)	2 (3%)	66.5 (97%)	24.5 (19%)	103 (81%)	35.4 (16%)	184.6 (84%)

caused by the ORMLite package framework used for data storage on Android. On the other hand, the data storage on the PC was done with the help of MySQL [35] and Hibernate ORM [36] frameworks and needed far less time than for executing rules. A possible solution for avoiding time losses is to remove the feature of saving rules at each iteration and to set up a Saving Phase for the Rules Engine only once, e.g., at the shutdown event of the mobile devices. Another important conclusion is that running a big amount of rules on the smartphone and tablet is a time-consuming process and, for the moment, these devices are able to run only limited sets of Proactive Rules, while still being able to provide real-time services to their users. This amount of rules is directly related to the computing capabilities of each device and the libraries used to ensure different functionalities of the PEMD.

Other series of tests were conducted between the smartphone and the tablet to measure the duration of their communication. In Figure 2 from section III, we illustrated the steps necessary to send a message and to get the confirmation that the message has been successfully received. Both operations need 3 steps. We measured how much time it took for sending and receiving the message, meaning the first 3 steps of the communication process, between the two registered devices. After 10 tests, in average, it took 751.3 milliseconds for the following operations: the creation of the message on device 1, sending the messages to the relay server, sending the message to Google's Cloud Messaging server and then, receiving the message on device 2. The message used in the experiments was the same one as the one as presented in Figure 4. It contained specific instructions for activating a rule on device 2. The Relay Server was running on the PC with the software and hardware configurations shown in Figure 5. The entire communication process also depends on external factors like the network bandwidth and latency, and on the response time of Google's Cloud Messaging Server, which is also not part of our system and cannot be controlled.

Figure 6 and Figure 7 illustrate better the differences of running one iteration between the different devices. They include the average results of all the 10 sets of Proactive Rules that were performed on all 3 devices. The scale for the execution time of one iteration was kept on purpose to show the major difference between amount of time needed for one iteration with the saving phase and without the saving phase. In Figure 6 we can notice quite big fluctuations in the execution time of one iteration on the smartphone. It is not linear like in the tablet's case and the PC's case. If until running iteration of 800 rules the time increased in an expected manner, afterwards, it started to rise up quickly. It means that for iterations with more than 800 rules the smartphone is starting to consider the PEMD quite heavy in terms of the processing resources needed. It can also slow down other applications that need to access the same resources. In Figure 7 , however, the

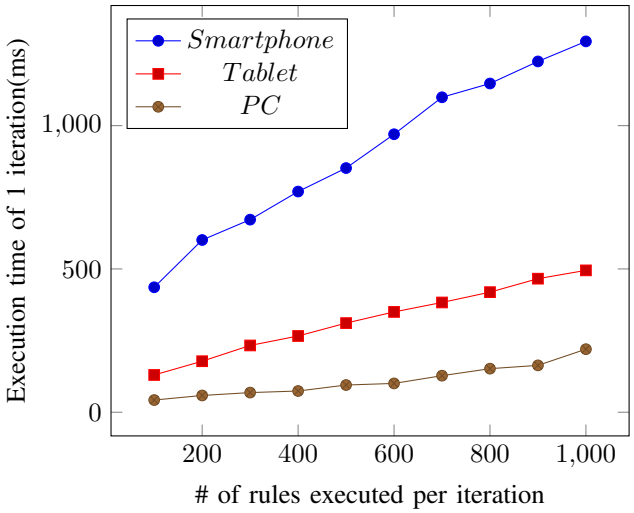

Figure 6. Total time of one iteration on all the devices when saving the rules at the end of the iteration

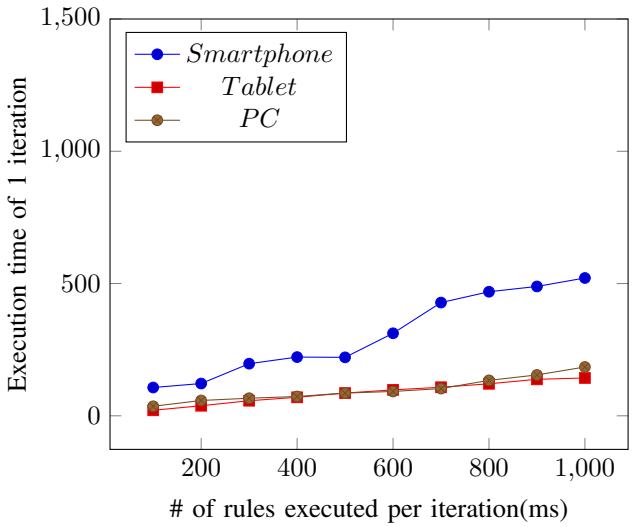

Figure 7. Iteration time on different devices without saving the rules at the end of the iteration

distribution of execution time without saving the rules in the database is increasing constantly, meaning that the duration time can be anticipated for various number of rules.

D. Battery Consumption

Mobile devices obtain the necessary energy for performing complex operations from their batteries ,which implies that analysing power consumption on these devices is very important. Our approach to calculate energy consumption was to measure the battery level on the smartphone using Android's internal system functions calls [37].

1) Benchmarks: We ran three types of benchmarks. The first one was designed for testing only the PEMD alone, which was executing sets of 100 Proactive Rules each 30 seconds. The Proactive Rule used in these tests was designed to simulated rules that would be used in different real-world applications. In the second benchmark, we simulated the interaction of a user with the screen of his/her smartphone by using a wakelock application that woke up the screen of the device, at 100% brightness, for a total of 18 minutes per hour. And, in the last benchmark, both the PEMD and the wakelock application were running simultaneously on the smartphone.

For all three benchmarks, 10 executions of 1 hour each were performed in order to compute their averages. During the tests the smartphone's Global Positioning System (GPS), Wireless fidelity (Wi-Fi), Bluetooth and the other mobile data connections were turned off.

2) Results: The results, shown in Table II, indicate first that executing Proactive Rules on a running PEMD, during 1 hour, takes only 1.5% of the total amount of the battery of the smartphone and second that a standard application consumes significantly more energy than a PEMD. The results of the third benchmark confirm the difference obtained between the first two benchmarks.

TABLE II. Average Battery Consumption

Applications	Average
PEMD	1.5%
Wakelock app	4.3%
Wakelock app + PEMD	5.4%

E. Discussions

Our tests were necessary for estimating the optimal number of Proactive Rules, which can be executed in one iteration by the Rules Engine on a smartphone and on a tablet, without having big delays. This aspect is very important when we want to design applications that can execute Proactive Rules on each device.

Our energy consumption analysis on the smartphone indicates that PEMDs do not take much battery consumption, which is a key aspect when developing models for smart applications.

In the future, the PEMDs should be able to allow multiple applications to execute concurrently Proactive Rules. This opens new perspectives and new challenges. If we take 5 applications with peak periods of approximately 250 rules instances for each we reach easily a number bigger than 1000 rules. And, if for now, running 1000 Proactive Rules on mobile devices does not raise performance issues, for bigger sets of Proactive Rules we could imagine slight problems. Solutions for this problem can be developed either by dividing rules into optimal sets of rules for running at each iteration or setting up a priority mechanism to distinguish between the crucial rules that need to be executed and the rules that can wait a couple of more iterations to be executed.

V. CONCLUSION AND FUTURE WORK

In this paper, we indicated the need of a rule-based engine for mobile devices capable of executing complex tasks like on the desktop computers. We also showed how few and limited possibilities are currently available for having rule-based systems on mobile devices. Then, we introduced a new

model capable of enhancing mobile devices with Proactive Computing properties and with support for executing collaborative applications. Our experiments indicate that Proactive Engines can be successfully integrated into mobile devices, that the model is able to run on different mobile devices and that the processes of our model are very efficient from a computational point of view and do not affect the overall performance of a device. Moreover, the performance of the PEMDs on smartphones and tablets was analysed and compared to the performance of PEs on desktop computers. A solution was provided for improving significantly the execution time of Proactive Rules by saving them when shutdown events occur instead of saving them during each iteration of the PEMD.

More tests are to be completed, to check the duration of the communication operation when multiple devices are involved in this process. For instance, we would like to know how much time it would take for a message to arrive to its destination when there are hundreds or thousands of this kind of operations performed at the same time. Also, future tests will include measurements to check how a device handles multiple receiving operations at the same time, like, for example, when more than 100 messages are sent to the same device at the same time. Also, our next evaluations will include services like Pushy instead of GCM for exchanging messages between the Proactive Engines, in order to check if these frameworks affect the overall performance of the mobile devices.

Future work will include developing smart applications capable of collaborating together and of performing joint complex actions. Our current work includes the development of a version of an PE for mobile devices running on iOS, and, on developing modified version of the Proactive Engine for wearable devices, which are capable of performing complex tasks, like the new generation of smartwatches.

REFERENCES

[1] I. D. C. I. W. M. P. Tracker., "Worldwide smartphone shipments edge past 300 million units in the second quarter," Tech. Rep., [Retrieved: May, 2015]. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS25037214

[2] D. James, "Recent advances in memory technology," in Advanced Semiconductor Manufacturing Conference (ASMC), 2013 24th Annual SEMI, May 2013, pp. 386–395.

[3] A. e. a. Wang, "10.3 heterogeneous multi-processing quad-core cpu and dual-gpu design for optimal performance, power, and thermal tradeoffs in a 28nm mobile application processor," in Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International, Feb 2014, pp. 180–181.

[4] N. Cacho, K. Damasceno, A. Garcia, T. Batista, F. Lopes, and C. Lucena, "Handling exceptional conditions in mobile collaborative applications: An exploratory case study," in Enabling Technologies: Infrastructure for Collaborative Enterprises, 2006. WETICE '06. 15th IEEE International Workshops on, June 2006, pp. 137–142.

[5] D. Tennenhouse, "Proactive computing," Communications of the ACM, vol. 43, no. 5, 2000, pp. 43–50.

[6] A. Salovaara and A. Oulasvirta, "Six modes of proactive resource management: a user-centric typology for proactive behaviors," in Proceedings of the third Nordic conference on Human-computer interaction. ACM, 2004, pp. 57–60.

[7] R.-A. Dobrican and D. Zampunieris, "Moving towards a distributed network of proactive, self-adaptive and context-aware systems," in ADAPTIVE 2014, The Sixth International Conference on Adaptive and Self-Adaptive Systems and Applications, 2014, pp. 22–26.

[8] D. Shirnin, S. Reis, and D. Zampunieris, "Experimentation of proactive computing in context aware systems: Case study of human-computer interactions in e-learning environment," in Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2013 IEEE International Multi-Disciplinary Conference on. IEEE, 2013, pp. 269–276.

[9] R. Dobrican, S. Reis, and D. Zampunieris, "Empirical investigations on community building and collaborative work inside a lms using proactive computing," in World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education, vol. 2013, no. 1, 2013, pp. 1840–1852.

[10] S. Coronado and D. Zampunieris, "Continuous proactivity in learning management systems," in Education Engineering (EDUCON), 2010 IEEE. IEEE, 2010, pp. 199–204.

[11] S. Marques Dias, S. Reis, and D. Zampunieris, "Personalized, adaptive and intelligent support for online assignments based on proactive computing," in Proceedings of the 12th IEEE International Conference on Advanced Learning Technologies, Rome, Italy 4-6 July, 2012. IEEE Computer Society Conference Publishing Services, 2012, pp. 668–669.

[12] D. Zampunieris, "Implementation of a proactive learning management system," in Proceedings of" E-Learn-World Conference on E-Learning in Corporate, Government, Healthcare & Higher Education", 2006, pp. 3145–3151.

[13] R.-A. Dobrican and D. Zampunieris, "Supporting collaborative learning inside communities of practice through proactive computing," in Proceedings of the 5th annual International Conference on Education and New Learning Technologies, Barcelona, Spain 1-3 July, 2013. IATED, 2013, pp. 5824–5833.

[14] J. Janeiro, T. Springer, and M. Endler, "A middleware service for coordinated adaptation of communication services in groups of devices," in Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on, March 2009, pp. 1–6.

[15] H. Chung, "Toward implementing a mobile collaborative system," in Systems and Informatics (ICSAI), 2012 International Conference on, May 2012, pp. 1248–1252.

[16] V. Sacramento, M. Endler, H. Rubinsztejn, L. Lima, K. Goncalves, F. Nascimento, and G. Bueno, "Moca: A middleware for developing collaborative applications for mobile users," Distributed Systems Online, IEEE, vol. 5, no. 10, Oct 2004, pp. 2–2.

[17] M. Caporuscio and P. Inverardi, "Yet another framework for supporting mobile and collaborative work," in Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on. IEEE, 2003, pp. 81–86.

[18] "Rule-based system jess." [Retrieved: May, 2015]. [Online]. Available: http://www.jessrules.com/jess/

[19] "Rule-based system clips," [Retrieved: May, 2015]. [Online]. Available: http://clipsrules.sourceforge.net/

[20] C. L. Forgy, "Expert systems," P. G. Raeth, Ed. Los Alamitos, CA, USA: IEEE Computer Society Press, 1990, ch. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, pp. 324–341, [Retrieved: May, 2015]. [Online]. Available: http://dl.acm.org/citation.cfm?id=115710.115736

[21] R. Dobrican and D. Zampunieris, "A proactive approach for information sharing strategies in an environment of multiple connected ubiquitous devices," in Proceedings of the International Symposium on Ubiquitous Systems and Data Engineering (USDE 2014) in conjunction with 11th IEEE International Conference on Ubiquitous Intelligence and Computing (UIC 2014). IEEE, 2014, pp. 763–771.

[22] S. Bobek, G. Nalepa, and M. layski, "Challenges for migration of rule-based reasoning engine to a mobile platform," in Multimedia Communications, Services and Security, ser. Communications in Computer and Information Science, A. Dziech and A. Czyewski, Eds. Springer International Publishing, 2014, vol. 429, pp. 43–57, [Retrieved: May, 2015]. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-07569-3_4

[23] C. Choi, I. Park, S. J. Hyun, D. Lee, and D. H. Sim, "Mire: A minimal rule engine for context-aware mobile devices," in Third IEEE International Conference on Digital Information Management (ICDIM), November 13-16, 2008, London, UK, Proceedings, 2008, pp. 172–177.

[24] "Ifttt (if this then that)," https://ifttt.com/, [Retrieved: May, 2015].

[25] N. Peers. Your online life made simpler, thanks to ifttt. [Retrieved: May,

2015]. [Online]. Available: http://blog.1and1.co.uk/2014/10/02/your-online-life-made-simpler-thanks-to-ifttt/

[26] G. J. Nalepa, "Architecture of the heart hybrid rule engine," in Proceedings of the 10th International Conference on Artifical Intelligence and Soft Computing: Part II, ser. ICAISC'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 598–605.

[27] S. Coronado and D. Zampunieris, "Towards a proactive learning management system using early activity detection," in SITE08-Society for Information Technology & Teacher Education International Conference. AACE, 2008, pp. 306–311.

[28] S. Marques Dias, S. Reis, and D. Zampunieris, "Proactive computing based implementation of personalized and adaptive technology enhanced learning over moodle (tm)," in Proceedings of the 12th IEEE International Conference on Advanced Learning Technologies, Rome, Italy 4-6 July, 2012. IEEE Computer Society Publications, 2012, pp. 674–675.

[29] "Ormlite - lightweight object relational mapping (orm) java package." [Retrieved: May, 2015]. [Online]. Available: http://ormlite.com/

[30] "Parse push," [Retrieved: May, 2015]. [Online]. Available: https://parse.com/products/push

[31] "Pubnub global data stream network," [Retrieved: May, 2015]. [Online]. Available: https://www.pubnub.com/

[32] "Urban airship," [Retrieved: May, 2015]. [Online]. Available: http://urbanairship.com/

[33] "Reliable push notifications," [Retrieved: May, 2015]. [Online]. Available: https://pushy.me/

[34] E. Nava, "Pushy a new alternative to google cloud messaging," January 2015, [Retrieved: May, 2015]. [Online]. Available: http://eladnava.com/pushy-a-new-alternative-to-google-cloud-messaging/

[35] "Mysql. my structured query language database management system." [Retrieved: May, 2015]. [Online]. Available: http://www.mysql.com/

[36] "Hibernate orm. hibernate object/relation mapping framework." [Retrieved: May, 2015]. [Online]. Available: http://hibernate.org/

[37] "Battery management, android developers." [Retrieved: May, 2015]. [Online]. Available: http://developer.android.com/reference/android/os/BatteryManager.html