An Experience with the Implementation of a Rule-Based Triggering Recommendation Approach for Mobile Devices

Sergio Ilarri
I3A, University of Zaragoza
Zaragoza, Spain
silarri@unizar.es

Irene Fumanal
University of Zaragoza
Zaragoza, Spain
758325@unizar.es

Raquel Trillo-Lado
I3A, University of Zaragoza
Zaragoza, Spain
raqueltl@unizar.es

ABSTRACT

In the current Big Data era, mobile context-aware recommender systems can play a key role to help citizens and tourists to make good decisions. Ideally, these systems should be proactive, able to detect the right moment and place to offer suggestions of a specific type of item or activity to the user. For this purpose, push-based recommender systems can be used, exploiting context rules to decide when a specific type of recommendation should be triggered.

However, experiences regarding the implementation of these types of systems are scarce. Motivated by this, in this paper, we describe our design and implementation efforts focusing on the ability to fire suitable recommendations, without user intervention, whenever it is required. In our proposal, the mobile user can activate, deactivate, parametrize, and define rules in an easy way, to obtain a better user personalization. Besides, the recommendation triggering is performed on the mobile device, which allows minimizing the amount of wireless communications and helps to protect the user's privacy (as context data is evaluated locally on the device, rather than by an external server). We have analyzed several technological options and evaluated the performance and scalability of our proposal, showing its feasibility.

CCS CONCEPTS

• **Information systems → Data management systems**; **Recommender systems**; • **Human-centered computing → Mobile computing**.

KEYWORDS

data management, mobile computing, mobile context-aware recommender systems, push-based recommendations, context rules, mobile devices

ACM Reference Format:
Sergio Ilarri, Irene Fumanal, and Raquel Trillo-Lado. 2021. An Experience with the Implementation of a Rule-Based Triggering Recommendation Approach for Mobile Devices. In *The 23rd International Conference on Information Integration and Web Intelligence (iiWAS2021), November 29-December 1, 2021, Linz, Austria.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3487664.3487806

1 INTRODUCTION

Modern citizens are facing unprecedented challenges for which the development of suitable data management techniques is key [16]. One area where software systems can help is by facilitating decision-making through the pre-selection of options that could be relevant to a user, filtering among a usually-overwhelming set of possible choices. Recommender systems (RS) [22] are specialized in learning user preferences and suggesting items (e.g., points of interest, hotels, restaurants, etc.) that fit those preferences.

More specifically, *Context-Aware Recommender Systems* (*CARS*) [1] extend the classical 2D $User \times Item$ paradigm of RS to a three-dimensional space $User \times Item \times Context$. The key idea is that considering the context of the user (e.g., his/her location, time of the day, weather conditions, etc.) can lead to recommendations that are more appropriate. Besides, in a scenario where the users are moving and interacting with a mobile device, proactive recommender systems (offering push-based recommendations when appropriate, without the need of an explicit user request) [12] are usually preferred over reactive recommender systems (pull-based recommendations) [8].

A basic building block required to build a proactive recommender system is a decision module that determines, based on the context of the user, whether a recommendation of a specific type should be triggered or not. For example, for a certain tourist visiting a city, it may be appropriate to trigger a restaurant recommendation if the user is wandering the city streets and it is lunch time. Suitable context rules could be defined, and customized according to the preferences of the user, in order to detect situations where a given recommendation process should be activated. Several technological options could be exploited for this purpose but, as far as we know, no recent study compares the existing options and evaluates a proposal concerning the development of recommender systems. Moreover, rule-based engines are usually assumed to be executed on servers rather than on mobile devices.

Motivated by this, in this paper we describe our experience with the definition and implementation of a proactive rule-based recommendation architecture. The structure of the rest of the paper is as follows. Firstly, in Section 2, we present the architecture that we have defined to tackle the problem of providing rule-based push-based recommendations in mobile environments. Secondly, in Section 3, we provide an overview of an analysis of some alternative technologies that could be used for the development of a prototype. Thirdly, in Section 4, we describe our prototype. Fourthly, in Section 5, we evaluate the performance and scalability of the proposal. Fifthly, in Section 6, we discuss some related works. Finally, in Section 7, we summarize our conclusions and outline some lines of future work.

2 ARCHITECTURE OF THE SYSTEM

In this section, we present a high-level overview of the architecture proposed to deploy recommender systems that provide proactive (push-based) recommendations to mobile users when appropriate, without requiring any user intervention. The architecture is based on our previous proposal presented in [12] (whose preliminary implementation was described in [15]), that we have extended to support the definition and evaluation of rules. As shown in Figure 1, the main novelty is the focus on the detection of circumstances that should trigger a recommendation process, through the use of a *Complex Event Detection* (*CEP*) module executing on the mobile device. The mobile device captures environment data, through the use of sensors of different types, and sends them to the CEP engine. When the device decides, based on the CEP engine, that a specific type of recommendation should be triggered, then it communicates it to an *Entity Manager* (*EM*), which is a server in charge of providing recommendations, to start the recommendation process; in fact, the mobile device could interact with several EMs at the same time, to communicate the recommendation request to them, in case the user is simultaneously within several recommendation *environments* (areas of influence controlled by an EM). Finally, the EM will return a set of recommended activities that may be filtered on the mobile device based on private user's data not communicated to the EM. In the following, we remark three key aspects of the proposal.

Firstly, we highlight the *execution of the recommendation triggering phase on the mobile device.* The decision about the type of recommendation that should be triggered (if any) is taken on the mobile device. For this reason, a CEP engine that can be executed on mobile devices is needed. The alternative would be to host the CEP engine on a fixed server on the network (e.g., in the EM), but in that case the device should constantly send context data changes to that server, to re-evaluate the conditions that determine if any rule should be activated; this would lead to an increase of wireless communications and therefore drain the battery life of the device due to a higher energy consumption.

Secondly, we mention *privacy preservation.* As the recommendation triggering phase is executed on the mobile device, there is no need to send context data to a server in order to decide whether a specific type of recommendation should be activated or not. Moreover, once a recommendation of a certain type has been triggered, the user is always in control of the data that he/she is willing to share with an external recommendation server (the EM). Thus, for example, shareable non-sensitive preferences can be exploited by the EM (and its recommendation engine) to perform a prefiltering, but private user's data (such as sensitive user preferences and/or sensitive context data) will not leave the mobile device and will be used only as a postfiltering step that will be executed on the device itself.

Finally, a third key feature is the support for *customization of the triggering behavior.* The user can define and customize his/her preferences regarding the activation of recommendations of different types. For this purpose, on the mobile device of the user it is possible to define two different types of rules: 1) *context rules* specify context conditions/situations that may be relevant to the user, and 2) *recommendation triggering rules* (or, for brevity, just *triggering rules*) specify when a certain type of recommendation should be automatically activated (based on the context rules).

As an example of the first type of rules, the user could define a location-based context rule called "In Millennium Park" (that is satisfied whenever the user is in Millennium Park in Chicago, i.e., around the GPS location <41.882702, -87.619392>), a location-based context rule "At home" (which evaluates to true if the user is at his/her home), a time-based context rule "Time to wake up" (indicating the time interval when the user is expected to wake up), or a time-based context rule "Lunch time" (representing the expected time interval for lunch, which could be set depending on the preferences of the user and habits of the region/country where the user is located). As an example of triggering rule, the user could define a rule "Lunch outside", that activates the recommendation of restaurants if it is lunch time and the user is not at home. It is possible to define several recommendation triggering rules to activate the same type of recommendation (e.g., the user could define two rules, "Lunch outside" and "Dinner outside", to activate a recommendation of restaurants for lunch time and also for dinner time). Besides, if the user wants to avoid receiving some types of recommendations at the same time, he/she can define *exclusion sets* (sets of types of recommendations that should not be activated at the same time) and their priorities within the set; for example, the user could define an exclusion set "FirstEatThenWatch" to avoid the simultaneous recommendation of restaurants and museums and giving higher priority to the recommendation of restaurants.

The two types of rules mentioned above have to be managed by a CEP engine running on the mobile device. Basically, the goal of the CEP engine is to detect high-level events that should trigger specific types of recommendations. In the proposed architecture, all the event detection rules are processed on the mobile device, which requires a CEP engine that can be completely executed on the mobile device. In the absence of such a technology, we would need to execute the CEP engine on a fixed server, which would require an efficient communication policy between the mobile device and the remote CEP engine: simpler rules could be executed on the mobile device in order to decide is a specific context variable change should be communicated (i.e., it could potentially trigger a recommendation process) or not. Another possibility would be to implement appropriate context variable update policies. Update policies have been extensively studied in the case of location data [27] (e.g., dead-reckoning policies send an update when the error of the predicted location obtained by considering the last communicated location and speed exceeds a certain threshold); for general sensor data, approaches like the one proposed in [17], based on the use of prediction functions, could be applied.

3 TECHNOLOGIES CONSIDERED

In this section, we describe the technologies that we have considered as potentially useful for our prototype.

3.1 Mobile Development Platforms

As described in [15], there are three main approaches that can be followed for the development of mobile apps:

- Develop a native mobile app, considering a specific platform (e.g., Android or iOS). The main advantage of this solution is

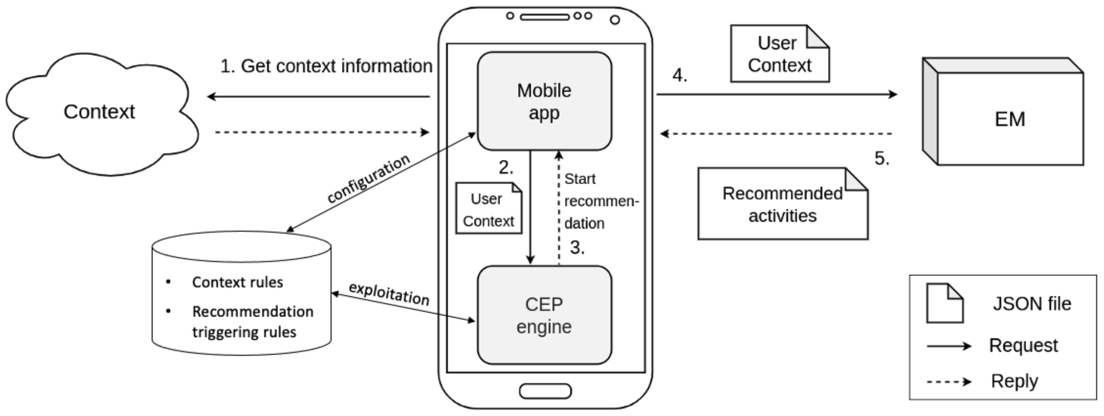


Figure 1: High-level architecture focused on the triggering of recommendations

that we can directly exploit the resources and functionalities of the target platform. The clear disadvantage is the need to develop a different implementation for each platform, that besides will need to be maintained independently.

- Develop a mobile web app, by exploiting the functionalities of standard web technologies (HTML5, CSS, and JavaScript). In this way, the solution developed could be run on any mobile platform with a suitable web browser.
- Develop a hybrid mobile app, using a mobile development framework (e.g., Apache Cordova, Ionic, React Native, Xamarin, Flutter, Framework7, or Appcelerator Titanium). This approach tries to find a balance between flexibility and development cost. It exploits web technologies but also allows access to platform-dependent functionalities (e.g., easy access to sensors and native look-and-feel).

Nowadays, using a development framework is a popular choice. The existing mobile development platforms usually support an easy access to sensors embedded in the mobile device, which obviously play an important role in mobile context-aware recommendation systems [14], thus facilitating the capture of context data. In [5], a recent comparative study evaluating the performance of several cross-platform mobile development frameworks is presented. Among these platforms, we have selected React Native [10] for our project, which is one of the most popular frameworks and offers nice functionalities like convenient programming using JavaScript and the React library, hot reloading (possibility to reload the app without recompiling, even keeping the same app's state), and support for wrapping native code. In an online survey questionnaire described in [4], React Native achieved the highest score in terms of framework interest.

3.2 Rule-Based Context Detection

Complex Event Processing (CEP) systems [6, 7, 11] allow the specification of rules for the detection of patterns, including complex event patterns (defined through aggregation and composition of other events). Our goal is to use these kinds of systems in our architecture in order to decide if a specific context is appropriate to trigger the recommendation of a certain type of item, by defining suitable context rules. In the following, we briefly discuss some technologies that could be considered for our purposes.

3.2.1 Esper. Esper (https://www.espertech.com/esper/) is one of the most well-known CEP technologies. It is open source and available for Java and .NET. It is well-document and, even though it appeared in 2006 (Esper 0.7.0 Alpha was released on January 16, 2006), new versions are still being released (the latest version at the time of writing is 8.7.0, released on January 27, 2021). It supports a rule-definition language based on SQL, called *Event Processing Language* (*EPL*), which facilitates the learning curve and provides a rich and easy-to-use syntax. In 2013, Esper was adapted for Android mobile devices, through the project *Asper* –Esper for Android– (https://github.com/mobile-event-processing/Asper), by addressing dependency restrictions due to missing Java packages and classes in Android. However, Asper has not been updated since then, and the current Asper version is based upon Esper 4.8.0 (released in March, 2013). This is a major inconvenience, since using Esper on Android would require considering a quite old (and outdated) version of Esper.

3.2.2 Siddhi. With a shorter history than Esper, Siddhi (https://siddhi.io/) [25] is another relevant open-source CEP technology (Java library), under Apache License. The first version of Siddhi available was published on April 24, 2015 (Siddhi 2.2.0 Release) and the last current version on March 19, 2021 (Siddhi Core Release 5.1.19). The project is well-documented and integrated with the open-source WSO2 Enterprise Integrator (WSO2 EI) as part of its *Streaming Integrator Tooling*. It supports defining rules using the language *SiddhiQL*, which has a syntax similar to SQL. It can be used as a Java or Python library and also as a microservice in Docker or Kubernetes. Besides, it provides a desktop tool (*WSO2 Integration Studio*), developed by WSO2, that can be used to define and debug rules, allowing the simulation of events to check that they work correctly. Siddhi can be run on an Android application. As an example, an approach using Siddhi for monitoring patients in a health context was presented in [9]. This use case illustrates the use of "the edge" to perform complex event detection autonomously: rather than continuously communicate sensor data, a complex event is communicated to the remote IoT Hospital Server (IHS), using the MQTT (Message Queuing Telemetry Transport) protocol, only when it is detected by the CEP engine running on the mobile device.

3.2.3 Apache Flink. Flink (https://flink.apache.org/) is an open-source data stream processing technology that includes a library to process complex events, called *Apache Flink CEP*. Flink is a very popular solution for the processing of data streams, including CEP support, but it is not specifically a CEP engine. Flink focuses on distributed data processing and has not been designed to be executed on mobile devices. As far as we know, it is not possible to execute it on Android devices. Besides, we are not aware of any approach trying to port Apache Flink CEP for Android.

3.2.4 Drools. Drools (https://www.drools.org/) is another popular Java-based open-source CEP technology or, as stated in its website, a Business Rules Management System (BRMS) solution. The language *DRL* (*Drools Rule Language*) is used to define the required business rules in .drl text files. As usual, rules are composed, at least, by conditions (when) and actions (then). According to [26], the use of resources by Drools is higher than in the case of Esper or Siddhi. Besides, it is not possible to directly execute Drools on an Android mobile device. However, in [23], which focuses on smart home systems, the authors mention the use of a ported version of Drools for Android; this seems to be an ad hoc solution applied by the authors and not publicly available. Porting issues with Drools, regarding the use of memory, are mentioned in [20].

3.2.5 Microsoft StreamInsight. Microsoft StreamInsight (https://docs.microsoft.com/en-us/archive/msdn-magazine/2012/march/microsoft-streaminsight-building-the-internet-of-things) is a commercial and well-known CEP technology. In this case, this solution is not open source and a SQL Server license is required to use it. Rather than defining rules, queries can be processed over the streaming data using the LINQ (.NET Language-Integrated Query) language. This technology is nowadays part of Microsoft SQL Server and cannot be executed directly on mobile devices.

3.2.6 Conclusions: Chosen CEP Engine. Our main goal is to have a CEP engine that can be executed on a mobile device. This is a requirement only satisfied by Asper and Siddhi (the other technological solutions could be used in case we adopted a different architecture where the rule detection could be performed on the EM, as briefly mentioned at the end of Section 2). To choose between Asper and Siddhi, we developed a simple mobile app in Android, with three simple buttons: one to start the CEP engine, another one to stop the CEP engine, and finally a third one to send an event to the CEP engine. We can note the following:

- The sample mobile app was successfully implemented by adding the Asper library to an Android Studio project. However, as mentioned in Section 3.2.1, Asper is based on Esper 4.8, which is very old. We tried to overcome this difficulty by testing if more recent versions of Esper (Esper 7.1 y 8.7) were directly compatible with Android (without using Asper). However, we noticed problems with dependencies (required classes that were not available in Android).
- The mobile app was also implemented, with no problems, using Siddhi. We tried with both the latest version of Siddhi and also with version 4. It would have been useful to be able to use and define custom functions in JavaScript to process some context data (e.g., to compare location coordinates and determine if they are equal), to be used as filters in the rules, but unfortunately Siddhi's JavaScript Extension, that allows to do this, is not well supported in Android (because some classes are not available in Android); therefore, we had to define the rules and the required computations using a more complex/verbose syntax using SiddhiQL.

There are some recent relevant surveys where other CEP technologies are analyzed [7, 11], but with no focus on their execution on mobile devices to be applied in the context of mobile recommender systems.

4 PROTOTYPE

In this section, we describe our prototype, focusing on the integration of the CEP engine, using Siddhi, and on the functionalities related with the definition and personalization of user rules. We have built our solution considering our previous work [15]. With the same spirit, we also use React Native (https://reactnative.dev/) [10] and the main code of the mobile app is written in JavaScript. Siddhi's engine can consume events coming from different sources and process them as required by the application. Once the events have been processed, it provides the results through different channels, such as HTTP, Kafka, or email. In our case, we use the Siddhi library to send and receive events using Java code.

We deployed Siddhi as an Android service, which allows executing short or long-term duration operations in the background, without affecting the Graphical User Interface (GUI) of the mobile app. There are three types of services in Android: foreground services, background services, and bound services. Specifically, we have used a bound service because it defines a client/server interface that supports interacting with the service.

The main code of our mobile app is written in JavaScript, but we need to use Android's native code to access Siddhi. To address this difficulty, we have created an Android Native Module (called *SiddhiClientModule*) operating as a client of the Siddhi engine: it invokes Siddhi's operations (e.g., start the engine, stop it, send an event, or retrieve a result) from the main code of the mobile app in JavaScript. More specifically, we distinguish three types of native classes: 1) *SiddhiAppManager* handles the communication between the app and Siddhi's engine, 2) *SiddhiService* implements the logic of the bound service, and 3) *SiddhiClientModule* is the native module that acts as a communication bridge between the Android's code and React Native's code (the operations described in this module can be called from the JavaScript code of the mobile app).

With a specified *context update period* (e.g., every 30 seconds), the mobile app will send the context information to Siddhi's engine (this communication could be avoided in case there is no context change). Besides, the mobile app also needs to listen to potential results from Siddhi, that could indicate the need of triggering a recommendation process of a certain type. For this purpose, we use *Headless JS* (which allows to run tasks in JavaScript while the app is executing in the background), supported by React Native, to define a service that periodically sends the user's context data to Siddhi (*SendContextTask*) and another service that listens to results provided by Siddhi (*ListRecommendationResultTask*).

Siddhi's engine only reports recommendation triggering rules that get activated. When a triggering rule is activated (i.e., when the current context matches the rule's conditions), an event is stored in a *Result* stream, which is retrieved by using a ReactNative's *callback*, as indicated in the official documentation of Siddhi concerning its use as a Java library. Instead of directly considering the type of recommendation associated to the rule that has been fired, the mobile app collects all the types of recommendations corresponding to recommendation triggering rules fired within a specific batch time window (e.g., 5 seconds by default in our current prototype). In this way, an example of batched output obtained could be <contextId, {restaurantRecommendation, museumRecommendation}>. Then, the batched output retrieved is processed to determine which of those types of recommendations should actually be initiated, based on the possible exclusion sets defined by the user and the priority of types of recommendations within each set (as explained in Section 2). To implement this batching functionality, we use the concept of *batch windows* provided by Siddhi, applying Siddhi's *timeBatch* function. An auxiliary thread is created by the *SiddhiClientModule* to obtain the results from Siddhi, as its executing thread should not be busy executing a task that could potentially require some time to complete.

We show some examples of screenshots in Figures 2 and 3. On the left of Figure 2, we show the screen that allows the definition, modification, deletion, and temporary deactivation of recommendation triggering rules. In the middle of Figure 2, we show an example of recommendation triggering rule defined by the user: he/she wants the recommendation of restaurants to be activated when it is lunch time and he/she is not at home. On the right of Figure 2, we show the screen that allows the definition and modification of context rules. These conditions correspond with context rules that the user can customize according to his/her specific needs, as shown in Figure 3; regarding the left screen in Figure 3, notice that using the button "Get current GPS location" may return an estimated location value (or the last known location) when the user is indoors (using positioning mechanisms other that the GPS, like the WiFi network the user is connected to). Another example of triggering rules supported (not shown in Figure 3) are calendar-based rules.

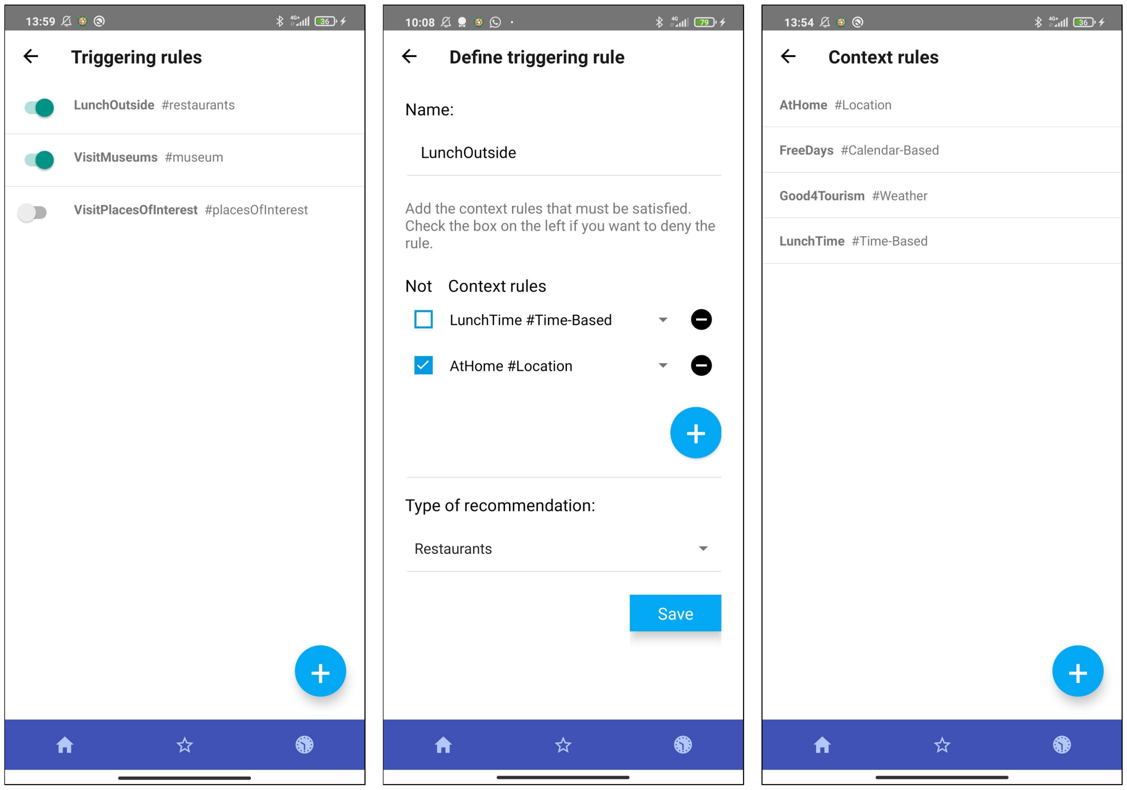
5 EXPERIMENTAL EVALUATION

We have performed some experiments to measure the performance of the recommendation triggering phase, which implies executing the Siddhi engine on a mobile device. Specifically, we have used a mobile device with a Qualcomm Snapdragon 626 processor (octa-core A53, 2,2 GHz), 4 GB RAM, and Android version 8.1.0.

First, we have defined a set of context rules based on 7 context variables: *dayOfTheWeek* (with possible values Monday, Tuesday, etc.), *season* (winter, spring, summer, or autumn), *partOfTheDay* (early morning, morning, afternoon, or night), *timeForDailyActivity* (time for lunch or time for dinner), *weatherStatus* (cloudy, clear, raining, etc.), *atHome* (with value true if the user is at home and false otherwise), and *onHolidays* (that indicates if the user is currently enjoying a holiday period). Then, we have defined a Python script that automatically defines recommendation triggering rules by randomly combining the previous context rules defined. The generation process avoids combining contradicting context rules in a single triggering rule (e.g., weatherStatus=clear AND weatherStatus=cloudy), as this would make no sense and the rule would be never satisfied. Each triggering rule has initially a single context rule; then, iteratively, with 25% probability, another context rule is added and with 75% probability no rule is added; this is repeated until it is not possible to add more non-contradicting context rules or the random decision taken at the last iteration does not add a new context rule. Following this process, 300 recommendation triggering rules have been synthetically generated for the experimental evaluation (each with a number of context rules between 1 and 7); in our experiments, the resulting average number of context rules per triggering rule is 3.36. Afterwards, by using another Python script, we have randomly generated a set of example contexts (that define values for each of the possible context variables that are used in the context rules) in JSON format, as required by Siddhi. 21 different contexts have been synthetically generated for the experiments.

Finally, we have performed the experiments. For each experiment, we define in the mobile device's app all the context rules, a subset of the recommendation triggering rules (the first *n* rules, depending on the number of recommendation triggering rules to consider in that particular experiment), and simulate context changes by sending to Siddhi each context defined (one context update every 20 seconds); then, we measure the *triggering latency* of each type of recommendation activated with each change of context, defined as the time elapsed between the context change and the time instant when the need to trigger a specific type of recommendation is detected. We measure this value using the module *SiddhiAppManager* (described in Section 4). As a change of context can activate several triggering rules at the same time, we collect the triggering latency of each single recommendation triggering rule activated by each change of context. In the following, we report the results obtained when considering the 21 context changes simulated.

On the left of Figure 4, we show the average triggering latency as well as the minimum triggering latency (triggering latency of the first rule activated by a context change) and the maximum triggering latency (triggering latency of the last rule activated by a context change). It can be seen that the highest triggering latency increases considerably with the number of recommendation triggering rules, as expected; however, the average triggering latency increases only very slowly; finally, the lowest latency is not affected by the number of recommendation triggering rules, as Siddhi checks the rules one by one, and therefore the cost of the first activation should be similar independently of the number of rules. It must be stressed that the latencies measured are always very small, not exceeding 220 milliseconds even in the worst case, that arises when the number of recommendation triggering rules is very high (200 recommendation triggering rules); we think that it is very unlikely that a mobile user will define such a high number of recommendation triggering rules and, even if he/she does it, the latencies will be anyway acceptable. On the right of Figure 4, we show the number of recommendation triggering rules activated depending on the number of recommendation triggering rules defined: obviously, the higher the number of recommendation triggering rules defined, the higher the expected number of rules that could be activated by a context change.



Triggering rules

LunchOutside #restaurants

VisitMuseums #museum

VisitPlacesOfInterest #placesOfInterest

Define triggering rule

Name:

LunchOutside

Add the context rules that must be satisfied. Check the box on the left if you want to deny the rule.

Not Context rules

☐ LunchTime #Time-Based

☑ AtHome #Location

Type of recommendation:

Restaurants

Save

Context rules

AtHome #Location

FreeDays #Calendar-Based

Good4Tourism #Weather

LunchTime #Time-Based

Figure 2: Screen to manage triggering rules (left), to define a new triggering rule (middle), and to manage context rules (right)

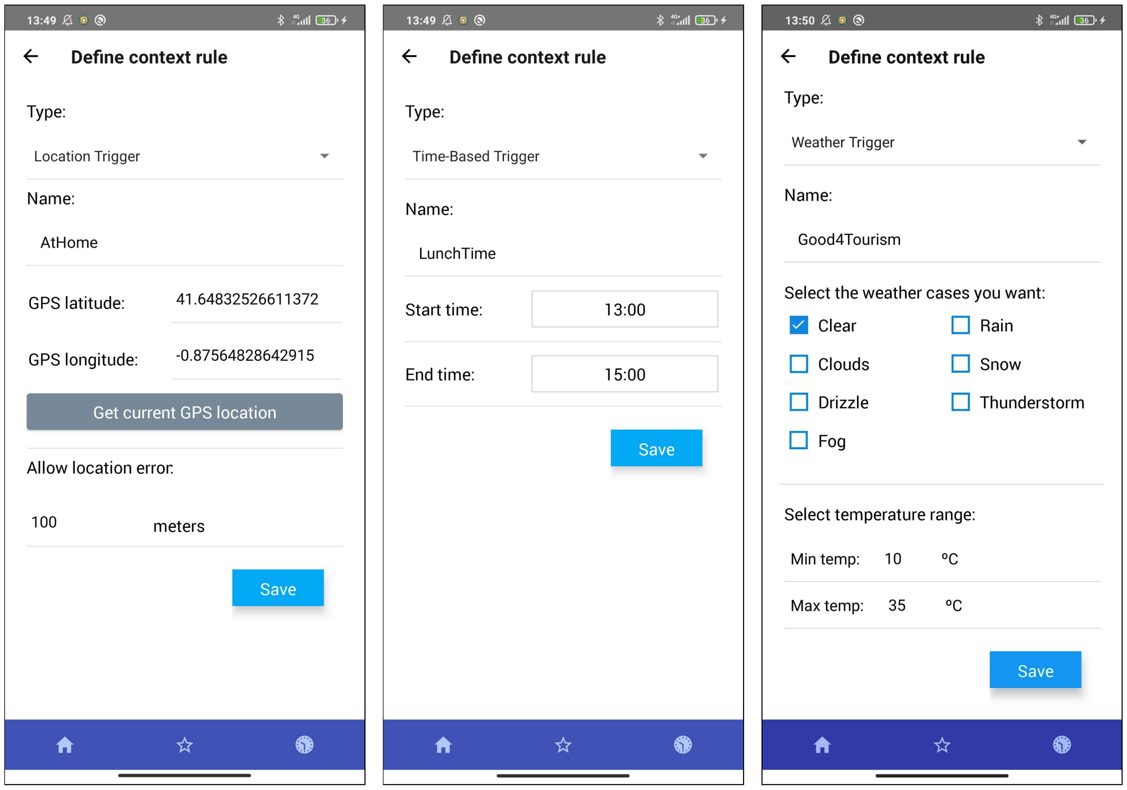
6 RELATED WORK

As described in Section 1, this work belongs to the field of recommender systems [22], and more specifically context-aware recommender systems [1]. Whereas a significant amount of research has been developed in the field of context-aware computing [2] and also in the field of context-aware recommender systems (CARS), it is more difficult to find practical experiences regarding the implementation of CARS using existing technologies and tools. The most related work is our previous proposal [15], based on the theoretical foundations described in [12], where we did not focus on the recommendation triggering phase, that is the core of this paper. Besides, in [13], we described some use case scenarios and presented some examples of context attributes and rules (with a SWRL-like syntax) that could be considered to trigger several types of recommendations; however, in [13] we did not carry out any practical implementation or testing.

In the proposal presented in this paper, we empower the user to be in control of the data he/she shares with external servers (thus protecting his/her privacy) and also to define and customize the context rules and recommendation triggering rules that are appropriate to him/her. The work presented in [24] also proposes the use of user-defined rules to activate recommendations (an example of rule shown in [24] is "If day is Monday then I would like to go to a Restaurant which is closer than 5 KM"): the users use a web form to specify the rule, which is then translated to a RuleML rule; however, that paper presents a model, which does not seem to be supported by an implementation/prototype.

The use of user-defined rules has also been proposed in other fields not related to recommender systems. For example, for the detection of network security situations, rules in SWRL are defined to compensate for the limited description ability of an ontology and improve the reasoning ability of the proposed model [28]. As another example, *SECE (Sense Everything, Control Everything)* [3] is a rule-based context-aware system that supports defining rules in a natural English-like formal language to compose different types of services based on events; an important difference with our work, besides belonging to a different research area, is that SECE is a web service (not something to be executed on a mobile device). As a final example, *LLA (Long-Life Application)* [18, 19] is a single context-aware distributed mobile application dedicated to everyday users; the idea is to inject the desired *situations* (context descriptions) and then matching situations to services in order to respond to contextual changes when it is suitable; appropriate situations may be injected by the user himself/herself, external providers (such as governments, business and private companies, and institutions/organizations/associations), or even a component of LLA that monitors the social environment of the user (e.g., tasks defined in Google Calendar) to suggest possible relevant situations.

There are also some rule-based end-user applications worth mentioning. For example, *IFTTT (If This, Then That)* [21] relies on a web service that allows automating some tasks and actions


13:49

Define context rule

Type: Location Trigger

Name: AtHome

GPS latitude: 41.64832526611372

GPS longitude: -0.87564828642915

Get current GPS location

Allow location error: 100 meters

Save

13:49

Define context rule

Type: Time-Based Trigger

Name: LunchTime

Start time: 13:00

End time: 15:00

Save

13:50

Define context rule

Type: Weather Trigger

Name: Good4Tourism

Select the weather cases you want:

☑ Clear	☐ Rain
☐ Clouds	☐ Snow
☐ Drizzle	☐ Thunderstorm
☐ Fog	

Select temperature range:

Min temp: 10 ºC

Max temp: 35 ºC

Save

Figure 3: Screen to define a location-based context rule (left), a time-based context rule (middle), and a weather context rule (right)

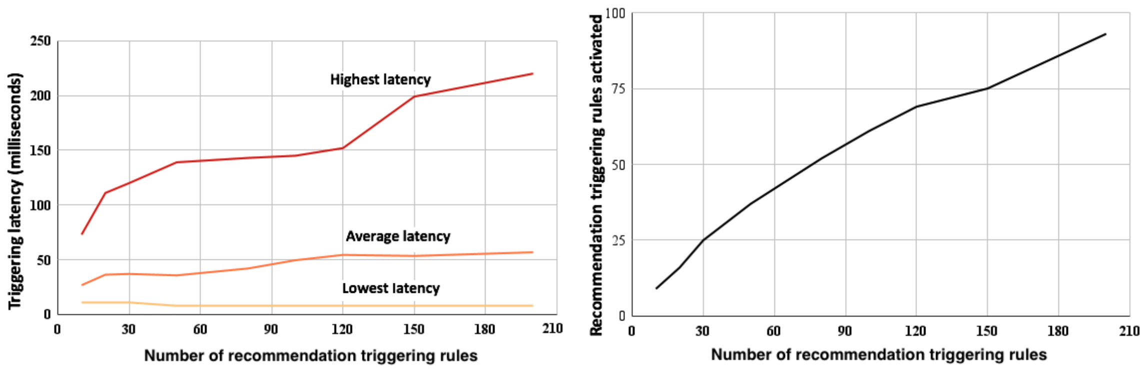


Figure 4: Evaluation of the triggering latency (left) and number of rules activated (right)

by connecting apps (e.g., the automatic publication in Twitter of a picture that the user has posted in Facebook, to relieve the user from the extra work of having to upload the same picture to several sites). The user can define those actions through the IFTTT's website or by using the IFTTT mobile app (available for Android and iOS). According to the web page of IFTTT, currently over 600 apps and devices work with IFTTT (e.g., apps like Twitter, Telegram, Google Drive, Gmail, and devices like Google Home, Alexa, etc.). Similarly, in the *Shortcuts* app (https://support.apple.com/en-gb/guide/shortcuts/), available for iOS devices, it is possible to define the conditions under which a specific task (shortcut) should run automatically (e.g., based on the time of the day or the location); specifically, four different types of triggers can be considered: event triggers (a specific time of the day, the use of an alarm on the user's phone, or an Apple Watch workout done by the user), travel triggers (when the user arrives in a location or leaves it, when the user connects or disconnects the CarPlay (https://www.apple.com/ios/carplay), or when a commute by the user is going to occur), communication triggers (reception of specific types of emails or messages), and setting triggers (related to modifications like activating the airplay mode, changes in the battery level, connecting to a WiFi network, etc.).

Finally, as mentioned in Section 3.2, there are some surveys that analyze CEP technologies [7, 11], but with no focus on their execution on mobile devices to build proactive recommender systems. Thus, as far as we know, existing works mainly consider the execution of CEP engines on fixed servers, rather than on mobile devices. However, as explained in this paper (and particularly in Section 2), executing the recommendation triggering phase on the mobile device (rather than on a server) is expected to reduce the amount of wireless communications from the mobile device (communications of context updates) and thus the energy consumption; besides, this approach helps to keep the privacy of the users.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we have described our experience with the development of a proactive push-based recommendation architecture, with an emphasis on the definition and triggering of context rules. A key distinct advantage of the proposed architecture is that the rule-based engine is executed on the mobile device, which improves the performance and maximizes the user's privacy (as the context is evaluated locally on the device). Besides, our flexible proposal gives full power to the user, by allowing him/her to define and customize his/her own rules. We have analyzed the technologies that could be used for this purpose and evaluated the performance and scalability of our proposal, showing its feasibility.

Concerning the prototype, we are currently finishing the integration of some functionalities of our previous prototype [15] (that did not focus on the recommendation triggering phase, as we do in this paper). Furthermore, we have recently carried out a survey with users to identify types of context rules and recommendation triggering rules that they would find useful (we have received 85 answers at the time of writing), which can help us to refine our prototype by defining a more complete customizable rule library. Finally, the possibility to develop a prototype that works on iOS could be analyzed in more detail in the future.

ACKNOWLEDGMENTS

Research supported by the Government of Aragon – Aragon-New Aquitaine project PASEO 2.0 (AQ-8), the project PID2020-113037RB-I00 / AEI / 10.13039/501100011033, and the *Departamento de Ciencia, Universidad y Sociedad del Conocimiento del Gobierno de Aragón* (Government of Aragon: group reference T64_20R, COSMOS research group).

REFERENCES

[1] Gediminas Adomavicius, Bamshad Mobasher, Francesco Ricci, and Alexander Tuzhilin. 2011. Context-aware recommender systems. *AI Magazine* 32, 3 (Fall 2011), 67–80. https://doi.org/10.1609/aimag.v32i3.2364

[2] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. 2007. A Survey on Context-Aware Systems. *International Journal of Ad Hoc and Ubiquitous Computing* 2, 4 (June 2007), 263–277. https://doi.org/10.1504/IJAHUC.2007.014070

[3] Victoria Beltran, Knarig Arabshian, and Henning Schulzrinne. 2011. Ontology-Based User-Defined Rules and Context-Aware Service Composition System. In *Extended Semantic Web Conference (ESWC 2012) Workshops*. Lecture Notes in Computer Science (LNCS), Vol. 7117. Springer, Berlin, Heidelberg, 139–155. https://doi.org/10.1007/978-3-642-25953-1$_1$2

[4] Andreas Biørn-Hansen, Tor-Morten Grønli, Gheorghita Ghinea, and Sahel Alouneh. 2019. An Empirical Study of Cross-Platform Mobile Development in Industry. *Wireless Communications and Mobile Computing* 2019 (January 2019), 1–12. https://doi.org/10.1155/2019/5743892

[5] Andreas Biørn-Hansen, Christoph Rieger, Tor-Morten Grønli, Tim A. Majchrzak, and Gheorghita Ghinea. 2020. An empirical investigation of performance overhead in cross-platform mobile development frameworks. *Empirical Software Engineering* 25, 4 (June 2020), 2997–3040. https://doi.org/10.1007/s10664-020-09827-6

[6] Gianpaolo Cugola and Alessandro Margara. 2012. Processing Flows of Information: From Data Stream to Complex Event Processing. *Comput. Surveys* 44, 3, Article 15 (June 2012), 62 pages. https://doi.org/10.1145/2187671.2187677

[7] Miyuru Dayarathna and Srinath Perera. 2018. Recent Advancements in Event Processing. *Comput. Surveys* 51, 2, Article 33 (February 2018), 36 pages. https://doi.org/10.1145/3170432

[8] María del Carmen Rodríguez-Hernández and Sergio Ilarri. 2016. Pull-Based Recommendations in Mobile Environments. *Computer Standards & Interfaces* 44 (February 2016), 185–204. https://doi.org/10.1016/j.csi.2015.08.002

[9] Amarjit Singh Dhillon, Shikharesh Majumdar, Marc St-Hilaire, and Ali El-Haraki. 2018. A Mobile Complex Event Processing System for Remote Patient Monitoring. In *IEEE International Congress on Internet of Things (ICIOT 2018)*. IEEE, USA, 180–183. https://doi.org/10.1109/iciot.2018.00034

[10] Bonnie Eisenman. 2015. *Learning React Native: Building Native Mobile Apps with JavaScript.* O'Reilly Media, Sebastopol, California, USA.

[11] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos Garofalakis. 2019. Complex event recognition in the Big Data era: a survey. *The VLDB Journal* 29, 1 (July 2019), 313–352. https://doi.org/10.1007/s00778-019-00557-w

[12] Ramón Hermoso, Sergio Ilarri, Raquel Trillo, and María del Carmen Rodríguez-Hernández. 2015. Push-based recommendations in mobile computing using a multi-layer contextual approach. In *13th International Conference on Advances in Mobile Computing and Multimedia (MoMM 2015)*. ACM, New York, NY, USA, 149–158. https://doi.org/10.1145/2837126.2837128

[13] Ramón Hermoso, Sergio Ilarri, and Raquel Trillo-Lado. 2018. Proactive Mobile CARS in Action: A First Step Towards Making Sense of Context Rules. In *13th International Workshop on Semantic and Social Media Adaptation and Personalization (SMAP 2018)*. IEEE, USA, 69–74. https://doi.org/10.1109/SMAP.2018.8501879

[14] Sergio Ilarri, Ramón Hermoso, Raquel Trillo-Lado, and María del Carmen Rodríguez-Hernández. 2015. A Review of the Role of Sensors in Mobile Context-Aware Recommendation Systems. *International Journal of Distributed Sensor Networks* 2015 (November 2015), 1–30. https://doi.org/10.1155/2015/489264

[15] Sergio Ilarri and Manuel Herrero. 2018. Towards the Implementation of a Push-Based Recommendation Architecture. In *13th International Workshop on Semantic and Social Media Adaptation and Personalization (SMAP 2018)*. IEEE, USA, 87–92. https://doi.org/10.1109/SMAP.2018.8501875

[16] Sergio Ilarri, Raquel Trillo-Lado, and Thierry Delot. 2020. Social-Distance Aware Data Management for Mobile Computing. In *18th International Conference on Advances in Mobile Computing & Multimedia (MoMM 2020)*. ACM, New York, NY, USA, 138–142. https://doi.org/10.1145/3428690.3429164

[17] Sergio Ilarri, Ouri Wolfson, Eduardo Mena, Arantza Illarramendi, and Prasad Sistla. 2009. A Query Processor for Prediction-Based Monitoring of Data Streams. In *12th International Conference on Extending Database Technologies (EDBT 2009)*, Vol. 360. ACM, New York, NY, USA, 415–426.

[18] Riadh Karchoud, Arantza Illarramendi, Sergio Ilarri, Philippe Roose, and Marc Dalmau. 2017. Long-Life Application – Situation Detection in a Context-Aware All-in-one Application. *Personal and Ubiquitous Computing* 21, 6 (December 2017), 1025–1037. https://doi.org/10.1007/s00779-017-1077-2

[19] Riadh Karchoud, Philippe Roose, Marc Dalmau, Arantza Illarramendi, and Sergio Ilarri. 2019. One App to Rule Them All: Collaborative Injection of Situations in an Adaptable Context-Aware Application. *Journal of Ambient Intelligence and Humanized Computing* 10 (December 2019), 4679–4692. Issue 12. https://doi.org/10.1007/s12652-018-0846-8

[20] Chinmoy Mukherjee. 2017. *Build Android-Based Smart Applications: Using Rules Engines, NLP and Automation Frameworks.* Apress, USA.

[21] Steven Ovadia. 2014. Automate the Internet With "If This Then That" (IFTTT). *Behavioral & Social Sciences Librarian* 33, 4 (2014), 208–211. https://doi.org/10.1080/01639269.2014.964593

[22] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. 2011. *Recommender systems handbook.* Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-0-387-85820-3

[23] Markus Schinle, Johannes Schneider, Timon Blöcher, Jochen Zimmermann, Sebastian Chiriac, and Wilhelm Stork. 2017. A Modular Approach for Smart Home System Architectures Based on Android Applications. In *Fifth IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud 2017)*. IEEE, USA, 153–156. https://doi.org/10.1109/MobileCloud.2017.20

[24] Silky Sharma and Damandeep Kaur. 2015. Location based context aware recommender system through user defined rules. In *International Conference on Computing, Communication Automation (ICCCA 2015)*. IEEE, USA, 257–261. https://doi.org/10.1109/CCAA.2015.7148384

[25] Sriskandarajah Suhothayan, Kasun Gajasinghe, Isuru Loku Narangoda, Subash Chaturanga, Srinath Perera, and Vishaka Nanayakkara. 2011. Siddhi: A Second

Look at Complex Event Processing Architectures. In *2011 ACM Workshop on Gateway Computing Environments (GCE 2011)*. ACM, New York, NY, USA, 43–50. https://doi.org/10.1145/2110486.2110493

[26] Kenny Warszawski. 2020. *Complex Event Processing for Internet of Things: Open-Source Frameworks Analysis*. Master's thesis. University of Namur (Belgium). Available at https://researchportal.unamur.be/en/studentTheses/complex-event-processing-for-internet-of-things [Accessed: October 3, 2021].

[27] Ouri Wolfson, A. Prasad Sistla, Sam Chamberlain, and Yelena Yesha. 1999. Updating and Querying Databases that Track Mobile Units. *Distributed and Parallel Databases* 7, 3 (1999), 257–387. https://doi.org/10.1023/A:1008782710752

[28] Guangquan Xu, Yan Cao, Yuanyuan Ren, Xiaohong Li, and Zhiyong Feng. 2017. Network Security Situation Awareness Based on Semantic Ontology and User-Defined Rules for Internet of Things. *IEEE Access* 5 (2017), 21046–21056. https://doi.org/10.1109/ACCESS.2017.2734681