2013 19th International Conference on Control Systems and Computer Science

A Platform to Support Context-Aware Mobile Applications

Ciprian Dobre*
*University Politehnica of Bucharest
Bucharest, Romania
E-mail: ciprian.dobre@cs.pub.ro

***Abstract*—Today smartphones and tablet PCs are gaining more popularity due to cutting edge technology added on top of wearability, thus creating the scene for context-aware applications that are capable to sense and actively use context to provide the user with valuable information whenever and wherever is needed, even while on the move. In this paper we describe CAPIM, a platform designed to support the construction of such context-aware mobile applications. The platform provides capabilities for sensing and collecting data from sensors and external sources. It includes a layer where the raw context data is aggregated and derived into higher-level information. A dedicated rule execution engine is offered to support context-aware workflows. CAPIM integrates context-aware services that are dynamically configurable and use the user's location, identity, preferences, profile, and relations with individuals, as well as capabilities of the mobile devices to aggregate and semantically organize the context data. They react based on dynamically defined context-oriented workflows. We present the platform's architecture, implementation details, and present case study scenarios which show its potential to handle a variety of context-aware situations. CAPIM is fully functional and can be used in a variety of context-aware situations.**

***Index Terms*—context-awareness, pervasive adaptation, social service adaptation**

I. Introduction

Today smartphones are becoming commodity hardware. They are seen everywhere, as more people realize that having more sensing and computing capabilities in every-day situations is attractive for many reasons. Smartphones are in fact already used to optimize (e.g. by helping organizing tasks, contacts, etc.) and assist (e.g. with navigation, find information more quickly, access online data, etc.) users with their everyday activities. Their success is the basis for a shift towards developing mobile applications that are capable to recognize and pro-actively react to user's own environment. Such context-aware mobile applications can help people better interact between themselves and with their surrounding environments. This is the basis for a paradigm where the context is actively used by applications designed to take smarter and automated decisions: mute the phone when user is in meeting, show relevant information for the user's current location, assist the user find its way around a city, or automatically recommend events based on the user's (possibly learned) profile and interests.

CAPIM (Context-Aware Platform using Integrated Mobile services) is a platform designed to support the construction of such next-generation applications. It integrates services designed to collect context data (location, user's profile and characteristics, as well as the environment). These smart services are dynamically loaded by mobile clients, and make use of the sensing capabilities provided by modern smartphones, possibly augmented with external sensors. The data is collected and aggregated into context instance. This is also possible augmented with external and inferred data about possible situations, relations, or other events.

In addition, the platform includes a workflow engine designed to continuously evaluate the context and take automated decisions or actions based on customized rules and particular context events. We describe how such rules are constructed further within this paper. We also present CAPIM's visualization layer that allows intuitive and easy interaction for the user with the platform and its running services, but also with the user's environment.

The rest of the paper is structured as follows. Section II presents related work. In Section III we present CAPIM's architecture, and Section IV presents the context model. Section V presents examples of possible scenarios and applications. In Section VI we conclude and present future work.

II. Related Work

Several platforms for pervasive and context-aware systems to support rich contextual features were built in the past few years. Several of these systems are openly available. However, no such systems are available for devices that are inexpensive, available off-the-shelf, and widely accepted by users in their everyday life, such as smart phones, the first real-world pervasive platform.

MobiPADS [1] is a middleware for mobile environments. It consists of Mobilets, entities providing particular services that are able to migrate between MobiPADS environments. Each Mobilet consists of a slave and a master. The slave resides on a server, while the master resides on a mobile device. Each pair cooperates to provide a specific service. MobiPADS is concerned with internal context of the mobile devices, which is used to adapt to changes in the computational environment. Thus, context types include: processing power, memory, storage, network devices, battery etc.

CARISMA [2] provides adaptable services for different applications. Each application has passive and active profiles. The passive one define actions the middleware should take when specific context events occurs, such as shutting down if battery is low. The active information defines relations between services used by the application and the policies that should be applied to deliver those services. Different environmental conditions may also be specified, which determine how a service is delivered.

CARMEN [3] transparently handles resources in wireless settings assuming temporary disconnects. If a user moves to another environment the proxy also migrates using wired connections. Each mobile user has a proxy which provides access to resources needed by the user. When migrating, the proxy makes sure that resources are also available in the new environment. This can happen by: moving the resources with the agent, copying the resources, using remote references, or re-binding to new resources which provide similar services.

From an utility point of view, CAPIM can also be compared with Google Now and Microsoft On{X} [4]. Google Now is able to predict what information an user need, based on his previous searches and on his context data. Microsoft On{X} lets the user set actions for states defined by his context data. When a certain state (previous defined by the user) is reached, a trigger is fired. Our platform supports this kind of approaches, being build as a framework for developers, not as a stand alone application, like Google Now or Microsoft On{X}.

These and similar other middlewares support differently pervasive and mobile computing based on context information. They all provide some method of adapting to changes in the context, and methods for collecting context, but otherwise use different entities and have different focus. We present a more complete and complex context model that integrates a wider spectrum of information, ranging from location to user's profile and social capabilities. The middleware allows collecting context information from a wide-area of data sources, aggregation including providing semantic relations, and an engine that is able to mimic the behavior of various context-aware applications.

III. Architecture

CAPIM's architecture consists of several layers (see Figure 1), each one providing a specific function: 1) collecting context information, 2) storing and aggregation of context information, 3) construction of context-aware execution rules, and 4) visualization and user interaction.

The core concepts are as follows. First the user installs on his/her smartphone the platform container. It is the execution framework on which all layers are built. We provide a service for dynamic

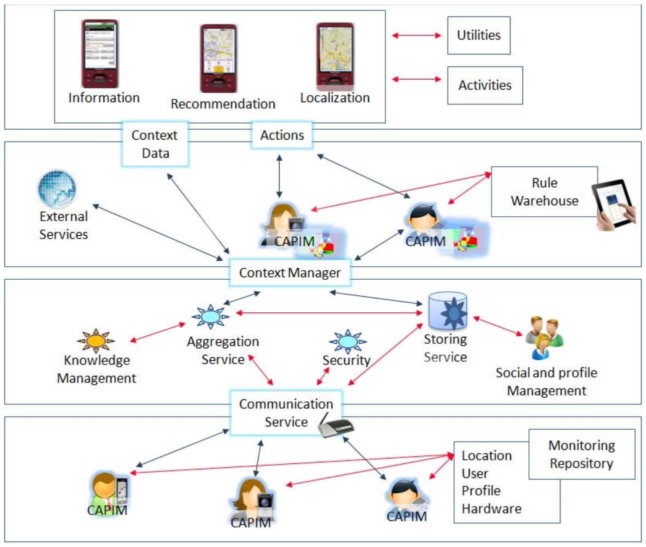


Figure 1. CAPIM's architecture.

discovery of available monitoring services, which are downloaded as required (e.g., depending on the requirements of the application, the battery or GPS data collecting module can be dynamically deployed into the platform), loaded and executed inside this container. For collecting context information, the first layer includes sets of monitoring services (collecting and first-stage storing on the local mobile device) for the context data.

Each monitoring service is packed in a digitally signed monitoring module. These modules are downloadable from remote repositories resembling application stores. Such monitoring services can be developed/maintained by third party organizations. This has several advantages: on the developer side updates are easily propagated, while for the end-user it means no more data collecting module are deployed than needed at all time (which make the monitoring layer less intrusive for the end-user's smartphone).

Each monitoring service is executed inside a separate container. This allows a separation of concerns (no service needs to know what other modules are deployed) and fault isolation.

The second layer deals with the aggregation and storing of context data. The components at this layer are running in a server environment, mainly because the aggregation involves collecting data from multiple mobile sources. It also involves higher computational capabilities that are available on the user's smartphone without interfering with his/her own activities. At this layer the information is received from several context sources and is organized according to the proposed Context Model. For example, the data from several sensors (GSM, WiFi, Bluetooth) is aggregated into current Location, and the user can experiment with various location algorithms. The user's characteristics are organized based on a FOAF and semantic technologies [5]. CAPIM is able to aggregate data into models describing actual relations between users, inferring information about their interests and activities. In an academic environment this allows defining rules specific to users interested in particular research area, or belonging to particular classes.

The next layer includes context Rule actions. Changes in the context may trigger different actions on the mobile phone according to a predefined rule set. The rules are expressed in an XML-based format and are stored in a remote repository. The user is therefore able to dynamically load and execute on the local mobile phone specific rules, depending on his/her own preferences. Finally, the fourth layer is responsible with the applications, expressed as rules and actions, which can be used for orientation, information and recommendation purposes. At this layer there are local utilities that can help with context-triggered actions. Also applications can use the context data to improve response to stimulus (an interior or exterior request). An application can react to changes in the current context and take specific actions depending on some predefined rules. For this, conditions are evaluated period as the data is retrieved. Third party applications and services can use the API provided by the context-aware services. They can use functions for obtaining particular context data, using filters, or can subscribe for context data. They can also declare new execution rules for users to install on their mobile devices.

IV. The Context Model

We model a generic *context*, seen as the information that can be used to characterize the situation of an entity [6]. Entity is any person, place, or object that is considered relevant to interaction between a user and an application including the user and application themselves. In accordance, the context is the collection of information used in some particular form.

The context model includes external data (relative to the environment of the mobile device executing the application, such as location, proximity) or internal information (to the device, such available disk space, battery, capabilities, etc.). The proposed context model aggregates this information into a unique set of data. The context is built based on all detectable and relevant attributes of the mobile wireless device, the application's user, the device's surrounding environment, and the interaction between mobile devices (creating an ad-hoc social interaction map) (see Figure 2).

The hierarchical context model has several layers. On the first layer is the device, grouping together location, time, the user's identity, and the information gathered from various hardware sensors. The device object also provides static information about the device, such as its identifier, operating system and platform, etc.

Location is obtained from several sources. For out-door locality we

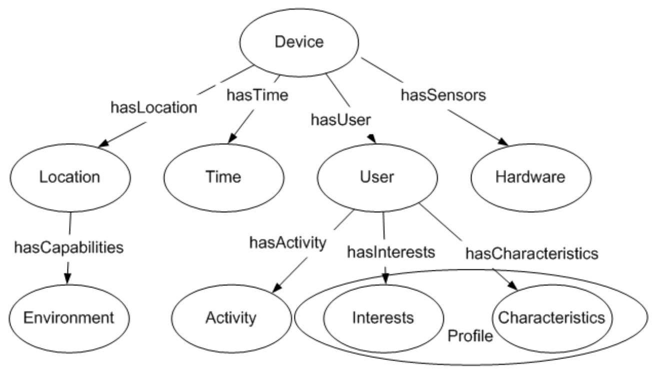


Figure 2. The proposed context model.

use the GPS or GSM capabilities of the device. For in-door location we combine information received from several sensors, such as GSM cells, WiFi access points, and hardware devices capable of recognizing bluetooth pairing. The platform also allows experimenting with various in-door locality algorithms and solutions. In this case first the user constructs a module (if one is not already available) for collecting information from sensors. It then aggregates the information into a recognizable form of location data (e.g., the user is in front of a predefined room).

The user's identity is made available from the certificates installed on the mobile smartphone. The identity information is used for discovering relevant services. It can also be used for situation-awareness where the application recognizes the user, its location and take an action to automatically open the door (as described in a subsequent Section).

If the user's identity is found, it is augmented with additional information, such as the user's profile and activities. The user's activities are discovered from his/her agenda, or from the user's academic schedule (if the user is a student, based on his certificate the schedule is discovered by interrogating the university's data management system).

The profile context includes information related to the user's research interests, academic interests, or social interests. For the research interests a special service collects and aggregates data from scientific research databases and provides a set of features including automatic collection of information, guided and focused retrieval of researcher profiles, aggregation and storage of structured data in time, aggregated and personalized view of collected information.

The user's profile is provided in either a static form (for example, based on the certificate the user's current academic profile can be easily extracted from the university's digital record database), or is inferred from social networks. For that the application uses as data sources several online social networks: Facebook and LinkedIn are among them [2]. These sources provide dynamic information about user's interests for example. But they also provide information about social relations between users. So, instead of asking users to insert their social preferences again, we learn them from the users' social networks and devise new connections based on the supplementary context information. This allows making queries to the system asking for the whereabouts of the user's current friends, representing users with current interests situated in the immediate proximity, or finding friends that can serve some specific events. The context also includes system information. For example, special designed collecting modules can use the mobile smartphone's sensors (for battery level, light intensity, accelerometer, etc.). In addition the hardware context includes information gathered from external sensors (from sensors in the environment).

Our vision is to use the context information as part of the processes in which users are involved. The context can support the development of smart applications capable to adapt based on the data relevant to the user's location, identity, profile, activities, or his/her environment (light, noise, speed, wireless networking capabilities, etc.). We propose the use of a context model that includes these parameters. Based on this model we propose *building smart and social environments capable to adapt to context using mainly the sensing and processing capabilities of users' mobile smartphones*.

In this sense the context model could support an academic environment in which users (students, teachers, etc.) may be endowed with a portable device which can react to changes in context by adapting the interface to the user's abilities (increase the luminosity when user is in a dark room, but not if a presentation is in progress) and profile (academic stuff are presented with a different set of services than students), increase the precision of information retrieval (use the context information relevant for the user's current action), or make the user interaction implicit (assume its interest based on his/her profile).

A. *Context acquisition modules*

The context monitoring modules are provided as packages which are downloaded from a remote repository (in case of the Android implementation) or from an App Store (in case of an iOS variant). A Module Loader component is responsible for the discovery and loading of the modules (see Figure 3). The module execution is part of the root platform container that the user installs first on his/her smartphone. On top of this platform, all monitoring services are dynamically discovered, downloaded as needed, loaded and executed inside this container.

Each monitoring service is executed inside a separate container. This allows separation of concerns (because no service needs to know what other modules are deployed) and fault isolation.

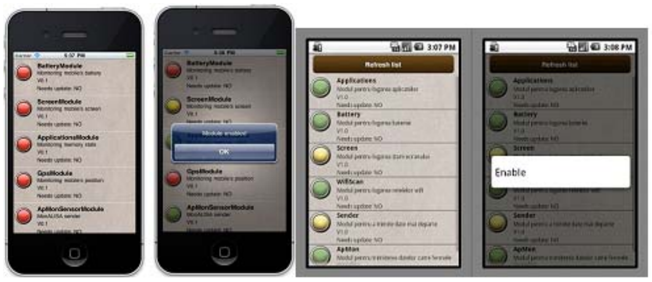


Figure 3. Management of monitoring modules (left, on iOS, right on Android).

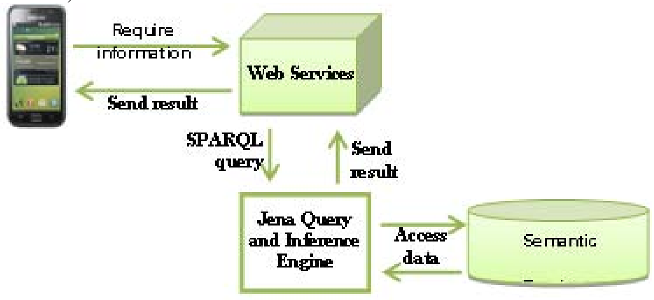


Figure 4. The flow of query the semantic service.

B. *A semantic-based model*

After the data is sensed and collected, it is further aggregated in CAPIM using a semantic model (other models are supported as well - for example, data can also be collected as time series, for long-term and near real-time processing guarantees). The semantic model provides a vocabulary to represent knowledge about context and to describe specific situations. The Context Ontology defines a common vocabulary to manage and share context data. The advantage of such an approach is sharing a common understanding of information to users, devices and services, because the ontology includes machine-interpretable definitions of basic concepts and relations.

The aggregation and semantic services are running on server-side (second level in Figure 1), as semantic aggregation involves collecting and aggregating together data from multiple sources. All mobile devices send context information to the aggregation service, where is further managed and semantically organized. The aggregation service is also responsible to infer the stored data and send aggregated information back to uses or applications (see Figure 4).

The aggregated semantic data is kept in a semantic database. CAPIM's repository implementation uses the Jena Semantic Web Toolkit [7]. The framework provides functions to add, remove, query even to infer data on generic RDF models.

The web service client is a CAPIM data receiver module - it receives context information from all running monitoring modules. This aggregation module packs the received information in a XML format data (Figure 4) and sends it to the aggregation web-service.

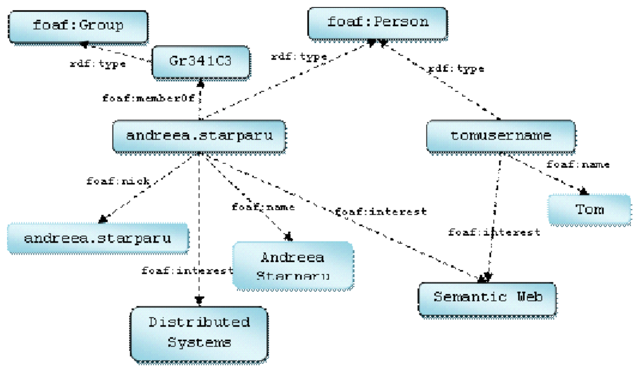


Figure 5. Part of CAPIM's Semantic Network.

The web-service stores the information in the semantic database.

The context ontology can capture all context information and model the basic concepts of person, interests and activities, describing the relationships between these entities. Considering as an example the pervasive computing which can be divided in a collection of sub-domains (e.g. home domain, school domain), we composed our ontology using domain-specific ontologies. Therefore, and considering its specific characteristics, CAPIM user's characteristics are organized based on FOAF [8] ontology (see Figure 5). In this way we can describe user's activities and his relations to other people and objects. To model a paper or a book we use PUBL [9] ontology, storing and linking in this way information such as authors, publishing company or the release date. To describe events, dates or locations we use the ICAL and GEO [10] / WAIL [11] ontologies.

The most known and used Semantic Web descriptive vocabulary is FOAF (an acronym of Friend of a Friend). It is a machine-readable ontology describing persons, their activities and their relations to other people and objects. FOAF allows groups of people to describe social networks without the need for a centralized database. FOAF is the most used specification for describing personal information and relations between people. For every context-aware application, location represents also an important data. For this reason, our semantic-based ontology models users' location using WAIL [11] and GEO [10] ontologies. WAIL (an acronym of Where Am I Language) provides a set of classes and proprieties very useful in describing the locations. It allows us to describe locations in as much or as little detail as we would like, being inspired by the Blogchalking project, which uses a similar (but stricter) system to help weblog authors find other authors near them. GEO vocabulary begins an exploration of the possibilities of representing mapping/location data in RDF. This ontology defines a class "Point", whose members are points. Points can be described using the "lat", "long" and "alt" properties. For example, we might use an externally defined property such as "foaf:based_near" or "wail:homeAddress" for representing a pointer to a location. Also, In CAPIM application context, it is important to model events. To describe an event we have to use proprieties like location, start date, end date and event's name - all of them described in ICAL ontology.

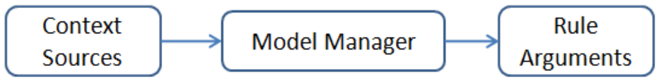


Figure 6. From context parameters to rule arguments.

The main benefit of using domain-specific ontologies is that we can dynamically plug and unplug them from our model when the environment has changed. We prefer to create the CAPIM ontology based on others already implemented ontology because in this way the redundancy can be avoided and our semantic stored data can be easier linked with other information on the web.

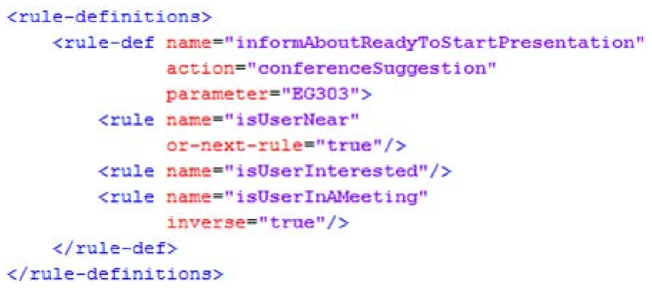
Using the context ontology in a CAPIM academic scenario, for example, we can query and infer the stored data finding out new useful information easier. To illustrate our modeling concept we can describe a typical scenario: to socialize, in a break, first year computer science student Tom wants to discuss about Semantic Web with his interested mates. For this he just needs to use CAPIM service. It will interrogate the aggregation service, which will send to device the required data. With a relational model, the service should have to iterate through all CAPIM users, to find their locations and their interests. This semantic model has all this data linked, so the result is obtained faster without being affect its validity.

C. The rule engine

CAPIM also offers an integrated rule engine. Changes in the context may trigger different actions according to a predefined rule set. First, the context information is translated into a list of parameters (name-value pairs) that are used for defining the conditions inside the rules configuration file (see Figure 6).

The rule definition contains a list of rule elements that are periodically evaluated by the engine. A rule is composed of several elements: *Conditions* (expressed as Boolean expressions, based on rule implementations), *Actions* (an action is triggered when the rule conditions are met), and *Action* parameters (strings which are passed as parameters to the action).

The rule implementations specify different expressions used to evaluate the context. For example, *rules.StringFieldEquals* expresses equality between a context parameter and a string. This rule can be applied to context parameters which have string values, such as the user's name. By default CAPIM provides several such rule implementations: *StringFieldEquals*, *IntFieldEquals*, *IntFieldGreaterThan*, *IntFieldLessThan*, *IntFieldBetween*, etc. The user can also specify higher-level functions in two ways: he can combine these rules using boolean algebra, or he can specify component that implements an



```
<rule-definitions>
    <rule-def name="informAboutReadyToStartPresentation"
              action="conferenceSuggestion"
              parameter="EG303">
        <rule name="isUserNear"
              or-next-rule="true"/>
        <rule name="isUserInterested"/>
        <rule name="isUserInAMeeting"
              inverse="true"/>
    </rule-def>
</rule-definitions>
```

Figure 7. Example of rule definition.

aggregation function. In the second case, the data is first passed to that component, and the result is further used in the rule evaluation.

These rules represent operations between base types and allow one to formulate different restrictions on context parameter values. When combined they can lead to more complex conditions:

$$Rule = Rule\ OR\ Rule \mid Rule\ AND\ Rule \mid RuleImpl$$

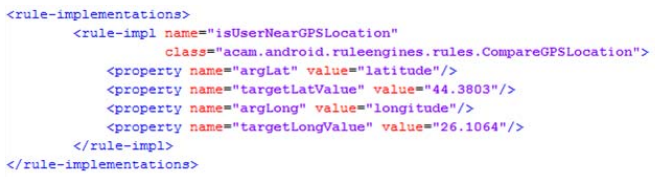
The rule XML file is structured in two parts: *rule-definitions* (see Figure 7) and *rule-implementations* (see Figure 8). The rules specified in the *rule-definitions* part under the *rule-def* tag are complex rules, the ones that trigger a certain action if they are evaluated to true. Complex rules are described with a *name* attribute, an attribute to specify the action to trigger and possibly an attribute for the parameter of the action.

In defining a complex condition as a list of simple rules, if the *or-next-rule* attribute of a simple rule is not specified as having the value “true”, then the current rule is in an *logical AND* relation with the next rule. On the other hand, simple rules may have the inverse attribute set to true such that we obtain a negation of the rule. Taking this into consideration, the rule in Figure 7 can be translated to the following logical proposition:

```
1  if $user\ is\ near$ OR $user\ is\ interested\ in\ the\ subject$
2   $of\ the\ presentation$ AND (NOT $user\ is\ in\ meeting$)
3      then $inform\ user\ about\ presentation$
```

A very important aspect when writing complex rules containing both logical operators (*OR/AND*) is the fact that the simple rules are not evaluated from first to last. If a rule doesn’t specify the *or-next-rule* attribute as true, only the current rule’s result is relevant. When dealing with an *OR* rule, the engine takes all consecutive *OR* rules and evaluates all of them and uses the result of the *OR* operation. In other words, from the way the rule engine is build, *OR rules* have precedence over *AND rules*, like they were between brackets.

The rule-implementations part of the rule XML file presents implementations for simple rules, each simple rule being presented in a *rule-impl* node (see Figure 8). In general, a *rule-impl* node contains



```
<rule-implementations>
        <rule-impl name="isUserNearGPSLocation"
                   class="acam.android.ruleengines.rules.CompareGPSLocation">
            <property name="argLat" value="latitude"/>
            <property name="targetLatValue" value="44.3803"/>
            <property name="argLong" value="longitude"/>
            <property name="targetLongValue" value="26.1064"/>
        </rule-impl>
</rule-implementations>
```

Figure 8. Example of rule implementation.

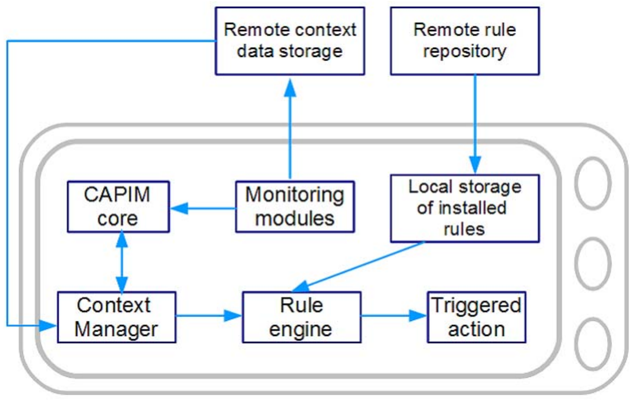


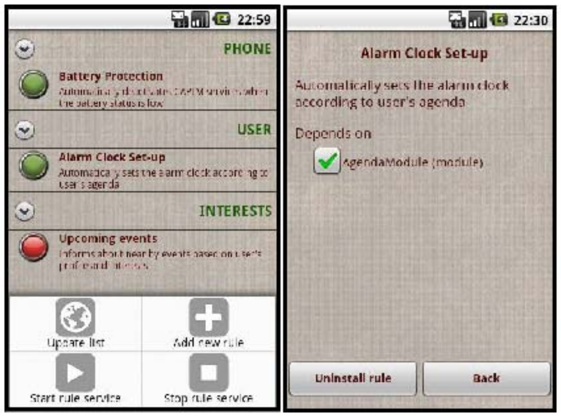
Figure 9. Components of the rule evaluation process.

the name of the rule, the class used to verify specific conditions and a list parameters list.

When defining the implementation of a rule, the list of property elements equals the number of attributes in the comparer class. Actually, each property’s value attribute is assigned to the attribute of the comparer class having the same name as the property. This aspect can be a drawback when creating rules automatically because rules become too dependent on the names of the implementation of comparator classes. This problem could be solved by creating a database of comparer classes with attribute name and description.

CAPIM includes a rule engine capable of interpreting context-based rules, thus providing the basis for developing context-aware applications able to react and take smart automated decisions at context data changes. The components involved in the process of rule evaluation are presented in Figure 9. The figure illustrates the relations between components. Furthermore, it shows which components run on the smart phone and which belong to a remote server or repository.

Each rule is, therefore, evaluated to either false or true. When a rule has an attribute which refers to an action, the rule engine will try to execute it. The rule engine uses the information provided by the Context Manager to evaluate the rules. In order to detect the changes in the context, the rule engine analyze the rules periodically, at a fixed polling interval. The value for the update interval constant is subject to further investigations, considering the power constraints on mobile devices. The use of a small value causes the rule engine to evaluate


22:59

PHONE

Battery Protection

Automatically deactivates CAPIM services when the battery status is low

USER

Alarm Clock Set-up

Automatically sets the alarm clock according to user's agenda

INTERESTS

Upcoming events

Informs about nearby events based on user's profile and interests

22:30

Alarm Clock Set-up

Automatically sets the alarm clock according to user's agenda

Depends on

[✓ AgendaModule (module)]

Figure 10. CAPIM's interface for managing rules.

the values too frequently, while a big value can miss some of the context changes. The approach in CAPIM consists of scheduling the next evaluation when the next context parameter used by the rules is about to expire. Common user activities take hours, weather is again a slow process, and some context data, like the phone's features, never change. If the rules under the administration of the rule engine refer only this kind of parameters, it can be inefficient to evaluate them every minute.

Other heuristics are also applied to gain maximum performance. For AND expressions is useless to evaluate all context parameters if the first one doesn't match the required condition. Again, for AND expressions, the rule evaluation interval is given by the context parameter having the longest life span. If this parameter does not change to true, the rule is certainly false. A rule for which none of the context parameters has expired is not evaluated at all, even if the rule engine has been activated.

The second part of a rule consists of actions to be executed when context is met. When a rule passes, the rule engine will trigger the associated action. An action can be either an application (running on the smartphone) that must be launched when conditions are met. But it can also represent an execution of a method provided by the CAPIM's visualization API. In the second case, the action can be a message presented using the notification system, or a pinpointed landmark presented on the map used for navigation.

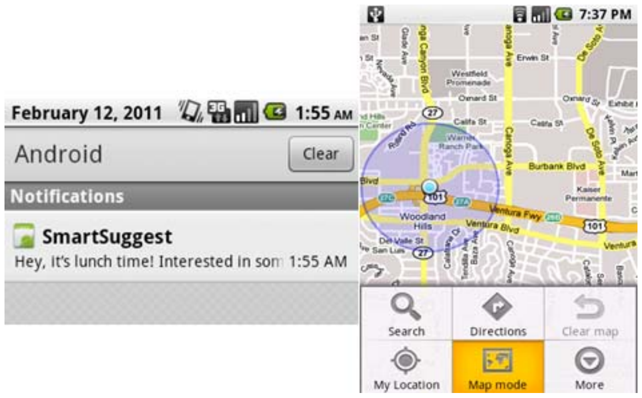
An important aspect that should be emphasized is that actions runs as distinct application, and are managed separately by the underlying operating system. This fault tolerant approach will prevent any fault inside an action to cause the CAPIM platform to fail.

V. Possible applications

A possible application of the proposed platform and context services is an automated support for people in an university, who may be endowed with a portable device which reacts to changes of context by (a) providing different information contents based on the different interests/profiles of the visitor (student or professor, having scientific interests in automatic systems or computer science, etc), and on the room he/she is currently in; (b) learning, from the previous choices formed by the visitor, what information s/he is going to be interested in the next; (c) providing the visitor with appropriate services - to see the user's university records only if appropriate credentials are provided, to use the university's intranet if the user is enrolled as stuff; (d) deriving location information from sensors which monitor the user environment; (e) provide active features within the various areas of the university, which alerts people with hints and stimuli on what is going on in each particular ambient.

```
<?xml version="1.0" encoding="UTF-8"?>
<rules-config>
    <rule-definitions>
        <rule-def name="showRestaurantSuggestion"
        action="category.PLACE_SUGGESTION"
        parameter="restaurants">
        <rule name="isLunchTime" />
    </rule-def>
</rule-definitions>
<rule-implementations>
    <rule-impl name="isLunchTime" class="rules.IntFieldBetween">
        <property name="argField" value="TIME"/>
        <property name="targetStart"  value="13"/>
        <property name="targetEnd" value="14"/>
    </rule-impl>
</rule-implementations>
</rules-config>
```

Figure 11. Example of a context-based rule.


February 12, 2011 1:55 AM

Android

Notifications

SmartSuggest

Hey, it's lunch time! Interested in som 1:55 AM

Figure 12. Expanded notification (left), and suggestion of nearby restaurants (right).

The proposed context-aware platform can be used for the experimental evaluation of many solutions. Users can evaluate methods for gathering context information, for aggregating data using semantics, ontologies. Such approaches were demonstrated in [12].

VI. Conclusions

Smartphones are today becoming commodities. Considering that even today more than half a billiard people have at least one smartphone, the previous affirmation is not so far-fetched. The advances in mobile technologies allowed people to have in their pockets, wherever they go, powerful computing devices, which can be of great help in their activities. Besides portability, these gadgets present another great feature: they have the necessary hardware capabilities to sense the environment. These advantages can be used to make mobile devices and the applications they host to be aware of the context they work in.

CAPIM is a platform designed to support the shift towards massive quantities of real-time information becoming access push rather than demand pull on a global case. It integrates services to monitor and a context for adapt with the user's context using sensors and capabilities of smartphones. It integrates context-aware services that are dynamically configurable and use the user's location, identity, preferences, profile, and relations with individuals, as well as capabilities of the mobile devices to manifest themselves in many different ways and re-invent themselves over and over again. Such services aggregate and semantically organize the context data. They react based on dynamically defined context-oriented workflows, and the platform includes an execution engine that supports context-aware actions for orientation, information, and recommendation.

A pilot implementation of the contextualization platform has proven the great advantages it provides in terms of simplicity and flexibility.

VII. Acnowledgment

This work was supported by project "ERRIC -Empowering Romanian Research on Intelligent Information Technologies/FP7-REGPOT-2010-1", ID: 264207. The work has been co-funded by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of Labour, Family and Social Protection through the Financial Agreement POSDRU/89/1.5/S/62557. CAPIM's official site and source code repository are available at http://cipsm.hpc.pub.ro/capim.

References

[1] A. Chan and S.-N. Chuang, "Mobipads: a reflective middleware for context-aware mobile computing," *Software Engineering, IEEE Transactions on*, vol. 29, no. 12, pp. 1072 – 1085, dec. 2003.

[2] L. Capra, W. Emmerich, and C. Mascolo, "Carisma: context-aware reflective middleware system for mobile applications," *Software Engineering, IEEE Transactions on*, vol. 29, no. 10, pp. 929 – 945, oct. 2003.

[3] P. Bellavista, A. Corradi, R. Montanari, and C. Stefanelli, "Context-aware middleware for resource management in the wireless internet," *Software Engineering, IEEE Transactions on*, vol. 29, no. 12, pp. 1086 – 1099, dec. 2003.

[4] "Microsoft onX," [Accessed February 9th, 2013]. [Online]. Available: \url{http://www.onx.ms}

[5] J. Banford, A. McDiarmid, and J. Irvine, "Foaf: improving detected social network accuracy," in *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing - Adjunct*, ser. Ubicomp '10 Adjunct. New York, NY, USA: ACM, 2010, pp. 393–394. [Online]. Available: http://doi.acm.org/10.1145/1864431.1864453

[6] A. K. Dey, "Understanding and using context," *Personal and Ubiquitous Computing*, vol. 5, pp. 4–7, 2001. [Online]. Available: http://dx.doi.org/10.1007/s007790170019

[7] B. McBride, "Jena: a semantic web toolkit," *Internet Computing, IEEE*, vol. 6, no. 6, pp. 55 – 59, nov/dec 2002.

[8] "Foaf website," [Accessed February 9th, 2013]. [Online]. Available: \url{http://xmlns.com/foaf/spec/}

[9] "Publ website," [Accessed February 9th, 2013]. [Online]. Available: \url{http://ebiquity.umbc.edu/ontology/publication.owl}

[10] "Geo website," [Accessed February 9th, 2013]. [Online]. Available: \url{http://www.geonames.org/ontology/}

[11] "Wail website," [Accessed February 9th, 2013]. [Online]. Available: \url{http://www.eyrie.org/~zednenem/2002/wail/}

[12] C. Dobre, "Context-aware platform for integrated mobile services," in *Emerging Intelligent Data and Web Technologies (EIDWT), 2011 International Conference on*, sept. 2011, pp. 198 –203.