CAPIM: A Context-Aware Platform using Integrated Mobile Services

Ciprian Dobre*, Flavius Manea*, Valentin Cristea*
*University POLITEHNICA of Bucharest, Romania
E-mails: flavius.manea@cti.pub.ro, {ciprian.dobre, valentin.cristea}@cs.pub.ro

Abstract

As smartphones integrate more sensors and provide larger computing capacities, they enable a shift towards massive quantities of real-time information becoming access push rather than demand pull on a global case. We present a platform for supporting generic context-aware mobile applications. It is composed of a monitoring layer, capable of collecting and aggregating data from various sensors related to the mobile phone, a rule execution engine capable of supporting context-aware workflows, and an application layer capable of supporting the interface between mobile applications and the human operator. We present the architectural decisions of our solutions, together with implementations details and experimental evaluation results. Our solution is fully functional and can be used in a variety of context-aware situations.

Keywords: ***context-awareness, mobile computing, smartphones, pervasive adaptation and computation carried by users, social adaptation of services.***

1. Introduction

Context-aware applications can sense their surrounding environment. They gather information about the circumstances under which they operate and react based on rules or stimulus. The context data generally falls under several categories [1]: *physical, system, application,* and *social* context. An example of physical context is the user's location. The system's environment is used to tailor the interaction based on, for example, nearby computing resources or network traffic. Application data such as the current text selection provides context information that can help anticipate user actions. The social environment consists of information about people, such as their presence, their identity, or their activity, and may be used by systems to tailor the system's behavior to a user's needs or preferences.

While context was initially perceived as a matter of user location, lately the notion has been part of a process in which users are involved; thus, sophisticated and general context models have been proposed to support context-aware applications [3]. Such applications use the context to adapt their interfaces, tailor the set of application-relevant data, and increase the precision of information retrieval. Such an application may know that the user is in a meeting room, is sitting down, and therefore may conclude that the user is currently in a meeting and reject any calls classified as unimportant [6].

Context-aware systems are concerned with the *acquisition of context* (e.g. using sensors to perceive a situation), *the abstraction and understanding of context* (e.g. matching a perceived sensory stimulus to a context), and *application behavior based on the recognized context* (e.g. triggering actions based on context) [5]. For a long time, context-aware applications were hard to develop because the devices able to support them were unavailable. Nowadays, almost every smartphone is equipped with sensors and communication systems which can provide a lot of information about the environment, sufficient to act as input for such applications.

Several platforms for pervasive and context-aware systems to support rich contextual features were built in the past few years. MobiPADS [3] is a middleware for mobile environments. It consists of Mobilets, entities that provide particular services that can migrate between MobiPADS execution 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 data includes: processing power, memory, storage, network devices, battery etc.

CARISMA [4] provides adaptable services for different applications. Each application has passive and active profiles. The passive one defines 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 [5] transparently handles resources in wireless settings assuming temporary disconnects. Each user has a proxy which provides access to the set of resources needed by the user. The proxy is able to migrate when the user moves to another environment. By 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.

These and other middlewares support differently pervasive and mobile computing based on context information. They all provide some methods of adapting to changes in the context, and for collecting context. We present a versatile middleware that includes 1) a wider spectrum of context sources, 2) advanced aggregation to compose the context, 3) dynamic workflows as the basis for context-aware applications, and 4) the support for the execution of such applications that can adapt to the environment, make recommendations and inform the user about context events, help the user better interact with the environment. It is based on a context model that integrates a wider spectrum of information, ranging from location to user's profile and social capabilities.

The rest of this paper is structured as follow. Section 2 presents the architecture of the proposed context-aware platform. In Section 3 we present details about the context acquisition modules, while the context model is presented in Section 4. Section 5 presents the context-based rule engine, and in Section 6 we present an evaluation application built on top of the platform. In Section 7 we conclude and present future work.

2. Architecture

As the smartphone market is increasing, their presence in everyone's pocket is becoming common. Smartphones today integrate more sensors, provide larger computing and storage capabilities, and use advanced wireless solutions. This opens the era for mobile wireless applications that can actively help users stay connected in the most remote places, and sense its context and help him/her take informed decisions.

Still, developing such applications can prove to be a problem. Modern smartphones integrate various sensors, based on different hardware architectures. They differ widely based on their architectural constraints.

We present **Context-Aware Platform using Integrated Mobile Services** (short-named CAPIM), a middleware specially designed to facilitate the development of context-aware applications. It provides a direct, uniform way to access the information. CAPIM sits between the operating system and the context-aware applications. The general design was previously presented in [7]. Here we present implementation details, together with insights on the context execution rule engine.

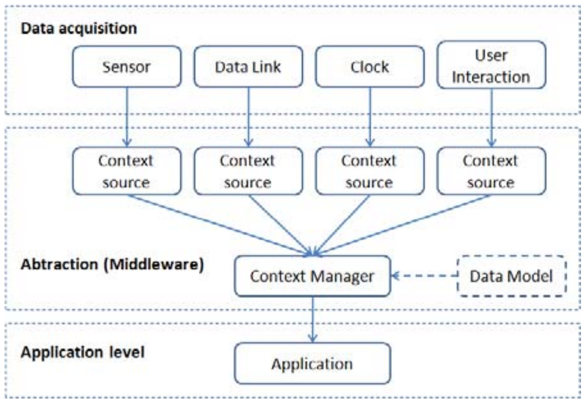


Fig. 1. CAPIM's components.

The diagram in Figure 1 presents the main components of CAPIM, organized on several layers. *Data acquisition* is responsible for gathering information from the surrounding environment. Based on the current capabilities of mobile devices we use different context providers: sensors, data links, clock, and user data. *Sensors* are used to capture information such as current temperature, altitude, location, the movement of the user (acceleration or direction), or the light intensity in the environment. Their type and performance varies. CAPIM also includes support for *data links* (through Wi-Fi or Bluetooth technologies we test the presence of nearby devices, access and exchange information), *clock* (leads to discovery of current user activity), and *user data* (outside the inference mechanism, the user is able to control the system response to context changes). The data acquisition layer is detailed in the next Section.

The second layer deals with *Abstraction*. Information from context sources is gathered by the context manager and organized based on concepts from a predefined model. This actually represents an abstraction layer which is used by applications to access context information. The domain described by the model acts as a contract between the middleware and the applications.

The last layer deals with the *Applications*. Applications can include the context in response to stimulus (interior or exterior requests) so to better serve the user's needs. The application can react to context changes and take actions depending on some predefined rules. For this, data is retrieved and conditions are evaluated periodically. CAPIM includes a rule engine capable of interpreting context-based rules. It provides the base for developing context-aware applications that can change their appearance, and provide more dynamic interaction

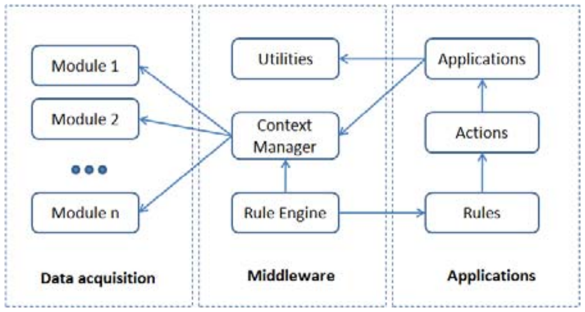


Fig. 2. The architecture of CAPIM.

All these components are interconnected according to the architectural design presented in Figure 2. The context acquisition is performed by specialized **Monitoring Modules**. The diversity of context sources (different sensors, technologies, mobile architectures) leads to a design which includes *diversity* and *expendability*. Our proposed solution consists in the use of dedicated monitoring modules. A module can collect the data for a specific sensor on some specific operating platform.

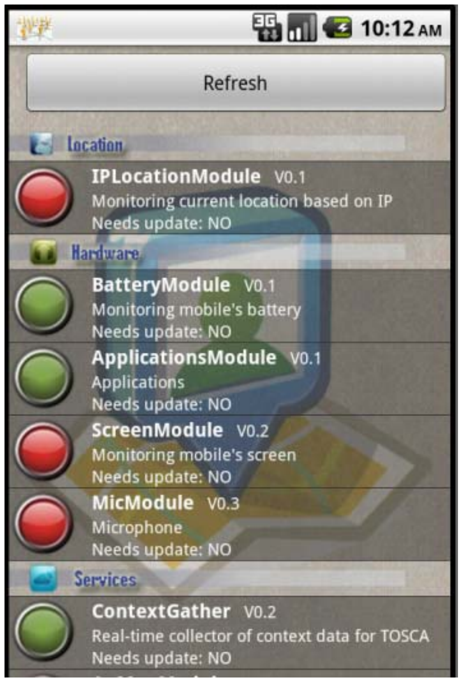
The **Context Manager** is responsible with the management of these modules. It provides context information to the applications in a structured form. The context depends on the information provided by the context modules. In our approach the user can download from a central repository the context modules his/her current application(s) require. So, instead of allowing the modules manage the context, the applications and user are in control of the type of information composing the current context model.

The Context Manager is responsible with the management of the dynamically loaded context modules. It maintains a directory with the information currently provided by the loaded monitoring modules. When a request for a specific context parameter is received, the manager mediates the request towards the corresponding context monitoring module.

A very common scenario in context-aware computing is when changes in the context trigger a specific action: for example, when the time turns 8 a.m. the alarm goes off and announces the user that a new day of work is ready to start. Another important function of the context-aware middleware is to serve such necessities, again in the most flexible manner as possible. In our proposal we designed a built-in rule engine that evaluates business rules from an input XML file and decides which actions should be started based on the current context parameters.

3. 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.



IPLocationModule V0.1
Monitoring current location based on IP
Needs update: NO

BatteryModule V0.1
Monitoring mobile's battery
Needs update: NO

ApplicationsModule V0.1
Applications
Needs update: NO

ScreenModule V0.2
Monitoring mobile's screen
Needs update: NO

MicModule V0.3
Microphone
Needs update: NO

ContextGather V0.2
Real-time collector of context data for TOSCA
Needs update: NO

Fig. 3. The Module List screen.

The management of the modules is facilitated by special screens (see Figure 3). This are part of the platform container that the user installs first on his/her smartphone. It is the execution root framework on which all layers are built. For example, the monitoring services are dynamically discovered, downloaded as needed, loaded and executed inside this container. So, for collecting context information, CAPIM includes monitoring services (collecting and first-stage storing on the local mobile device) for the context data.

The monitoring services can be developed and maintained by third party organizations. For example, a manufacturer might construct a module to collect data from its own sensor, therefore integrating its data within the user's context.

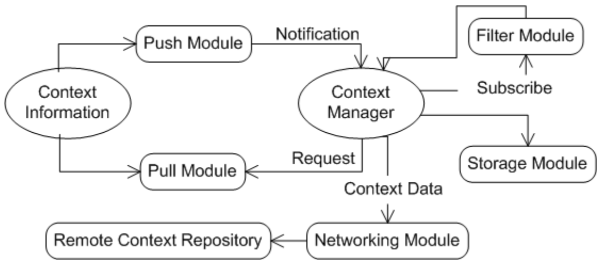


Fig. 4. Flow of monitoring information.

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.

Depending on the general function, the monitoring services are further grouped into several categories (see Figure 4). The central component is the Context Manager, orchestrating the flow of information between the monitoring services.

The Push and Pull monitoring services are directly responsible for collecting context information. They collect context information generally directly from sensors. The Push service reacts to changes of the context, which in turn triggers notifications to be sent to the Context Manager. The Pull service is periodically or on-request interrogated for the current values of the monitoring parameters.

All the context information is further sent to Filter, Storage and Networking services. The Filter service subscribes to specific context information. The Context Manager forwards the data of interest to the Filter service, which in turns can produce new context information (possible from multiple data sources). Such a construction allows for first-stage aggregation of context information.

The Storage service can store data locally for better serving the context-execution rules. Finally, the Networking service is responsible for sending the collected context information remotely to aggregation services (the Remote Context Repository component located in the next layer). It is here that we can experiment with different network protocols and methods of sending data, whilst balancing between costs and energy-consumption.

Each monitoring service is also responsible for a particular type of monitoring information. Thus, these services fall into different categories: location, user, profile, hardware.

4. The Data model

Context-aware applications access information through a dedicated *IContextManager* interface. The interface provides a method which returns a list of all context parameters that can be resolved (with the help of the available context modules). An application can enable or disable some of its features based on the type of information composing the current context. In turn, context parameters are identified using string keys, and are managed within the platform by using them (from modules to the context manager and even applications). A *getContext* method returns a special data model which logically organizes common context parameters.

Our proposed context model uses an acceptation of context initially proposed in [2]: *Context* is the information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to interaction between a user and an application including the user and application themselves.

In accordance, CAPIM considers three major types of information. The *computing context* contains the information related to computational aspects of the system (such as the application context: agenda events, websites visited, emails received; and also system context: the network traffic, status of resources, bandwidth, etc.). The *user context* consists of information related to the service requestor (such as personal context: schedule, activity, etc.; and social contexts: group activities such as classes, social relationships derived from social networks, nearby persons, etc.). Finally, the *physical context* consists of information related to the physical aspects of the system (the physical context: location, time; the environmental context: light, noise; and informational context: data aggregated from other mobile devices).

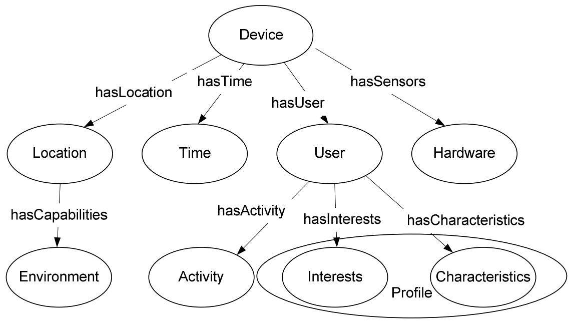


Fig. 5. The proposed context model.

The proposed context model aggregates this information into a unique set of data. We construct the context starting from 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). The resulting context model is presented in Figure 5.

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 use GPS or GSM data. For in-door location we combine information received from several sensors: GSM cells, WiFi access points, and hardware devices capable of recognizing bluetooth pairing. The platform allows experimenting with various in-door locality algorithms and solutions. In this case 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 room).

The user's identity is obtained based on certificates installed on the mobile smartphone. The security infrastructure is based on the use of X.509 certificates. The identity is used for discovering relevant services. It can be used for situation-awareness where the application recognizes the user, its location, and take an action to automatically open the door. If the user's identity is found, it is further 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 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).

5. The rule engine

An important functionality associated with CAPIM is offered by the 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).

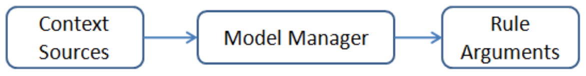


Fig. 6. From context parameters to rule arguments.

An example of a configuration file which contains rules for the rule engine is presented in Table 1.

Table 1. A rule example.

```
<?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>
```

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, IntFieldGreater Than, 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 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 | Rule AND Rule | RuleImplementation

Each rule is 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 (see Figure 7).

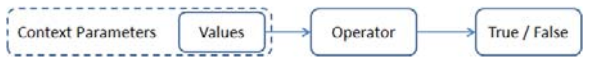


Fig. 7. The rule evaluation process.

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 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.

6. An experimental application – SmartSuggests

SmartSuggests is a sample application that uses the context-aware platform to monitor changes in the context and offer the user suggestions that it beliefs he might be interested in. For example, *SmartSuggests* detects that it is lunch time by consulting the user's agenda. It then determines the user's current location and can use Internet services to determine restaurants nearby. A notification is then brought up. If the user is interested he/she can access more details about the suggested nearby restaurants.

In another situation, the application observes that the user is in a free time interval according to his agenda and place (location), and also that the weather is sunny (using weather internet services). According to the user's preferences, SmartSuggests will present parks nearby where the user can take a walk or other similar outdoor possible activities close to the current location.

Actually, the application is able to support an endless number of such scenarios with the help of the Rule Engine component that allows us to define the conditions that have to be met for a suggestion to be shown.

The Rule Engine reads these rules from an input file (in XML format) at startup and evaluates them periodically to see if the conditions to show a specific suggestion are fulfilled.

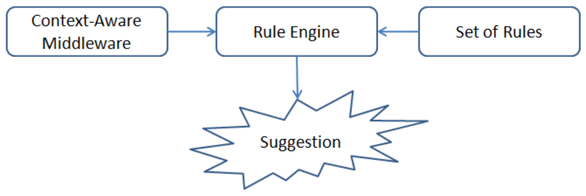
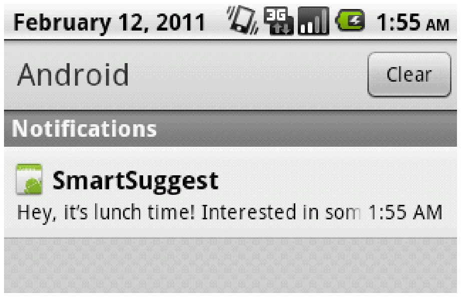


Fig. 8. The suggestion mechanism.

A possible suggestion is registered in CAPIM as a context action, thus it has a context rule which triggers the suggestion (see Table 1), and the action itself, installed as an application on the mobile smartphone. The *place suggestion* action (see Table 1) further searches places of different types close to the user's current location and displays them as a list, together with their descriptions and locations.

In SmartSuggets suggestions made by the application are presented to the user in the form of *notifications*. A status bar notification adds an icon to the system's status bar (with an optional ticker-text message) and an expanded message in the "Notifications" window. The developer can also configure the notification to alert the user with a sound, a vibration, and flashing lights on the device.

The screenshot in Figure 9 shows the notification's expanded message in a typical Notifications window. The user can reveal the Notifications window by pulling down the status bar (or selecting *Notifications* from the Home options menu).



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

Fig. 9. SmartSuggests expanded notification.

If the user chooses to accept the notification, by clicking on it, an activity is opened presenting information related to the suggestion. For example, the place suggestion for restaurants will open up a Google Map (see Figure 10). The map will be centered on the user's current location and will show what restaurants are near him. Clicking on a restaurant location will bring up a new screen with detailed information about it. The suggestion action could access context information as well and point out only the locations that match the user's culinary preferences.

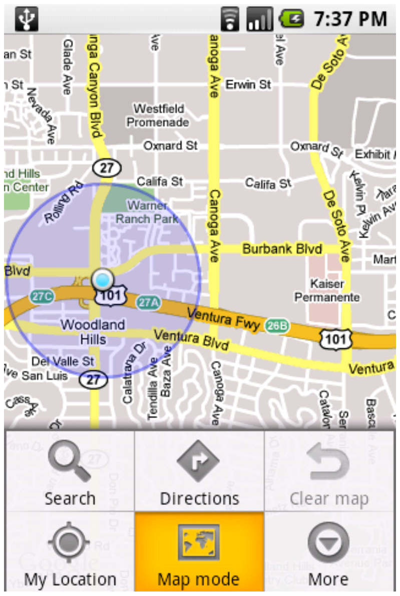


Fig. 10. Interest points nearby on a Google Map.

There are all kind of suggestions that can pop-up as context changes and help the user in his activities. Still, an administration application provides the means to enable or disable suggestions and allow the user to control his/her phone's behavior.

CAPIM includes different services for building context-aware applications. For example, it is able to present different interest points around the user's current location. Its library includes special tools that allow one to localize different objectives on a Google Map for example.

7. Conclusions and future work

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.

Context-aware applications can adapt to new context conditions, can understand more easily the user needs, and communicate with him/her more efficiently. In this we proposed a software platform which can help applications to access context information in a simple and transparent way.

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

Future improvements of CAPIM involve the increase of energy efficiency, which is a critical property for software on portable devices. To achieve this, several optimizations of the rule evaluation procedure have been already implemented.

Acknowledgments

The research presented in this paper is supported by national project: "TRANSYS – Models and Techniques for Traffic Optimizing in Urban Environments", Contract No. 4/28.07.2010, Project CNCSIS-PN-II-RU-PD ID: 238. 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.

6. References

[1] Salber, D., and G. D. Abowd, "The Design and Use of a Generic Context Server", In the Proceedings of the Perceptual User Interfaces Workshop (PUI '98), San Francisco, CA, November 5-6, 1998. pp. 63-66.

[2] Dey, A.K., "Understanding and Using Context". *Personal Ubiquitous Comput.*, 5(1), January 2001, 4-7.

[3] Chan, A.T.S., and S.-N. Chuang, "MobiPADS: A Reflective Middleware for Context-Aware Mobile Computing", *IEEE Trans. Softw. Eng.*, 29 (12), 2003, pp. 1072-1085.

[4] Capra, L., W. Emmerich, and C. Mascolo, "CARISMA: Context-Aware Reflective Middleware System for Mobile Applications", *IEEE Trans. Softw. Eng.*, 29 (10), 2003, pp. 929-945.

[5] Bellavista, P., A. Corradi, R. Montanari, and C. Stefanelli, "Context-aware Middleware for Resource Management in the Wireless Internet", *IEEE Trans. Softw. Eng.*, 29 (12), 2003, pp. 1086-1099.

[6] Dey, A.K., G.D. Abowd, and D. Salber, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications", *Hum.-Comput. Interact.* 16 (2), December 2001, pp. 97-166.

[7] Dobre, C., "Context-Aware Platform for Integrated Mobile Services", Workshop on Services for Large Scale Distributed Systems, International Conference on Emerging Intelligent Data and Web Technologies (EIDWT-2011), September 2011, Tirana, Albania.