UbiPOL: A Platform for Context-aware Mobile Device Applications in Policy Making

Mihai Barbos
Software Department
S.C. IPA S.A.
Bucharest, Romania
e-mail: mihaibarbos@ipa.ro

Eugen Pop
Software Department
S.C. IPA S.A.
Bucharest, Romania
e-mail: epop@ipa.ro

Habin Lee
Brunel Business School
Brunel University
Uxbridge, UK
e-mail: Habin.Lee@brunel.ac.uk

Luis Miguel Campos
Research and Development Department
PDM&FC
Lisbon, Portugal
e-mail: luis.campos@pdmfc.com

***Abstract* — UbiPOL is a context-aware platform for policy making. At its core, the UbiPOL platform has an essential framework of web services and APIs. It is intended to aid and support the development of location-based and context-aware applications in the field of policy making. Software developers can make use of UbiPOL generic web services and APIs to develop new context-aware policy making applications for mobile devices, leveraging the built in support for Location-based Services of the platform. The paper gives a generic presentation of the UbiPOL platform focusing on the provisioning of location-based and context-aware mobile device applications in the field of policy making.**

Keywords – context; context-aware; Location-based Services; UbiPOL; mobile applications.

I. INTRODUCTION

Literature identifies several challenges and barriers in e-Participation. One of them is poor citizen's involvement due to lack of interest for relevant government policies. As a result, there is a need for ICT tools that motivate citizens to participate in policy making processes.

UbiPOL is a context-aware participation platform for policy making. The concept of UbiPOL is based on the assessment that citizens will be more motivated to participate in policy making processes if they can find connections between their everyday life and government policies. In UbiPOL, context-awareness allows linking policy making processes to the everyday life pattern of citizens in order to increase motivation and involvement at all participation levels.

The paper gives a generic presentation of the UbiPOL platform focusing on the provisioning of location-based and context-aware mobile device applications in the field of policy making.

First, in Section II, we make a brief presentation on the state of the art. We introduce the concepts of context-awareness and Location-based Services (LBS) which are essential to UbiPOL. A subsection on e-Participation is also included, emphasizing on the need for ICT systems that increase citizen motivation.

Section III briefly describes the concept and novelty of UbiPOL, focusing on how context-awareness can improve citizen motivation to participate in policy making processes.

Section IV defines the context in relation to generic common tasks of an UbiPOL user. In this section we also reveal some of the features and elements that make UbiPOL context-aware.

Section V is a generic description of the UbiPOL system architecture, including all the platform components.

In Section VI we focus on the front-end API, one of the UbiPOL platform components. We also provide some examples on how the API can be used to develop and implement context-aware mobile device applications.

Finally, the conclusions highlight some advantages of the UbiPOL system and platform.

II. STATE OF THE ART

A. *Context-aware computing and Location-based Services*

An important part of context-aware computing is the context. Context can be defined as "any 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 the interaction between a user and an application, including the user and applications themselves" [1]. Several variables or pieces of the "constantly changing execution environment" can be part of the context, including: available processors, user input devices, network capacity, connectivity, bandwidth, user identity, location, time, collection of nearby people and social situation [1].

Location, identity, time and activity are primary data for context-aware computing, because they provide indices for other contextual information, referred as secondary information. Examples of this sort of data are: addresses, phone numbers, e-mails, people, activities, situations near the entity, etc. In order to perform context-aware computing, one must first analyze and define the context specific for a certain application scenario, including secondary environment data. Unlike primary data (like user location) that is present in all context-aware applications, secondary information is specific and will differ from one system to another. A context-aware application should gather and processes all information about the surrounding environment that is relevant to the user's task. If certain information is useful to describe the status of a participant that takes part in an action, that information represents context [1].

"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task" [1], [2]. Context-awareness can also be defined as the capability of software applications to find out and behave according to the modifications that occur in their surrounding environment [3], [4]. Originated from ubiquitous computing, context-awareness is flexibly associated with moving entities or processes. Context-awareness is usually referred to be complementary to location-awareness, because information about position is relevant for evolving processes [5]. Out-of-the-box access to location makes mobile devices appropriate software application hosts in the field of context-aware computing. Whereas position is significant as context information, context-aware computing is related with Location-based Services (LBS).

Virrantaus defines Location-based Services as "information services accessible with mobile devices through the mobile network". Location-based Services employ "the ability to make use of location" available on mobile devices [6]. Another similar definition by Steiniger refers to a location-based service as "a wireless-IP service that uses geographic information to serve a mobile user." According to Steiniger's definition a LBS can be "any application service that exploits the position of a mobile terminal" [6].

Literature review identifies 2 types of Location-based Services: push services and pull services [6], [7]. Both types rely on the use of location in delivering information to users. But the difference between the two is in the trigger (the event that initiates the transaction). For push services the trigger is not directly linked to the user. The trigger can be an event like the user entering a specific area or a timer. "Push services deliver information which is either not or indirectly requested from the user" [7]. On the other hand, for pull services the information is requested by the user explicitly. Moreover, Virrantaus separates pull services in two categories: "functional services like ordering a taxi or an ambulance by just pressing a button on the device" and "information services like the search for a close Chinese restaurant" [6].

B. *The need for increased citizen involvment in e-Participation*

E-Participation can be defined as "the use of information and communication technologies to broaden and deepen political participation by enabling citizens to connect with one another and with their elected representatives" [8].

Macintosh defines 3 levels of participation in policy making processes: E-enabling ("the use of technology to enable participation"), E-engaging ("the use of technology to engage with citizens") and E-empowering ("the use of technology to empower citizens") [9].

The first level of participation, E-enabling, refers to the provision of "relevant information in a format that is both more accessible and more understandable" [9].

The second level of participation, E-engaging, refers to "the top-down consultation of citizens" by policy makers. This level is characterized in terms of user access to information and reaction to policy maker led initiatives [9].

The third level, E-empowering, supports active participation and facilitates "bottom-up ideas to influence the political agenda". This level emphasizes the strong need to allow citizens to influence and participate in policy formulation [9].

All 3 levels of participation refer to *the use of information and communication technologies for reaching a wider audience and increasing citizen participation*. According to Macintosh these elements are useful indicate the scale of participation [9].

Moreover, literature review further identifies the complexity of participation, indicating a social barrier comprised of several factors like: "the large and diverse range of stakeholders which have different needs and preferences; have diverse backgrounds, perspectives, and linguistic and technical capabilities". Macintosh underlines the need for e-Participation approaches which reflect these differences [8].

The effectiveness of e-Participation systems can be maximized only when the end users (citizens) are committed and having a proactive attitude towards policy making processes [8].

Some researchers motivate the failure of many unsuccessful e-Participation initiatives by lack of citizen involvement. "Though the technology platform appears deceptively simple and cheap to implement, many efforts fail to attract widespread interest amongst citizens or politicians, are unrepresentative, lead to poor information or poor quality of debate, or are monopolized by a few vocal contributors. A serious problem with these forms of e-Participation is citizen engagement – citizens do not necessarily become more willing to participate simply because net-services are provided for them" [10]. A key factor for the success of e-Participation is citizen involvement. E-Participation initiatives are "dependent on citizen engagement, interaction and social networking because democratic systems favor the interests of larger groups of citizens – the more voices behind a political proposition, the greater its chances of success" [10].

One of the reasons that make citizens de-motivated is the ignorance of relevant policies [11]. "Citizens often feel there is a glass barrier between their everyday life and the policy making processes in government" [12].

III. THE CONCEPT OF UBIPOL

In previous sections we augment that the failure of other e-Participation initiatives is determined by poor citizen involvement in policy making processes. Literature reveals that poor citizen participation comes from lack of interest and motivation.

The concept of UbiPOL is based on the idea that "the more citizens find connections between their everyday life and relevant policies, the more they become pro-active or motivated to be involved in policy making processes"[12].

To effectively address known obstacles in citizen commitment to policy making processes at all participation levels, UbiPOL makes use of context-aware and location-based services.

Both the concept of context-aware and Location-based Services are essential to UbiPOL. In UbiPOL, the context of the end user (citizen) triggers different system response and behavior for all participation levels.

Relevant information is provided to citizens on mobile devices based on their location, context, preferences and needs (E-enabling). And because UbiPOL makes use of mobile devices for service delivery, relevant policy making information is more accessible to end users "on the fly".

UbiPOL enables *top down consultation of citizens* on mobile devices (E-engaging). Policy makers can define consultations on matters of public interest. Citizens are informed and can express their opinion.

In UbiPOL citizens can also report on or generate issues to influence and *participate in policy formulation* at their own initiative (E-empowering). Other users can comment on citizen generated issues.

Context-awareness makes UbiPOL sensitive to user needs linking policy making processes to the everyday life of citizens at all participation levels. This increases the citizen's motivation to be involved in policy making processes leading to wider audience and increased participation.

IV. CONTEXT-AWARENESS IN UBIPOL

Information about location, identity and time is mandatory for most context-aware applications. Those variables are prerequisites of the primary data that the context-aware system must know in order to determine the specific contextual data or secondary information relevant to the user's task. As such, location and identity of the user and time are primary data variables for context in UbiPOL. The secondary information that constitutes specific contextual data for UbiPOL is closely related with the tasks made available by the platform to its users.

At its core UbiPOL is a participation platform for policy making. Its main objective is to enable the participation of citizens in the "policy making process from the middle of their everyday life, overcoming spatial and time barriers" [12]. UbiPOL deploys its services to end-users on mobile devices. UbiPOL mobile device applications are based on platform APIs available also for third party developers. The platform APIs provide basic mobile application development features facilitating the implementation of functionalities specific to policy making processes. One common generic task of the UbiPOL mobile application end-user is to be involved in the policy making process. In relation to this end-user task, we can define the specific contextual data for UbiPOL.

The following concepts constitute UbiPOL specific contextual data in close relation to common generic end-user tasks:

- Policy issues, defined by the policy maker through a web application (a UbiPOL server side component), in order to provide relevant information to citizens;
- Questions, questionnaire forms, voting polls also defined by policy makers and proposed for consultation to the citizens;
- Reported issues, opinions, comments and proposals raised or defined by other citizens, regarding various types of problems of interest.

All the concepts introduced above have corresponding entities in the UbiPOL domain model. The entities used to encapsulate policy making information are linked with location data. A policy issue for example can be related to one specific point of interest (like a school, a library, a museum) or more.

A brief reiteration enables us to define the information that makes up context in UbiPOL. The context in UbiPOL is comprised of primary data and secondary information (determined by the system based on the primary data). The primary data is location and identity of the user and time as for most context-aware systems. And policy making information available in the system (including policy issues, reported issues, questions, questionnaire forms, voting pools, opinions, comments, proposals etc) constitutes the base for secondary contextual data. Because the user is on the move the entities located in close vicinity are constantly changing.

Location and identity of the user are primary context data for UbiPOL. Location-based support in UbiPOL is provided through front-end APIs and core web service components. The front-end APIs provide access to built-in mobile device location functions enabling the use of GPS or network positioning methods. UbiPOL front-end APIs also include a communication manager component providing web service client modules. The core web service components used in conjunction with the front-end APIs provide both pull and push location-based support for mobile application development.

Up to now we've emphasized on context and location-based support in UbiPOL. Although location-awareness is a prerequisite for context-awareness, knowledge of the geographical position and identity of the user is not sufficient. One other essential feature of context-aware applications is to find out and behave according to the modifications that occur in their surrounding environment [3], [4]. In order to provide context-aware support UbiPOL is employing user profiles. A user profile in UbiPOL includes specific information that defines citizens' detailed preference in relation to policy making processes. Each citizen is allowed access to create and modify policy making filters from the mobile device. Citizens (UbiPOL end-users) are on the move and their location is constantly changing. Their common generic task (in relation to UbiPOL) is to be involved in policy making processes. The entities that comprise context for UbiPOL are linked with locations as well. So, as citizens move according to their everyday life pattern, the UbiPOL execution environment (entities surrounding them) is changing. Based on those changes and on user predefined preference, citizens receive notifications about relevant surrounding entities (policy issues, reported issues, etc) enabling them to get involved and participate in policy making processes.

V. GENERIC UBIPOL SYSTEM ARCHITECTURE

UbiPOL is a policy making platform with support on a wide rage of device types and mobile operating systems. It provides both front-end and back-end APIs to enable development of e-participation applications in the field of policy making. Besides a full set of customizable front-end and back-end APIs, UbiPOL also provides five web services exposing system features to third party developers. UbiPOL employs the system oriented architecture (SOA). With access to platform APIs and web services, UbiPOL developers can design and implement new policy making applications.

For cross-platform portability reasons the development programming language being used in UbiPOL is Java. Front-end APIs and components are based on the Java ME platform. Back-end APIs, components and web services are based on Java EE 6.

The typical system architecture, also employed in UbiPOL field trials, is shown Figure 1.

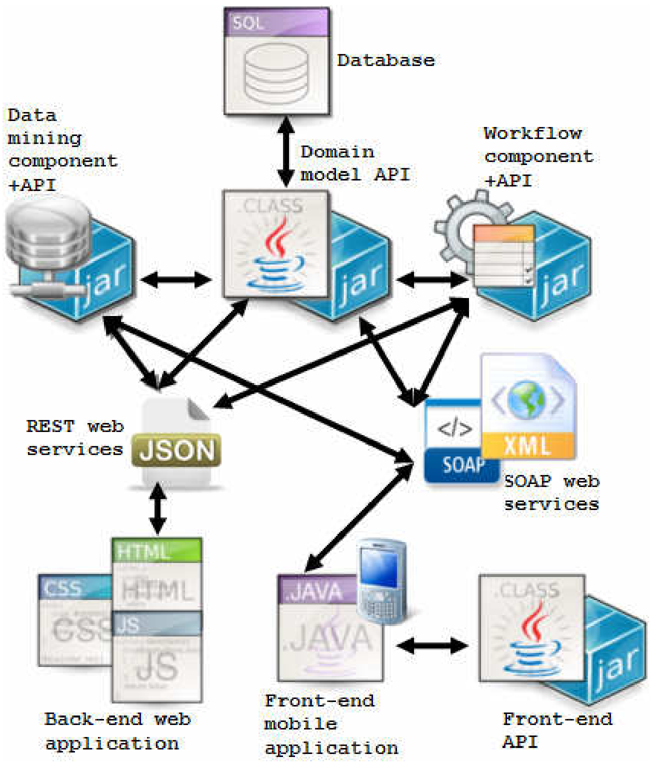


Figure 1. Typical UbiPOL system architecture

In Figure 1 we emphasize on relation between logical system components and APIs without highlighting hardware and communications infrastructure. Variations from this architecture are possible and fully up to the developer. When developing UbiPOL applications it is not mandatory to make use of all platform APIs, components and web services. The extent of platform components required is determined based on the scope of the policy making applications being developed.

A generic description of all components in the typical UbiPOL system architecture from Figure 1 is provided below:

- The UbiPOL database stores all system information. UbiPOL employs relational database architecture. All system specific contextual data introduced in Section IV is stored here: policy issues, questions, questionnaire forms, voting polls, reported issues, opinions, comments and proposals. The UbiPOL relational database model was determined based on analysis of 4 different policy making processes in 2 countries: UK and Turkey. Based on the analysis database table elements were designed to support specific policy making process data. The database also stores information related to system users, administrators, user profiles, fitters etc.
- The UbiPOL domain model which relates classes to tables from the database. Generally, each table in the UbiPOL database has its own counterpart in the domain, following the table per concrete class strategy. Based on this strategy we are able to have identical attributes on database tables and domain classes (like policy issue for example which is both a table in the database and an entity in the domain both having the same attributes). There are some particular exceptions when the joined table strategy is used. And since UbiPOL back-end components are based on Java EE 6, all constructs in the domain are JPA entity classes.
- The UbiPOL domain model API which handles all persistence operations in UbiPOL. This core platform API is comprised of EJBs with common persistence support for each entity class in the domain. All other UbiPOL back-end components rely on this API to handle persistence.
- The workflow component which manages and executes the policy making processes. This component is based on a set of back-end APIs and it facilitates the flow of information, tasks, and events for policy making processes in UbiPOL. It improves transparency by providing policy tracking functionality to end-users (citizens) throughout the execution of the process. This component relies on the domain model API for data access and persistence.
- The data mining component employs natural language processing and sentiment analyses techniques to identify and extract subjective information from citizens free text comments stored in the UbiPOL database. This component relies on the domain model API for data access and persistence.
- UbiPOL SOAP web services are ideal for exposing UbiPOL business logic to front-end applications.

Client SOAP based support is available out-of-box on a wide range of Java enabled smart phones and PDA devices. The use of JSR 172 APIs on UbiPOL client applications facilitates stub code generation. There are 5 UbiPOL web services available. Two of those provide location-based support: the Notification Service (push LBS support) and the Retrieval Service (pull LBS support). UbiPOL SOAP web services also define an intermediary layer of DTOs (Data Transfer Objects) used in XML serialization for the communication with mobile devices. This layer is mandatory in order to reduce the number of successive web method from the UbiPOL client. From this perspective a DTO can include information provided by several entity classes in the domain model that would otherwise be retrieved independently (each through a separate web method call). DTOs also facilitate convenient conversions to compatible JSR 172 data types before XML serialization. UbiPOL SOAP web services rely on the domain model API for data access and persistence. Also, some of the UbiPOL SOAP web service methods make use of other components and APIs (like workflow component APIs and data mining APIs) to provide required functionalities to front-end clients.

- UbiPOL REST web services are also used to expose business logic to UbiPOL applications. Although SOAP web services are also available, REST is used for exposing UbiPOL business logic to back-end applications. As for SOAP, UbiPOL REST web services also make use of some back-end components and APIs like: the domain model API, the workflow component API, the data mining API.
- UbiPOL back-end web applications provide support to policy makers allowing them to ask citizens for their opinion by defining policy issues, questions, questionnaire forms, voting polls etc. UbiPOL back-end web applications are fat-clients making use of little server resources and network connectivity. They rely on REST web services to handle some amount of the business logic.
- UbiPOL front-end applications are intended to be used by citizens on mobile devices. In UbiPOL, mobile device applications are also based on platform APIs (described in more detail in the next section). Built on platform front-end APIs, UbiPOL mobile applications provide multi-language support and a map based interface for policy issue visualization.

Although third party developers may decide on the extent of components required by their implementation, employing the Notification and Retrieval web services in conjunction with some front-end platform APIs is mandatory in order to deliver location-based and context-aware UbiPOL mobile applications. This topic is addressed in the next section of the article where we also provide some generic information about UbiPOL front-end APIs.

VI. Developing Location-based and Context-aware UbiPOL Mobile Device Applications

Context can be applied in a flexible way to entities that are on the move. That is why context-awareness is complementary to location-awareness. Built-in support for access to location makes mobile devices appropriate software application hosts for context-aware computing. UbiPOL enables the development of context-aware mobile applications in the field of policy making. Although several components are part of the UbiPOL platform (as described in Section V), only those front-end and back-end APIs and services required to deliver context and location-aware policy making applications are detailed in this section.

Primary contextual information in UbiPOL is location and user identity. Information about user location is determined at the front-end side. A policy map API is available for UbiPOL front-end application developers. The policy map API includes location support classes to determine user position and display markers at specific geographic coordinates. Markers on an UbiPOL map have policy information attached to them.

Secondary or specific contextual data is provided through web services. As mentioned before, two of the five UbiPOL web services are location-based. Those are the Notification service and the Retrieval service. UbiPOL front-end applications determine the position of the user based on the APIs provided by the platform. User position and identity are included as parameters in UbiPOL Notification and Retrieval web service method calls. Based on identity information the system is able to determine user preference and policy filters (part of the user's profile) from the database. Based on location information the system is able to determine the entities in vicinity of the user in relation to filters, preference and specific user task.

The common generic user task in relation to UbiPOL is to be involved in policy making processes. In order to be involved the user must first be informed. Information about policy making processes in UbiPOL is triggered at the user's request (pull LBS) or through notifications (push LBS). Notifications in UbiPOL are essential for context-awareness. They are intended to dynamically apprise the user in real time about his/her vicinity to locations with relevant policy information attached. One specific feature of context-aware applications is to detect and behave according to the modifications in their surrounding environment. Accordingly, UbiPOL mobile applications will deliver notifications to users about the changes in execution environment (policy making entities surrounding them). And, because UbiPOL users have different profiles and preferences, the system will behave differently for different users. Two citizens using the same UbiPOL system implementation might receive different notifications although in vicinity to each other and surrounded by the same policy making entities. In other words, the system behaves according to relevancy to the user's task.

In order to implement location-based and context-aware UbiPOL mobile device applications, the following minimal configuration of front-end APIs is required:

- The policy map API;
- The communication manager API;

Of course, a typical UbiPOL system implementation would employ the architecture described in the previous section making use of all platform components, APIs and services. But here we only emphasize on development of UbiPOL mobile applications with location and context-awareness support. Calls to Notification and Retrieval web service methods are handled by the front-end communication manager API. All UbiPOL front-end APIs described below are based on the Java ME platform. The entire UbiPOL front-end API comes packed into a JAR file for use by developers.

A. *The policy map API*

The policy map API is a graphical user interface class library. It is a wrapper around the Google Maps API providing general functionality required to display maps on Java ME enabled mobile devices. The Light Weight User Interface Toolkit (LWUIT) is used for implementation of the UbiPOL policy map API to preserve the same consistent look and fell on different mobile device platforms and operating systems.

The policy map API includes dedicated Java classes enabling support for the following generic functionalities:

- Sliding the map to different locations;
- Zooming in and out;
- Displaying makers at specific coordinates;
- Centering on the user's location.

Policy information in UbiPOL is related to location. Markers displayed at specific geographic coordinates on an UbiPOL map have policy data attached to them. In order to determine user position and display markers at specific geographic coordinates, the policy map API has dedicated location support classes. By default the policy map GUI object centers to the users position when displayed on a mobile device. So unless the user specifically slides to a different location, the map will be centered on the actual user position showing the surrounding markers.

Policy map location support is based on the Java JSR-179 API. Location specific classes in the policy map API extend on Java ME location classes (JSR-179) providing event driven functionalities on position changes to the application.

An essential component of the policy map API is the map class. This class is the actual GUI class providing map based user interface support for implementations. In order to make it deployable on different device types the map class was designed to automatically adapt to different mobile phone screen sizes and resolutions without any need of customization by the developer. The map GUI class is also capable to respond to sensor events triggered by changes in device orientation. Support for this functionality is based on the Mobile Sensor API (JSR-256). This feature does not need any customization by the developer as well.

The policy map GUI class was also programmed to respond to different mobile device input methods including: touch screen events, action key events and track-ball events. Figure 2 shows a sequence diagram that exemplifies the typical use of the policy Map GUI class in an UbiPOL mobile application.

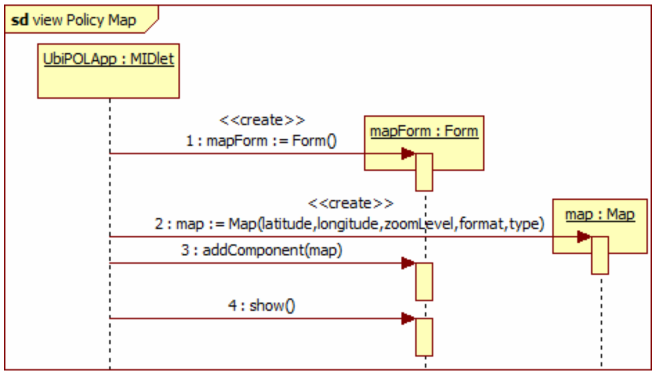


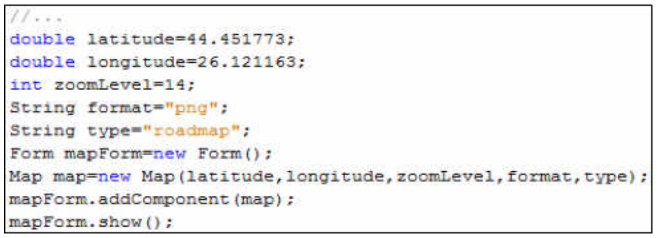
Figure 2. Sequence diagram for displaying the policy

Source code implementation for the sequence diagram in Figure 2 is pretty straight forward. The Map class extends the Container class from the LWUIT toolkit. So it can only be added and displayed on LWUIT Form object. Implementation of the first step in the sequence diagram requires instancing a LWUIT Form class by calling the constructor method. Then the Map class is instanced at steep 2. The Map class has a set of predefined constructors besides the implicit one. The one used at step 2 in the sequence diagram above requires passing the following arguments:

- The latitude and longitude that will be used as default initial position if location is not enabled. The values provided here will not be used if location is enabled.
- The initial zoom level that the map should display. The user can change the level at run time by zooming in or out (this feature is preprogrammed in the map class which responds to user generated zoom in and out events).
- The map format that will be used to display tiles. This can have one of the following Google predefined values: png8 or png, png32, gif, jpg, and jpg-baseline. The values can be changed at runtime.
- The map type that will be used to display tiles. This can have one of the following Google predefined values: roadmap, satellite, terrain, and hybrid. The values can be changed at runtime.

At step 3 in the sequence diagram from Figure 2 the new instance of the map object is added to the LWUIT form. Then at step 4 the Form object is displayed by calling its show method.

The following snippet is the source code implementation required for the sequence diagram in Figure 2. The arguments for the Map class constructor method are assigned values and defined ahead of the constructor call. When used in an MIDP application, the source code snippet from Figure 3 will display the UbiPOL policy map as shown in Figure 4. The screenshot from Figure 4 was made on a N8 smart-phone device from Nokia, for UbiPOL front-end API experimentation purpose.



```
//...
double latitude=44.451773;
double longitude=26.121163;
int zoomLevel=14;
String format="png";
String type="roadmap";
Form mapForm=new Form();
Map map=new Map(latitude,longitude,zoomLevel,format,type);
mapForm.addComponent(map);
mapForm.show();
```

Figure 3. Source code example for displaying the policy map

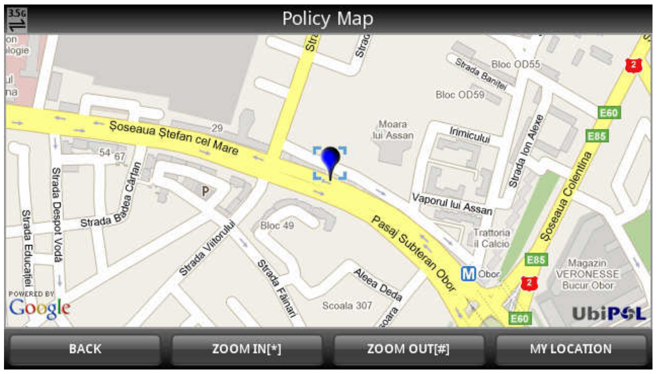


Figure 4. The UbiPOL front-end Map object running on a Nokia N8 smart-phone

The following 2 figures show the behavior of the Map class object as response to sliding and zooming events generated by the user. Sliding and zooming are built-in features of the UbiPOL front-end Map object. As a result, the developer does not need to write any source code to support those features. By default the Map class implementation will execute sliding operations as response to arrow key events on phones with key pads. For touch screen phones pointer drag events are supported. The predefined keys for zooming in and out on phones with keypads are the POUND key (for zoom out) and the STAR key (for zoom in). For touch screen devices the developer has to add menu buttons for the 2 operations to catch user generated events (see Figure 4). But the zooming feature is built-in through 2 Map class methods that the developer can call. On phones with both touch and keypad input devices both options are available by default at the same time.

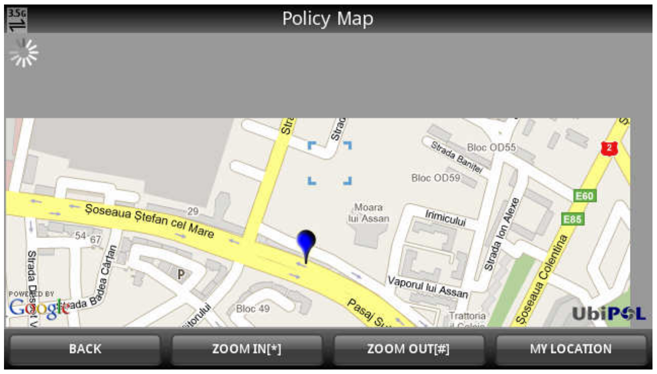


Figure 5. UbiPOL front-end Map object slide action on a Nokia N8 smart-phone

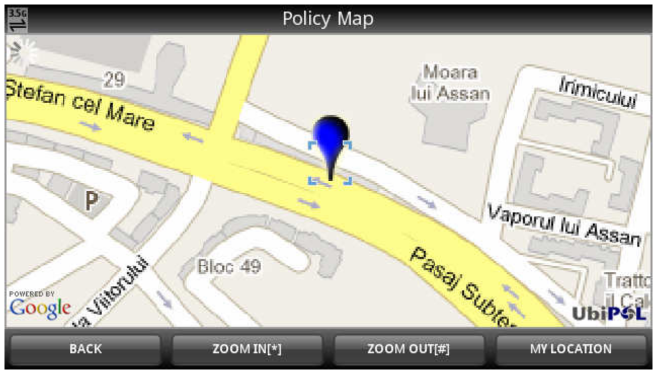


Figure 6. UbiPOL front-end Map object zoom in action on a Nokia N8 smart-phone

B. *The communication manager*

Policy making data in UbiPOL is delivered to mobile devices through SOAP web services. Aside from back-end web service implementations, UbiPOL provides a front-end API for communication with the server. The front-end communication manager API is made up of 5 sub-packages containing Java classes. There are 5 UbiPOL SOAP web services available: Authentication, Knowledge Sharing, Notification, Retrieval, and Tracking. So the front-end communication manager is structured in 5 corresponding sub-packages (one for each UbiPOL web service).

The UbiPOL front-end communication manager is based on the Java ME web service specification API (JSR-172).

Each corresponding web service client sub-package in the communication manager API has a generic structure including:

- An interface extending the "java.rmi.Remote" interface. This interface defines method signatures for all the operations exposed by the specific service. The method signatures are the same as for the server side operations exposed.
- A stub class that implements the service interface (mentioned above) and the "javax.xml.rpc.Stub" interface.
- A set of supporting classes mapping the data types (DTOs) returned by the specific service. The supporting classes are be simple POJOs with attributes, setters and getters.

As mentioned in the previous sections of the article, two of the 5 UbiPOL web services are essential for location-based and context-aware support. Those are Notification and Retrieval. So, in order to develop context-aware mobile applications the use of both Notification and Retrieval classes in the front-end communication manager API is mandatory. Also, in order for an UbiPOL mobile application to gain access to any back-end web service operations, the use of the front-end Authentication classes in the communication manager API is required.

The Retrieval service provides pull LBS support. Policy making information related to location can be retrieved at the user's specific request. User preference and policy filters are applied to the queries besides location constraints.

The Notification service provides push LBS support. The user does not specifically request the data although subscribing to the service is required. There are 2 Notification service operations that provide support for subscribing and unsubscribing a user. One other relevant web method is the “notify” operation available for delivering the actual notifications. The “notify” web method receives location and user identity information as input parameters from the mobile device. It factors in user preference, policy filters and location constraints in the reply. This web service operation only provides a Boolean response (true or false) in relation with the availability of relevant policy information in the vicinity of the user. But the actual relevant policy information must be acquired by calling Retrieval web service operations. In other words, both Retrieval and Notification front-end communication manager classes are required for context-aware support.

The policy making data retrieved can be displayed through a map based interface (using the policy map API) or through a set of custom design GUI objects available in the UbiPOL front-end API.

The following 3 figures show examples for delivery of notifications to the user, displaying the retrieved data on a map and also with a custom GUI policy issue list object available in the UbiPOL front-end API.

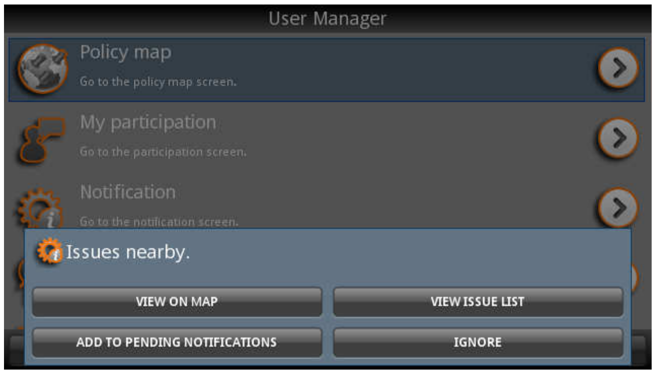

User Manager
Policy map
Go to the policy map screen.
My participation
Go to the participation screen.
Notification
Go to the notification screen.
Issues nearby.
VIEW ON MAP
VIEW ISSUE LIST
ADD TO PENDING NOTIFICATIONS
IGNORE

Figure 7. Delivery of notifications to the user on a Nokia N8 smart-phone

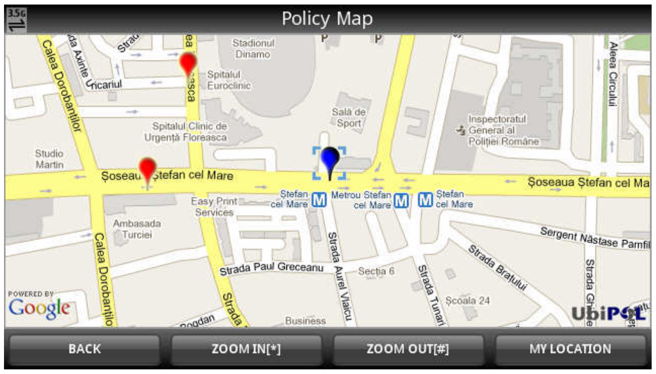
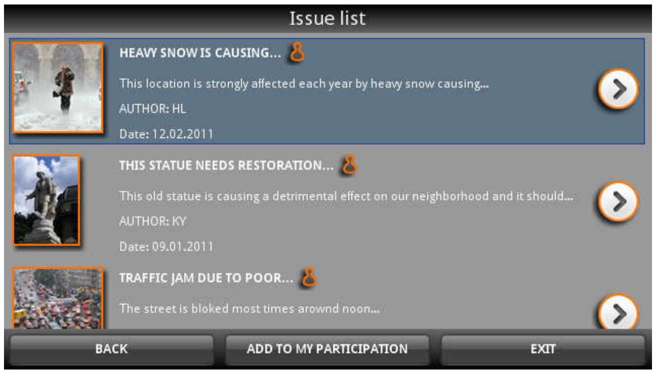


Figure 8. Displaying policy data on the UbiPOL front-end map


Issue list

HEAVY SNOW IS CAUSING...
This location is strongly affected each year by heavy snow causing...
AUTHOR: HL
Date: 12.02.2011

THIS STATUE NEEDS RESTORATION...
This old statue is causing a detrimental effect on our neighborhood and it should...
AUTHOR: KY
Date: 09.01.2011

TRAFFIC JAM DUE TO POOR...
The street is bloked most times arownd noon...

Figure 9. Displaying policy issue data with a custom GUI list object from the UbiPOL front-end API

VII. CONCLUSIONS

Throughout the article we talk about context-awareness and Location-based Services (LBS) which are essential to UbiPOL. Based on literature review we underline the need for ICT systems that motivate citizens to be involved in policy making processes. We define the common task of users in relation to UbiPOL and then we specify the elements that comprise context for the platform. We show how UbiPOL makes use of context-awareness to increase citizen motivation in policy making processes. A generic UbiPOL system architecture is also included. In the last sections we focus on the provisioning of location-based and context-aware UbiPOL mobile device applications.

To conclude, we briefly reiterate and summarize some of the obvious advantages of UbiPOL presented in previous sections of the article.

The concept of UbiPOL is based on linking policy making processes to the everyday life of citizens at all participation levels. This will increase the level of motivation and commitment of citizens leading to wider audience and increased participation. Context-awareness is essential for the concept of UbiPOL. The citizen (UbiPOL user) is constantly on the move. The entities (policy issues, reported issues, etc) that comprise context for UbiPOL are linked with locations. So, as citizens move according to their everyday life pattern, the UbiPOL execution environment (entities surrounding them) is changing. Based on those changes and on user predefined preference, citizens receive notifications about *relevant* surrounding entities (policy issues, reported issues, etc) enabling them to get involved and participate in policy making processes “on the fly”.

UbiPOL involves citizens in policy making process at all participation levels. It employs both approaches: top-down consultations (policy maker led initiatives) and bottom-up participation (citizens can generate policy issues).

UbiPOL targets both policy makers (who can deploy it as it is) and developers (who are provided with a set of generic front-end and back-end APIs). UbiPOL front-end and back-end APIs are implemented to support generic requirements determined based on the review of 4 different policy making processes in 2 countries.

UbiPOL allows citizens to define their own context by considering the different needs and requirements of its users. In UbiPOL citizens can define their preference and interests with regard to relevant policy issues. This way, citizens can influence how the UbiPOL system reacts to them.

Both UbiPOL front-end and back-end APIs are based on Java. This makes UbiPOL APIs and applications deployable on a wide range of desktop and mobile operating systems.

UbiPOL also provides a set of 5 core web services to address the needs of third party developers that are implementing on platforms and operating systems that don't support Java (like iPhone for example).

ACKNOWLEDGMENT

The work presented in this article was carried out in the FP7 UbiPOL (Ubiquitous Participation Platform for Policy Making) project (grant agreement no. 248010).We acknowledge that parts of the content in the paper are based on the discussion among UbiPOL partners.

REFERENCES

[1] A. K. Dey, and G. D. Abowd , "Towards a better understanding of context and context-awareness," Proceedings of the first international symposium on Handheld and Ubiquitous Computing (HUC '99), Springer-Verlag London, UK ©1999, ISBN:3-540-66550-1.

[2] A. K. Dey, "Providing architectural support for building context-aware applications," Ph.D. thesis, College of Computing, Georgia Institute of Technology US., 2000, unpublished,http://www.cc.gatech.edu/fce/ctk/pubs/dey-thesis.pdf, retrived on 19.09.2011

[3] B. N. Schilit, and M. M. Theimer, "Disseminating Active Map Information to Mobile Hosts," IEEE Network, Volume 8

[4] A. Schill, "Context-aware applications," Technische Universitat Dresden, Department of Computer Science Institute for System Architecture, Chair for Computer Networks,http://www.rn.inf.tu-dresden.de/lectures/MCaMC/09_Context-aware%20Applications.pdf, retrived on 20.09.2011

[5] M. Rosemann, and J. Recker "Context-aware process design: exploring the extrinsic drivers for process flexibility," In T. Latour & M. Petit (Eds.) The 18th International Conference on Advanced Information Systems Engineering. Proceedings of Workshops and Doctoral Consortium, Luxembourg, Namur University Press, Grand-Ducy of Luxembourg, 2006, pp. 149–158.

[6] K. Virrantaus, J. Markkula, A. Garmash., Y. V. Terziyan, "Developing GIS-Supported Location-Based Services," Proceedings of First International Workshop on Web Geographical Information Systems (WGIS'2001), Kyoto, Japan, pp. 423–432.

[7] S. Steiniger, M. Neun, and A. Edwardes, "Foundations of location-based services, lesson 1 CartouCHe1 - lecture notes on LBS, V. 1.0," Technical Report, University of Zurich, 2006,http://heanet.dl.sourceforge.net/project/jumppilot/w_other_freegis_documents/articles/lbs_lecturenotes_steinigeretal2006.pdf, retrived on 19.09.2011

[8] A. Macintosh, "Challenges and barriers of eParticipation in Europe?", 2007, Paper presented at the Forum for Future Democracy, http://www.sweden.gov.se/content/1/c6/08/49/42/9d411e53.pdf, retrived on 17.09.2011

[9] A. Macintosh, "Characterizing E-Participation in Policy-Making", 2004, Proceedings of the 37th Hawaii International Conference on System Sciences

[10] Sæbø, O., Rose, J. and Nyvang, T. (2009) The Role of Social Networking Services in eParticipation in Macintosh, A. and Tambouris, E. (Eds.) (2009) eParticipation, LNCS 5694, pp. 46–55

[11] J. G. March, and J. P. Olsen, "Institutional perspectives on political institutions.", In M. Hill (Ed.), The policy process: A reader (2nd ed., pp. 139–155). London: Prentice Hall/ Harvester Wheatsheaf

[12] Irani Z, Lee H, Weerakkody V, Kamal MM, et al. (2010) Ubiquitous Participation Platform for Policy Makings (UbiPOL) - a Research Note, International Journal of eGovernment Research, 6 (1), 78 - 106.