Enabling Cross-Platform Mobile Application Development: A Context-Aware Middleware

Achilleas P. Achilleos and Georgia M. Kapitsaki

Department of Computer Science, University of Cyprus,
1 University Avenue, Nicosia, Cyprus
http://www.cs.ucy.ac.cy

Abstract. The emergence of mobile computing has changed the rules of web application development. Since context-awareness has become almost a necessity in mobile applications, web applications need to adapt to this new reality. A universal development approach for context-aware applications is inherently complex due to the requirement to manage diverse context information from different sources and at different levels of granularity. A context middleware can be a key enabler in adaptive applications, since it can serve in hiding the complexity of context management functions, promoting reusability and enabling modularity and extensibility in developing context-aware applications. In this paper we present our work on a cross-platform framework that fulfils the above. We elaborate on the need for cross-platform support in context-aware web application development for mobile computing environments identifying gaps in the current state of context support. The paper introduces the architecture of the middleware that fills these gaps and provides examples of its main components. An evaluation based on the development of a prototype, web-based, context-aware application is detailed. The application is compared against an analogous hybrid mobile application showing the evolutionary potential introduced via the middleware in delivering context-aware mobile applications.

Keywords: web applications, context middleware, context-awareness, HTML5, mobile computing.

1 Introduction

The growth of mobile devices capabilities offered by a variety of mobile platforms (e.g., iOS, Android, Windows Phone) increases the interest of users in personalized applications that can be accessed from any place and platform. This is reflected in a mantra coined in 1991: "*giving all the freedom to communicate anywhere, anyplace, anytime, in any form*" [1]. This vision is now more profound due to the explosion of hardware, software and communication technologies provided by an abundance of smart devices. Mobile users are best served if the above goal is fulfilled by keeping users requirements and their need for control in top priority, without implementing applications that overwhelm users with redundant information and futile functionality. Such a personalized technology view

B. Benatallah et al. (Eds.): WISE 2014, Part II, LNCS 8787, pp. 304–318, 2014.

has been adopted by various researchers to improve the user experience of web applications [2], [3]. Currently the personalized perspective has been enriched by the widespread presence of mobile sensors, such as GPS (Global Positioning System) receivers, accelerometers, compasses, microphones and cameras that are integrated in a mobile device [4], as well as the huge volume of social network data available from different locations.

In order to keep up with these advancements, technologies should adapt to the end-user's perspective and aid in tailoring applications to the surrounding context (i.e., location, situation, social data) exploiting the capabilities of smart devices and platforms. Realization of this smart vision demands mechanisms for ubiquitous and reliable acquisition, analysis and sharing of information to improve user requirements, by anticipating user needs while the user remains undisturbed by the underlying technology. Also, application developers that target those environments should have access to a transparent infrastructure and a set of reusable context modules that support cross-platform development of such context-aware applications.

Nowadays mechanisms and technologies that target cross-platform development in the field of web applications are emerging, e.g., HTML5 [5]. Nevertheless, a universal web application framework tailored to the needs of context-awareness is missing. In this work we elaborate on these issues presenting the current state on cross-platform web development and proposing a solution that facilitates the development of such applications providing software engineers with reusable and extensible elements. The contribution of this paper is twofold: on the one hand, it provides an analysis and review on the HTML5 support in mobile web browsers of widely-used mobile devices and, on the other hand, it gives on overview of the proposed HTML5 Context Middleware (H5CM) that serves as a facilitator for cross-platform development of context-aware web applications.

Regarding the former part of our contribution we have performed a study on HTML5 support provided by widely used mobile browsers in different mobile platforms. For the latter part we have exploited the results of this study in order to design and implement a generic context management middleware that relies on the HTML5 specification and the latest developments in mobile platforms and browsers. The H5CM framework provides reusable context sensing and reasoning modules in the form of plugins and offers the ability to extend this pool of plugins through the development of additional modules. Modules are provided in an hierarchical ordering. H5CM supports context-management functions through a common interface to multi-domain context sources for facilitating the development of web-based context-aware mobile applications.

The rest of the paper is organized as follows. Section 2 introduces the notion of context-awareness in mobile platforms settings that is acting as motivation for our work, whereas section 3 analyses the support of HTML5 features relevant to context-awareness on popular web browsers. We then describe in section 4 the main concepts of the H5CM framework. The framework's plugin concept is analysed through examples in section 5 along with the presentation of how the plugins are used in a mobile application. A comparison and evaluation of

the context-aware application is performed in section 6, against an analogous mobile application developed using hybrid technology (i.e., PhoneGap). The final section concludes the paper.

2 Context-Awareness in Mobile Platforms

Context-awareness is a key enabling requirement for enhancing the ability of mobile users to exploit efficiently different software applications and services using different mobile devices (e.g., smartphones, tablets, and netbooks) at different locations. In mobile computing context-awareness encompasses various aspects spanning from sensing information at the hardware and network information level, to context-based recommendations at the application level, such as the case of context-based music recommendation presented in a previous work [6].

The popularity gained by different mobile platforms enables users to perform everyday tasks and use diverse devices for various work and leisure activities [7]. In such mobile settings, the user needs to be dynamically assisted, by tailoring the application and providing proactive behaviour. Mobile devices offer clear-cut user benefits, but there are application usability and ease of use limitations (e.g., small screen, low battery) that must be considered. Perhaps the key issue that needs to be addressed is this diversity of platforms even when only one user is considered, since users own and use different devices.

The term of context-awareness as already introduced describes the process of acquiring, managing and distributing different pieces of context to intelligently adapt the application behaviour. We adopt the definition of Dey and Abowd [8] which states that context is *any information that can be used to characterize the situation of an entity, in which the entity can be a person, a place, or a physical or computational object that is considered relevant to the interaction between the entity and the application.* Directly from this definition it is apparent that the requirement arises to be able to support diverse context sources via a modular, reusable and extensible mechanism. In specific, four driving requirements, namely modularity, extensibility, code reusability and cross-platform development, motivated us to define and develop a context-aware, web-based middleware to handle context-awareness support in different environments.

Comparable approaches that handle context data in different platforms can be found in the literature, such as MUSIC [9], [10] for the Java platform and Really Simple Context Middleware (RSCM) [11] for the Android platform. The essential difference of the current work with existing solutions lies on the application type focus, which is on the web development, rather than on native applications that are not universally exploitable in various environments. The former case of MUSIC refers to the results of a European project on Self-Adapting Applications for Mobile Users in Ubiquitous Computing Environments and its proposed middleware runs using Java OSGi. The latter is available in Google code and focuses on Android, allowing the development of context-aware applications using context plugins.

Differentiating from approaches that are tailored to specific platforms the main driving motivation of this work is platform-independence. This point has

also been reflected in hybrid technologies that constitute a first attempt to satisfy the assortment of mobile devices and platforms. Examples of such technologies can be found in PhoneGap [12], Apache Cordova, i.e., the open-source engine that runs PhoneGap, and Titanium[1]. The above solutions provide cross-platform mobile development environments via a set of uniform JavaScript libraries that can be invoked by the developer, wrapping (i.e., calling) device-specific native backing code through these JavaScript libraries. This provides access to native device functions, such as the camera or accelerometer from JavaScript. The main criticisms against hybrid development environments is that the developer needs to learn how to use the native libraries for each platform, but most importantly that mobile devices are not fast enough to smoothly run a hybrid application [13]. On the other hand, native applications offer benefits in terms of performance and API coverage. Still native applications lack in terms of instant deployment, since they require manual installation and/or upgrades, and flexibility to combine data from different resources [14].

In contrast to existing solutions we propose to exploit the rapidly developing HTML5 capabilities of mobile browsers to define a middleware that supports cross-platform application development using only web technologies [15]. In this way, performance issues that are apparent in hybrid applications, and complex deployment issues that appear with native applications can be avoided.

3 HTML: Features and Browser Support

The current landscape of mobile application development can benefit from the new features of web technologies that provide interactivity and animation, and support cross-platform development. HTML5 is the W3C specification referring to the new version of HTML [5]. The first public working draft of the specification was made available in January 2008, while the specification is currently at a very mature stage and undergoes continuous development.

The question on whether HTML5 features are supported and can be exploited in miscellaneous platforms remains open and is an issue that is constantly evolving. In this section we present the results of a study performed on different mobile platforms, examining their support for HTML5 features. The comparison was performed with the aid of a simple prototype application that we developed for this purpose. While the survey provides also data on the general support of HTML5 features, the focus of this work is on browser APIs that support features with a context-awareness flavour, such as mobile device geolocation tracking and monitoring, device orientation, motion and acceleration, monitoring battery level, ability to connect to RESTful and SOAP Web Services, etc. The main conclusion drawn is that mobile and tablet browsers support most features including context-relevant ones, since vendors try to keep in line with the evolving HTML5 specification. Moreover, as new sensors are added to mobile devices additional features are continuously added by browser vendors, by implementing new APIs in different browsers.

[1] `http://www.appcelerator.com/titanium/`

Table 1. Browser Support - [Samsung Galaxy Note 3 running Android 4.4.2 - Also tested on S3, Tab 2], [Apple iPad 2 running iOS 7.1.2: Also tested on iPhone 4S]

Name	**Default** (Android)	**Dolphin Browser HD 10.3.1** (Android)	**Firefox Mobile 31.0** (Android)	**Opera Mini 7.5** (Android)	**Opera Mobile 22.0** (Android)	**Chrome Dev 36.0.1985.128** (Android)	**Safari Browser - Default** (iOS)	**Dolphin Browser 6.5.1** (iOS)	**Opera Mini 8.0.1** (iOS)	**Chrome Browser 36** (iOS)	**Mercury Browser 8.6.1** (iOS)
Popularity (Max. 5)	-	4.5	4.5	4.4	4.5	4.5	-	4	-	4	4.5
Score (Max. 555)	475	463	483	53	486	492	410	410	410	410	410
Device Orientation	✓	✓	✓	x	✓	✓	✓	✓	x	✓	✓
Device Motion	✓	✓	✓	x	✓	✓	✓	✓	✓	✓	✓
Geolocation	✓(x)	✓(x)	✓	x	✓	✓(x)	✓	✓	✓	✓	✓
Battery Status	x	x	✓	x	x	x	x	x	x	x	x
Device Media	x	✓(x)	x	x	✓	✓(x)	x	x	x	x	x
Device Network	✓	✓	✓	✓(x)	✓	✓	✓	✓	✓	✓	✓
Web Sockets	✓	✓	✓	x	✓	✓	✓	✓	x	✓	✓
RESTful services	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SOAP services	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Legend - Supported: ✓, Not supported: x

The growing collection of HTML5 features and the ability to display 3D graphics natively in a browser, i.e., via WebGL that is the key standard supported by many browsers [16], make cross-platform development of applications a tangible target. The above table showcases the features considered critical to enable context-awareness. A score is shown as calculated by the website html5test.com in respect to browser support coverage for all HTML5 features. Popularity is also shown based on available user reviews and the total number of downloads, as retrieved for each application from the respective platform store.

In particular, Table 1 presents the support in different browsers for the Android and iOS platforms. Note that the values shown in parenthesis in the table indicate the features that are not supported on the Galaxy Tab 2 tablet, which are though supported on smartphones Note 3 and Galaxy SIII. In overall, smartphone and tablet devices provide support for the HTML5 features (in five out of six main browsers) up to a level of 86.8%, while most of the key features that enable context-awareness are supported. Only the Battery Status and Device Media APIs are currently lacking support in most browsers, with browsers like Firefox Mobile 31.0 and Opera Mobile 22.0 providing support also for these features. In this regard an application that needs to support, e.g., Battery Status, can be executed at least on Firefox Mobile 31.0 for Android.

The iOS platform and the browsers available for iOS offer high support in terms of general HTML5 features, up to the level of 68.8%. At the same time support for context-aware specific features is also at the highest level with 7 out of 9 features being supported through APIs provided by the different browser vendors. In particular, only two features are not supported and these refer again to the Battery Status and Device Media APIs. On the basis though of the continuous and progressive strive to implement new APIs (especially in terms of media support) we strongly believe that support for these features will soon arrive also on the browsers running on the iOS platform [17].

Finally, HTML5 support was also tested for the MS Windows Phone platform on the Nokia Lumia 800 mobile device running Windows Phone 7.1. A detailed presentation is omitted due to the low support encountered in this case in terms of HTML5 features. In specific, the support was exceedingly limited for context-aware features, with mainly the Device Network and Geolocation APIs being supported in some of the five browsers (i.e., Explorer, UC Browser, Explora, MetroUI and Incognito) that were considered in the test. Moreover, the support of generic HTML5 features was approximately 25% for the MetroUI browser, while support in all other browsers was basically non-existent. In the future we aim to test HTML5 support in browsers, as soon as they become available in newer mobile devices that support the Windows Phone 8 platform or higher.

4 HTML5 Context Middleware

Having in mind the necessity of providing cross-platform access to context-aware elements we have designed the HTML5 Context Middleware to allow developers to reuse existing modules making their code more compact. The implementation of the H5CM takes into consideration the four basic requirements of modularity, extensibility, reusability and the "develop once deploy on any platform" approach. H5CM follows a hierarchical structure: at the lower level there are context-sensor modules (*s-m*) that allow acquiring and distributing low-level context information that is relevant to the mobile device, the end-user and the environment. At the second level of the hierarchy there are context-reasoner modules (*r-m*) that accept low-level context from one or more sensor modules and apply the appropriate reasoning logic so as to create high-level context information. The context-aware application is at the top of the hierarchy and is

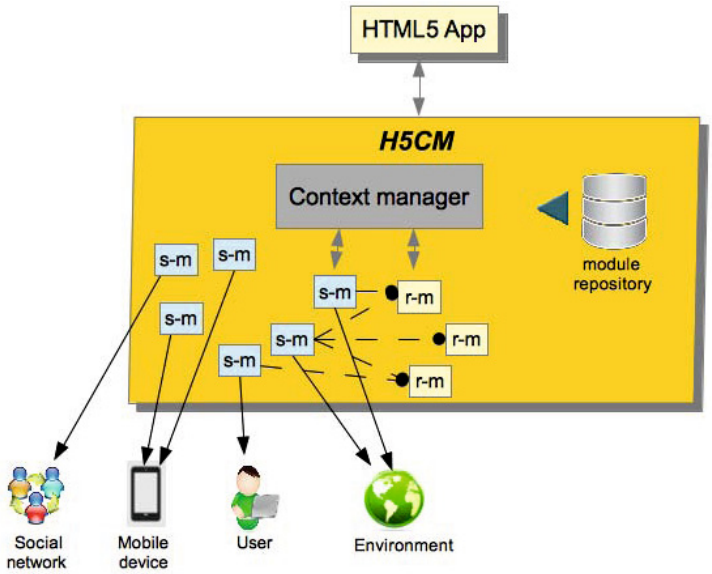

Fig. 1. Main elements of the HTML5 Context Middleware

able to communicate at the software level with sensor and/or reasoner modules to acquire context information that enables the adaptation of the application's logic.

The key architectural elements of H5CM are illustrated in Fig. 1. The *Context manager* constitutes a central point that handles information originating from different sources as gathered by respective plugins and distributes this information accordingly to the context-aware applications that require it. These modules (or plugins) act as enablers of context-awareness, empowering applications to be adapted to end-user preferences and circumstances. The plugins cover four main categories:

- modules that allow context acquisition directly from the mobile device (e.g., battery level)
- modules that provide information retrieved from social networks (e.g., LinkedIn, Delicious)
- modules that obtain input from the end-user, and
- modules that obtain information from the environment connecting to local or remote sensors or servers (e.g., room temperature)

In specific, the *Context manager* orchestrates the management of modules that retrieve information from the environment, the user and the device. The *Context manager* is implemented as a singleton JavaScript web module, while the context plugins are also implemented as JavaScript web modules. Hence, the context-aware application can be also implemented using web technologies (i.e.,

HTML5, CSS, JavaScript). Through the *Context manager*, context modules (i.e., sensors, reasoners) required for an application are registered and loaded. In fact, H5CM and the *Context manager* are based on the popular Q JavaScript framework[2], which offers the Promise object and complies fully with the Promises/A+ Specification[3]. A promise represents the result of an asynchronous operation. Interacting with a Promise object through its *then* method allows registering callbacks and receiving a fulfilled promise return value or the reason for failure. Using this approach it is guaranteed that the context plugins required by the application are loaded successfully.

At this point the application is able to interact and communicate with the loaded context plugins (e.g., GPSCoordinates, BatteryLevel). The loaded plugin(s) is(/are) actually registered with the *ContextManager* singleton, which initializes the associated context property and creates a new *CustomEvent* assigning the name of the context property as the unique identifier for the event. The context-aware application defines an *EventListener* associated with the above context event, which allows monitoring and updating the application with relevant context changes. The context plugin is responsible to update the context property and dispatch a unique event, when new context is available.

The above functionality is accompanied by an extensible and reusable *Context Repository*. The modules define an extensible and reusable repository. On the one hand, they can be reused by developers, since they are generic and can be invoked from any context-aware web application. On the other hand, the module set can be extended by technical users that need additional functionality as features of HTML5 expand and the respective support in mobile browsers is extended. The modules to be loaded are defined as dependencies in the application. Since the values of context data change more or less frequently depending on the type of the data, H5CM is responsible for monitoring all context variables and updating the application when relevant context changes are detected. This functionality offers a context monitoring that allows the adaptation of the application while it is executing.

As aforementioned, in order to provide context information at different levels of granularity a concept introduced in H5CM is the differentiation between sensor- and reasoner-modules. The former provide access to raw context data in the form that these can be collected from context locations (e.g., user geographical coordinates), whereas the latter give access to higher level context information that are derived from the "basic" raw data and are meaningful to the end-user. As a reasoner example consider the use of the raw data provided by: 1) the battery status that actually comprises of the battery level of the device and the information on whether the device is actually charging, 2) the coordinates of the user location and 3) the device motion, to provide the information of whether the user is currently walking on a street or driving. Application developers are able to contribute at this level by developing additional reasoner-modules that aggregate content from different context-sensor modules.

[2] Q Promises JavaScript Framework - `http://documentup.com/kriskowal/q/`.
[3] Promises/A+ Specification - `http://promises-aplus.github.io/promises-spec/`.

5 Context-Aware Sensor and Reasoner Modules

In this section we present the H5CM in more detail giving information on how to make use of the context-middleware modules to build context-aware applications. The implementation of the H5CM is available at Google Code[4], which includes the context manager web component and the available context sensor and reasoner modules. The presentation of these context modules serves as a guideline for developers on how to develop additional context-aware sensor and reasoner modules, enriching thus the functionality of the middleware, but also how to use these modules for the implementation of context-aware applications. At the time we have developed 8 sensor and 2 reasoner plugins that constitute the current state of the middleware. We present for demonstration purposes in more detail the *BatteryLevel* and the *FacebookConnect* sensor modules along with the *BatteryAnalyzer* reasoner module.

Listing 1. Battery Level Sensor Module.

```
1.  function getBatteryLevel() {
2.     var singletonContextManager = ContextManagerSingleton.getInstance("batteryLevel");
3.     var battery = navigator.battery || navigator.mozBattery || navigator.webkitBattery;
4.     if (battery != undefined) {
5.        var bLevel = Math.round(battery.level * 100);
6.        battery.addEventListener("levelchange", updateBatteryStatus);
7.        return bLevel;
8.     }
9.   . . . . .
10. function updateBatteryStatus() {
11.    var bLevel = Math.round(battery.level * 100);
12.    singletonContextManager.batteryLevel.batteryLevel = bLevel;
13. }
```

The *BatteryLevel* sensor-module retrieves and continues monitoring the level of the device battery (e.g., 74%), as required by the context-aware application. This is a hardware specific sensor plugin, while the *FacebookConnect* is a social sensor plugin that provides the means to acquire profile data of the user (i.e., public user data). It also allows asking the user to grant access for retrieving additional restricted data (given in comma separated string values). As a prerequisite the user of the mobile, web application needs to provide his/her Facebook credentials and allow the application to access these restricted data. *FacebookConnect* sensor module provides the connection point to Facebook, but similar functionality can be achieved by implementing modules for additional social networks, such as Google+, LinkedIn or Delicious. Note that security and privacy of user data is supported and respected through the use of advanced mechanisms provided by the different social networks. Moreover, hosting and deployment of the context-aware application on a secure web server (e.g., Apache) provides the capability of layering the Hypertext Transfer Protocol (HTTP) on top of the Secure Sockets Layer (SSL) protocol, thus adding the security capabilities of SSL to standard HTTP communications.

Listing 1 presents part of the implementation of the battery level sensor module that allows at first to register this plugin to the Singleton *ContextManager* instance (i.e., line 2) that in turn attaches this context to the *ContextListener*. Following, the

[4] `https://code.google.com/p/h5cm/`

battery is accessed by using one of the three web engines powering different browsers (i.e., line 3) and the level of the battery is obtained, while an event listener allows monitoring changes to battery level (i.e., lines 4-8). As soon as battery level change event is fired the *updateBatteryStatus* function is invoked (i.e., lines 10-13) that updates and distributes via the singleton *ContextManager* instance the new battery level to the application or reasoner depending on the application scenario.

Listing 2. Invoking Battery Level Sensor and Reasoner modules.

```
1.  var cms = ContextManagerSingleton.getInstance("application");
2.  . . . . .
3.  batteryLevel = getBatteryLevel();
4.  document.getElementById("batteryLevel").innerHTML = batteryLevel;
5.      reasoning = getBatteryAnalyser(parseInt(batteryLevel),30);
6.   . . . . .
7.  var coords = getGPSlocation();
8.  cms.gpsCoordinates.watch("gps", function (id, oldval, newval) {
9.  . . . . .
10. if (reasoning == false){
11.  document.getElementById("gps").innerHTML = "Latitude: " + myLat + " Longitude: " + myLng;
12. }
13. else if (reasoning == true){
14.  document.getElementById("gps").innerHTML = "Latitude: " + myLat + " Longitude: " + myLng;
15.  mapThisGoogle(myLat,myLng);
16. }
17. return newval;
18. });
19. . . . . .
20. var fbData =new Array("name","email","gender","username","birthday");
21. getFBInfo(fbData);
```

Based on the battery level we have defined a module that allows reasoning and adapting the application. The developer is allowed to set a cut off value for the battery level that defines if the application should restrict its functionality to preserve resources. The example application code shown at line 3 (Listing 2), allows invoking firstly the *BatteryLevel* sensor module presented in Listing 1. Then as shown in Listing 2, based on the value detected for the battery level the *BatteryAnalyser* reasoning module (see Listing 3) is invoked, which takes the final decision by comparing the two values and returning the result in the form of a boolean variable. In the example demonstrated this allows to make a decision (lines 7-16), when the location tracking sensor module is called, whether to show the location of the user using a text-based representation or using Google Maps.

Listing 3. Battery Analyser Reasoner Module.

```
1.  function getBatteryAnalyser(batteryLevel, batteryLevelCutoff){
2.     var singletonContextManager = ContextManagerSingleton.getInstance("batteryAnalyser");
3.     if (batteryLevel>=batteryLevelCutoff) {
4.        return true;
5.     }
6.     else if (batteryLevel<batteryLevelCutoff) {
7.        return false;
8.     }
9.  }
```

Listing 4 presents a small part of the sensor module that allows connecting to Facebook. In particular, after user credentials are validated and the permissions to access the profile data are confirmed, the Facebook API is invoked for retrieving these data.

Then via the singleton *ContextManager* instance the application is notified that the profile data have been retrieved and the web application needs to only parse and display these data. This specific module can be invoked from a context-aware web application as shown in lines 20-21 of Listing 2.

Listing 4. Facebook Connect Sensor Module.

```
1.     function getFBInfo(data) {
2.         var singletonContextManager = ContextManagerSingleton.getInstance("facebookInfo");
3.         FB.api('/me', function(response) {
4,            for (i=0;i<data.length;i++){
5.               var data_next = data[i];
6.               if (profile){
7.                  var profile = profile + "@%@" + response[data_next];
8.               }
9.               else{
10.                var profile = response[data_next];
11.              }
12.           }
13.          singletonContextManager.facebookInfo.facebookInfo = profile;
14.       });
15.    }
```

For the development of these modules we used HTML5 APIs to implement context-sensor plugins that enable access to device-specific hardware information, along with additional APIs that enable acquisition of network-specific information from RESTful and SOAP Web Services. Moreover, we have exploited APIs of social networks to define web-based context-sensor modules that allow retrieving the user profile and other data. Reasoners are also defined that allow acquiring low-level information from context-sensors and applying reasoning logic to generate high-level information.

6 Use Case Evaluation

As a use case for our evaluation we have chosen to implement a typical context-aware application: the restaurant finder application. Usually restaurant finder applications rely on the current user position and propose nearby restaurants allowing users to view the location of the restaurant, browse restaurant reviews and consult opening hours. Our version of *My Restaurant Out There* offers similar features in a context-aware fashion utilizing information collected through a number of context modules. Specifically, the application utilizes the following information:

- The location of the user that is used to return restaurant options in the vicinity of the user,
- The battery level of the device based on which it can either display the returned results in a tabular format (e.g., for battery level $< 30\%$) or on Google maps that offer a resource-demanding alternative (e.g., for battery level $>= 30\%$).

Existing sensor-module plugins from H5CM are used to offer the above functionality to the restaurant finder application. The *BatteryLevel* is used to understand the battery level and status of the device, whereas *Geolocation* gives the current user coordinates or the city the user is located in. Regarding the battery use, instead of using directly the respective sensor-module the *BatteryAnalyser* reasoner presented in the previous section is called, since it offers a finer representation of the context value returning true or false based on the battery level.

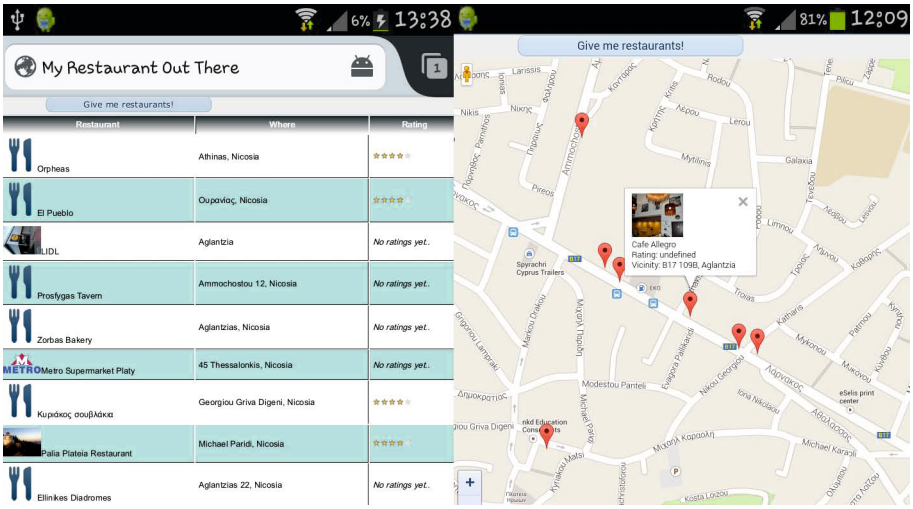

Fig. 2. Execution of My Restaurant out There

For evaluation purposes we have implemented two variants of the restaurant finder application: the first variant utilizes the proposed framework, whereas the second is based on one of the most popular framework for hybrid technologies PhoneGap. The latter variant was implemented for Android. Screenshots of the two applications under execution are shown in Fig. 2 with the left side indicating the list of restaurants as returned from the web applicaiton on FireFox (that supports the battery status feature of HTML5) for low battery level levels (i.e., 6%) and the right side depicting a map of restaurants as returned by the hybrid application for higher battery level (i.e., 81%).

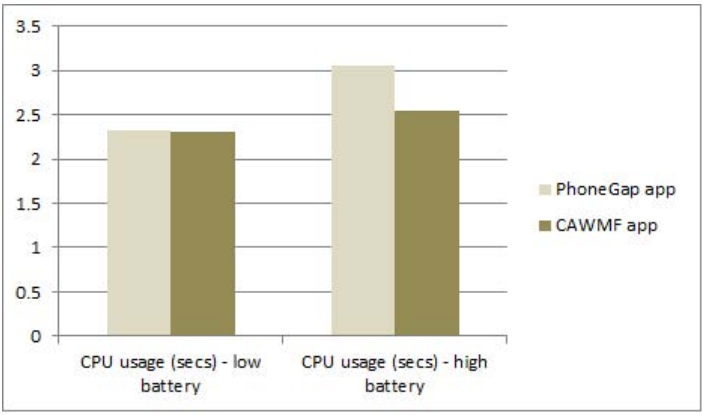

Fig. 3. CPU usage for the H5CM and PhoneGap variants of the restaurant finder application

Since one of the driving forces of the design of the proposed framework was the minimization of resources required from the mobile device we have measured the following parameters for the two variants: (i) CPU usage and (ii) network traffic usage including both outgoing and incoming traffic. For the evaluation a Samsung galaxy S3 device with Android version 4.3 was employed. The measurements were based on the "show CPU usage" feature of Android (available from Developer options) that depicts among other information the CPU usage in the last minute and the Traffic Monitor Android application[5] that provides traffic data information per application for different time intervals including measurements for the current date. We have measured the approximate values for each metric when Firefox or the restaurant finder hybrid application is running on the device, along with a number of Android applications or services (roughly 9). The results are displayed in Figs. 3 and 4 respectively. Note that for network traffic the diagram indicates the traffic measured only for the case when the application is running on a device with low battery level. We have omitted the high battery level case, since the traffic is in that case highly dependent on the interaction of the user with Google maps (zooming etc.) and no concrete conclusions could be drawn.

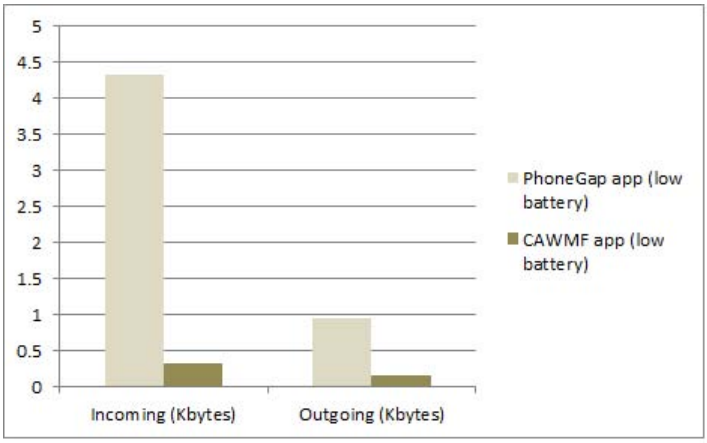

Fig. 4. Network traffic for the H5CM and PhoneGap variants of the restaurant finder application

Regarding CPU usage the results for the two variants are comparable. This is plausible, since CPU usage is highly dependent on the number of applications and services running on the device (in the last minute based on the measurements used). For both incoming and outgoing traffic there is a difference in the data volume between the two variants. This may be attributed mainly to the additional traffic that needs to be handled by a native application, since the PhoneGap application is in essence installed as a native application on the mobile device, and to the fact that PhoneGap-based applications make use of dedicated libraries of PhoneGap. Overall, our experiments indicate that the H5CM-based application is less resource-demanding. However, the application scale is also a parameter that needs to be considered for drawing general conclusions, since the application use case is rather simplistic.

[5] `https://play.google.com/store/apps/details?id=com.radioopt.tmplus`

From the above initial proof of concept and the description of the proposed procedure many benefits can be observed from the developer side. These are mainly found in the elements of modularity, extensibility, code reusability and cross-platform development provided by the H5CM as introduced earlier in the paper. An additional benefit can be found in an increase in code quality, since the application design based on the framework modules produces more compact and comprehensible code, an essential element also for the phase of software maintenance.

7 Conclusions

Cross-platform support in the development of context-aware applications is a desirable feature to simplify application development. In the field of web applications the most popular choice towards this direction is HTML5. Support for HTML5 features on mobile web browsers of popular platforms is necessary in order to take advantage of access to context data as facilitated by HTML5 constructs making thus the user experience adaptive and personalized.

In this paper we have performed a review on the support of HTML5 features on widely used mobile browsers, observing that many features are indeed supported. This tendency will strengthen as the HTML5 specification evolves to embrace the emerging needs of mobile users, while the modular architecture of the HTML5 Context Middleware allows supporting new HTML5 features as soon as they become available, by implementing additional modules. Moreover, we have introduced our context-aware framework, namely the H5CM, that alleviates developers workload and puts application development for different mobile devices in a common perspective by providing a selection of plugins that make the access to context data seamless. Context data are also monitored in order to reflect changes to the actual application. Also, the evaluation served as an initial proof of concept and showcased that HTML5 context-aware applications designed using the H5CM are less resource-demanding than mobile applications developed using hybrid technologies; especially in terms of network load.

As future work we intend to expand the library of plugins available on Google Code. This will enable developers to use additional context-aware features and thus reduce the time in developing context-aware applications. We will investigate also the integration of additional security and privacy guarantees in the use of H5CM, on top of social networks guarantees and the use of an HTTPS-enabled web server, since protection of user-relevant data is vital in context-awareness.

References

1. Weiser, M.: The Computer for the 21st Century. Scientific American (September 1991)
2. Rodden, K., Hutchinson, H., Fu, X.: Measuring the user experience on a large scale: user-centered metrics for web applications. In: Proc. SIGCHI Conference on Human Factors in Computing Systems, pp. 2395–2398 (2010), doi:10.1145/1753326.1753687
3. Rossi, G., Schwabe, D., Guimarães, R.: Designing personalized web applications. In: Proc. 10th Int'l Conf. World Wide Web, pp. 275–284 (2001)
4. Lee, Y., Ju, Y., Min, C., Yu, J., Song, J.: MobiCon: A Mobile Context-Monitoring Platform. Communications of the ACM 55(3), 54–65 (2012), doi:10.1145/2093548.2093567.

5. Berjon, R., Faulkner, S., Leithead, T., Navara, E.D., O'Connor, E., Pfeiffer, S., Hickson, I.: HTML5 - A vocabulary and associated APIs for HTML and XHTML. W3C Candidate Recommendation (February 2014), http://www.w3.org/TR/html5/
6. Han, B.-J., Rho, S., Jun, S., Hwang, E.: Music emotion classification and context-based music recommendation. Multimedia Tools and Applications 47(3), 433–460 (2010), doi:10.1007/s11042-009-0332-6.
7. Tarkoma, S., Lagerspetz, E.: Arching over the Mobile Computing Chasm: Platforms and Runtimes. IEEE Computer 44(4), 22–28 (2011)
8. Dey, A.K., Abowd, G.D.: Towards a Better Understanding of Context and Context Awareness, Workshop: What, Who, Where, When, and How of Context Awareness. In: ACM Conf. Human Factors in Computer Systems, The Hague, Netherlands (2000)
9. Paspallis, N., Achilleos, A., Kakousis, K., Papadopoulos, G.A.: Context-aware Media Player (CaMP): Developing context-aware applications with Separation of Concerns. In: IEEE Globecom 2010 Workshop on Ubiquitous Computing and Networks (UbiCoNet), Miami, Florida, USA, December 6, pp. 1741–1746 (2010)
10. Floch, J., Fr, C., Fricke, R., et al.: Playing MUSIC building context–aware and self–adaptive mobile applications. Journal Software: Practice and Experience (2012)
11. Ioannides, F., Kapitsaki, G.M., Paspallis, N.: Demo: Professor2Student – connecting supervisors and students. In: Daniel, F., Papadopoulos, G.A., Thiran, P. (eds.) MobiWIS 2013. LNCS, vol. 8093, pp. 288–291. Springer, Heidelberg (2013)
12. Wargo, J.M.: PhoneGap Essentials: Building Cross-platform Mobile Apps. Addison-Wesley Professional (2012)
13. Gai, D.: Hybrid VS Native Mobile Apps (2013), http://www.gajotres.net/hybrid-vs-native-apps/
14. Mikkonen, T., Taivalsaari, A.: Reports of the Web's Death Are Greatly Exaggerated. IEEE Computer 44(5), 30–36 (2011)
15. Mikkonen, T., Taivalsaari, A.: Apps vs. Open Web: The Battle of the Decade. In: Proc. 2nd Annual Wksp. Software Engineering for Mobile Application Development (2011)
16. Khronos Group, WebGL Specification, Editor' s Draft (2011), http://www.khronos.org/registry/webgl/specs/latest/
17. Melamed, T., Clayton, B.: A comparative evaluation of HTML5 as a pervasive media platform. In: Mobile Computing, Applications, and Services, pp. 307–325. Springer, Heidelberg (2010)