Journal of Web Engineering, Vol. 16, No.1&2 (2017) 075-102

AN APPROACH FOR BUILDING MOBILE WEB APPLICATIONS THROUGH WEB AUGMENTATION

GABRIELA A. BOSETTI SERGIO FIRMENICH

LIFIA, Facultad de Informática, UNLP, 50 y 120, La Plata, Argentina

CONICET, Argentina

{gabriela.bosetti, sergio.firmenich}@lifia.info.unlp.edu.ar

SILVIA E. GORDILLO

LIFIA, Facultad de Informática, UNLP, 50 y 120, La Plata, Argentina

CIC, Argentina

gordillo@lifia.info.unlp.edu.ar

GUSTAVO ROSSI

LIFIA, Facultad de Informática, UNLP, 50 y 120, La Plata, Argentina

CONICET, Argentina

gustavo@lifia.info.unlp.edu.ar

Received November 10, 2015
Revised July 25, 2016

Mobile Web Applications combine traditional navigation access enriched with location-based services, which results in a more complex development process since there are a myriad of issues to consider while integrating these kinds of behaviours. This complexity increases even more if the integration of another specific functionality is considered, as personalization or context-aware services. In this article we present a novel approach to facilitate the development of Web applications that enhance existing ones with mobile features through client-side Web Augmentation. Assuming the existence of a set of Web pages that could be associated to a physical object and some mechanism for location sensing, we allow developers to define mobile services or adaptations according to their own interests. We present a detailed comparative analysis of the features we provide against other similar approaches, in order to clearly highlight those aspects that distinguish our work from the existing ones. Finally, we show that this approach is feasible and effective by presenting two prototype applications for two possible scenarios and the results of our first experiment.

Keywords: Mobile Web Applications; Web Augmentation; Context Awareness; Mobile Hypermedia.
Communicated by: G-J Houben & M. Bielikova

1 Introduction

The increasing availability of mobile technologies has driven a steady evolution of Web applications, providing diverse kinds of location-based behaviours in a great variety of domains, like tourism, marketing or entertainment. Building them is generally a complex endeavour, since developers must consider a vast number of aspects, like variety of user devices, sensor technologies, etc. These topics have been studied by other authors, such as in [22], who consider the building process very complex. As a consequence, most Web applications are developed without considering the user's location for adapting the provided services or content. On this basis, useful services are generally neglected; among them, we can name mobile guides [12, 34] or some Mobile Hypermedia [4] features.

Some authors [13, 38, 6] have proposed to enhance Web applications with Context Awareness (CA) for providing adapted services or content, but the majority of these contexts are predefined and they do not cover a wide range of mobility aspects. Some organizations, as museums or tourism offices, need to provide some kind of location-based information and, for that end, they use their own legacy Web sites combined with barcodes. Under this scenario, end users receive the corresponding encoded data when reading the barcodes by using a mobile device. QRpedia[1] codes use a slight variation of this strategy to provide location-based information, by associating Wikipedia Web pages with QR codes; such relation can be used for making the URL to play not only the role of digital resource identifier, but also of physical objects in the real world. These solutions solve the initial problem of providing some kind of information at a very low cost. However, the resultant combination of such mechanisms and the accessed Web pages do not provide the mobile user experience with a vast range of –external, third party– functionalities.

As concrete example, the Natural Sciences Museum of La Plata provides location-based information through QRpedia. This museum has some volatile thematic tours according to specific exhibitions. For economic reasons, these tours are seldom shown in the museum's official Web site, for example, the Darwin digital tour[2]. These tours are not simple to implement as physical tours, because they need to provide additional assistance while mobile users are walking the Museum. The complexity is increased because it is not possible to directly manipulate the sources of the Wikipedia pages in order to meet the demand of a concrete mobile functionality for such exhibition. An alternative could be having QR codes referring to the Museum's Web pages instead of using QRpedia. Providing physical tours, in this case, implies creating these services as part of the Web site. To this end, it is required to have development knowledge and skills for incorporating this functionality. The situation gets worse if new tours arise very often. It implies a constant cost that the Museum might not be able to afford. Providing a solution for creating flexible applications and delegating these tasks to the crowd may be an interesting improvement.

In this paper we present a detailed description of our approach for augmenting existing Web applications with mobile features. We describe a flexible framework for facilitating developers the application creation through client-side scripting, and a mobile browser plug-in for executing such applications' scripts. Resultant applications can be demanded, shared and used by other users, as typically happens in Crowdsourcing [28] communities, like Userscripts[3].

---

[1] QRpedia: http://qrpedia.org/
[2] Darwin tour: http://www.fcnym.unlp.edu.ar/museo/educativa/darwin/darwinenmuseo.html
[3] Userscripts : http://userscripts-mirror.org/

To ensure our solution supported scalability and reuse, we have strived to make our framework architecture flexible by implementing it under the foundations of modularity and variability, as studied in [16]. We composed it with several separate modules –e.g. for generic applications, augmenters, positioning– that act independently, but also together, for achieving different kind of applications development and execution goals.

Variability in mobile applications is also important during the design stage, because they are subject of continuous changes brought by the incorporation of new technologies, contexts of use, environments, etc. In addition, and in concordance with our approach, this is even more important if we consider a) the wide spectrum of functionality the developers could want to create, according to the application domain, and b) the fact they are reusing existing –and probably third party– Web content, which also carries its own changes and challenges. In this work, we identified the main variability features of the solution, so we highlight two of its associated open variation points related to applications and augmenters. We also provide implementation proofs of some common variants of such points.

As seen in [23] and [36], people are used to interact with smart or physical objects [37], so we believe there is no operational-feasibility inconvenient in carrying out our implemented interaction strategy. The only requirements for building applications under our approach are: a target Web page for augmentation, some extra content to be included in an augmentation layer, and some mechanism of location sensing, such as GPS or barcode decoding technologies. A supporting tool of our software framework is in charge of linking locations to application objects in specific Web pages.

We use a client-side approach to create the augmentations. Thus, developers can create their personalized scripts to enrich Web applications with mobile features. These scripts run in mobile devices and they may provide not only the static information (e.g. the Wikipedia page) but also specialized behaviour, for example, some mobile hypermedia functionality such as walking links. When users are sensed in a specific location they receive the augmented Web page.

Summarizing, the main contributions presented in this paper are the following:

- A flexible approach to enhance Web applications with mobile features, through client-side scripting and the extension of the browser's capabilities, which contemplates variant features support for allowing developers to extend and reuse the building artefacts in different contexts and domains as they need.
- A software solution supporting two kinds of user roles: developers, who extend the framework artefacts by creating scripts in specific domains, and end users, the ones who use such scripts through a mobile browser plug-in.

We have performed a comparative analysis covering existing works to date, which allowed us not only to highlight features of our approach against others but also to determine the right variables to compare in our hypothesis. We present a body of evidence on the feasibility and power of our approach supported by the implementation of two different-domain cases studies and by the results of an evaluation with real users.

The remainder of this paper is structured as follows. Section 2 presents a background of Web Augmentation and Mobile Applications, and Section 3 describes related works to date. In Section 4,

we explain our approach in a nutshell and in Section 5 we present more technical details about it. In section 6, we analyse existing works in detail and present a comparison of the contemplated features. Then, an evaluation is presented in Section 7, in order to validate our approach. Finally, we provide some final conclusions and mention further works.

2 Background

2.1. Web Augmentation

Web Augmentation [10] is a set of techniques, usually applied by end users of Web applications, for adapting existing Web pages according to the user's requirements. Two main approaches have emerged for this purpose: proxy-based transcoding systems [2, 11] and DOM manipulation through client-side scripting. Their main goal is to allow users to modify what they perceive of the applications by manipulating the original documents obtained through HTTP requests.

Transcoding Systems are usually developed as a proxy system, responsible of resolving the Web Page adaptation at a dedicated server or proxy, and then of making it available to the clients. The other approach is carried out through client-side scripting; this technique is applied generally by DOM (*Document Object Model*) manipulation. The DOM acts as an API for accessing to structured documents based on a markup language, and it allows altering documents by removing, adding, or changing its components. When the client sends a request, it still receives the original Web site but, once the content is loaded, it is augmented at client-side with new features. The main idea of this approach is to manipulate the DOM once it is loaded and parsed by the Web browser. In this case, there are no browser's external systems intermediating and the adaptation itself is performed at client-side. Different tools and approaches have emerged, especially for desktop Web browsers. However, with the evolution of mobile devices and their sophisticated Web browsers, this strategy is becoming also a new trend, and mobile browser's extensions are emerging[4].

2.2. Mobile Applications

The definition of what is considered a *Mobile Application* has been discussed over the years. Basically, *Mobile Applications* are information systems designed for being executed on mobile devices platforms. Some authors argue that mobile computing also involves other aspects, as ensuring continuous access to remote resources while the user is on the move [15, 35]. But it is sufficient to access any mobile application market for seeing a myriad of so-called mobile applications that does not contemplate the mobility feature. In this work we do call mobile application to the ones considering this feature, because we believe the added value of *Mobile Applications* over others information systems comes precisely from the fact they can move and make use of the information retrieved from the context of use by the device sensors, and use it to tailor its functionality or content.

Mobile applications can be classified according to what values of the context they use, and what do they use for. For example, we can mention Location Based and Location Aware, Context Aware, Mobile Augmented Reality, Tour Guides, and Mobile Hypermedia applications. One fact is certain:

[4] Firefox Mobile extensions: https://addons.mozilla.org/en-US/android/

Mobile Computing evolves exponentially and today there is a big number of features for taking into account and allowing new domains to arise, as in [32].

In this work, we are interested in supporting a large variety of mobile applications, but we just focalize in *Context Aware* and *Mobile Hypermedia* ones for performing a proof of concepts and validate extensibility and reuse. The first of the mentioned fields is well known and, in short, it involves applications that can adapt their behaviour according to different situations. Most of the common context types sensed by this kind of applications are related to the environment and the device, such as location, orientation, noise, light, time, energy consumption or network conditions. But the definition of a context can involve much more types than those [12], and it can be extended with, for instance, user or social related aspects, as the user activity or the emotional, relationship or interaction states.

Mobile Hypermedia applications [4] extend the concept of navigation used in the traditional Hypermedia, to the real world. These applications are considered a particular kind of *Mobile Application*, not only because it includes mobility features but also the navigation aspects. In these systems the user accesses the information in two different ways: digitally and physically. Digital access, as in traditional hypermedia applications, is enriched with the usual digital links. However, physical access requires users standing in a specific location or point of interest (PoI), thus some sensing mechanism is required to provide information related to their location. Then, users accesses to this information by navigating "walking" links (or real-world links as mentioned in [29]) and also receives the hypermedia information associated with the location (also called "digital-physical link" in [20]). Navigating a walking link is more complex than a digital link, because it involves the physical movement of the user, as explained in [21]. When users select one of these links, they express the intention to walk to that target. In order to assist the user to such target, the application should present instructions for arriving to the related location. For instance, the application could display a map with a path to the selected object. When users will arrive to the marked location, they will be sensed and notified. In Figure 1 an example of Mobile Hypermedia information access is shown.

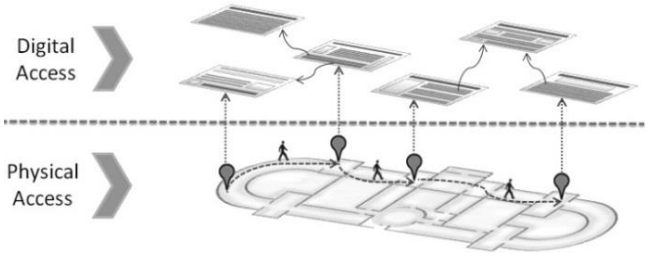

Figure 1. Example of Mobile Hypermedia.

We have experience in different aspects related to the development of *Mobile Hypermedia* applications [16, 17]. In particular, we have presented a model-based approach for creating *Mobile Hypermedia* applications. In these models, we have used the concept of separation of concerns for supporting flexibility and scalability. We separated pure hypermedia from mobility aspects; this separation allowed us to think how to integrate an existing Web application with new features, for example, supporting positioning information and walking links. If we look again at the mentioned

scenario concerning the *Darwin tour*, the pure hypermedia is the Wikipedia pages and the physical access is implemented through QRpedia codes. Note that a physical tour could be created by adding walking links between the pieces of the museum.

3 Related Work

Mobile Guide Applications are mobile applications which provide assistance and enrich the users' experiences when they are moving in the real world. These kinds of applications can cover a wide range of domains [12], for example guides in museums exhibitions or urban areas. These applications have evolved constantly to improve services or functionalities offered according to the users' needs. This can be seen in earliest prototypes as Cyberguide [1], which provided simple context-aware services, to the most modern mobile guides [9, 12, 34] that offer complex ones. Evolution increases considerably the complexity of the applications, depending on the requirements of the users. This issue, evolution of software [25], is presented in any product of software and Mobile Guide Applications are not the exception.

Specifically, the authors in [1] designed a family of mobile context-aware tour guide prototypes. They made and tested some prototypes, by supporting different user profiles, recording user activity, delivering suggestions based on context data, and trying available hardware, both for sensing the user's context and for displaying data. A complete study on mobile guidance applications and a proposed taxonomy are presented in [12]. A more recent work regarding tour guides is presented in [34]; the authors developed a mobile tourist guide for planning trips. The difference with traditional guides is that they offer users a full control of their tours; they first need to navigate, select and rate a set of PoIs, so routing algorithms can favour the subsequent use of certain PoIs in the route construction. Then the user is offered with alternative routes, considering a starting time and PoI, and an ending time and final PoI. The recommendation system will try to consider all the selected PoIs for building the routes, but it will finally include the ones who fit better in the scheduled time. At the end of such process and while users are performing the tour, they are allowed to modify the tour by adding/removing PoIs, reordering them, changing durations or the route itself.

Covering all the users' needs is a difficult task for the developer. To solve this, some works use the user's context to provide services or adaptations, especially in mobile applications [30]. This is one possible option to reach flexibility, but it is limited to the contexts that were defined when the application was created. There are some hybrid approaches that combine tour guides with recommendation systems based on the user's context, as presented in [14]. It is an Adaptive Tourist Recommendation System which possesses the capability of changing its behaviour based on the user's current situation, and providing a personalized assistance. It is reached by collecting the user's data, for example by using the Facebook or email client APIs, taking the traveller's stage of travel into account and the context values, collecting preferences about the offered recommendations and also by allowing advanced users to modify the recommendation algorithm parameters. The system monitors these characteristics and offers the proper adapted content.

A more focused Context Aware approach is described in [13]; it proposes an alternative for allowing Web applications to access the mobile phone characteristics and retrieving the perceived data from the context. The authors specify a set of extensible context aware XML tags that Web applications can use in the presentation layer to request information of the user's context. In addition,

they also created a Context Aware browser that interprets those tags, asks the user's permissions and executes their related tasks.

In [38], the authors present a generic approach for improving existing Websites with context-aware features on-the-fly. They implemented an application in order to prove their ideas, COIN, by using an existing Android-based Context Aware framework. This application enriches third-party Websites by extracting semantic information from these sites, then looking for matches with the user's context and finally enriching each page through DOM manipulation. A most recent approach was presented in [6]; they implemented a browser-based context framework that empowers Web applications with context sensing capabilities and allow interactions with smart objects. As example, they present an implementation for extending the Flickr Web site with the capability of sharing the current visited media content with the devices in the surroundings.

In the last few years, a lot of End User Development [27] tools have been developed for empowering the user to create different kinds of mobile applications. CASTOR [33] is another approach for creating, managing and playing in situ context aware stories. The authors present not only an authoring tool for applications creation in the same place the story should take place, but also a Web-based editor for allowing later changes and a context-dependant player. A Mashup construction system, which allows creating context-aware services based on rules, is described in [24]; the authors implemented MyService, an Android based application which allows clients to define context-based recommendation rules through a graphical interface, based on the specifications of a service directory. Then, a composition system is in charge of gathering the context data, selecting proper services and generating the Mashup code for the Web application.

CAMUS [8] is another Mashup development approach; the authors present a framework for creating Context Aware Mobile Web applications by combining sources through Mashups. Their resultant applications collect and integrate Web resources according to the context of use; this way the application's content is not pre-packaged but dynamic. These resources should be registered by an administrator in the platform's Universal CDT, so from thereon designers can include them in a tailored CDT for a specific application and construct the Mashup.

There is also a recent work about an authoring tool that combines Mashups with Augmented Reality [40]; it allow end users to create Augmented Reality (AR) applications for both indoor and outdoor environments, through the use of a native mobile client application. First, users should select a physical area as the AR zone, and take several pictures or record a video of the place, so images can be processed at server side, by extracting some 2D features, and the scenes are identified. Then, users have to link Web resources to the created zone and assign them a position, a scale, a rotation and an update rate for requesting the content from the server.

In the field of Web Augmentation, a multimodal augmentation platform is presented in [19]. It augments existing Web applications according to the context of use. For instance, if users change from a noisy to a silenced context, they are offered with a vocal interaction modality through augmentation. The authors implemented a context manager that manages "event-condition-action" rules and notifies the respective context changes. They resolve the adaptation content at server-side through a model based language that allow descriptions generation in different abstraction levels. Once the content is

processed, the adaptation is performed through DOM manipulation, by using a navigation proxy that allows external scripts and content injection.

The authors in [31] present an approach for End User Development of Web adaptations under Crowdsourcing principles. They mention the idea of providing the users with recommendations retrieved from a recommendation system, at server side, based on the user's context characteristics. These characteristics are previously processed by another component at server side, and a third one is in charge of storing the user's created artefacts. They also mention a client side adaptation toolkit that communicates with the components at server-side as services. A Crowdsourcing approach like ours is possible because the participation of users is not new in the Web, as is the case of the GreaseMonkey community. These ideas can be extended as explained in [23]. In such work, an analysis of opportunities and challenges is presented in relation with the Crowdsourcing and heterogeneous network environment fields. It proposes a conceptual architecture of a CA mobile Crowdsourcing platform, which allows crowdsourcers to define and offer context sensitive and personalized tasks –like writing a review or describing their physical surroundings–. Once defined, these tasks should be offered to mobile workers through a pull-service, and according to their interests, skills, availability, network load, etc.

Regarding personalization support, [39] presents an architecture, implementation and evaluation of PersonisJ, a framework for client-side personalization on Android mobile phones. They also cover privacy data protection, by storing the user data model at client side. Even so, the architecture presented has also server-side components. Client applications can access the user data model by declaring some security permissions, which the user must grant access, in order to be allowed to execute "read/write/tell" operations. The user model is composed as a hierarchical structure of contexts which are updated by subscribing the corresponding context listener, and being notified of the changes in the context.

All the approaches mentioned above allow users to create new applications or to add new features to the existing ones, but with some limitations. Generally, the created applications are difficult to increase or evolve. For that reason, we focus on providing a flexible approach for allowing developers to create their adaptations by enhancing existing Web applications with mobile features. There are some interesting projects in the presented areas; for example, a Context Aware Mobile Crowdsourcing approach in which the tasks are assigned according to the user's context, as presented in [36]. This confirms that users are willing to participate in the creation of applications or in data harvesting processes; therefore an approach like ours is not only feasible but also promising.

4 Our Approach in a Nutshell

MoWA (*Mobile Web Augmentation*) is an approach for enhancing Web applications with mobile features. Existing Web pages are enriched with an augmentation layer and associated to a specific location. As a result, different kinds of mobile Web experiences can be created, for example, with mobile hypermedia features. A brief outline of our approach is depicted in Figure 2. The specification about how to augment the exiting Web pages is deployed as a MoWA application, which is basically a JavaScript script. Developers just need to extend a MoWA application class, define an augmentation layer for each Web page involved in the application, and specify some geographical locations where

the users should receive each augmented Web page. Once a MoWA application is finished and shared, any user can install and run it through the MoWA Plug-in on the user's mobile Web browser.

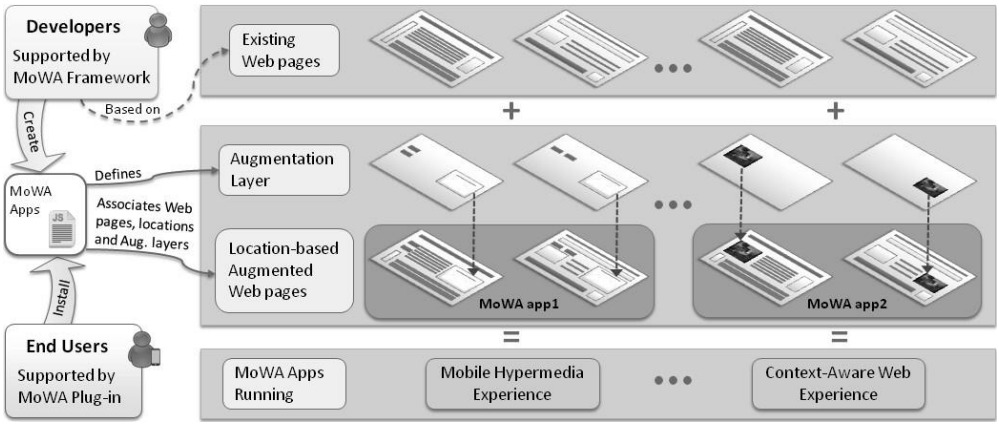

Figure 2. The MoWA approach.

Suppose a developer wants to create a specific physical tour for the museum of La Plata; to that end, defines a MoWA application called *DarwinTour*, by extending our framework. In this case, for each object involved in the tour there will be an associated Wikipedia Web page and a physical location, provided through QRpedia. It is also created an augmentation layer for each tour's object. These layers specify the information for enhancing each page, for example, some comments *Darwin* made about the concrete objects and some walking links to connect all the objects of the tour. When the application is deployed, any user could install and use it through our plug-in. As result, every time a user scans a QRpedia code of any object of the physical tour, the related Wikipedia page is opened and augmented with the respective *Darwin* comments and walking links.

The MoWA architecture can be represented through two software pieces: a framework and a plug-in, as shown in Figure 3. To show how developers can extend some the framework's extension points, a concrete MoWA application was created, which represents the *DarwinTour* application described above. The framework provides the necessary behaviour to match locations with Web pages, load the pages and execute adaptations. Each concrete application is responsible of defining the relevant Web pages, its location and augmentations. Our framework is used not only to create MoWA applications but also to collaborate with our plug-in artefacts to enhance Web pages at runtime. When our plug-in detects a change in the user's location, it notifies the change to every running application, and these ones use the data and the general behaviour of the framework for enhancing the accessed Web page.

Our framework presents some open variability points, which represent the supported variant features and allow developers to reuse MoWA artefacts functionality for different contexts; it is possible to extend a large set of artefacts: applications, users, points of interest, augmenters, space representations, context types (e.g. location, noise level) and context types' sensors (e.g. GPS sensor, Noise sensor). Some extensions require a little more effort, not only for the complexities in their implementation, but because they depend on other classes, making them necessary to extend as well. For example, there is no point in extending a location if there is no *LocationSensor* to be observed or a related update method in the subject, therefore, nor application capable of using it. In order to show

how developers can extend one of these extension points, a concrete MoWA application is created which represents the *DarwinTour* application described above.

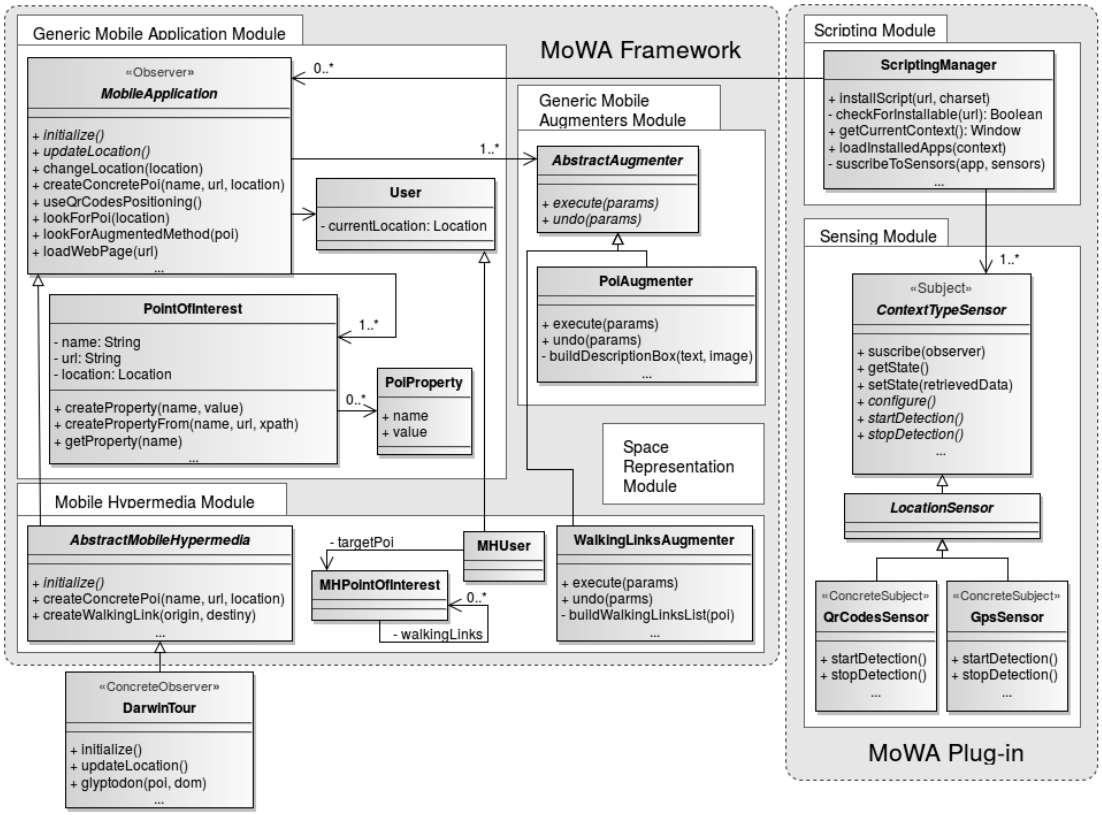

Figure 3. MoWA Architecture: Framework and Plug-in's main modules.

The modules presented in Figure 3 are depicted below. The *Generic Mobile Application Module* implements generic behaviours common to all mobile applications which consider the user's location for offering information. The *MobileApplication* class has a *Façade* role [18] and represents a mobile application which will enhance existing Web page(s). This class has associated a user, a set of PoIs and also the ability to augment the Web pages based on the user's location. The *User* class represents a person who is using the application. This class abstracts the user's current location and navigation history. The *PointOfInterest* class represents any PoI relevant to the application. For each PoI, it is required to define a name, an associated URL and a location (we define location as an interface according to [26]). The name could be used, for example, for matching the name of the augmentation method at runtime; the URL corresponds to the page which will be loaded and augmented when the user is at that PoI; the location defines where the augmentation should be triggered. The framework provides a mechanism to add other properties[5] (e.g. a description or a picture) to the PoI; to do that, it uses the class *PoiProperty*; a generic class to represent the name of the property and its value. Generally, these additional properties are used to enhance the Web page.

[5] These properties can be defined using a value from an external resource.

It is required for each concrete subclass of *MobileApplication* to define the `initialize` method with the starting values of the application. This involves the instantiation of each PoI, the definition of a space representation, defining where each PoI is located in the physical space, and defining the sensing mechanisms relevant for the application. To define these mechanisms, the inherited methods could be used, for example, through `useQRCodesPositioning`. It is also required for each concrete subclass to define an augmentation method with the same name as the PoI. These methods may perform DOM manipulations by defining the augmenters that will be executed in runtime.

The common inherited behaviour for any subclass of *MobileApplication* is presented below. When a user changes the current location, the application searches the corresponding PoI, using the `lookForPoI` method. If there is a PoI associated with this location, its Web page is loaded through the `loadWebPage` method, and it is enhanced using the augmentation method which matches with the PoI's name. The `lookForAugmentationMethod` is used to find the corresponding method which then it is executed to adapt the page.

The *Generic Mobile Augmenters Module* provides specific augmenters classes which help developers to enhance each PoI's related Web page. Augmenters provide the mechanisms to adapt and augment existing Web pages with new content. The generic behaviour of augmenters is defined in the abstract class *AbstractAugmenter*. This class defines two abstract methods that all concrete classes must implement: `execute` and `undo`, and both methods receive the required data in the parameter `params` as a JSON object. The first method is responsible for enhancing the Web page, by receiving the necessary data for performing the augmentation as a parameter. The second is responsible for reversing the changes made by the first. MoWA provides a set of default augmenters, localizable at the `mowa` namespace, which can be instantiated and used directly in a script, but also can be extended to adapt them to new requirements. The *PoiAugmenter*, as shown in Figure 3, allows augmenting a Web page with information about a particular PoI, by showing its image and description in a new HTML div. To use this augmenter, it is required to pass the `image` and `description` properties as parameter, so they should be defined also in the PoI for providing them when sensed.

The specific mobile hypermedia features are defined in the module corresponding with the same name. These features are represented as extensions of those defined in the *Generic Mobile Application Module*, as shown in Figure 3. For example, *MHPointOfInterest* extends the *PointOfInterest* by adding the concept of walking links. Related to these kinds of applications, specific augmenters have been created. As the name suggests, the *WalkingLinksAugmenter* provides the mechanism to enhance Web pages with walking links; the ones created using the walking links collection defined in the *MHPointOfInterest*. When users click one of these links, they receive, in this particular implementation, a 2D map with a path traced from the user's current location, to the selected target. The behaviour for detecting the user arrivals to the target is included in the augmenter, but such information is retrieved by observing the plug-in's location management module.

As we mentioned before, the *DarwinTour* is a concrete MoWA application. So, it defines the required `initialize` method and the augmentation method for each PoI, for example, the `macrauchenia` or `glyptodont` methods. The functionality related to the representation of the physical space is defined in the *Space Representation Module*. This includes varied features like showing a map or finding a specific path to a target PoI. The developer may specify which space

representation is used for each MoWA application; for example a 2D map or an external service, as Google Maps[6].

Next, we describe the modules related to the MoWA plug-in. The *Sensing Module* comprises a set of classes which implement different mechanisms for detecting and notifying changes in context types, called context values [3]. Those context types comprise not only the ones related to the device's sensors but also the ones related to the user, the full environment, the system, the social issues and the service (e.g. availability, task sequence, constraints), as explained in [12]. We provide two sensing mechanisms for the key context type of any pure mobile application: the positioning. One is based in GPS positioning and the other is based in QR codes sensing. We also support a time sensor and a concrete user-category sensor, but the framework is not limited to such context types; sensors could be extended by developers for supporting new ones and also new sensing mechanisms, as we will show later in Section 5.2.

We provide a *ContextTypeSensor* superclass playing the Subject role in the Observer Pattern [18]. It implements the required mechanisms to start and stop the sensing of a concrete feature, and when a significant change is detected, a *ContextTypeSensor* calls its `notifyChange` method, which is in charge of notifying every listener subscribed to the subject; therefore, applications can subscribe to sensors and be notified about any context sensing change. The relation between applications and sensors is set by the *ScriptingManager*, when applications are loaded, as specified in its specification (defined in the `initialize` method of each application). *ContextTypeSensor* subclasses should be implemented as singletons; this way, many applications may listen to the same sensor instance changes, allowing lazy instantiation and making it suitable for limited resources platforms. Moreover, there is no point in having more than one sensor by context type; for instance, the user location will be always the same, regardless of what application uses this data. When any *ContextTypeSensor*'s value changes, it is notified to every attached observer, through their respective update method. Consider a scenario where the *DarwinTour* application is using a *QrCodesSensor*. When the status of the sensor is changed, because an image containing a QR code was successfully processed, it throws its `onPositionSensed` method, which stores the sensed value and calls the notification method which, in turn, calls the `updateLocation` of every observer, in this case, the *DarwinTour* application. Each application can use the inherited behaviour of the *MobileApplication* to check, for example, if this new location matches with some PoI's location. If so, its Web page is loaded in the browser and it is enhanced with the corresponding augmentation method.

In Figure 3, we show an abstract class called *LocationSensor* and –up to now– two concrete subclasses: one based on geo-positioning and another based on QR codes. The first one uses the geo-location API of *Firefox for Android* to detect positional changes. The second one implements a mechanism for decoding QR codes from pictures captured from a live video streaming.

Finally, for the sake of simplicity, we only show the aforementioned *ScriptingManager* class in the *Scripting Module*. It is also responsible for creating, editing, suppressing and executing scripts, subscribing applications to the specified sensors, and managing the high-privileged objects and operations at browser-side; for example, getting the native window for displaying messages to the user, or loading extra files in the current Web page's context.

---

[6] Google Maps API: https://developers.google.com/maps/documentation/javascript

5 Case studies

In this section we describe how to create new applications using our framework, especially the *DarwinTour* application depicted in Section 4. Then we present an example of Context Aware application. Finally we show how programmers can create new augmenters in order to contemplate other kind of behaviour for enhancing the Mobile Web.

5.1. A MoWA Mobile-Hypermedia application

Suppose a developer uses the *Darwin* digital tour provided by the museum to enhance the Wikipedia pages with a *Darwin* description and walking links, by creating a physical tour.

Figure 4 shows a simplified script which is used to describe the guidelines that developers need to follow in order to create a MoWA application. First, the metadata of the script is defined, like the name of the application and its namespace. Developers need to extend one of the open variability points; in this case *DarwinTour* extends from *AbstractMobileHypermedia*. The required `initialize` method is defined in lines 5 to 9. A 2D map is used for representing the physical space, and a sensing mechanism is defined by calling the inherited `useQRCodePositioning` method.

In this script, the PoIs are defined in the method `createPoIs`. For instance, the PoI with the name "*Macrauchenia*" is defined in lines 11 to 18. Its associated URL is the *Macrauchenia*'s Wikipedia page and its location is represented through a *QrLocation*. Two additional properties, `image` and `description`, are defined in relation to *Macrauchenia*'s page of the *Darwin*'s digital tour. When the PoIs are defined, they could be connected by creating some walking links between them, e.g. a walking link from *Macrauchenia* to *Glyptodon*. Finally, the augmentation methods are defined. Lines 24 to 28 show the `macrauchenia` method executing two MoWA augmenters.

```
1  // @name        DarwinTour
2  // @namespace   dev01
3  // ...
4  function DarwinTour() {...};
5  DarwinTour.prototype = new AbstractMobileHypermedia();
6  DarwinTour.prototype.initialize = function(){
7      this.createPois();
8      this.create2DMapRepresentation();
9      this.useQrCodePositioning();
10 }
11 DarwinTour.prototype.createPois = function(){
12     var name = "Macrauchenia";
13     var url = "https://es.m.wikipedia.org/wiki/Macrauchenia";
14     var location = new QRLocation(url);
15     var macrauchenia = this.createConcretePoi(name, url, location);
16     macrauchenia.createProperty("image", "http://www.fcnym.../macrauquenia.jpg");
17     macrauquenia.createPropertyFrom("description", "http://www.fcnym.../pieza1.html",
18         "//*[@id='contenido']");
19     ...
20     var glyptodon = this.createConcretePoi(..., ..., ...);
21     ...
22     this.createWalkingLink(macrauchenia, glyptodon);
23     ...
24 }
25 DarwinTour.prototype.commonAugmentation = function(poi, dom, xpath){
26     var params = {"poi": poi, "dom": dom, "elemPosition":xpath};
27     this.augmenters["mowa:WalkingLinksAugmenter"].execute(params);
28     this.augmenters["mowa:PoiAugmenter"].execute(params);
29 }
30 DarwinTour.prototype.macrauchenia = function(poi, dom){
31     this.commonAugmentation(poi, dom, "//*[@id='section_0']");
32 }
33 DarwinTour.prototype.glyptodon = function(poi, dom){...}
34 ...
35 var myApplication = new DarwinTour();
```

Figure 4. Darwin Tour script using MoWA.

After an end user installs the script through our plug-in, the adaptation mechanisms can be triggered when the location of the user changes, e.g. by reading QR codes. When the user captures an image with the QR-codes reader integrated in our plug-in, the result is sent to the *QrCodesSensor*. The propagation of the new location continues as mentioned in Section 4. Figure 5 shows the interactions of the objects involved in the augmentation execution, once the Macrauchenia's code is read, decoded and sent to the application.

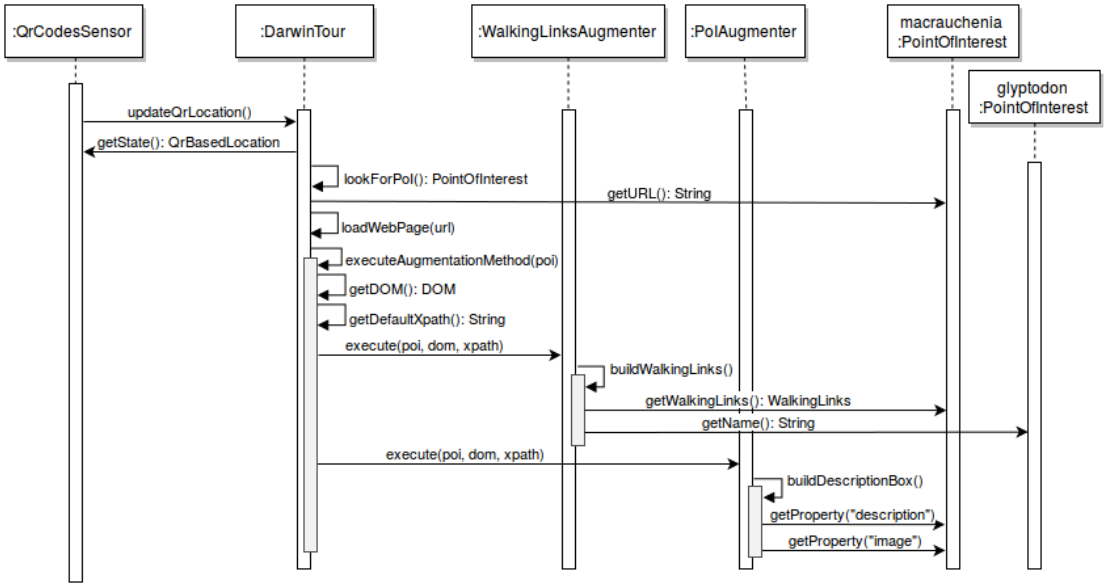

Figure 5. Darwin Tour script running.

5.2. A MoWA Context-Aware application

In order to validate flexibility and reuse of our framework's artefacts, we also extended it with another kind of application for supporting Context Awareness. In the same way as with Mobile Hypermedia implementation, we created an *AbstractContextAware* class. Considering this kind of application, imagine now a second scenario where a developer extends this abstract class with *AdaptiveDarwin* in order to augment the same Darwin pieces, but this time focusing not on physical navigation assistance but in the following context types: location, user-category and time.

The first two types are user-related. Regarding location, augmentations should be displayed according to the current visited PoI, as in the previous scenario. The second context type, user-category, performs some adaptations to the augmentations, according to the following pre-existing user profiles: general public, researcher and kinder. Even though all profiles offer a common augmentation, based on the Museum's official Web site information, this separation will enhance the user experience by providing specialized versions of the content and even some extra resources.

A user with general public profile will see the augmentation generated just by a *PoiAugmenter*'s execution; the paleontological-specialized public will be provided with the same augmentation plus a new section for related academic articles about such pieces; and Kinder visitors just with the application name as a title and some multimedia content, avoiding textual descriptions so they can see how the animals look(ed) like (if extinct or not) in their natural habitat.

The third characteristic is an environmental one; the time. This includes making the application aware of the Museum's closing time, the time the user started the tour, and an estimated time for completing the whole tour. The goal of these variables is to determine the convenience of showing -or not- multimedia content to the user. In order to make a simple demonstration, a wider spectrum of complementary timing variables was avoided.

In an optimal scenario, where user has enough time for consuming extra and external multimedia material, a general user will be offered with the b) view in Figure 6; a researcher visitor will see the c) augmentation alternative; and a kinder will be presented with the d) option. Option a) is the original Web page, without augmentations.

Figure 6: Augmentation alternatives of the AdaptiveDarwin

In order to implement such application, developers need to extend the *AbstractContextAware* class and implement the subscription to the subjects they want to listen, as shown in Figure 7. As mentioned in Section 4, sensors can be as generic as listeners of the device's sensors values –as *CurrentTimeSensor*–, but also as specific as listeners of a particular kind of user profile used for a concrete application kind –as *ADUserProfileSensor*–. When any ContextTypeSensor's values change, every attached observer is notified through their respective update method.

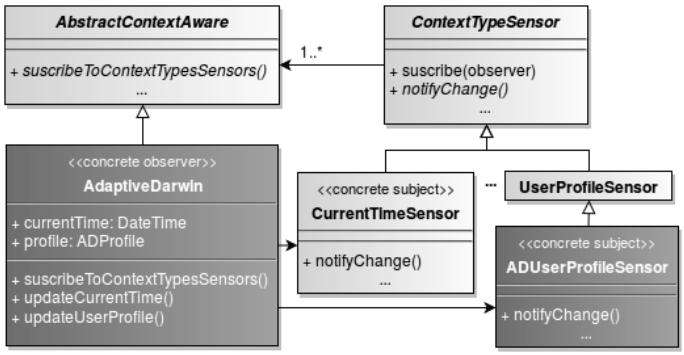

Figure 7. Extending the Context Aware application superclass

In the same way, *AdaptiveDarwin* can notify their respective augmenters of those changes in the context. The reason why the responsibility of observing the sensors' changes is not delegated to

augmenters is because in some cases the application needs to perform other tasks before. For instance, an application could need to reload a Web page before notifying changes, so augmenters have to be executed again; or it could have to decide whether the context change is relevant for a concrete scenario, avoiding each inserted augmenter to take this decision by its own.

Augmenters, in this case, are context aware. They implement an update method, and therefore subject applications can notify them the context changes; they are generally composed by primitive augmenters, which are executed and undone when required, so they know how to adapt the content to each case.

5.3. Extending MoWA augmenters

MoWA augmenters can also be extended, allowing developers to add new augmentations according to their needs. For doing that, developers need to extend *AbstractAugmenter* or any of its subclasses. For instance, we mentioned a concrete augmenter that offers specialized content for researchers in the field of palaeontology, by adding related academic articles about the current visited PoI. As shown in Figure 8, the *RelatedDocsAugmenter* is defined with the correspondent behaviour for inserting a new navigational menu in a certain position of the DOM, containing a title and an ordered list of links to Google Scholar paleontological documents. They are automatically retrieved through a query with a specific keyword.

```
// @name        RelatedDocsAugmenter
// @namespace   dev01
// ...
function RelatedDocsAugmenter(){};
RelatedDocsAugmenter.prototype = new AbstractAugmenter();
RelatedDocsAugmenter.prototype.execute= function(params){
    var container = this.createContainer(params. title, params.dom, params.elemPosition);
    var links = this.searchInGoogleScholar(params.keywords);
    this.buildLinks(container, links);
};
RelatedDocsAugmenter.prototype.createContainer = function(title, dom, elemPosition){...};
RelatedDocsAugmenter.prototype.buildLinks = function(container, links){...};
```

Fig. 8. Extending the framework with a new augmenter.

This new augmenter can be installed by opening its file in the Web Browser. If the MoWA plug-in is installed, the user will be suggested with its installation. In Figure 9 we show how the developer can use this augmenter from any script.

```
var rdaParams = { "title": "Some related articles:", "dom": dom,
                  "elemPosition":"//*[@id='section_0']",
                  "keywords": [poi.getProperty("name")] };
this.augmenters["dev01:RelatedDocsAugmenter"].execute(rdaParams);
...
```

Figure 9. Using the new augmenter created by the developer.

Our plug-in was initially developed to be run on *Firefox for Android*, even though its design is portable to other browsers that support a similar kind of extensions.

5.4. Extending MoWA sensors

We have used some already supported sensors by MoWA in previous sub-sections; they report changes of location, user-category and time. But these three context types are not always enough; the description of a context depends directly on the domain to be resolved. For instance, some sectors of a Museum may have a minimal lighting to preserve certain pieces of their collections. Keeping the

mobile screen on, in this context, could not be allowed or it can disturb other users. Therefore, a developer could consider augmentations that turn off the screen whenever a lower level of light is perceived and, if a headset is plugged into the mobile device, it starts reading the Webpage content. For that purpose, he needs to create the proper augmentation with the reading capability; as an example, he can use meSpeak.js[7], a client-side JavaScript library which converts text into an audio, and can be used to extend augmenters as explained in the previous sub-section. We will focus this subsection in extending the framework with the light level detection capability to shown an example of extending the notion of context in MoWA. As we show in the next paragraph context extensions are straightforward due to the modularity properties of MoWA.

The window object in the Firefox for Android browser has a *DeviceLightEvent*[8]; it provides information about the perceived ambient light level by the device, coming from photo sensors or similar detectors. This feature is experimental but available in the browser, and it is currently a draft specification[9] by the W3C. For obtaining luminance values, the developer should add an event listener to the window object, as shown in line 17 of Figure 10. The value of the received event argument is expressed in lux, the unit of luminance set by the International System of Units.

The developer should chose a value as the minimum allowed light to keep the screen on. Building surroundings, streets and parking areas use to have lux values lower than 50. Let us say the developer chooses such value. In Figure 10, you can see this value assigned in the *maxDarkness* local variable, at the end of line 3. It is also used in line 6, for setting the current perceived lux level, a significance; if it is equal or lower than *maxDarkness*, we infer the user is in a darker ambient. Then, this significant value is compared with the last inferred one, so we can detect when the user moves from the brightness to the darkness, and back. When such kind of change is detected, then the state of the sensor is updated and a notification is sent to every application listening for changes, as shown in lines 8 and 9.

```
1  function LightLevelSensor(){
2
3      var me = this, fromTheDark, maxDarkness = 50;
4      this.handler = function(event) {
5          //Detects significant changes in the perceived light
6          var toTheDark = (event.value <= maxDarkness);
7          if( fromTheDark != toTheDark ){
8              me.setState(event.value);
9              me.notifyChange();
10         }
11         fromTheDark = toTheDark;
12     };
13 }
14 LightLevelSensor.prototype = new ContextTypeSensor();
15 LightLevelSensor.prototype.startDetection = function(){
16     //Starts listening for lux changes
17     window.addEventListener('devicelight', this.handler);
18 }
19 LightLevelSensor.prototype.stopDetection = function(){
20     //Stops listening for lux changes
21     window.removeEventListener('devicelight', this.handler);
22 }
```

Figure 10. Extending the framework with a new sensor for light level detection.

[7] meSpeak.js: http://www.masswerk.at/mespeak/
[8] Firefox devicelight event: https://developer.mozilla.org/en-US/docs/Web/Events/devicelight
[9] Ambient Light Sensor API: https://w3c.github.io/ambient-light/

Finally, it is worth mentioning that sensors can be easily imported into MoWA, through the corresponding MoWA sub-menu, available in the browser main menu. When a file is installed, MoWA also imports as much files and directories as the same root directory of the selected file has, so other libraries can be included for the sensor execution. Sensors can work at the default frequency a developer has set or the browser notifies, but developers can also provide UI controls for configuring such values by implementing the *configure* method that every *ContextTypeSensor* inherits. This method is called when the user ask MoWA to configure the sensors, but a default message is shown if they do not implement it. In the presented case, the browser notifies a change to the sensor every time a new change is available from the light sensor, through the *devicelight* event; in a similar way, the navigator.geolocation[10] object allow constantly obtaining the user positions, by subscribing to its changes through the `watchPosition` method. However, it is also possible for the developer to avoid the sensor to be notified of every change, and just ask for location through the `getCurrentPosition` method every certain period of time, and such value could be set by the developer or configured by the end user, if the developer implemented the sensor's `configure` method.

6 Comparative Analysis

Based on the presented related works, we highlighted some similarities and differences against our approach through a comparative analysis. We discuss below several characteristics, which are listed as columns in Table 1, by analysing each of the mentioned works in Section 3; every row represents one of them, where ticked fields are used to match a correspondence, and unchecked values mean that such characteristic is not supported or mentioned in the referenced writing.

The first column (a) in the comparative table concerns the system making use of a limited data consumption method. This means that the related writings describe data consumption limited just to one modality, for instance just by using Web services, consuming pre-packaged data, or just through Web augmentation techniques. Our approach contemplates all these alternatives, and due to the flexibility wrought by the fact that a developer can define their own adaptations, it can be extended with other ways of retrieval. CAMUS [8] is another flexible tool that allows defining remote APIs and in-house services into their platform for being reused from the created applications, according to the context of use. Ambient AMP [6] contemplates the use of Flickr API, and it also allows injecting additional resources by specifying it in the plug-ins primary's JavaScript file. Finally, PersonisJ [39] just allows adding evidence to the profile database by "tell" operations, but such method can be called through the application that is using this modelling framework, and there are no limitations mentioned about how to consume this new content.

The remaining works have no alternative for data consumption, or it does not fit with their main goal; the architecture in [13] just focuses in context data acquisition, not restricting the application domain to the ones who consume or not external data; Cyberguide [1], which stores data in different formats in their database, even Web pages, but it does not dynamically retrieve the up-to-date online version; the architecture proposed in [23], whose main focus is Crowdsourcing not external data consumption; MyServices [24] that just uses services; the authors of the proposed platform in [31] prefer to offer layout adaptation, such as the position of the DOM elements, instead of providing content or functionality; [34] focuses on the tour guide and the way of providing recommendations; the

[10] Geolocation in Firefox: https://developer.mozilla.org/en-US/docs/Web/API/Geolocation

approach in [38] also enhances existing Web-sites on the fly but with adaptations, not augmentations, so there is no point considering this feature; platform in [19] augments Web pages with multimodal adaptations, but does not contemplate the use of any external content to augment the current Web page; [9] uses external Web services for retrieving both geographical information and multimedia contents; the implementation in [14] uses Facebook and Outlook APIs for extracting user's data, and WikiTravel API to retrieve data from the PoIs, but it is not clear how –or just if they do– support data consumption from another APIs (in fact, they conclude they still need to increase integration and collaboration with other systems); the authoring tool in [40] allow users to select data sources by providing the URI to a Web service or via Web service adapters, but in both cases are restricted to existing services; in CASTOR [33], users create and they are the providers of their own stories.

The second compared characteristic (b) is closely related to the first one, but different in the sense that it does not matter in how many ways the external content consumption is supported, just takes into account if the system uses dynamic third party content. In this sense, dynamic excludes pre-packaged data and refers to the on-the-fly retrieved one. The base of any MoWA application is any existing Web application, so MoWA is undoubtedly included in this category. Another applicant approaches are MyServices [24], Ambient AMP [6], CAMUS [8], the system in [14] and the authoring tool in [40] through diverse Web services consumption. The not-applicant cases here share the same arguments than the previous characteristic.

Then we analysed which applications are built in the bases of other ones by or improving them, as shown in the third column (c) of the table. The difference with the previous column resides in the fact that systems could reuse external content for creating applications from scratch, or for improving an existing one. As previously said, MoWA enhances existing Web applications with mobile features, so this characteristic also applies to our work. Platforms in [19, 31], and implementations in [38, 6] are also included because they are founded in Web Adaptation or Web Augmentation principles, respectively. Architecture in [13] allow creation of new applications that define the contextual tags they provide, but they argue that as the XML tags are added at presentation layer, it is easy to redefine some of existing Web pages' source code for including their features.

We also took into account which implementations offered the attached content in a fluid, unified view with the improved application (column "d"). This requires DOM manipulation and external content consumption, so that limit the applicant cases to Web Augmentation approaches, like ours and [6, 19, 31]. We also could say that Mashups also present data in an integrated view, but all their sources are not improving per se an existing one; they are creating a new one that uses another's content to provide a new service. PersonisJ [39] can also be included in this category because it allows a flexible development by providing the requested data and letting the application present it in conjunction with other content.

In the same way we filtered characteristics just dependant of Web Augmentation, we were also interested in which of the works allowed improving existing sites through Web Adaptations; in this meaning we can include also Web Augmentation approaches, because the same implemented features can be used to reach just the adaptation goal. This is presented in column "e", and the embraced approaches are the same group concerning column "d" and [38].

We also observed, and presented in column "f", which of the analysed works require altering the existing Web application sources. We only found one applicant here; architecture in [13].

Table 1. Comparative analysis of existing works and MoWA

	a) Limited data consumption methods	b) Dynamic third party content reuse	c) Improving existing applications aim	d) Additions are presented in a fluid, unified view	e) Web adaptation	f) Requires altering the Web application sources	g) Unrestricted domain/discipline	h) Supports user location	i) Supports other context types	j) Supports personalization	k) Distributed architecture supporting the application execution	l) Extends the browser capabilities	m) Follows Crowdsourcing principles	n) Generates a Web application
MoWA	✗	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓	✓
Cyberguide [1]	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✗	✗	✗
Ambient AMP [6]	✗	✓	✓	✓	✓	✗	✗	✗	✓	✗	✓	✗	✗	✗
CAMUS [8]	✗	✓	✗	✗	✗	✗	✗	✓	✓	✓	✓	✗	✗	✗
Framework in [9]	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✗	✗	✓
Architecture in [13]	✓	✗	✓	✗	✗	✓	✗	✓	✓	✗	✗	✗	✗	✗
System in [14]	✓	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗
Platform in [19]	✓	✗	✓	✓	✓	✗	✗	✗	✓	✗	✓	✗	✗	✓
Architecture in [23]	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✗	✓	✓
MyServices [24]	✓	✓	✗	✗	✗	✗	✗	✓	✓	✗	✓	✗	✗	✓
Platform in [31]	✓	✗	✓	✓	✓	✗	✗	✓	✓	✗	✓	✗	✓	✗
CASTOR [33]	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗	✓	✗	✗	✗
Tour guide in [34]	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✗	✗	✗
COIN [38]	✓	✗	✓	✗	✓	✗	✗	✓	✓	✓	✓	✗	✗	✗
PersonisJ [39]	✗	✗	✗	✓	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗
Authoring tool in [40]	✓	✓	✗	✗	✗	✗	✗	✓	✗	✗	✓	✗	✗	✗

Having compared those characteristics related to existing applications, there is another interesting common point: the resultant kinds of applications. They were restricted to just one domain/discipline, as presented in column "g". While [34] and [14] focus in recommendations in tourism guides, [13] does it in CA features availability, and [23] in Crowdsourcing. But others present a combination of approaches, for example in [24] and [8] it predominates Context Awareness but also Mashups; in [1], tourism guides and Context Awareness; in [31], Context Awareness and Crowdsourcing; in [38], Context Awareness and Web Adaptation; in [6], Ambient AMP focuses in Web Augmentation and Smart Objects interaction; in [19], Context Awareness and Web Augmentation; in [40], mashups and Augmented Reality; in [33], Context Awareness and *in situ* narrations creation. In contrast to all these approaches, MoWA was conceived from its origins to support the extension of any kind of mobile application, and we demonstrated its feasibility through a proof of concept, by extending the

framework with two concrete subclasses, covering Mobile Hypermedia and Context Awareness. Another considerable approach is PersonisJ [39]; although it strictly focuses on building customized applications, flexibility in application deployment does not limit the type of target application and, for example, we can conceive the idea of creating personalized Mobile Hypermedia, Augmented Reality-based or Mashups applications.

While some platforms contemplate the support of the user location and its usage to provide content or functionality according to where the user is, others present a similar behaviour but taking into account a wider quantity of context types, like orientation, noise level or light level. This is shown in column "h" and "i", respectively. Applications providing location also use to provide another context type sensing mechanism. [1] contemplates location, orientation and user profile for offering their services; [13] defines tags for sensing and allowing applications to be aware of the light level, compass values, the orientation, the location, the audio capture and the camera; [23] uses location, but also the user's interests, skills, availability, network load and ratings; [24] allows dynamic Mashup development, taking into account context types as location, user-related and times; [31] uses context types related to the device and the user for recommending adaptations; [36] uses multiple dimensions for providing personalized tasks, like location, dates, hours, user age and gender, movement speed and call status; [38] enhances Web sites according to the user's preferences, characteristics and goals, the device type and characteristics, and environmental context types like location and sensory data; [34] takes into account location, the desired starting and finishing times for the visit, the poi's opening time, the user interests and collaborative ratings; Ambient AMP [6] mentions sleep state, heart rate, activity recognition, light control, but not location. We can assume it could be extended, but we do not ensure what is not mentioned. With [8] is possible to consider dimensions as time, place, the user's current company or his interests for consuming Mashup services; in the example implementation of [19], the authors contemplate sensing specific light and noise levels, hearth rate, respiration rate and user movements, and despite they don't use location, they mention the contemplated context types are extensible, so it could be covered but not mentioned; PersonisJ [39] allows sensing the user's personal context, including location; in [9] user's preferences and PoI's characteristics are taken into account for allowing users to build their own personalized itinerary; [14] uses several information for performing recommendations, like the traveller's age, gender, occupation, location, education, relational status, history of check-ins, and the visited place's climate, health and safety; [33] allows building and delivering geo-localized Context Aware stories, based in the user location, weather, time, season and listeners. [40] tracks the devices location and orientation for positioning the AR content, but these are basic context types of any AR application and do not necessarily makes it a pure CA one.

Some of the presented context types in column "i" were user-related, but taking into account just these bounded group, applications can provide accommodation for specific individuals. This is what we know as personalization, and the matched cases are presented in column "j"; the matching cases are [1, 8, 9, 14, 23, 34, 38]. In this work we presented an example supporting a slight level of personalization, by taking into account user roles as one of the context types for displaying the augmentations.

Regarding the architectures, we analysed which of them used a distributed approach for supporting the generated application execution; this is presented in column "k". It is worth mentioning that we are not referring to the consumption of third party data, but components that perform a vitally process for

the application over other platforms. We can observe that there are not many applications running at client-side, except our approach and [13, 14, 39]. This could be due to mobile resources limitations, but we tested several libraries and optimized the source code of our implementations in order to reach full client-side solutions. Regarding [13], the authors implemented a Web browser for supporting their tags interpretation. At the time they proposed their approach there was no mechanism for allowing Web applications to access the device's sensors for sensing the context with existing browsers, so the solution was building their own native application for accessing low level APIs. PersonisJ [39] is another solution intended for client-side; their aim is also to keep the user model at client side for generating confidence regarding the user-data privacy. The adaptive tourist recommendation system in [14] performs everything at client-side, but in contrast to us, they do not generate a mobile Web application and they are not limited to mobile capabilities.

The following approaches contemplate a client side solution, but some of their functionality depend on server-side components: in [1], applications connect to the server for accessing communication services and, although this is not essential for the application execution, they wanted to exploit server side functionality but they could not do it due to technical limitations; [23, 31, 36] are Crowdsourcing approaches, so they naturally involve a distributed architecture; [24] implements a context engine and stores a recommendation rules database at server-side; in [38], the semantic information extraction, the content matching and the adaptation itself are performed at server-side; [34] has a server component for data update and collaborative filtering; Ambient AMP [6] contemplates a local Web server listening for requests of bookmarklet plug-ins, the ones that augment the Web pages; [8] maintains the universal CDT at server side; the multimodal augmentation in [19] is server-based; [9] stores user activities and allows retrieving personalized data; [40] uses server side for processing images and videos for scenes detection, and also for providing the augmented content; in [33] the content consumed by the applications is generated by other users, and it must be stored in the server at creation time, and then stories can be accessed and edited from a desktop Web application, and delivered to another users depending on their context values.

Regarding implementation, we also were interesting in knowing which ones extended existing browser capabilities, for example, through extension/plug-in development. As we can see in column "l", MoWA applies because it was implemented as a Firefox for Android extension. Remaining works implemented their client-side applications as native applications or Web applications, but none of them extended the browser capabilities. It is necessary here to clarify why three of these approaches were excluded from this characteristic, and to say this consideration is open to debate; one of them is [23], that is still a conceptual architecture, so they did not face implementation yet; the second approach was [31], and despite their idea consist in providing end users with a toolkit for adapting existing Web sites directly in the browser, they do not provide a detailed level about the implemented toolkit, so we can't say if such implementation offers a completely new browser, as in [13], or if they provide their functionality through a browser plug-in implementation, like us; the last one is Ambient AMP [6], which allows performing the augmentation through bookmarklets, but it is necessary the browser having this feature implemented so, in fact, it is not about extending the browser functionalities but using them.

Among the studied works, we also wanted to identify which of them were built under the foundations of Crowdsourcing principles. We included MoWA here, because despite we lack the

repositories implementation by now, we do contemplate the existence of –at least– two user roles and we outlined how they should interact under our approach [7]. Other applicable works are fully focused on Crowdsourcing [23, 31].

Finally, we observed which implementations generate –or are based in– a Web application as the result of their process. Our approach allows developers to build their own applications, which are executed through the MoWA plug-in and improve another existing Web application; the client-side part of the Mashup construction system presented in [24] is Android based, but as result it generates a lightweight client Mashup in JavaScript and HTML, which makes it consumable for any mobile browser; [19] performs Web adaptation through a proxy, and then users consume and interact with the modified version of the Web page; the mobile component of platform in [9] is generated as a Mobile Web application.

In the same way as explained in description of column "l", we cannot strictly know implementation details of [31] about how the authors reached their goal, so we did not considered as applicant. A good argument for this consideration is that performing Web adaptation requires privileges for altering third party applications, and that can be achieved by manipulating sources from server-side, creating a native application, or extending the browser capabilities through extensions development, but we discarded this last alternative because such paper was presented in a conference in June 2011, and by then Android mobile browsers did not support extension development. One of the first navigators supporting this feature, if not the first, was Firefox for Android, available from December 2013[11]. Ambient AMP [6] is another approach discarded, because its augmentations are not consumable for non-bookmarklet browsers and its functionality relies on the background service provided by a native application. Nor we contemplate CASTOR [33] here because the implementation is PhoneGap[12] based, so it is about hybrid applications, not fully native or purely Web.

Taking into account the analysed approaches in the table, we can summarize that MoWA is part of the 25% of solutions that do not limit the data consumption methods; of the 37.5% contemplating dynamic third-party reuse; of the 37.5% whose aim is based on improving existing applications and of the 93.75% not requiring the modification of the source code of existing applications; of the 31.25% presenting the external content in a fluid unified view; of the 31.25% performing Web Adaptation techniques; of the 12.5% covering applications for unrestricted domains; of the 87.5% considering the user location for content or behaviour adaptation; of the 93.75% performing adaption according to another context types besides location; of the 56.25% supporting personalization; of the 75% whose architecture is distributed; of the 18.75% following Crowdsourcing principles and of the 31.25% of tools based or generating a Web application. It is the only one that extends the browser capabilities.

7 Evaluation

The illustrations and examples introduced in previous sections show the feasibility of building Mobile Web applications with our approach. In this section we present some discussion points about our first evaluation, introduced in [7]. Below, general details of this evaluation are described for helping the

[11] Firefox for Android report: https://blog.mozilla.org/futurereleases/2013/12/10/firefox-for-android-optimized-for-devices-that-support-intel-x86-chipsets/
[12] PhoneGap framework: http://phonegap.com/

reader to understand the discussion. Through the evaluation, we compared our approach for building mobile Web software against the "conventional" ones. We chose not to compare against the existing approaches because they could not reach an outcome of the same characteristics as we expected. Starting from the idea that our goal is to reuse existing resources, there was only five approaches, of the ones presented in section 3, having this characteristic [6, 13, 19, 31, 38]. But as was explained in the previous section, works in [13, 19, 31, 38] present some limitation regarding dynamic third party data consumption. This limits the group of applicable applications to only one [6], but it focuses in augmenting Web with smart objects interactions. In fact, the provided examples in their work are not based on sensing users but smart objects, and the only available plug-ins in their public repository[13] were focused in that field; they allow discovering and controlling nearby UPnP and AirPlay media devices, and detecting light level, battery level or installed applications in the device. This way, extending and using the existing framework carries almost the same complexity for understanding and implementing user positioning and Mobile Hypermedia support.

Reuse existing resources involves, for instance, extending a non-mobile Web site for supporting location-based access. With this idea in mind, we have set the following hypothesis: *Creating a Mobile Web application with Mobile Hypermedia features and based on existing Web sites is more efficient by using MoWA than traditional Mobile Web development techniques*. We wanted to measure the effort needed to build a *Mobile Hypermedia* application with both methods, so we recruited participants and divide them in two groups: one for implementing with traditional techniques (Group 1) and another with *Web Augmentation*-based (Group 2) methods. Each group has 8 participants which different experiences and skills in the field of Web applications. The main task to be performed by all participants was to develop a *Darwin Tour* application, which is defined with tree specific requirements. More details about the set-up of the experiment and the evaluation method are described in [7]. It is important to mention that we defined a general questionnaire for asking the participants for the satisfied requirements, and the level of difficulty perceived for each of them. Meanwhile, just participants of Group 2 completed a SUS [5] questionnaire about the use of MoWA.

The results of the experiment were, in general, auspicious for our approach. First, it was important for us to know that building *Mobile Hypermedia* applications based on MoWA was clearly feasible. We measured two main aspects: effectiveness (i.e. how many participants fulfilled all requirements) and efficiency (i.e. how much time it took for each participant to accomplish with all requirements). In terms of effectiveness, only 4 participants from Group 1 could finish the application by satisfying all requirements. The other 4 participants from the same group did not accomplish any requirement. Regarding Group 2, 7 of the 8 participants finished the application by satisfying all requirements; the last participant also finished the application but he did not accomplish one of the requirements. If we compare the performance of both groups in terms of effectiveness, we may appreciate clearly how participants from Group 2 obtained better results. Developing the *Darwin Tour* application was easier by using MoWA. There is one important comment about a threat to validity: all participants of Group 2 had JavaScript background, so it is difficult to determine if a developer without JavaScript skills could have finished the MoWA application.

[13] Ambient Amp plugins: http://www.ambientdynamix.org/plugins

In order to measure the time needed for building the application, we took into account only the successful cases; the ones where all requirements were satisfied. This includes 4 participants from Group 1 and 7 participants from Group 2. The average time of Group 1 was 18 hours and 52 minutes (with a standard deviation of 7 hours 28 minutes), and Group 2's average time was 5 hours and 17 minutes (standard deviation of 3 hours). As an interesting point, we should highlight that one participant from Group 2 finished the application in just 1 hour and 40 minutes, while other two from the same group did it in around 2 hours and 20 minutes. Not casually, these three participants were the most experienced from Group 2 in *Mobile Hypermedia* Applications development, but they did not consider themselves as experts in the field.

The final SUS score of MoWA was 81.3, which is an *acceptable* result taking into account that this is our first version of the framework. Participants from Group 2 also gave us valuable feedback regarding possible improvements of MoWA. For instance, a participant suggested developing a tool for collecting pieces from the museum on the fly and exporting the skeleton of a MoWA application. This idea may be extended to collect any PoI, which was in synchrony with the opinion of other participants from Group 2. They say, in general, that the hardest and most tedious part of developing the MoWA application was specifying the PoIs.

8 Concluding Remarks and Further Work

In this paper we presented a novel approach, called MoWA, for building Mobile Web Applications through a client-side augmentation approach. Developers can create their own augmentations based in just two assumptions: i) a set of Web pages exists, and ii) there is some mechanism of location sensing available. Our framework is in charge of mapping locations to application objects (existing Web pages), and then enhance them with the corresponding augmentation. In this way, our approach provides support for personalization and adaptation.

For helping developers, we built the MoWA framework which provides several extension points that make easier the process of specifying a new kind of mobile application, by extending the abstract base class *MobileApplication*, or new augmentations for adapting Web pages in a particular way. This way, developers can create artefacts according to their needs. It is required for each MoWA application to have defined: the PoIs, the space representation, the relevant sensing mechanisms and the augmentation methods. Specifically, we have shown how to create an application for a particular domain and scenario; the *Darwin Tour*. Our plug-in allows running MoWA scripts in the mobile browser. According to the preferred scripts installed by the end user, original Web pages can be personalized. Therefore, MoWA gives support for two kinds of users: developers and end users, through the framework and the plug-in respectively.

An exhaustive analysis of related works was performed; we were not only surveying existing contributions but also verifying the existence of similar purpose tools. We have carefully analysed fourteen features over fifteen approaches, argued the reasons for applicability and then compared them against our approach. We began the study expecting to find alternatives against which to conduct experiments and compare results, but eventually we concluded that there is no solution addressing the same capabilities as MoWA.

We performed a first evaluation that shows that our approach is feasible and effective than a "traditional" development approach. The discussion presented in the paper shows the feasibility of applying our approach to existing Web applications, and that it can be considered promising for developing mobile WA applications.

We are also improving the presented Crowdsourcing approach by implementing the artefacts repository at server side, enabling interactions from the MoWA plug-in and also working on visual tools for pre-building specific artefacts, in order to simplify the development process of creating personalized Mobile Web applications. Such tools are oriented to users with no or minimum programming skills. One of the goals is to facilitate the work of developers in the process of gathering specific information about each PoI while walking on a place, and then generate a source code template that they can manipulate later. Another opportunity is an End User Development tool for allowing users to specify and generate MoWA applications through visual components. Such authoring tool will also enable them to build their own solutions in the same place the application will run. The integration of this *in situ* approach will also carry agile development benefits.

References

1. Abowd, G., Atkeson, C., Hong, J., Long, S., Kooper, R. and Pinkerton, M. Cyberguide: A mobile context-aware tour guide. Wireless NetworksWINET-3(5)(1997), 421–433.
2. Asakawa C. and Takagi H. Transcoding. In Web Accessibility (Springer London, 2008), pp. 231-260.
3. Baldauf, M., Dustdar, S., & Rosenberg, F. (2007). A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computing, 2(4), 263-277.
4. Bouvin, N.O., Christensen, B.G., Gronbaek, K. and Hansen, F.A. HyCon: a framework for context-aware mobile hypermedia. In The New Review of Hypermedia and MultimediaNRHM-9(1) (2003), 59–88.
5. Brooke, J. (1996). SUS-A quick and dirty usability scale. Usability evaluation in industry, 189(194), 4-7.
6. Carlson D. and Ruge, L. Ambient Amp: An open framework for dynamically augmenting legacy Websites with context-awareness. In IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing ISSNIP, IEEE , 2014, pp. 1-6.
7. Challiol, C., Firmenich, S., Bosetti, G.A., Gordillo, S.E. and Rossi, G. Crowdsourcing Mobile Web Applications. In Current Trends in Web Engineering, eds. Q.Z. Sheng and J.Kjeldskov, (Springer International Publishing, 2013), pp. 223–237.
8. Corvetta, F., Matera, M., Medana, R., Quintarelli, E., Rizzo, V. and Tanca, L. Designing and Developing Context-Aware Mobile Mashups: The CAMUS Approach. In 15th International Conference on Web Engineering ICWE (2015), Springer International Publishing, pp. 651-654.
9. D'Amico, G., Ercoli S. and Del Bimbo, A. A Framework for Itinerary Personalization in Cultural Tourism of Smart Cities. In XIII Conference of the Italian Association for Artificial Intelligence AI*IA (2013).
10. Díaz, O. Understanding Web augmentation. In Current Trends in Web Engineering, eds. F. Daniel and F.M. Facca, (Springer Berlin Heidelberg, 2012), pp. 79–80.
11. Díaz, O. and Arellano, C. The Augmented Web: Rationales, Opportunities, and Challenges on Browser-Side Transcoding. ACM Transactions on the Web TWEB, 9(2), 8.
12. Emmanouilidis, C., Koutsiamanis, R.A. and Tasidou, A. Mobile guides: Taxonomy of architectures, context awareness, technologies and applications. Journal of Network and Computer ApplicationsJNCA-36(1) (2013), 103–125.

13. Espada, J.P., Crespo, R.G., Martínez, O.S., Cristina Pelayo, G., Bustelo, B. and Lovelle, J.M.C. Extensible architecture for context-aware mobile web applications. Expert Systems with ApplicationsESWA-39(10)(2012), 9686–9694.
14. Etaati, L. and Sundaram, D. Adaptive tourist recommendation system: conceptual frameworks and implementations. Vietnam Journal of Computer Science 2(2)(2014), pp. 95-107.
15. Forman, G. and Zahorjan, J. The challenges of mobile computing. In Computer, 27(4)(1994), pp. 38-47.
16. Fortier, A., Rossi, G., Gordillo, S.E. and Challiol, C. Dealing with Variability in Context-Aware Mobile Software. Journal of System and SoftwareJSS-83(6)(2010), 915–936.
17. Fortier, A. Challiol, C., Fernández, J.L., Robles, S., Rossi, G., and Gordillo, S.E. Exploiting personal web servers for mobile context-aware applications. Knowledge Engineering ReviewKER-2(2)(2014), 134–153.
18. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software (Pearson Education, 1994).
19. Ghiani, G., Manca, M., Paternò, F. and Porta, C. Beyond responsive design: context-dependent multimodal augmentation of web applications. In: Mobile Web Information Systems (Springer International Publishing, 2014), pp. 71-85.
20. Hansen, F.A. and Bouvin, N.O. Mobile Learning in Context — Context-aware Hypermedia in the Wild. International Journal of Interactive Mobile TechnologiesiJIM-3(1)(2009), 6–21.
21. Harper, S., Goble, C. and Pettitt, S. proXimity: Walking the Link. Journal of Digital Information JoDI-5(1)(2006).
22. Hong, J.Y., Suh, E.H. and Kim, S.J. Context-aware systems: A literature review and classification. Expert Systems with ApplicationsESWA-36(4)(2009), 8509–8522.
23. Korthaus, A. and Dai, W. Crowdsourcing in Heterogeneous Networked Environments-Opportunities and Challenges. In Proc. of 15th International Conference on Network-Based Information Systems (NBiS'12), IEEE, Korea, 2012, pp. 483–488.
24. Lee, E. and Joo, H.J. Developing lightweight context-aware service mashup applications. In Proc. of 15th International Conference on Advanced Communication Technology (ICACT'13), IEEE, Korea, 2013, pp. 1060–1064.
25. Lehman, M.M. Laws of software evolution revisited. In Software Process Technology, ed Oquendo, F. (Springer Berlin Heidelberg, 1996), pp.108–124.
26. Leonhardt, U. Supporting location-awareness in open distributed systems. Doctoral dissertation, Imperial College, 1998.
27. Lieberman, H., Paternò, F., Klann, M., & Wulf, V. (2006). End-user development: An emerging paradigm. In End user development (pp. 1-8). Springer Netherlands.
28. Luz, N., Silva, N. and Novais, P. A survey of task-oriented crowdsourcing. Artificial Intelligence Review, 2014, pp. 1-27.
29. Millard, D.E., De Roure, D.C., Michaelides, D.T., Thompson, M.K. and Weal, M.J. Navigational hypertext models for physical hypermedia environments. In Proc. of 15th ACM Conference on Hypertext and Hypermedia, ACM, New York, USA, 2004, pp. 110–111.
30. Musumba, G.W. and Nyongesa, H.O. Context awareness in mobile computing: A review. International Journal of Machine Learning and ApplicationsIJMLA-2(1)(2013), Article No. 5.
31. Nebeling, M. and Norrie, M.C. Context-Aware and Adaptive Web Interfaces: A Crowdsourcing Approach. In Current Trends in Web Engineering, eds. F. Daniel and F.M. Facca, (Springer Berlin Heidelberg, 2012), pp. 167–170.
32. Pejovic, V. and Musolesi, M. Anticipatory mobile computing: A survey of the state of the art and research challenges. ACM Computing Surveys (3)(2015), pp. 47.

33. Pittarello, F. and Bertani, L. Castor: Designing and experimenting a context-aware architecture for creating stories outdoors. Journal of Visual Languages & Computing JVLC 25(6)(2014), 1030-1039.
34. Schaller, R. Mobile tourist guides: bridging the gap between automation and users retaining control of their itineraries. In Proc. of the 5th Information Interaction in Context Symposium, ACM, pp. 320-323.
35. Talukdar, A.K. Mobile Computing (2)(2010), Tata McGraw-Hill Education.
36. Tamilin, A., Carreras, I., Ssebaggala, E., Opira, A. and Conci, N. Context-aware mobile Crowdsourcing. In Proc. of the 2012 ACM Conference on Ubiquitous Computing, ACM, New York, USA, 2012, pp. 717–720.
37. Tomitsch, M., Mitchell, M.C. and Weng, H. Designing for mobile interaction with augmented objects. In Proc.of 2012 International Symposium on Pervasive Displays, ACM, New York, USA, 2012, Article No. 5.
38. Van Woensel, W., Casteleyn, S. and De Troyer, O. A generic approach for on-the-fly adding of context-aware features to existing Web sites. In Proc. of 22nd ACM conference on Hypertext and hypermedia, ACM, New York, USA, 2011, pp. 143–152.
39. Wasinger, R., Fry, M., Gerber, S., Iivonen, O., Kay, J. and Kummerfeld, B. Client-side User Modelling across Multiple Mobile Applications. University of Sydney, School of Information Technologies, 2011.
40. You, Y. and Mattila, V. Visualizing web mash-ups for in-situ vision-based mobile AR applications. In Proc. of the 21st ACM international conference on Multimedia ACM MM (2013), ACM, pp. 413-414.