Hindawi
Mobile Information Systems
Volume 2017, Article ID 2525367, 28 pages
https://doi.org/10.1155/2017/2525367

Research Article

An End User Development Approach for Mobile Web Augmentation

Gabriela Bosetti,[1] Sergio Firmenich,[1,2] Silvia E. Gordillo,[1,3] Gustavo Rossi,[1,2] and Marco Winckler[4]

[1]*LIFIA, Facultad de Informática, UNLP, La Plata, Argentina*
[2]*Consejo Nacional de Investigaciones Científicas y Técnicas, Buenos Aires, Argentina*
[3]*Comisión de Investigaciones Científicas, Buenos Aires, Argentina*
[4]*ICS-IRIT, University of Toulouse 3, Toulouse, France*

Correspondence should be addressed to Gabriela Bosetti; gabriela.bosetti@lifia.info.unlp.edu.ar

Received 12 October 2016; Revised 5 December 2016; Accepted 26 December 2016; Published 20 February 2017

Academic Editor: Jose M. Barcelo-Ordinas

The trend towards mobile devices usage has made it possible for the Web to be conceived not only as an information space but also as a ubiquitous platform where users perform all kinds of tasks. In some cases, users access the Web with native mobile applications developed for well-known sites, such as, LinkedIn, Facebook, and Twitter. These native applications might offer further (e.g., location-based) functionalities to their users in comparison with their corresponding Web sites because they were developed with mobile features in mind. However, many Web applications have no native counterpart and users access them using a mobile Web browser. Although the access to context information is not a complex issue nowadays, not all Web applications adapt themselves according to it or diversely improve the user experience by listening to a wide range of sensors. At some point, users might want to add mobile features to these Web sites, even if those features were not originally supported. In this paper, we present a novel approach to allow end users to augment their preferred Web sites with mobile features. We support our claims by presenting a framework for mobile Web augmentation, an authoring tool, and an evaluation with 21 end users.

1. Introduction

The increasing growth of both the Web and mobile technology has raised new forms of participation of end users. Mobile devices started to be used as personal computers, competing and even replacing desktop computers for daily life tasks and, while the resources of these devices increased, so did, and still does, the number of innovative functionalities they can benefit from. Today, it is usual that end users interact with the same Web applications from both, desktop computers and mobile devices.

Some of the most popular Web applications (like Facebook, Twitter, YouTube, and others) provide native mobile applications, and some of them (like Booking) also provide nice location-based features to profit from the mobile nature of user devices. However, many Web sites are still accessed from Web browsers in the device, and they do not provide any mobile features. For instance, you can observe that there are search engines of popular Web sites (we choose them from http://www.alexa.com/topsites) like IMDB, Amazon, Google, or Reddit that do not offer different results while the user is performing the search in noisy or silent environments. A possible benefit from this feature is giving priority to audio-visual content when the user is in a quiet places and textual content when he is in noisy environments. Providing users with access and desktop versions of Web sites is not enough; they might need to be designed to meet the particular requirements of their context of use, as explained in [1]. This phenomenon might be caused either by economic reasons (to build the native application), by the difficulties to modify the Web application to support the mobile features, or just for lack of interest. In any case, end users lack the possibility of better access the information and services provided by these applications.

This situation is worse for Web applications providing information that is naturally location-based, such as the one

Table 1: Total scripts and installations by three popular userscripts communities.

	Userscripts-mirror	Greasyfork (https://greasyfork.org/fr/scripts)	Openuserjs (https://openuserjs.org/?p=1)
Total scripts	130956	13603	3919
Installations	1203605530	1259866	13305857

presented by museums, city halls, or tourist applications. In these cases, accessing parts of such information in-situ (e.g., visiting the museum with a smart-phone) could certainly enrich the visit, providing the end user with locative media experiences, as it has been shown in dozens of cases [2, 3]. Some of these institutions provide a location-based feature by adding QR codes to some of the points of interest (artworks, monuments) so users can explore some information with their devices, for example, by using QRpedia (http://qrpedia.org/). More complex scenarios, like itinerant or temporary exhibitions, might complicate things further. In this paper we present an approach which aims to empower end users to implement mobile Web applications by profiting from information (and services) already existing in the original Web sites, but enriching them with different kinds of mobile functionality. The underlying philosophy of our approach is the one provided by the concept of Web augmentation.

During the last years, end users started to, unofficially, add new functionalities to Web applications when some of their requirements were not contemplated originally [4]. Big communities of userscripts and browser extensions (as the ones at https://addons.mozilla.org/ or https://chrome.google.com/webstore/category/extensions) support this claim. For instance, userscripts used to be a repository that recorded a large number of scripts per year. The site shut down in 2014 but a mirror (http://userscripts-mirror.org/) is still accessible, and it registers a total of 130956 scripts and 1203605530 installs (accumulated among all scripts) (we obtained the numbers on December 18, by running a script on Greasemonkey for traversing all the site's pages and obtaining such numbers by analysing their DOM. In the same way, we got the values presented in Table 1). In Table 1, we present these quantities also for other two similar repositories that remained active since 2014.

The existence of these sites shows that the consumption of scripts to adapt the Web is a trend and grows quickly. From the Greasyfork site, we could also know that 4224 scripts were created in 2014, 4276 in 2015, and 5103 in 2016. This means that the quantity of scripts in that site grew more than three times since 2014 to date. In addition, we also got the number of latest updates per year of such scripts (see Figure 1), which demonstrates that most scripts tend to be updated by the community over time.

In this way, users not only began to participate under the role of *consumers* of Web applications, but also learned how to become *producers* of their own solutions. Being a producer is not necessarily synonymous with having, or not, technical knowledge or expertise. Within this category of end users, you can find people with skills for textual or visual programming, as well as those who do not have much technical knowledge but can also build their applications by using simple

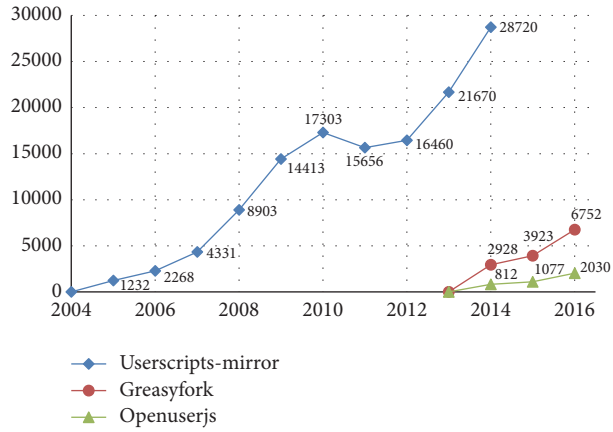

Figure 1: Number of ≪last updates≫ by scripts.

or assisted tools, as form-based wizards or tools supporting Programming By Example [5].

A very popular practice for adapting existing third-party Web applications is Web augmentation (WA) [6, 7], and there are different strategies for achieving it; one of them is client-side scripting, which consists in manipulating the applications' user interface (UI) when a particular site is already loaded on the browser. WA makes it possible to change the content, style, structure, and functionality of a Web page without the need for changing the source code at server-side. The scripts that perform a specific adaptation are called *augmenters* [7], and it is usual that their creators have some level of expertise in JavaScript. The amount of features that can be opportunistically added to an existing application are countless, moreover, taking into account the possibility of consuming information from a wide range of sources. For instance, EBay products can be augmented with more information for the user to decide whether to purchase it or not by summarizing opinions on the sellers or checking the price of the product in other sites. In the same way, books in Goodreads can be augmented with more options for buying a copy, promoting local online stores, and actors in IMDB may also include pictures from Google Images in the case there are no, or few, images in their profiles.

In this paper, we focus on mobile Web augmentation, adding different kinds of mobile-based features to existing Web applications. Applying Web augmentation on mobile devices implies that, besides the common aspects that may be adapted typically (e.g., look and feel, personalization, recommendations), mobile and context-aware features can be contemplated as well. For instance, we could take into account the user's position for augmenting a news portal with geopositioned and content-related videos and tweets. By taking

the current user's position through the Geoposition Web API (https://developer.mozilla.org/en-US/docs/Web/API/Geolocation), it is possible to use it for building geolocated queries through the Data API of YouTube (https://developers.google.com/youtube/v3/docs/search/list) and the REST API of Twitter (https://dev.twitter.com/rest/reference/get/geo/search). Then, the retrieved content could be injected into a news portal's Web page by using the Addon-SDK (https://developer.mozilla.org/en-US/Add-ons/Firefox_for_Android) of Firefox. We can listen for device's orientation changes (DeviceOrientationEvent (https://developer.mozilla.org/en-US/docs/Web/API/DeviceOrientationEvent)) and augment Google Maps' Web page with a real world view (camera capture) and an arrow pointing to the target location when the user tilts the device vertically. We can calculate the perceived sound level by using the WebRTC API (https://webrtc.github.io/samples/src/content/getusermedia/volume/) for automatically adapting the volume of YouTube videos at middle levels or stopping/resuming the reproduction at the high ones.

Although there are already some approaches for augmenting Web applications from mobile Web browsers, such as [8–10], they are aimed and limited to *producers with programming skills* (from now on, *developers*). In a previous work, we have presented a mobile Web augmentation (MoWA) approach [11, 12], which comprises a software framework and a set of tools for developers. Basically, MoWA provided developers with a framework for creating mobile Web applications based on client-side adaptation, including the addition of new (e.g., location-based) contents and functionality (such as context-awareness) directly on the front-end, that is, the Web browser. In this paper, by incorporating EUD principles, we change our target audience from developers to broader ones: producers with no programming skills (from now on just *producers*).

Providing *producers* with the tools for specifying how their preferred applications should be augmented is a good solution to help them to augment the Web from mobile devices, thus, to create better mobile Web experiences. This strategy is reasonable since many recent studies have demonstrated that there is a global tendency of end users meeting the concrete needs of domain-specific scenarios by creating their own applications by using end user development (EUD) tools [13, 14].

In this work, we extended the MoWA approach with the aim of creating a general purpose authoring tool, in such a way that end users can develop their own applications without the need for having programming skills or even having to write a single line of code. As the main contributions of this paper, we aim to

(i) analyse how to overcome the challenge of developing mobile Web applications by using an EUD approach based on augmentation;

(ii) outline an approach in which *developers* create domain-specific components called *builders*, which are composable constructs available through an authoring tool, with the aim of empowering end users *(producers)* with the capability of creating domain-specific applications.

The remainder of this paper is structured as follows. Section 2 presents some background in regard of the main topics faced in this approach: End User Development, mobile applications, Web augmentation, and our previous approach, called MoWA [11, 12]. In Section 3, we present our contribution: an end user development approach for mobile WA applications. Section 4 presents our supporting tool and a case study. Evaluation procedures and results are presented in Section 5, involving participants with diverse characteristics: education levels, fields of study, ages, genders, used to different mobile platforms, using mobile devices with different frequencies, and having diverse levels of expertise in the use of such technologies. Section 6 outlines the existing work in EUD concerning Web applications, mobile applications development, from both, desktop and mobile platforms, and mobile Web applications development. Finally, Section 7 draws the conclusions of this work.

2. Background

2.1. End User Development. Some studies [13, 14] indicate that there is a strong tendency demonstrating that the end user is starting to create, modify, or extend their own software artefacts by using programming environments that abstract, somehow, the complexity of the development. This tendency gave rise to what we actually call end user development (EUD) [5, 15] and was motivated by the need of users to quickly build their own solutions for the needs they have in their daily lives or circumstantially. Different to «traditional» software engineering approaches, in EUD, the same person plays the role of developer and end user; he is the one who knows his context and his needs better than anybody, and who does not necessarily have formal education in development processes. EUD comprises a set of methods, techniques, and tools that empower the user with the capability of creating, modifying, or extending software artefacts.

For achieving the aforementioned, EUD relies on some well-known programming techniques, like Programming By Example (a.k.a. by Demonstration) [16], Extended Annotation (a.k.a. extended parametrization) [5], Visual Programming [17], and Scripting Languages [5]. The first technique consists in recording the sequence of actions of an end user in a system, so the application is built on the specification of the user's demonstration, and it does not require the user to code. The generated application should reproduce the commands he executed and also allow him to parameterize some data objects used in the demonstration. The main benefit is that the user is writing programs through the user interface he is already familiar with, but it limits the end user to use already existing functionality in the base system or to add control structures to the recorded program. The second one is about associating a new functionality with the annotations the user makes, for example, allowing the user to annotate the DOM elements of a Web page and associating a specialized behaviour to the element, as in [18]. Visual Programming is another set of techniques, all of them intended to build a program by

letting the user combine visual constructs. Languages here have a visual counterpart, a representation of the available conceptual entities and operations (the programming constructs). In contraposition, text-based languages cover just one dimension and require the user to learn the construct names and a syntax. Although this last technique can be a bit confusing, because it is hard to imagine an end user writing code, there are already widely used tools that implement this technique and which are considered traditional examples of EUD, such as applications created using formulas in spreadsheets [13, 15].

EUD applications started spreading in the Web, where large number of users joined different online communities to create and share their applications through public repositories. For instance, the aforementioned Userscripts or GreaseFork allow users to share Greasemonkey (http://www.greasespot.net/) scripts that adapt and augment the Web. Even traditional applications, such as desktop spreadsheets, began to be conceived as part of the global Web scenario with the use of online, shareable, and multiple access applications, like Google Sheets (https://docs.google.com/spreadsheets/). This application is a target not only for being considered an EUD environment, but also because it can be augmented through the mentioned techniques. EUD expanded its scope to the mobile applications field and it represented new challenges of integration with the possibility of providing both, the development process and the resultant application, with features based in diverse context types, like positioning, orientation, or noise perception level. Studies like [19] have been performed and it demonstrated, despite specific issues concerning their respective work, the steady growth of this new tendency.

2.2. Mobile Web Applications. One of the main benefits brought by the development of mobile applications [20] was the possibility of providing applications with mobile features (e.g., location-based) to offer the users customized services according to their environment. For instance, context-aware applications [21] constantly monitor the users' environment and adapt the behaviour of the application accordingly. mobile hypermedia applications [22] make use of the position of the users and the points of interest (PoI) for assisting them in their navigation through the real world. There are also mobile augmented reality applications [23] that consider the user's position for computing the position of digital objects in the real environment and draw them into a virtual layer.

During a long time, this kind of applications were developed with native code or specialized intermediate frameworks, which allowed developers to create native or hybrid applications [24]. The latter ones are Web-based but not pure mobile Web applications, because they depend on native components for providing the end user with Web content. At that time, mobile Web applications had little significance because there were not mechanisms for direct access to the device's internal services. Then, some works started to develop strategies for allowing mobile Web applications to access the context information. For example, in [25] the authors presented a custom browser that interprets applications that are enhanced with specific XML tags. Such tags

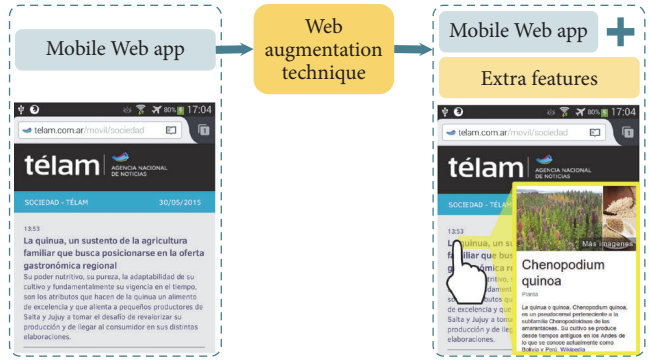

Figure 2: Web augmentation technique exemplification on mobile.

requested specific context information, and the information communication is achieved through the implementation of RESTful web services or POST requests. In [9] the authors adapt existing Web applications according to the context of use, for example, sensing light and noise levels, hearth rate, and user movements, but consuming the contextual information through native applications, not the Web application itself, called «context delegates» that notify the server-side components for resolving the adaptation, which will be finally delivered to the client Web browser.

Nevertheless, due to recent advances in mobile Web browsers, Web applications can use the devices sensors values for adapting their behaviour, and new and interesting functionalities can be created. For instance, two of the most popular Web browsers provide each a Web API (Firefox Web API: https://developer.mozilla.org/en/docs/WebAPI, Google Chrome's Web API: https://developer.chrome.com/extensions/api_other) that allow regular Web applications to access some contextual information (e.g. geolocation, proximity status, battery status, audio, video, etc.). Moreover, the W3C is working on the standards of such APIs, for example, the ones regarding the battery (https://w3c.github.io/battery/) and the proximity (https://w3c.github.io/proximity/) sensor are currently an editor's draft, and the geolocation API (https://www.w3.org/TR/geolocation-API/) is already a Web standard.

2.3. Web Augmentation. Web augmentation (WA) [6, 7] is a set of techniques for manipulating and enhancing existing Web pages with new features. This makes it possible to meet needs that were not contemplated when a Web application was implemented, either because (a) there was no such need at that time and could not be anticipated, (b) it was not detected at the requirements elicitation stage, or (c) simply because it was intentionally ignored. Augmentation is suitable for third-party stakeholders with very particular interests that have not been implemented; augmentations might also convey useful information for the owners of Web applications, because it can be used to identify uncovered needs of customers. For example, in Figure 2 we can appreciate how Télam, a news' portal, can be augmented with the capability of looking definitions in an online encyclopedia, when the user holds a word on the screen of his mobile.

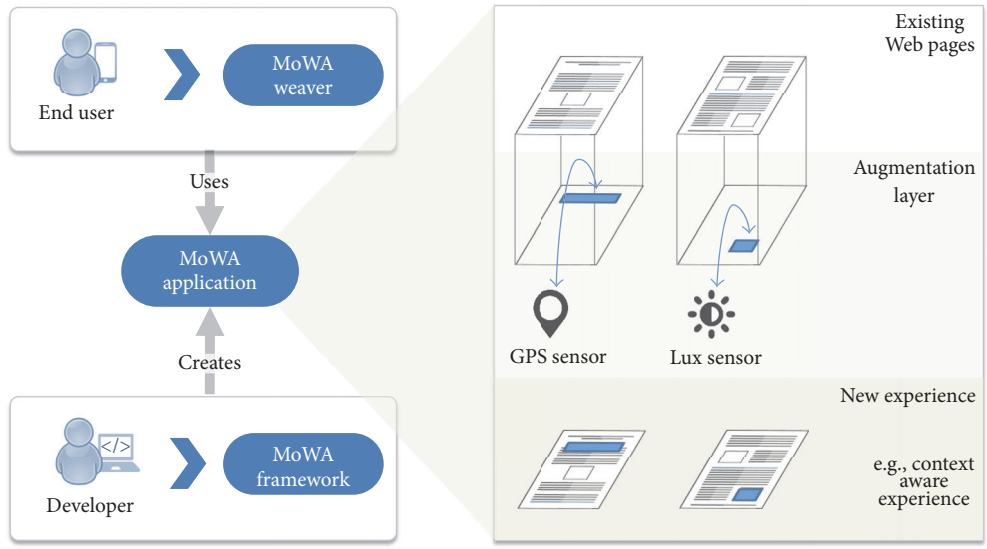

Figure 3: The MoWA approach.

There are several ways for implementing WA, but in general terms, we can classify them according to where the process takes place: at client-side or remotely (usually proxy servers). In all cases, augmentations are usually materialized through the manipulation of Web Sites DOM, which is basically the UI that users perceive on client-side.

Web augmentation on client-side is appealing since different users can share or eventually improve the same *augmenter* on their own devices, without depending on third-party proxy servers such as transcoding [7, 26]. On client-side, there are several ways to address the deployment of such augmenters, but usually it implies installing some Web browser extension. These extensions may be found in the corresponding browsers markets (e.g. Firefox, Chrome, etc), and they are usually designed for augmenting a specific Web application (for instance, Amazon, YouTube, etc.). There are other kind of extensions which act as weaving engines enabling the execution of augmenters written in a Web browser agnostic way. Such augmenters are usually referred as userscripts. Most of these engines are available for all well-known Web browsers; they allow the reuse of the augmenters, something that is not possible with specific Web browser extensions.

Without taking into account the deployment strategy, as we already mentioned in Section 1, we can find several hundred thousand augmenters. The most used, such as *magic actions for YouTube* (https://chrome.google.com/webstore/detail/magic-actions-for-youtube/abjcfabbhafbcdfjoecdgepllmpfceif), have been downloaded more than two and a half million times by users, others such as *Flatbook* (https://chrome.google.com/webstore/detail/flatbook/kadbillinepbjlgenaliokdhejdmmlgp) (an extension for augmenting Facebook) near to one million, and others less known around one hundred thousand, such as *Plus for Trello* (https://chrome.google.com/webstore/detail/plus-for-trello-time-trac/gjjpophepkbhejnglcmkdnncmaanojkf) or *Koc Power Bot* (https://greasyfork.org/es/scripts/892-koc-power-bot). There are even «official» extensions that provide better experience to customers, such as Amazon Assistant (https://chrome.google.com/webstore/detail/amazon-assistant-for-chro/pbjikboenpfhbbejgkoklgkhjpfogcam), with more than three million users. Besides that, all these repositories allow users to contribute and send feedback about the augmenters. The reader may see that for the most used ones; there are hundreds of comments suggesting changes or improvements. A deeper survey about exiting artefacts for the augmented Web can be found in [7].

2.4. MoWA Framework for Developers. In a previous work [11] we have presented an approach for augmenting existing Web applications that originally do not contemplate mobile features. The approach, called mobile Web augmentation (MoWA), comprises an augmentation framework and a weaver for running client-side scripts (MoWA applications) on top of existing Web pages, after a Web page was parsed and loaded into the browser. Developers instantiate the framework classes and extend them with a new kind of MoWA application, which can be finally instantiated. As shown in Figure 3, applications are installed into the MoWA weaver, a mobile browser extension, and their underlying augmentations are triggered either because the end user manually navigated to a target page, or because it was loaded in response to a change in the context (e.g., the user position). In both cases, the weaver runs the corresponding MoWA applications, which, in general terms, modify the UI by adding extra components. Such components reside in an augmentation layer and they are adaptive to the user's context; this means that style, content, or behaviour may be adapted when some context value changes.

MoWA relies on the definition of *sensors* as observers [27] of *context types* [28] (e.g., location, time, and orientation). At the same time, such sensors play the role of subject for those MoWA applications that are subscribed to them; the goal is to adapt the Web when a proper change in the observed context happens. Such changes are performed by augmentation components, introduced by MoWA applications over

the content of a target Web page. To this end, the application knows one or many URLs of the target Web pages to augment. For instance, consider a MoWA application that augments YouTube by adapting the application to changes in the noise level perceived by the phone; the volume of a video could be increased when it is being played in noisy environments and decreased in the silent ones. Reusing content (from the target Web page to augment and from some external ones) and accessing the mobile sensors are possible because the framework runs under the context of a mobile browser extension, which has access to the proper low level APIs.

MoWA requires some basic knowledge of JavaScript to instantiate and combine the framework features, and it offers several hot spots [29] that are easy to extend. Some of them address specific behaviour related to the supported context types. For instance, supporting orientation requires implementing a sensor but also a proper context type, in order to define some concrete context values of interest for the application. In this case, the context type is the *PointInSpace* class, which has one attribute for each axis in the tridimensional space, but other possible *ContextTypes* are *lux*, *Decibel*, *GeoLocation*, *QrBasedLocation*, *BatteryCharge*, *TelephonyStatus*, *UserProfile*, and so on. Other hot spots deal with the augmentation aspects, enabling the incorporation of new augmenters, interpreting new kinds of context values, or supporting new context types representation systems (e.g., an outdoors map, a 2D floor plan, a decibel scale, and a brightness scale), domain functionalities, and so on.

As already mentioned, a MoWA application knows which Web pages to augment and, for each of them, there are a set of associated augmenters that developers can extend. What an augmenter can do depends on what a developer wants and can do; it could be as simple and generic as displaying a new sentence (e.g., supporting a sticky note) or as complex and specific as supporting a whole tour guide through a city (whose points of interest have a digital counterpart). In both cases, we are referring to augmentations related to the addition of new content; in the latter, we are also attaching new behaviour to the Web page. Nevertheless, other augmentations are also possible in relation to the manipulation of style and structure of the underlying Web documents. For instance, consider visualizing an extra control on the top of any Web page for increasing and decreasing the font sizes, or for applying a new layout based on the selection of some elements listed on the page. Moreover, each augmenter, despite which context type it observes, can also have defined a reference for external content. Therefore, it is possible to conceive the same augmenter checking the current time and ordering elements (which have an associated time) to be reused in diverse contexts, as movies in a cinema Web site or guided tours listed in the web site of a city Hall.

Summarizing, MoWA empowers developers with a set of software features bringing together both the mobile and the augmentation worlds and the underlying concepts for a concrete MoWA application are

(i) an existing *Web application* to be augmented or adapted with mobile features;

(ii) the specification of the *context types* that will be used for the mobile experience; these types are based on a particular mobile device capability, such as GPS geolocalization or light sensors; context values are made available to applications through software *sensors*, which either are part of the MoWA framework or can be defined by the developer; for instance, the location, as a context type**,** can be tracked for assisting the end user to traverse a tour's path;

(iii) the association of sensors to a particular URL (through a concrete MoWA application), to be opened and augmented when interesting events are notified by those sensors;

(iv) a set of *augmenters* that specifically implement the augmentation layer considering the underlying application domain and the observed sensors.

While validating the approach [11], we gathered opinions and suggestions from participants; they emphasized the need of tools for automating some of the tasks they had to perform, for example, gathering the information required for the definition of the PoI and the positioning of their markers in the map. Since such tasks imply a constant repetition of code and actions, we decided to create a visual tool such that it could be used not only by *developers*, but also by *producers*.

3. Our Approach

In our research, we aim to empower end users with (1) the capability of augmenting existing Web sites with mobile features and doing so (2) according to their own requirements and (3) from their own mobile devices (although it may be done form desktop computers). It is worth mentioning at this point that there are some platforms allowing the authoring process from mobile environments [30–33], but none of them conceives the creation or adaptation of Web applications; they depend, at some point, on a native or hybrid functionality. There is also an approach contemplating the EUD of augmentations [4] but not considering mobility for the development process nor the resultant applications, and it is based on a nonvisual language. Finally, there are applications augmenting Web sites from mobile devices [8–10], but they lack EUD support. None of them perform the enhancement as part of the mobile browser, and their operation depends on a native component (e.g., for sensing and propagating the user's position). We present further details concerning existing works in Section 6.

Our challenge was providing end users with a usable tool for creating mobile Web augmented applications and finding if they were able to easily understand and apply the required concepts of the approach (listed in Section 2.4). In this direction, we developed a domain-specific authoring tool for creating mobile Web applications, to enable end users to create their own experiences on-demand and in-situ. The benefit is the users not only create their own solutions but also facilitate the validation of functional requirements, since the same person setting out the requirements for the application is also in charge of meeting them under the same context in which the application will be used.

Below, we present some possible scenarios to be faced with our approach, the supported user roles, and the main concepts of our approach.

3.1. Motivating Scenarios. We next present different scenarios designed to show both the potential and the flexibility of our approach. Although different scenarios may be combined, for the sake of clarity, we separated them into two categories: one focusing on the use of context-awareness in general and another, more specific one, in tours, by connecting diverse points of interest. We are aware that finer grained scenarios can be devised and tools that are more specific can be developed for this aim; we briefly comment this possibility in Section 7.

3.1.1. Context-Aware Scenarios. The idea of augmenting existing Web applications with context-aware features is about enabling mechanisms for retrieving context values from the environment through the device sensors so the application can adapt its content and behaviour accordingly. The following scenarios are characterized, precisely, by using context information in their augmentations.

(i) Mobile Multimedia: hundreds of Web sites containing embedded multimedia resources (such as YouTube/Vimeo videos) could be adapted in order to play those resources according to some context types, like light and noise levels.

(ii) Yellow Pages Web Sites: this kind of Web sites could be improved considering the current user's location. In this way, the results for a specific search may be enriched with a map showing how to reach some of the resulting places from the current location. The mobile experience could start a step before, considering to (auto) fill the search form considering the current user's city.

(iii) Mobile Cinema portals: in many cities, the cinemas make their billboard public in a common Web site (e.g., in La Plata, http://www.cinemacity.com.ar). In these sites, visitors may see the available movies and their functions. We can adapt portals like Cinema City into a full mobile Web application supporting different functionality.

  (a) Reordering movies according to the current time: the movie schedules could be sorted to give a quick overview of the ones that are starting soon.

  (b) Recommending the nearest cinema for a specific movie: considering the walking time for the movie that a user wants to see, the application may recommend the best cinema to go.

  (c) Adding a map showing the path to the cinema: when the user chooses a cinema, the application could add a map showing the path from the user location.

3.1.2. Tour-Based Scenarios. Tour guides represent a typical example of mobile applications; they provide people with assistance and information about a set of PoI spread across a city, museum, educational establishment, and so on. They usually consider (1) a set of PoIs, (2) a predefined path for navigating among these PoIs, and (3) a method for sensing the user's current position (QR codes, GPS, etc.). Below, we present some examples.

(i) Fixed Indoor Exhibitions in Museums: some museums use QR codes to identify each piece of their collection, and scanning them allows the mobile device to access a digital counterpart of the object. For instance, consider the codes generated by QRpedia (http://qrpedia.org/) that allow redirecting to concrete articles in Wikipedia. However, these digital counterparts of the pieces are not always linked, and the user could benefit from having assistance for touring the full or a concrete subset of the collection.

(ii) Itinerant exhibitions, designed to be performed at the facilities of any building or open space, presenting a collection focused in diverse kinds of artistic, historical, cultural manifestations that can be represented no matter the place. The physical points of interest in this kind of tours could be distributed, for example, in different areas of a school, and could be linked to a digital object, such as a Wikipedia article. Augmentations can be offered based on the information provided in the catalogue of the tour and also by retrieving public comments to encourage discussion and the development of a critical attitude in the visitors.

(iii) Temporary exhibitions, where a single art gallery is in continuous change, present different collections of artistic works usually for periods of less than two months. Augmentations can also be performed over Wikipedia articles (if the exhibition does not have its own Web site), adding quotations from the authors, comments by the visitors, and touring assistance along the way.

(iv) Outdoor tours through the most significant heritage sites in a city, as the one presented by some city halls, allow tourists to read about the best places for experience and appreciate the rich architectural history of the city. Augmentations here can be performed over each PoI Web page in the site, providing the touring assistance and also retrieving public review content through the some Trip Advisor Content API (https://developer-tripadvisor.com/content-api/).

3.2. Targeted User Roles. Before deepening into details, it is worth to make a clear distinction between two end user roles that are considered by our approach. In this research, we distinguish between

(i) producers, who create their own applications by using an EUD programming environment,

(ii) consumers, the ones who install and execute applications (created by a developer or a producer).

In order to understand which are the requirements for being a producer, we have surveyed the literature to identify common learning barriers on end user programming systems. According to [34] there are six learning barriers (design, selection, coordination, use, understanding, and information) in the context of end user programming systems. In this regard, and following some authors recommendations and findings [32, 35, 36], we defined a producer's live programming environment, mainly based on prebuilt widgets and forms. In this way, from our point of view, producers require to know what kind of mobile experience they want to have on a target Web site, and which context information (and the associated sensors) they want to use. They need to be capable of filling forms, placing widgets by drag and drop and, in order to let producers to understand their creation, they should be able to manage the preview of the mobile augmentations in any moment to check if the augmentations are working as expected.

For designing our solution, we took into account the main variability issues listed in [35] (dynamic context model, sensor support, and domain-specific adaptation) and adopted a form-based assistant for the in-situ authoring process, as in [32], as their work demonstrated that it is feasible for end users with no programming skills (they conducted their experiment with kids). While they allow students to incrementally compose a story, so we do with the values of interest for an application (e.g., concrete coordinates or lux levels) and the augmentation units (e.g., to display a title, a PoI information, and some walking links). They also offer a mechanism for automatically retrieving the user's current position; we also contemplate it for a wide range of context types (e.g., location, lux, dB, and orientation) but we did not make it mandatory to use it. Due to the domain of the applications we present in this paper (case study and experiment set-up), we also took into account the «display of anchors» and «link traversal support» explained in [36].

3.3. Mobile Web Augmentation for End Users. The main idea behind our approach comprises, at least, a producer creating a mobile Web application with a domain-specific authoring tool and a consumer using such application through a mobile browser weaver. There are other possibilities such as developers extending or instantiating the framework and improving the end user tool, but we will concentrate on the former for the sake of clarity and conciseness.

We provide three tools to support this process: (1) MoWA authoring, an extension to Firefox for Android, supporting the creation of mobile Web applications at client-side; (2) MoWA crowd, an online platform supporting crowdsourcing and sharing services for MoWA artefacts; (3) MoWA weaver, another extension to Firefox for Android supporting the management and execution of MoWA applications.

As shown in Figure 4, a producer can interact with MoWA authoring, basically, for creating mobile Web experiences from existing Web applications. He can create such applications based on his own requirements, but he can also do so upon the request of consumers, materialized in an app request posted in MoWA crowd. In both cases, creating an application may involve the extension of an existing one; to do so, the user must select an existing application, either in the local storage or in the remote repository. For the second option, he should search and download one of his applications in the repository, or a public one created by another user. The user may follow a series of steps that we explain later in Section 4.1 and, at the end of the process, it involves automatically saving a copy of the created application in the local storage of the browser extension, so it can be further edited or just executed in the producer's EUD environment. Finally, the producer has the chance of sharing his application in MoWA crowd. He can do so with the aim of sharing a new user experience, or for completing the app request process started in the MoWA crowd context.

A consumer can install MoWA weaver in his mobile browser and start experiencing the augmented mobile Web. To do so, he may execute MoWA applications that may be retrieved from the MoWA crowd repository or the ones he already has in his local storage (in the case, he already has imported any or if the same user plays both roles). If the consumer cannot find an application facing his requirements in the public repository, he can start a new app request in MoWA crowd. Finally, a consumer can also manage their local applications. For example, he can edit or delete them, change the order of execution in relation to other applications, and so on.

4. Supporting the Authoring Process

In this section, we present the architecture of the tool supporting our approach, the authoring process, and the involved steps, and finally, we use our authoring tool for solving and presenting a case study, by following each step of the mentioned process.

4.1. The Authoring Control Flow. For supporting the authoring process of a mobile Web application (the «create Mobile Web app» use case in Figure 4) we started by defining it as a series of stages, represented as activities in Figure 5.

These stages are as follows.

(1) Setting the application base data: here, the tool asks the user the base information for the application, like a name, a namespace, a filename, etc. The builder is in charge of getting the values that the end user should provide: in this case, just the application name. The rest of the required data is transparent for the end user.

(2) Selecting the augmentation strategy. Two alternatives are offered to the end user to specify what to do when a context change is notified in the application. A first strategy augments any Web page the end user is currently navigating, whether the URL matches the ones defined for the application. A second strategy checks if the sensed context value is contained in the list of values of interest for the application and, if so, it opens a concrete URL to augment and then applies the augmentations.

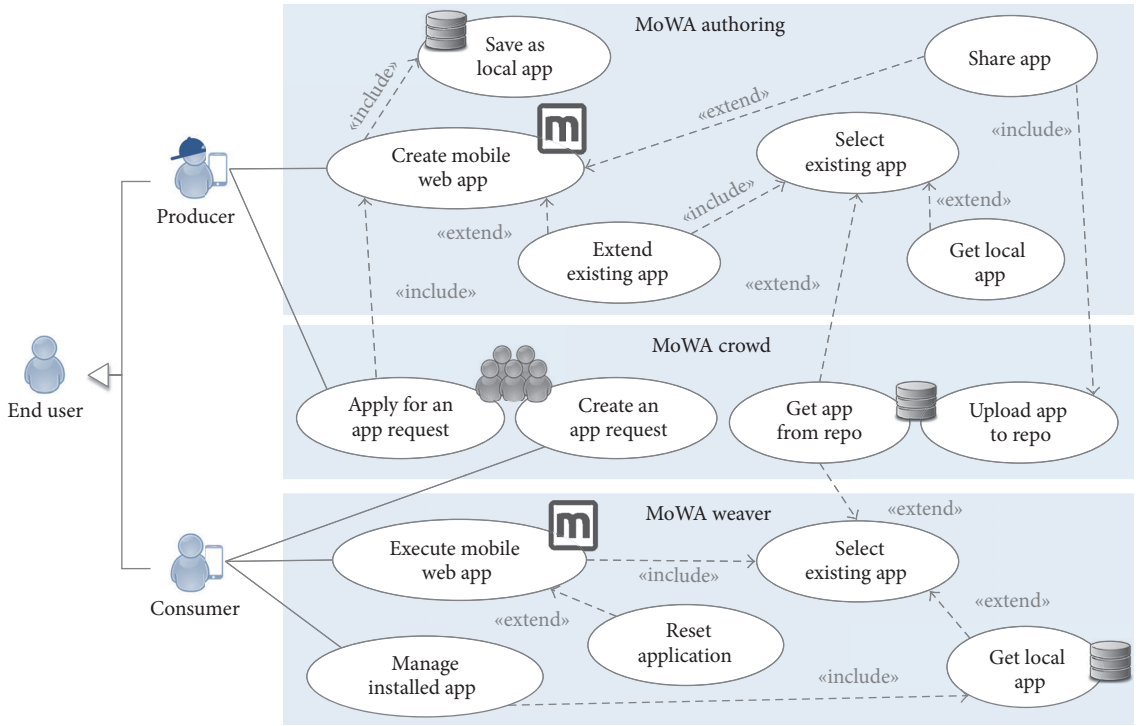

Figure 4: A crowdsourcing platform for mobile Web augmentation.

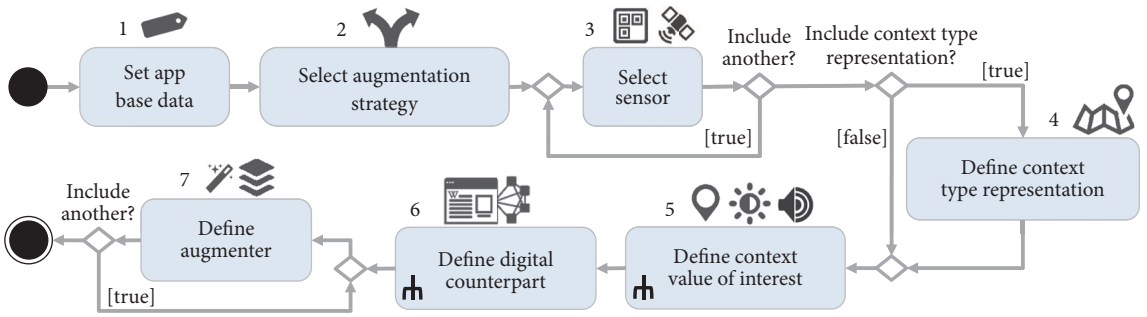

Figure 5: The supported process by the MoWA authoring tool.

(3) Selecting the sensor(s): then, for each selected context type, the user is offered with a set of available sensors for listening to their changes. For example, both a GPS sensor and a QR sensor could be on charge of sensing the user's location; a lux sensor can notify about changes in the level of light perceived by the mobile device; a dB sensor can track changes in the noise level in the ambient.

(4) Defining a context type representation: as will be further explained in Section 4.2.2, some augmenters may need to use a visual representation of the context values. To do so, they need to be placed in the context of a containing space, like a map for a marker (e.g., related to a QR-based location) or a scale for a lux point (matching a lux value). As some augmenters may not use this representation, this is an optional step.

(5) Defining context values of interest for the application: every sensor notifies changes of a context type to the subscribed augmenters (that will be defined in a following stage), but in order to support such augmenters to use the sensed context values, it is required that the application knows which values are representative for its purposes. For example, as the end user is building a pure mobile application, the application needs essentially to know a set of locations for triggering the augmentations. Such locations are represented as points of interest and they have some optional properties that can be specified, for example, external content related to every point of interest or the specification of a navigability order through the set of PoIs. In the same way, an application subscribed to a light sensor needs to know what are the bounding values that represent a significant change in the light level,

and optionally it could define some associated data to every light level, as a default description of it. Therefore, every application is capable of showing a set of configurations according to the selected sensors.

(6) Defining digital counterparts: every context value has associated a digital counterpart that knows a URL of a Web page to augment, and a set of augmenters. Moreover, a context value may have some user-defined properties. For instance, imagine an end user is creating a tour for La Plata city, a digital counterpart may indicate that the URL to open and augment when the consumer is very close to the coordinates «−34.920609, −57.954393» is https://es.m.wikipedia.org/wiki/Plaza_Moreno (2) and an instance of a *TitleSection* will be executed. Such augmenter will be defined in the next step, but it can consume information defined for the digital counterpart, for instance, a name or some description the user manually entered or has selected from an external Web page.

(7) Defining augmenters for a digital counterpart: at this stage in the process, our tool asks the user to define a set of augmenters for each Web page to augment. We provide the producer with a set of augmenters according to the sensors he has chosen, and the context type representations he has defined; we suggest them based on a simple tagging mechanism, defined as metadata in the augmenter's class file. Augmenters are defined in the context of a digital counterpart, and the producer can add as many as he wants. Each augmenter needs some input values to be properly executed. We provide the end user with three alternatives for defining such parameters' values: (1) he can reuse the defined data related to a concrete context value, for example, the PoI name, by accessing the data model and selecting a property; (2) he can manually input such data in the form; or (3) he can use an assistant for retrieving external content.

4.2. The Underlying Architecture

4.2.1. The Overall Picture. The software architecture supporting the approach is presented in Figure 6. Two mobile browser-based tools are shown at both sides of the figure; they are in charge of enabling the authoring process and executing the resultant applications. There are also server-side components supporting application sharing and some crowdsourcing services, so consumers can demand for new solutions and producers and developers can meet such need.

At the right of Figure 6, you can see the customer's device, which has installed the MoWA weaver extension. He can import already defined applications from our repository and use it for visiting existing Web sites with augmented features (e.g., he can download applications solving the aforementioned scenarios in Section 3.1). Such applications can be created in two ways: by client-side scripting (for developers) or by using the authoring tool (for producers). The first ones are downloaded from the repository as JavaScript files, and our weaver has a specialized interpreter in charge of loading the required classes for each application and instantiating them in the context of a set of concrete Web pages [11]. In the second case, another engine interprets authored applications, the ones created with MoWA authoring, that are materialized as an XML file, and specified according to the XSLTForms data model. This engine is in charge of interpreting such specification, instantiating the proper classes with the values in the data model, and then also cloning such objects in the context of the original Web page to augment.

Then, in the middle, we have MoWA crowd. It is an online platform supporting crowdsourcing tasks management and a repository of MoWA artefacts, among them, a collection of applications ready to be installed and used by *consumers*. Concerning the source of such applications, they can be both: the ones created by developers, as discussed in [11], or the ones created by producers with the MoWA authoring tool.

At left of the figure, the MoWA authoring tool architecture is presented. It is installed as an extension of the mobile browser on the producer's device. It comprises a set of specialized classes that have a reference to the base classes that compose a MoWA application and extend their functionalities with the ability of asking the end users for the required parameters through a form-based assisted process, in order to instantiate such classes with the required values. For every component of a MoWA application, we have a builder supporting a part of the authoring process by helping end users to configure the specific instances to run properly.

There is a main builder, the one in charge of orchestrating the full authoring process: the Mobile Application Builder. It knows all the builders in charge of the application's construction process and their dependency to execute the application while it is being authored. The order matters; consider an augmenter dependent on the user's position. It cannot be properly set up if, at least, a Web page has not been defined and if a sensor has not been selected. Therefore, it is important not to allow customers classes to access those subsystems directly. This way, the Mobile Application Builder provides a unified interface and delegates the configuration of the more specialized application's components to other builders in the subsystem, in a specific order.

Builders might also depend on some values of the user's context for configuring the underlying application's component. For example, consider a producer walking a city and building a tour with MoWA authoring; he may want to retrieve his current position for creating a marker in a map or detecting noisy areas in the city for defining an augmenter that increases the volume of a video. If the user is building an indoor tour based on existing QR codes in the building, he needs the tool to be capable of decoding those QR codes in order to associate the data with a physical position. And if the user selects a context-dependent augmenter for his application, it also needs to be subscribed, somehow, to the changes in such context type. This depicts how the builders supporting the in-situ authoring process also depend on the MoWA sensors to be notified when their values change; the application's components are not the only ones that need to be context-aware.

Our authoring tool provides end user with the capability of reusing existing content but not just taking it as a base

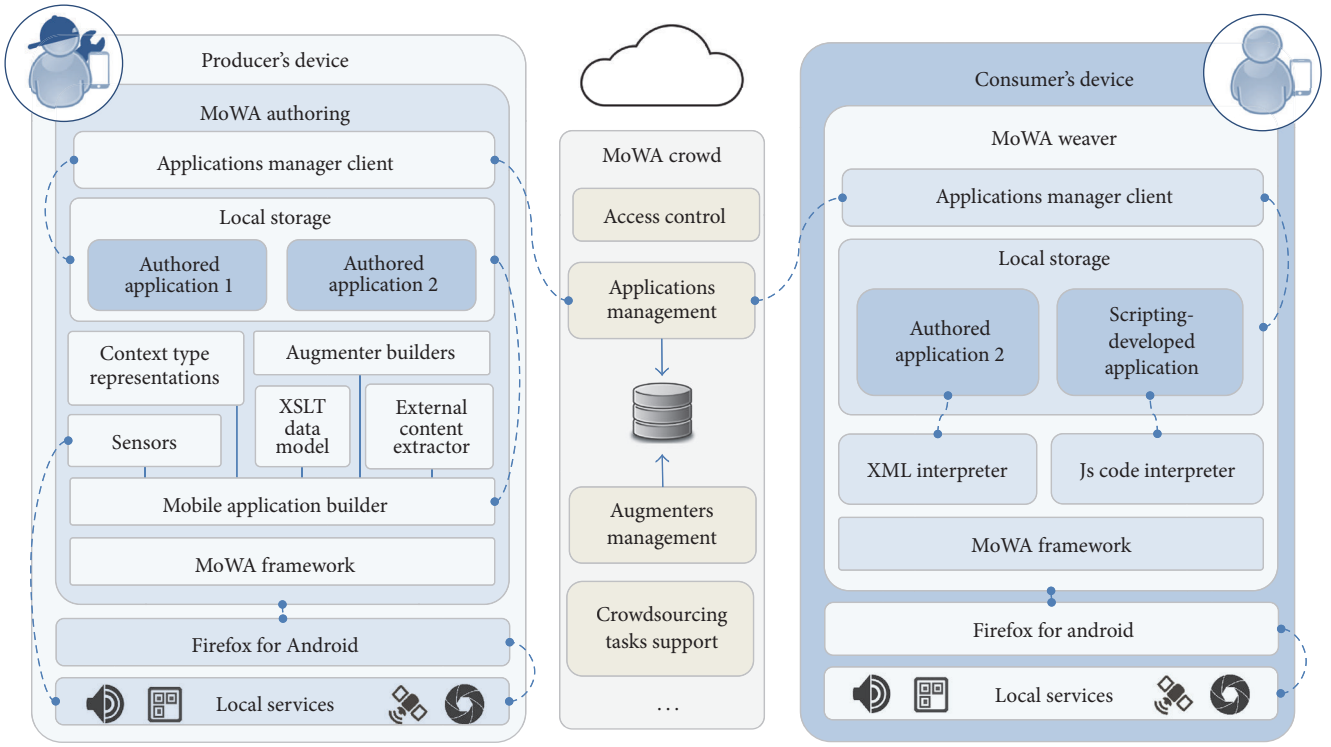

Figure 6: High-level representation of the MoWA platform architecture.

for the augmentations; it also allows him to extract it from external Web pages, usually third-party ones, to be injected into a new context. For example, he can take the actors' profile at IMDB as target Web pages and augment them with a carousel of related trailers from YouTube videos when the device is in landscape orientation. The content extraction is responsibility of a common component, available for every builder, named External Content Extractor. Such component is instantiated in the privileged context of the browser extension, so it makes it possible to append extra behaviour in any Web page, enabling user interactions for selecting their DOM elements of interest. This allows the manipulation of DOM elements to obtain their positioning data in the DOM (e.g., the XPath) and dynamically consume their content, even from external contexts (other Web pages that do not share the same domain name).

4.2.2. The Authoring Tool. For deepening into details about how we support the authoring process, it is appropriate to present first the expected structure of a MoWA application. As shown in Figure 7, a MoWA application knows how to be subscribed to the full range of existing sensors. When some subscription message of the application is called, let us say *suscribeToQrSensor(sensor)*, it asks the received sensor to add herself as an observer [27] of the sensor, and it also instantiates and keeps a reference to a concrete subclass of *ContextValuesManager*. This way, when a change in the context is perceived by the sensor, it sends an update message to all its listeners. A change in a sensor's value may be manually or automatically triggered; it depends on the implementation of the sensor. For instance, a *QrLocationsManager* will be manual, because the user needs to scan a QR code, but a *GpsBasedSensor* can automatically watch the changes by using the browser's API. If the application is subscribed to a *QrBasedLocSensor*, when a change is detected, the latter calls the application's *updateQrLocation(sensor)* message with a reference to itself, so the subscribed applications can ask the sensor for the current value. When the latter happens, the application delegates the augmentation process to a concrete strategy [27] *(AugmentationStrategy)*, by passing the reference of the received sensor. We currently support two kinds of strategies:

(1) *MatchPageAndAugmentStrategy* that simply checks if the current Web page matches a series of URL defined as augmentable and executes the augmenters;

(2) *OpenAndAugmentStrategy* that will run the augmentation whether the sensed value matches one of the values of interest defined in the MoWA application; to do so, it asks to the concrete application's *ContextValuesManager* (e.g., the *QrLocationsManager*) to compare such a value against the ones it contains, and if it matches, the proper URL is loaded in the browser by adding it to its associated *DigitalCounterpart*. The last one may also have a collection of custom properties *(DCPropery)* defined by a developer or a producer, but it contains for sure a set of augmenters (e.g., a *Panel*, a *TextAugmenter*, a *WalkingLinksAugmenter*) to be executed after the Web page is loaded. The case study in Section 4.3 and the evaluation requirements in Section 5 are based on the last strategy.

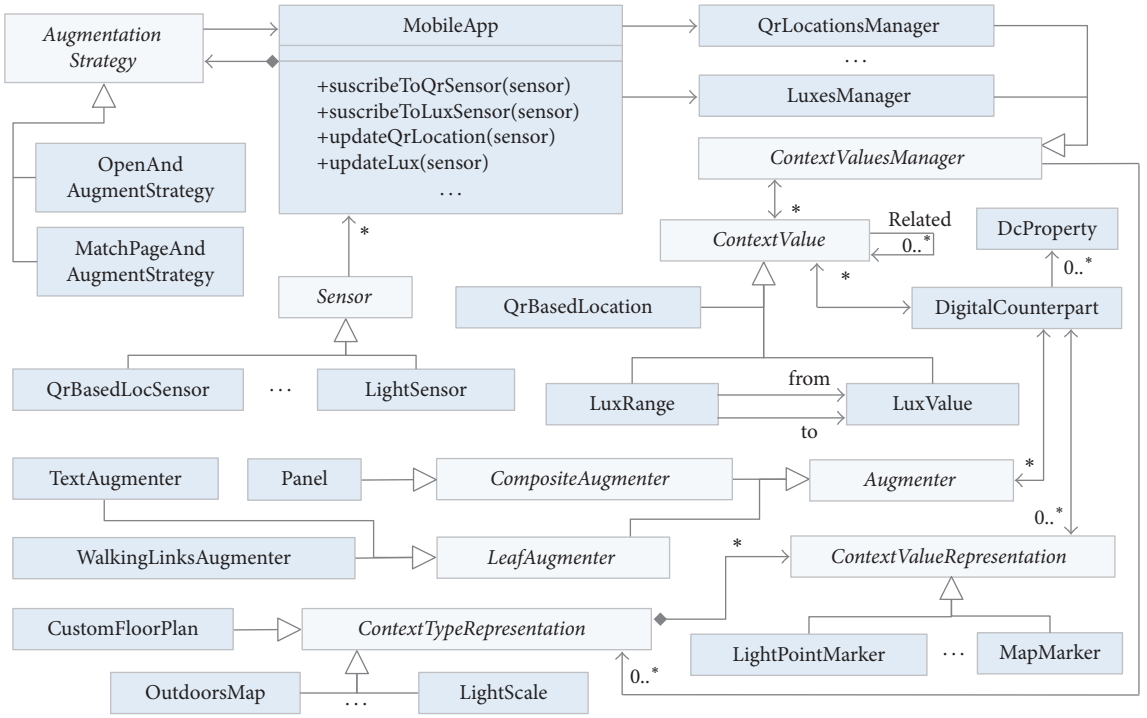

Figure 7: A MoWA application expected as a result of the authoring process.

The *DigitalCounterpart* may also have some a *ContextValueRepresentation*, and that implies that the *ContextValuesManager* has a *ContextTypeRepresentation*. Concrete subclasses of the first one implement a visual representation of a space (e.g., a scale, a map) that will contain the representation of a set of context values (e.g., a visual point in a scale, a marker in a map). The reason of these components being nonmandatory is that some augmenters may need such information for implementing their features, so they should be configured for that purpose. For instance, the *TextAugmenter* and the *Panel* augmenters do not need such information; they can show a text or contain leaf augmenters (see composite pattern in [27]), respectively, with no need of displaying a *MapMarker* in a *CustomFloorPlan* or a *LightPointMarker* in a *LightScale*. However, a *WalkingLinksAugmenter* is in charge of displaying the following PoI of a tour in a textual list, but also in a map, so the user can understand where to go next. In this case, the augmenter can ask the *CustomFloorPlan* to display the current user's position and the following location, according to the registered PoI.

Regarding augmenters, in addition to those already mentioned we implemented the following ones in order to verify the technical feasibility of the examples mentioned throughout the article.

(i) *TitleSection*: in the same way as the *TextAugmenter*, it shows a div with a specific text but with a special style: within a beige container, center-aligned, and with a 1.5 em size bold font. This text can be a fixed value as well as a value extracted dynamically in relation to a text already available in the Web (e.g., from an external site).

(ii) *WalkToTarget*: it is possible to configure this augmenter with a series of locations to be displayed as markers in a map. A route is traced starting at the user's current location and ending at one or more specified locations. Unlike the *WalkingLinksAugmenter*, this does not impose an order while traversing the points of interest; it simply shows the alternative targets and how to reach them.

(iii) *ImageAugmenter*: it shows an image at a specific position of the DOM. Such image can be defined in terms of a Base64 encoded string or from the URL of an existing image on the Web.

(iv) *PoiAugmenter*: it adds a div at a specific position in the DOM with a description and a picture of a PoI. It is composed by a *TextAugmenter* and an *ImageAugmenter*.

(v) *VisitedLocations*: based on a list of locations, it adds a div with a list of locations that are already visited by the user. Locations may have associated a descriptive name and they are marked as «visited» if the user was physically sensed near such location.

(vi) *UnvisitedLocations*: in a similar way to that *VisitedLocations*, this augmenter adds a list of locations that were not visited by the user.

(vii) *YoutubeVolumeLeveler*: it has no visual representation on the Web page, but it increases or decreases the volume of a video in a Web page according to the perceived level of noise in the user's surroundings.

(viii) *LightsCameraAction*: it starts playing or resumes a concrete video in a Web page when a change in the perceived level of luminance in the user's surroundings is up to 25 luxes in less than one second, and the last sensed value is under 5. If the change is the opposite, from a dark environment to a brighter one, it pauses the video.

(ix) *DontLetMeWaste*: it disables the interactions with all the audios and videos found in the current Web page while the battery level is under 25%.

(x) *RelatedTweets*: it adds a div containing a set of tweets matching the textContent (https://developer.mozilla.org/en/docs/Web/API/Node/textContent) attribute of any DOM element set as target.

(xi) Based on the definition of search engines UI-APIs [18]: in this sense, it is possible to transparently perform a search through the UI of a Web page and to obtain a set of results. This involves being aware of the structure of the DOM in order to identify some interaction elements (as the input and search buttons) and the returned elements. Under this category of augmenters, we have the *RelatedGoogleImages* showing a set of similar images to the one set as target and the *SimilarAmazonProduct* and *SimilarBookInStore* adding a list of similar products and books, respectively, according to the textContent attribute of the target DOM element.

(xii) *ListElementsSorter*: it orders a set of DOM elements in a list according to the textContent value of any of its child elements (that should be present in all the elements of the list). For instance, taking the list of movies in http://www.cinemacity.com.ar/cartelera.aspx you can order the elements by removing and reinserting the elements according to their titles or their schedule times. However, it is intended only for sorting items present in a single Web page and which do not require paging.

When it comes to the sensors, Figure 7 shows just two of them and the required components supporting their proper kind of context types, but it is necessary to clarify that the application could be also subscribed to other sensors and adapt itself according to their related context values.

Based on the structure of a MoWA application, we have created a set of classes that have the responsibility of allowing the end user to configure an application component; we call them builders. In order to show the relations between the application classes and their builders, Figure 8 presents some of the required builders for instantiating and configuring a very simple application, subscribed only to a single sensor. Nevertheless, it is possible to dynamically contemplate the use of other builders, as we explain later. In the figure, builders are the darker classes whose name ends with «Builder». Each of them must be a subclass of *ConfigurableComponent*, but for the sake of space, we do not show the superclass in the diagram. A *ConfigurableComponent* states that every builder should be capable of at least carrying out a fraction of the authoring process and persisting and validating the data entered by the end user. For this purpose all their subclasses should implement a *configure(window)* method, which is in charge of generating and loading a specialized form with controls for filling the required data, so the user can configure the component (e.g., an augmenter). For instance, the *PoiAugmenter* is in charge of adding a div at a concrete position in the DOM with a textual description and a picture of a PoI. If the user chooses such augmenter for his application, the *PoiAugmenterBuilder* will ask him to provide a text and an image for the augmenter to use. Both of them are required, so the builder should validate their values before enabling the execution of the augmenter (e.g., to preview the results) and then persist the data for later usage. Such builder is, then, in charge of achieving a part of the augmentation process.

In our model, each builder is responsible for defining the backbone of the part of the authoring process it is in charge. For example, the *MobileAppBuilder* implements the main algorithm process that is further explained in Section 4.1. At this point, it is enough to understand that the *MobileAppBuilder* is the most important builder; it is in charge of orchestrating the full process, configuring its own variables and collaborators that do not need a concrete configuration (e.g., sensors, augmentation strategy) and delegating the remaining to more specialized builders. For instance, the mobile app builder presents the user with a form asking for the application's name, the two possible augmentation strategies, and a list of available sensors. There is no much to configure regarding the augmentation strategy, but when the user selects a sensor, let us say the *QRBasedLocSensor*, the application is subscribed to the sensor and an instance of *QrLocationsManager* is now available for the application and the *QrLocationsManagerBuilder* for the *MobileAppBuilder*. The application now can ask such collaborator (and any other *ContextValuesManager*) to be configured, and it will do the same as the *MobileAppBuilder*, configuring the properties and delegating what requires more specialized behaviour to its related builders.

As enabling user interaction with the authored artefacts may entail errors at execution time, MoWA application components must support mechanisms for displaying extra messages to the end user in case an authored object was misconfigured, for example, if a map contains a wrong coordinate value, if the link of the image selected as a 2D floor plane is broken, or if an augmenter lacks a required parameter value. Achieving this requires that any *ConfigurableComponent* may be capable of checking the arguments it is receiving and be capable of displaying proper and useful messages to the *producer*. The *ConfigurableComponent* has defined an abstract method to that end: the *checkInputParameters*, which is executed by each Builder before saving the configuration of a component (while in authoring mode), and it is also useful for the component itself at execution time (while in authoring or regular execution mode with the weaver).

As already mentioned, the responsibility of a builder is to request the user, through a specialized form, the data necessary for the execution of the component that is building (e.g., an augmenter). Developers can manipulate the DOM (of the window that the configure method receives as a parameter)

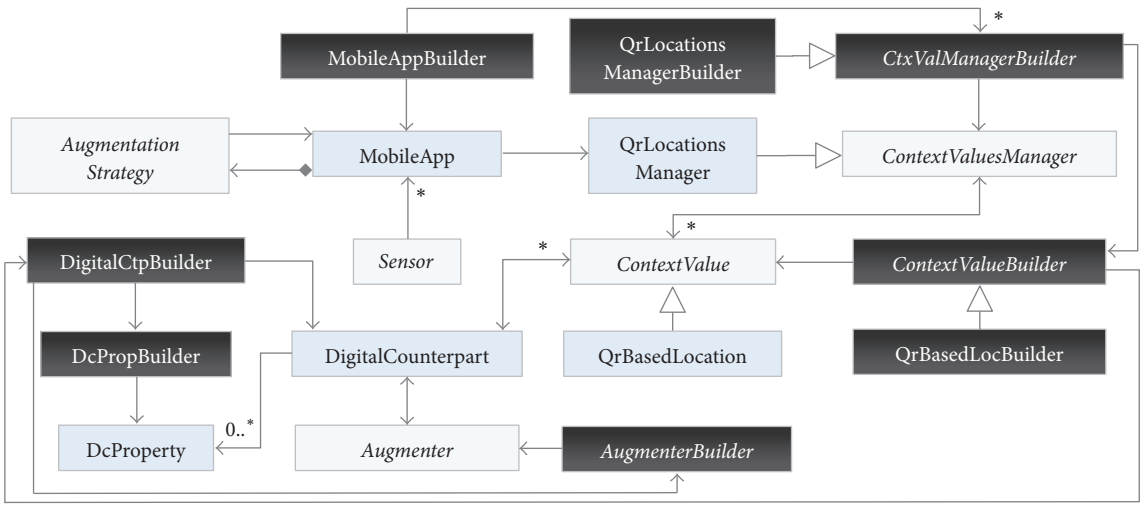

Figure 8: Structure supporting the authoring process of a MoWA application.

and change it according to their requirements, and as this code is executed in a privileged context (as an extension) there are no restrictions about which data to request or how to do it. For example, it is possible to contemplate manual mechanisms, where the user explicitly enters the information by typing in an *input* form control, but also more sophisticated ones that autocomplete the expected data, like suggesting properties already available in his workspace (as the properties of a *DigitalCounterpart*). An example of a more specialized form is the one for configuring context values of interest from a *CustomFloorPlan*; it presents a map where the user can add markers by holding on it. However, as we wanted to facilitate the in-situ development modality, we also added a button triggering a mechanism for automatically adding a marker at the user's current position.

4.2.3. Technical Details. Both client applications (the authoring tool and the weaver in Figure 6) are Firefox for Android extensions, running on version 41 of such mobile Web browser. However, they have successfully run in precedent versions, as discussed in Section 5.2. Both applications share a common base; first, the same mobile Web browser that allows them to access the underlying capabilities of the device. For instance, it exposes multiple APIs for accessing the battery, the camera, the geolocation, the orientation, and so on. This means that developers can access values of the user's context and take advantage of them in a direct way, for example, by using the navigator.geolocation.getCurrentPosition (https://developer.mozilla.org/en-US/docs/Web/API/Geolocation/Using_geolocation), but also by using the APIS for processing data and tracking other context types. Example of this is accessing the camera, asking the user to point a QR code, taking a picture, and analysing it for decoding a QR code, which contains information related to the user's position. Second, they share the same base framework for instantiating, extending, and executing applications: the MoWA framework. Finally, they also share some generic components for managing the user's installed applications in a local storage, but also for connecting to the server and upload, update, or download applications.

Concerning the supported interactions by the builders, broadly speaking, we introduced diverse modalities, like the ones required for managing a floor plan representation. In such a case, we used Leaflet Maps (http://leafletjs.com/) in conjunction with OpenStreetMap (https://www.openstreetmap.org), which provides a complete API for interacting with maps from both mobile and desktop environments in a light way. Thus, we support two interaction modalities with maps: data visualization and edition mode. In the first case, augmenters can use it for displaying the positions of the user and some PoIs, while in the second case it allows end users through a concrete builder to create, move, delete, and connect markers in a map and also to configure the map's zoom through touch events.

Regarding the persistence between browsing sessions, the data entered by the user is kept through the mechanisms provided by the XSLTForms engine [37] and its underlying data model. XSLTForms is a client-side implementation of XForms, whose benefits were extended by adding subforms management that allows us to easily support the authoring process in a wizard mode. The input values in XSLTForms are automatically bound to a data model that, at the end of the process, we can export and use as the specification of a MoWA authored application.

Finally, we also adopted a definition of language bundles for each building artefact for internationalization purposes, so the MoWA engine is able to provide the authoring experience according to the user preferences or the browser's language. We provide language bundles of Spanish, French, and English, and this allowed us to invite a broad spectrum of participants to our experiment; in fact, we conducted the experiment in the facilities of our laboratory in Argentina, and we had two participants in the experiment whose mother tongue was not Spanish. Nevertheless, they opted to create their application in Spanish.

4.3. A Case Study. Below, we present a case study matching every step of the aforementioned process in the domain of a mobile hypermedia tour in a museum; the process is similar in other domains and fields. We chose the hypermedia tour domain since the level of complexity required to build applications in this domain is much higher than the rest of the supported applications (e.g., those indicated in Section 3.1). In this sense, the end user must be able not only to configure sensors and augmenters to provide a Web application with context-based features, but also to configure navigational features at two levels, digitally and physically. This requires the end user to set up a space of representation, some points of interest with related information, and the walking links (a regular anchor supports a walking link, but triggering it expresses the user intention to walk to a target location in the real world; it involves the physical movement of the user, as explained in [38]) between them.

The chosen scenario for this section was also used for the experiment set-up (see Section 5.3). At this point, it is only necessary to say that the museum has a collection of pieces that can be visited in a certain order, and each of them is associated with a QR code that redirects to a Wikipedia page. This museum also has its own Web site that provides information about the tour and each of its pieces, or PoIs, but this information is not associated with the physical objects; the only digital counterpart of such physical objects is the Wikipedia articles.

In this context, a museum guide may want to create an application that augments the Wikipedia articles with the information in the museum's official Web site. To do so, he can carry out the authoring process, which contemplates the following stages.

(1) Setting the application base data: as shown in Figure 9, once the authoring mode (S_1) and the full-screen mode are enabled, the producer chooses to create a new application (S_2) and he sets a name for the application (S_3).

(2) Selecting the augmentation strategy: then, the user configures the augmentation with one of the augmentation strategies, in this case, as shown in «S_4», the *OpenAndAugmentStrategy*.

(3) Selecting the context sensor(s): in «$S_{5,6}$», the user chooses among the available sensors accessible through his browser (according to the mobile capabilities, not all of them can be instantiated). Such sensors are displayed in the meaning of the context types they notify (e.g., a location, a light level). As he has chosen the location and there are two available sensors for such a context type, he chooses, in «S_7», between two related sensors observing changes in location: GPS and QR-based sensors.

(4) Defining a context type representation: in this step, the end user just chooses if he wants to define a context type representation or not. If he takes this path, the tool offers him in «S_8» the possibility of choosing among the available alternatives (e.g., an outdoor map, a custom floor plan). He selects a custom 2D floor plan that should be configured with a base image and a name, as in «S_9» of Figure 10.

(5) Defining context values of interest for the application: as he has selected a QR-based location sensor and a floor plan as representation, now he should define locations by creating markers in a map. As we can see in Figure 10, the tenth step (S_{10}) requires the producer to define some locations as the context values of interest. To do so, he holds on the map and, in response, a marker is created in such position and a QR code should be scanned (S_{11}), in order to associate it as a *QrBasedLocation*.

(6) Defining a digital counterpart: after that, the user is automatically directed to the definition of a digital counterpart, which involves writing a name, a URL (it is also possible to scan a QR code to get it), and optionally defining some properties. Figure 11 draws such step in «S_{12}», and some external content is extracted and defined as properties of the PoI in «$S_{13,14,15}$». In «S_{15}», the user defined two properties: «poi-desc» and «poi-pic», which will be used by an augmenter in future steps (e.g., S_{19} in Figure 12). Once these parameters are defined, the producer is taken back to the «S_{13}» screen. When a second digital counterpart is defined, the user can create paths between two points by holding the markers and choosing «set as origin» or «set as target» from a context menu. «S_{19}» is the resultant view after all the digital counterparts were defined and connected.

(7) Defining augmenters: as shown in Figure 12, the producer can add multiple augmenters for each digital counterpart. In «S_{17}», the tool automatically loaded the URL previously defined for the digital counterpart in a frame. There is a carousel, at the bottom, that the user can swipe to navigate the available augmenters. By clicking one of them, he can insert an augmenter builder on the context of the loaded Web page. It will be inserted as a floating element at the middle-center of the screen, which should be positioned in the DOM by drag and drop as in «$S_{18,19}$». This element is always placed in relation to another existing DOM element, which serves as a base for generating an XPath for the augmenter's content. If the user places the augmenter in relation to an element that, in the time, is removed from the DOM (e.g., the Web page is dynamically rendered), the augmenter needs to be repositioned. When the tool fails in evaluating an augmenter's XPath, it is positioned as a floating element on the default position (middle-center of the screen), so the user can reposition it. After positioning it, the user must configure the augmenter's parameters shown in «$S_{20,21}$». He can write the input or he can link the configurable parameter with the application given name, as in «S_{21}». The user is taken back to the Web page to augment and he can preview the configured augmenter, as in «S_{22}». Then, he must repeat the process for each augmenter he wants to contemplate in his augmentation; he chooses the augmenter, places it

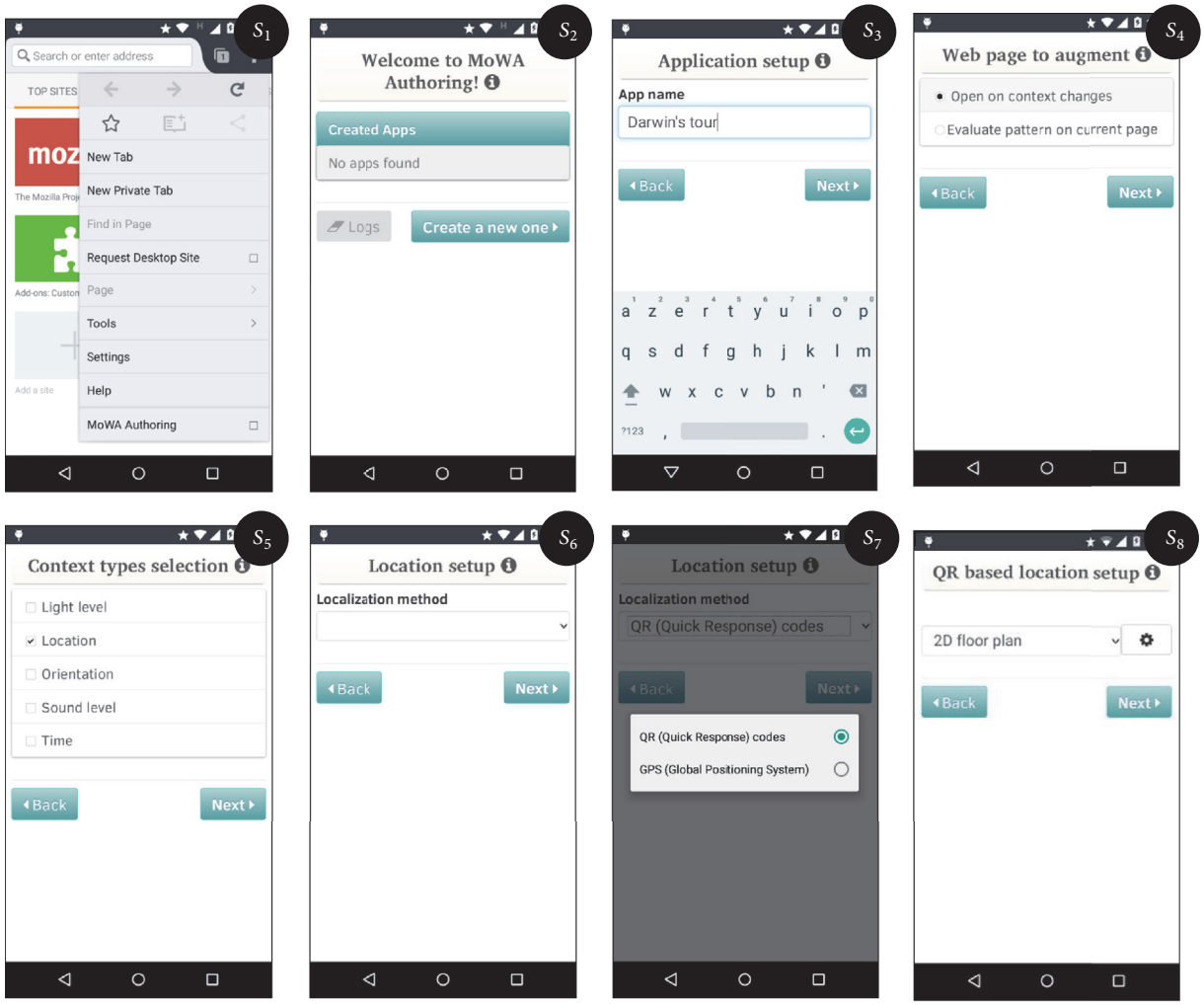

Figure 9: Setting up the application base data and sensors.

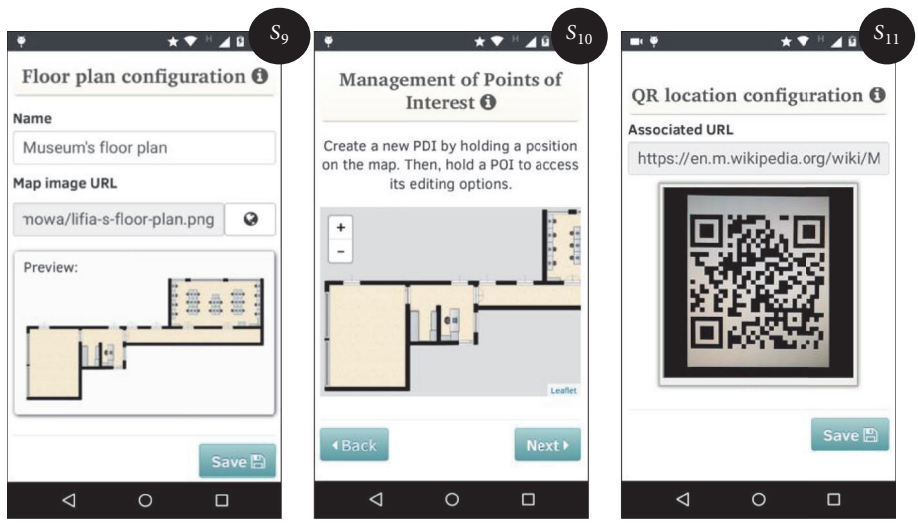

Figure 10: Configuring a context type representation and defining values of interest.

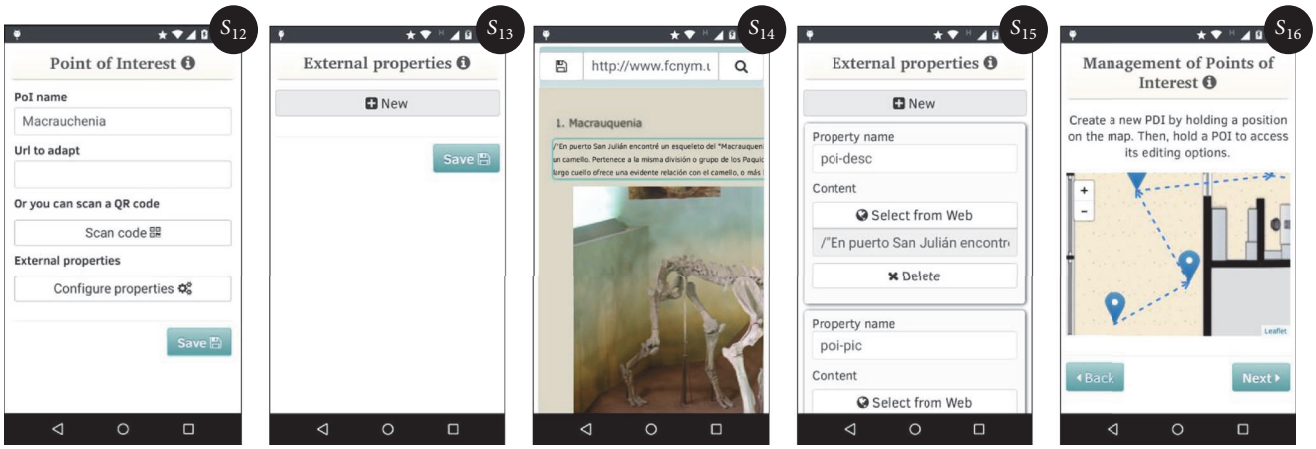

Figure 11: Configuring digital counterparts.

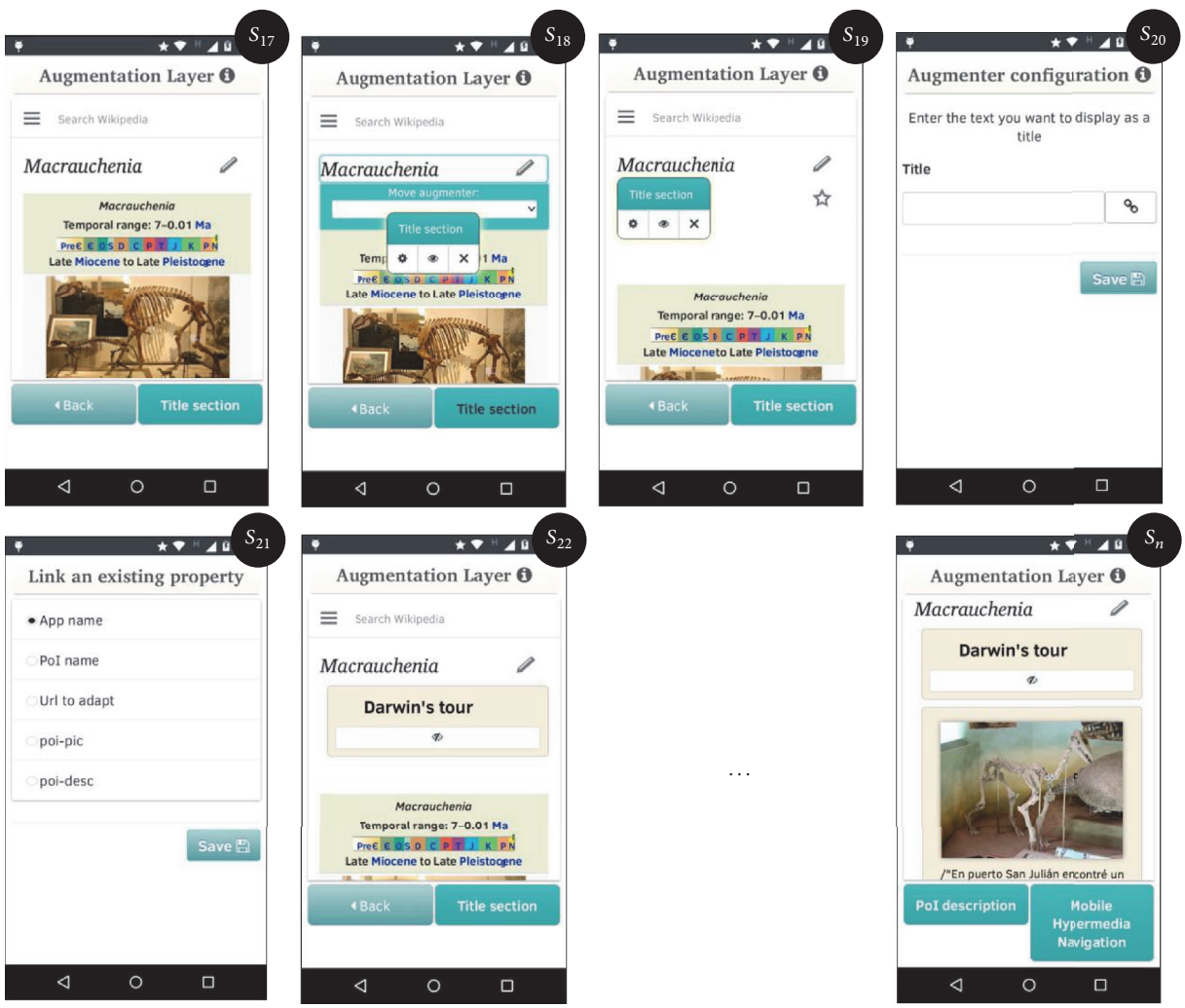

Figure 12: Creating and configuring the augmentation.

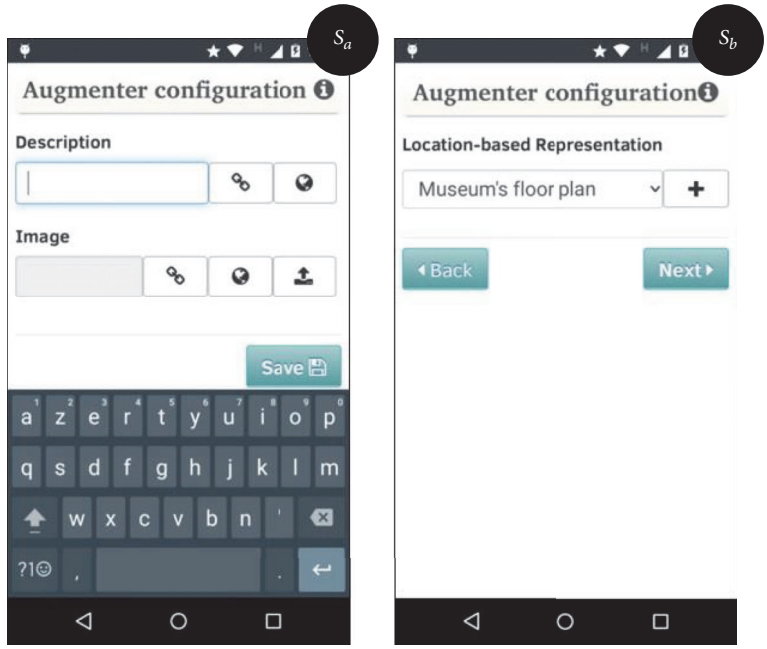

Figure 13: Remaining augmenters' configuration forms.

in the DOM, and configures their specific parameters through specific forms and, finally, he may execute the augmenter's previsualization. As each subclass of *AugmenterBuilder* implements its own configuration process in order to collect the required data for the augmenter's execution, the user will be presented with different and specialized forms while configuring each augmenter. This way, steps «S_{20}» and «S_{21}» are different for configuring other augmenters; Figure 13 shows the configuration form for a *PoiAugmenter* in «S_a» and for a *WalkingLinksAugmenter* in «S_b». At the end of this stage, the user can preview all the augmenters together and appreciate the layer's result, as in «S_n». Below, we list all the augmenters the user adds and configures in this concrete case study:

(a) *TitleSection* for displaying a title at the top of the augmentation

(b) *PoiAugmenter* to present the specific information (a textual description and a picture) of the PoI with external information to Wikipedia, extracted from the museum's official Web site

(c) *WalkingLinksAugmenter* to assist the user while walking to the museum (taking the tour, not building it), by showing a map with a path between the user's current position and the next PoI to visit in the tour

(d) *VisitedLocations* to display a list of PoIs that were not yet visited by the user

(e) *UnvisitedLocations* to present the PoIs that the users has visited

As a result of the process, a MoWA application specification is obtained, and the *producer* can share it with *consumers*. The MoWA weaver could be used for running such applications later. Let us say that the producer shares the script with Julio, his cousin, who is visiting the museum for the first time. Julio installs the authored application by using MoWA weaver and, once in the museum, he scans the first QR code he finds with the QR codes scanner provided by MoWA weaver. Such action will load a Wikipedia article in the browser and, if the code is part of the ones contemplated by the application, the Wikipedia article will be augmented with the extra information taken from the museum's official Web site. However, such information will be adapted according to the sensed context values. For example, if the visited piece is the one defined as the first of the tour, Julio will be presented with information about it and the path he should walk to arrive to the next PoI; otherwise, the augmentation may indicate to him how to walk from his current position to the first PoI. The same will apply to the following PoIs; the tour will assist Julio with information about each piece and the tour's order.

5. Evaluation

In order to validate our approach, we conducted an experiment with 21 participants. We asked real end users to use the MoWA authoring tool for solving a concrete scenario problem in the context of a mobile hypermedia application [22], oriented to the tourism domain. The ultimate goal is to demonstrate the feasibility of the tool to perform adaptation tasks and collect metrics of use of the tool. This study is rather preliminary and do not cover all the dimensions for usability

or UX. Nevertheless, it carries on to a better understanding of the potential of the tool for end users. For that end, we have assessed if any end user using MoWA authoring is capable of building mobile Web augmentation applications.

5.1. Participants. We conducted the experiment with 21 experimental subjects with a very wide range of demographic characteristics. Due to the large number of participants in the session, we chose surveys as collecting data mechanism. We found that 71.43% of the members were males, against a 28.57% of female ones. Their ages range was homogeneous, ranging from 22 to 39 years old, with an average of 26.9. They were of different nationalities: 14 Argentinean citizens, 3 Colombians, 2 French citizens, 2 Venezuelans, and 1 Peruvian. They were mainly ongoing students of different degree careers, representing the 52.38% of the population, preceded by a group of the same education level but different status: 28.58% of them already completed their studies. The remaining were, at equal frequency, people with completed high school level, uncompleted degree career, completed postdegree career, and postdegree studies in course.

As we focused on real end users with a wide range of interests, we searched for volunteers in various contexts, involving people coming from different fields of study. Most of the participants, 11 of them (52.38%), said they belong to the hard sciences, then 4 of them to social sciences, 2 to arts, another 2 to natural/life sciences, and one to economics and another one said he had no specialized studies.

Concerning the technological expertise of the participants, we focused on three aspects that allowed us to infer their level of technological background and accordingly classify them to observe if there is any influence in their performance. Such aspects were related to their programming experience, technological know-how, and the user perceived complexity in a series of related tasks related to our tool. As shown in Table 2, according to these three factors, we categorized them as novice, regular, and expert users.

Regarding the first aspect, we just ask them about their programming skills experience. 47.62% never experienced programming, 42.86% of them program frequently, and a 9.52% occasionally programmed. We considered them as novice users, expert users, and regular users, respectively.

In order to analyse the second factor, we presented them a set of questions based on how frequently they perform certain tasks. We asked about the usage of (1) smartphones, (2) the Android platform, (3) Web browsers in daily life activities, (4) Firefox for Android, (5) browsers' extensions, (6) Firefox for Android extensions, (7) location aware apps, (8) guided tours apps, (9) Web augmentation apps (10) authoring tools, (11) mobile Web forms, (12) QR codes scanning, (13) Web searching, (14) content selection and edition, (15) map markers management, (16) slice gesture for menu navigation, and (17) drag gesture for moving objects. We provided 5 options in each case: daily, weekly, or monthly usage, "a few times" and "never did it." The first option was related to expert users, the second and third ones to regular users, and fourth and fifth ones to inexperienced users. This way, we assigned every answer to an expertise level and took the higher occurrence percentage of them for representing the user expertise level.

Table 2: Technological expertise of the participants.

PS	Programming experience	Technological mastery	Complexity perception	Prevalent category
1	Novice	Expert	Regular	Regular
2	Expert	Expert	Expert	Expert
3	Regular	Novice	Regular	Regular
4	Expert	Expert	Regular	Expert
5	Novice	Novice	Regular	Novice
6	Regular	Expert	Expert	Expert
7	Novice	Novice	Regular	Novice
8	Novice	Regular	Expert	Regular
9	Novice	Expert	Expert	Expert
10	Novice	Regular	Expert	Regular
11	Novice	Novice	Expert	Novice
12	Novice	Novice	Expert	Novice
13	Novice	Novice	Expert	Novice
14	Expert	Novice	Expert	Expert
15	Expert	Regular	Expert	Expert
16	Expert	Novice	Regular	Regular
17	Novice	Expert	Expert	Expert
18	Expert	Novice	Regular	Regular
19	Expert	Expert	Expert	Expert
20	Expert	Expert	Expert	Expert
21	Novice	Novice	Regular	Novice

For the sake of simplicity, we show final results in third column of Table 2.

We were also interested about the user's perceived complexity of specific tasks related to our tool, not using the tool itself but of similar UI interactions. Such tasks were a subset of the previously presented, ranged from the one numbered 11th to the 17th. This time, the possible values were very easy, easy, normal, hard, and very hard. The criteria were the same applied to the previous procedure, so we matched the first case with expert users, the second and third cases with regular users, and the remaining two ones with novice users. We show the final results of this process in the fourth column of Table 2.

Finally, we assigned the users a category by selecting the predominant in the three analysed factors. When the three options are present, we chose the intermediate value, the regular user. This way, we have a population conformed by 28.57% of novice users, 28.57% of regulars, and 42.86% of expert ones.

5.2. Platforms. As the MoWA authoring tool was implemented as a Firefox for Android extension, it was a requisite to use any Android-based device capable of running such Web browser. Users carried their phones and installed the required software. The concrete authoring tool was successfully executed in every single device, which ranged among five different benchmarks and a total of 7 models: 8 times in a Samsung Galaxy S3 I9300, 6 in a Motorola Moto G, 2 in a LG Pro Light and a Motorola Moto E, and just once in a Blu Studio 5.5, a Huawei G6 L33, and a Samsung Galaxy S2.

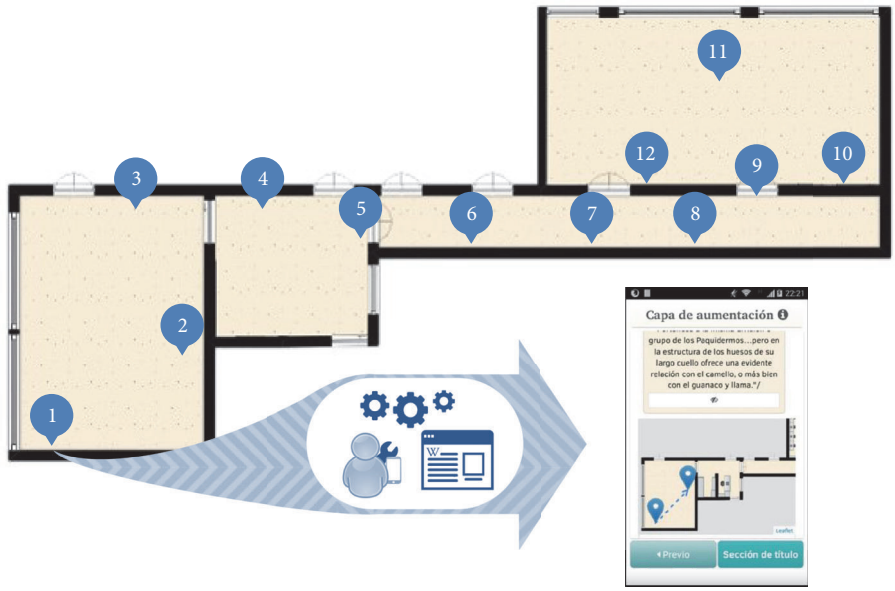

Figure 14: Physical distribution of PoIs, replicating the museum at our facilities.

In regard to the Android platform, mobile phones have installed 5 different versions. Android Jelly Bean with three API levels: one of the total devices supported the API 16 (v.4.1.2), another one the API 17 (v.4.2.2), and 11 of them the API 18 (v.4.3). The latter represents the 52.38% of the total cases. Remaining quantities correspond equally, with 4 devices each, to Android KitKat for API level 19 (v. 4.4.4) and Lollipop for API level 21 (v.5.0.2).

Our extension successfully was run in 4 versions of the mobile browser: most of them, 14 cases (66.67%), in the version 38.0, 5 cases in the 39.0, one in the 37.0, and another one in the 41.0. The former was the official stable version at that moment, the second was the Firefox Beta version, the third one was an old release previously installed by the user, and the latter was the Nightly version.

5.3. Setup. We asked users to create a museum tour guide app with our authoring tool. We simulated the *Museo de Ciencias Naturales de La Plata* in the facilities of our laboratory, by printing the same QRpedia codes found in the museum and placing them in four separate but connected spaces of our facilities, as seen in Figure 14. We decided to do it in several spaces, trying to simulate the museum environment in the best way, so that users have to walk over a large area and look for the pieces. The first room was a training space where users were allowed to interact with us and between them in order to understand the use of the tool; such points are not taken into account when analysing the results.

Before the experiment began, we explained to the participants the motivation of our approach and some technological and domain-specific concepts related to the kind of application required for performing the tasks of the experiment. They were introduced to Web augmentation, mobile hypermedia applications, and in-situ development in a 30-minute talk. After that, participants were asked to complete a survey for demographic data collection. Once every participant finished the survey, a scenario was explained to them.

You are about to visit the Museo de Ciencias Naturales of the Universidad Nacional de La Plata, in order to perform a special tour, that presents a set of pieces related to the Darwin's findings and descriptions made during his Voyage of the Beagle. You knew about this tour thanks to the museum's website, so you already read something about the pieces and also know they are presented in certain order.

When you arrive to the museum, you observe that the pieces, not only the ones of the mentioned tour, have a QRpedia code. Then you realize that some information is not being presented to those visitors who did not know about the Web site. You come up with the idea of combining the information submitted on the official Web site of the museum and the one presented in Wikipedia and also incorporate certain functionality that assists the user in visiting the pieces in the right order. In order to do this, you need to create a mobile Web application that supports

R1 augmentation of the Web page related to each piece with new information extracted from the Museum's official site,

R2 users assistance for showing the following piece to be visited,

R3 users assistance if they are in a wrong piece.

Once the statement was understood by the participants, they received printed instructions. They were explained they had to install the MoWA authoring tool, whose creation process is assisted through a series of form-based steps. We collected the data of each individual experience by logging every user action, registering a copy of their final productions and asking them to complete some questions. At the end of each step, the printed instructions requested the user to assign

Table 3: Percentage of completed tasks and full times by participant.

PS.	R1			R2 & R3				SR	Time
	a	*b*	%	*c*	*d*	*e*	%		h:mm:ss
1	1.00	0.56	0.78	0.56	1.00	1.00	0.85	**0.83**	0:49:20
2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**	0:57:53
3	0.61	0.61	0.61	0.89	1.00	1.00	0.96	**0.85**	0:57:52
4	1.00	1.00	1.00	1.00	1.00	0.75	0.92	**0.94**	0:58:20
5	0.11	0.00	0.06	0.00	0.11	0.00	0.04	**0.04**	0:16:04
6	1.00	0.00	0.50	0.00	1.00	1.00	0.67	**0.61**	0:46:29
7	0.50	0.50	0.50	1.00	1.00	1.00	1.00	**0.83**	1:05:57
8	0.89	0.89	0.89	0.89	0.89	0.38	0.72	**0.78**	1:02:22
9	1.00	1.00	1.00	1.00	1.00	0.63	0.88	**0.92**	1:12:09
10	0.56	0.22	0.39	0.22	1.00	0.38	0.53	**0.49**	0:54:07
11	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**	1:06:11
12	0.50	0.50	0.50	1.00	1.00	1.00	1.00	**0.83**	0:49:54
13	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**	0:59:13
14	1.00	0.78	0.89	0.78	1.00	1.00	0.93	**0.91**	0:49:02
15	1.00	1.00	1.00	1.00	1.00	0.50	0.83	**0.89**	0:35:55
16	1.00	1.00	1.00	1.00	1.00	0.63	0.88	**0.92**	0:54:41
17	0.81	0.83	0.82	0.89	0.89	0.63	0.80	**0.81**	0:33:22
18	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**	0:57:10
19	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**	0:23:06
20	1.00	0.50	0.75	1.00	1.00	1.00	1.00	**0.92**	0:22:22
21	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**	0:25:57
%			0.79				0.86	**0.84**	**0:48:27**

a level of difficulty of the performed tasks. At the end of the process, users were asked to complete another survey, this time, about their perception in the usage of the tool.

5.4. Results. In Table 3, we present the results of the requirements met by the 21 participants (PS). We wanted to know the completed tasks percentage for each participant, so we applied a concrete criterion for each requirement and obtained its completeness percentage.

Criterion for R1 was checking if applications have defined the following: (a) the 9 PoIs and (b) the PoI augmenter for the 9 PoIs' associated Web pages. In the first case, a percentage was obtained by analysing whether each PoI has defined a name, a URL to augment, and an external picture and description, both referenced from the museum's official Web site. In the second case, we analysed if the PoI augmenter has been rightly positioned and configured with the right external values. Each property was evaluated separately, and both made the final percentage value. When a property was partially configured, the property was marked as incomplete (0) and only right values with (1). Results shown in column *b* represent the average of both properties.

Criterion for R2 and R3 was the same, because both can be achieved by defining the mobile hypermedia navigation augmenter, the 9 PoIs, and the connections between them. We present the three issues in columns *c*, *d*, and *e*, respectively. Values in column *c* represent a percentage of rightly positioned augmenters for the 9 PoIs; values in column *d*, the percentage of defined PoIs in relation to the 9 that were required (just considering the name and the URL to augment as the mandatory properties to define); column *e*, the percentage of connected PoIs in relation to the 8 expected. Successful rates (SR) of the full tasks were obtained as the average of the percentages of the three requirements; this is,

$$\mathrm{SR}_i = \frac{\left(a_i + b_i\right)/2 + 2 * \left(\left(c_i + d_i + e_i\right)/3\right)}{3}. \quad (1)$$

Total result let us appreciate that end users could complete, in average, 84% of the requirements of the experiment. The standard deviation for such values is 22. Among the 21 analysed observations, 3 of them obtained a percentage below 75% and 18 above it.

With regard to the times, the whole process took participants an average of 00:48:27 h, ranging from 22 to 72 minutes. We started logging the user interactions when the «create application» option was triggered and stopped when the application was exported. We logged every single interaction of the user and save it to a file that finally was uploaded in conjunction with the application to our server. These times allowed us to observe the real consumed time by the user for the whole process. Times were obtained for the full process, not just for the tasks concerning the requirements.

Participants were also asked to rate the difficulty of the steps in the process, whose results are shown in Table 4. For each task in the first column of the table, users had to choose among five notes matching different levels of difficulty: (1) very easy, (2) easy, (3) regular, (4) hard, and (5) very hard. Averages were obtained for each step from the responses of

Table 4: Difficulty perceived by participants by tasks.

	Task	Average by users	Standard deviation	Rounded	Matching category
a	Installing the extension	1.40	0.60	1	Very easy
b	Setting up the app (name, augmentation strategy, sensor)	1.53	0.61	2	Easy
c	Setting up the context type representation	1.84	0.76	2	Easy
d	Defining context values of interest and their digital counterpart	2.58	0.84	3	Regular
e	Configuring augmenters (average of the grades for each augmenter)	2.66	0.86	3	Regular
f	Exporting the application	1.39	0.50	1	Very easy

the participants. However, the only step that accumulated 21 qualifications was the first one (a); in the remaining steps, there was a participant who refused to rate the tasks because he did not finish the process. Such participant chooses a «1» for the only task he ranked. No task was rated «5» by any participant.

Participants were also asked to optionally report to us their experienced problems and make suggestions about the tool and about other possible scenarios for which they think the tool may be useful. Regarding experienced problems, three participants considered the application was slow for them. The mechanism they considered to be a long wait was the scanning of QR codes (we used an embedded library for decoding them, so it depends on the processing capacity of the device). Five of them told us that printed instructions were not clear for them, and it took some extra time for them to understand what to do (even when it was explained in the initial talk). One participant had to reboot the browser 3 times, because the tool froze. This was the case of the participant using the 37th version of Firefox. One of them had problems at the beginning of the definition of values of interest; the first time the form was loaded, he could not create a new value of interest, but he reloaded the form and the problem was solved.

About improvements, one of them expressed that he would like to have bigger icons and another one that he would love to have an indicator of progress in the construction of the application, although he understood that it depends on the amount of context values of interest the application contemplates. Two participants tried, with no success, and wanted to have a copy-paste mechanism for text. We knew about this limitation, but we could not implement the feature for such versions of the browser API while it was in full-screen mode. Another one also suggested the copy-paste feature, but was intended for the list of properties defined for another PoIs. Other two participants considered that inserting augmenters should be materialized by drag-and-drop. We did this way at the beginning, but there was a cost on performance that we choose to avoid. Finally, one participant complained about the lack of zoom in the iframes when selecting external context from the Web.

Concerning other usages of the tools, all participants limited their scenarios to the domain of the augmenters used in the experiment scenario. Four of them said it is good for tourism in general, contemplating other museums and cities. One participant thought it could be useful to present information and other three for activities scheduling in the installations of a University. A last one considered that some game could be created using this application, like the treasure hunt.

5.5. Threats to Validity. As omitting threats can lead to wrong conclusions, we present below some issues regarding the validity of our experiment design and the obtained results. First, we mention some factors that could represent a difference in the presented experimental instance.

(i) User experience: while none of the users knew about or ever used MoWA authoring tool, we have conducted the experiment with an heterogeneous group of participants with different capabilities concerning the usage of mobile devices, mobile browsers, and browser extensions. We classify them by asking them some questions about their technological know-how. We collected results from 28.57% novice users, 28.57% regular users, and 42.86% expert users.

(ii) Learnability: some users tend to learn the required tasks faster than others, and some of them require more practice or repetitions to achieve the same results. We had some repetitive tasks, like defining a marker for the PoIs, filling the required data and extracting data from external sites, and placing and configuring augmenters.

(iii) Training: participants were allowed to interact during a first phase of the experiment with us and other participants by face-to-face speaking (not performing other participant's tasks). We cannot predict what would have happened if the evaluation had been conducted without such training phase.

(iv) Maturation: as the users had to create 9 PoIs (with 3 extras for training) and the creation process of the 9 ones took an average time of 00:48:27 h, psychological changes can be experienced, as being fatigued, causing an impact in the variable successful tasks. They can also feel frustrated and quit. In fact, we had a case of a participant not ending the process.

As our experiment was (a) controlled in a simulated environment where some conditions may differ in the real world, (b) conducted against just one sample of participants, and (c) designed for solving a problem under just one application domain, we are aware that our findings can be altered when applied in different settings. Therefore, some setting can alter the representativeness of results, making our conclusions

generalizable just under certain conditions. Below, we present some required points for experiment replications.

(i) Participant selection: we wanted a representative sample of end users to participate in our experiment, so a number of people were invited to participate in the experiment and they were the ones who decided volunteering in the experiment or not to do it. We did not discard any participant for any criteria, even if they were not used to mobile devices. In that sense, we were ensured to have a heterogeneous group; however there were factors with prevailing values of people who chose to participate. The sample had 71.43% of male members, 66,67% of Argentinean citizens, 90.48% of people with at least a degree career in progress, 52.38% of coming from hard sciences field, and 42.86% expert users, and with age range not falling outside 22 to 39 years.

(ii) Context selection: we conducted the experiment in the facilities of our laboratory, covering four independent spaces, to ensure that the user has to move and look for the different points of interest. However, the dimensions of the museum, the disposition, and representation of the pieces of the collection were quite different. In addition, our facilities provided a stable Internet connection through Wi-Fi.

(iii) Domain selection and tasks complexity: as we wanted to compare our results against our previous results in [11], we decided to conduct the experiment under the unique, same domain problem, considering an only level of complexity.

5.6. Discussion. Our main purpose, the assessment of our research question, was successfully achieved favoring our approach; we demonstrated a high probability for an end user, without programming expertise, to be capable of creating his own mobile Web experience by reusing and enhancing an existing Web application with context-aware features. Nevertheless, there are still many questions to answer, some of which derive from the following facts.

There was an unexpected behaviour during the experiment, concerning the single participant who quit the experiment, completing a low percentage of tasks; he achieved just a 4% of the process. He was in the right direction, but he told us he did not understand how to do it. We could also observe that, unlike other participants, he did not attempt to solve the problem by asking us or interacting with others. He was part of the novice users' category, and yet, the results of this category of participants were promising; they achieved, in average, a 78.49% of the requirements, although with a standard deviation of 37.26. In this sense, regular users and experts got better numbers: 80.91% with a standard deviation of 17.65 for the first ones and 88.91% with a standard deviation of 11.92% for the latter.

There were also unexpected behaviours on the other side of the ledger. There is a difference of just 2.42% between participants with a low level of expertise and regular users and 10.42% against expert users. Among novices, the 83.3% of participants never used Firefox for Android before, and none of them was used to use extensions for this browser. 66.6% told us that they have never used any kind of browser extensions, and in the same percentage, they never used Web augmentation applications or QR codes scanners. 83.3% of them never used an authoring tool, and they never managed markers in a map. Many of these were challenges that the user had to face for the very first time to carry out the experiment, and anyway they obtained promising results in relation to experienced users. Finally, we could also observe some complications related to the use of some UI elements. First, we used a wrong design for presenting the properties of a point of interest, and many participants had some trouble interpreting this distribution in a small screen. A second problem was related to the expected interactions by the end users. Concretely, we identified a problem in the interaction for inserting a new augmenter in the target Web page. As the contexts of the bar with the list of augmenters and the Web page content were different and also because of the processing limitations of a mobile device, it was hard to simulate a drag and drop among both contexts without losing performance. Instead, we implemented the insertion with a tap in the list of augmenters, and then, when the augmenter is in the Web page context, the user is able to drag-and-drop the thumbnail. There was also a problem with the capture of a picture for decoding QR codes; we opted to allow the capture by tapping any area on the screen (as in many mobile phone's cameras), but the lack of a button confused a participant, who reloaded a form a couple times before trying to touch the screen. Such participant was considered a regular user (according to the categories in Table 2).

6. Related Work

As already mentioned, there is a need for supporting the adaptation of Web applications [6, 7, 26]. Contents and services can be tailored according to the users' requirements, which may involve not just a user model but also a context model. In [39], authors present a model-driven approach supporting active context-awareness. They contemplate the context as an active actor, which can automatically trigger adaptations with no intervention of the end user when a concrete context of use is identified. The supporting framework is an extension of WebML, but although this approach is based on some visual components for the development of applications, it does not contemplate EUD programming techniques.

However, there are many approaches supporting EUD techniques and Web adaptation can be achieved in different ways. Users can create Web-based solutions from scratch, but also on the basis of existing Web resources. Concerning the Web field, there is a close approach to ours but with a different purpose, raised under the principles of a different programming technique and without considering aspects related to mobility, context information, or in-situ development. In [4], authors propose a DSL for end user development that abstracts the complexity and understanding of programming structures and provides trust, because even if an end user is not the author of a script, he can easily read and know what tasks it does perform. Their DSL constructs are based

on a metaphor, consisting in understanding the Web as a wall to be decorated with stickers, in the meaning of external content fragments. This pair conforms a Sticklet, which is an augmentation unit. In contrast to our work, they propose a set of constructs generic enough for creating Web augmentation solutions for a wide range of purposes, but their possibilities are beyond the scope of mobile or context-aware features.

Then, with the advent of mobile technology, other approaches emerged that allowed end users to start creating their own solutions contemplating mobile features, from different development environments and targeting applications of different nature. There are approaches conceiving the creation of mobile applications from desktop [31, 40], native mobile [30, 32, 33, 41], and mobile Web environments [42, 43]. In the same way, we found approaches producing native or hybrid mobile applications but not pure mobile Web ones; we present them below.

Along the rest of this section, we focused on a series of related works facing EUD and Web adaptation in relation to the mobile and mobile Web fields. We analyse each work considering the platform for which it was developed and the architecture of the tool; the resulting application kind; the adopted end user programming technique; the capability of the tool to solve different domain problems and to specify control flows; and the contemplation of crowdsourcing principles. Since the selected approaches are aimed for end users, we also consider the technical know-how of end users and if they conducted the experiment with prior training. In this regard, it is important for us to compare the expected expertise against the real expertise of participants at evaluation time, since we think it is important to demonstrate EUD approaches with people other than experts (e.g., computer science students).

When contextual information is used to support the software development process, end users can build their solutions in the same place where the needs and problems are presented. This increases the boundaries of opportunistically development a.k.a. situational applications [44]. In this setting, [32] presents CASTOR, an authoring platform with a storytelling purpose. They implemented three supporting tools: a hybrid mobile application for creating stories in-situ; a Web-based client for managing the collected data, and a hybrid mobile application for consuming the narrations. The Phonegap framework supports the resultant applications, and users can share them through a repository. The authoring process is assisted through forms that allow users to select the structure of the story, define stages and context values, write the plot of the story. Stories can cover different domains, for example, history and literature, but they are always restricted to a storytelling purpose and using a limited number of story structures (simple, sequential, or crossroad). The tool was evaluated with 19 primary school students with no programming skills, who were introduced with an initial briefing before usage.

Sometimes, users have needs that can be solved by combining existing Web content and services retrieved from different sources, and mashups [45] may represent a possible solution. [40] presents CAMUS, a framework for designing mobile applications through visual composition and high-level visual abstraction. It integrates and provides resources according to different contextual situations. It involves different user roles: an administrator with technical knowledge for registering resources in the platform and mapping them with context elements; a designer who defines how to mashup the services and to visualize the information; and a final recipient of the authored application. There is no mention of the adopted programming technique, but the provided screenshot suggests a WYSIWYG approach. CAMUS is aimed at creating context-aware mobile or Web-based applications. Nevertheless, the authors mention that execution engines are created as native applications for different mobile devices.

In the mobile field, [33] presents MobiDev, an Android-based development platform for mobile application creation from mobile devices. It allows the user to create the graphical interface by writing source code, designing mock-ups with a visual editor, or drawing a sketch on paper and taking it as a picture, so the system will analyse and interpret it to generate a visual design. The approach contemplates end users with no programming skills developing applications with a basic control flow, but also developers defining a more specific behaviour through JavaScript code. The approach was evaluated with 16 students belonging to computer science department, who were previously trained. The experiment was successful, but the requirements did not contemplate the use of mobile features.

[41] presents a native platform for enabling end users to compose native mobile applications from their own mobile devices, by integrating the mobile features provided by the same device but also from Web services. Users specify activities through visualization components, which are executed by an interpreter that automatically creates the user interface. The approach comprises a repository for enabling end users to share their productions and reconfigure them. The research team evaluated their approach with 40 students of the first year of the computer science career, and it was a requirement that they have no programming skills. Concerning the prior training, they give participants a lesson of 20 minutes about their tool and another extra 20 about another similar tool against which they compared results.

Puzzle [30] is an EUD framework for producing native but Web-based applications, targeting touch-based platforms. Users combine building blocks through a puzzle-based metaphor with color-equipped corners for pinpointing its combinability, which makes it suitable for end users with no programming skills. The authors mention that there is no need of plugins, but they end up presenting an implementation in the form of a native Android application. Diverse multipurpose combinations can be created, changing the application's logic. A repository of created artefacts is available, but there is no way for users to request the construction of a concrete application to the crowd. Puzzle was evaluated with 13 participants with no IT-related jobs, and they were not exposed to previous training.

The authors in [42] propose an approach towards EUD for multidevice mashups creation with composite resources. They implemented a framework and a UI centric tool using the WYSIWYG technique. Users should select among the

Table 5: Features covered by approach.

	MOWA	[42]	[40]	[30]	[41]	[43]	[31]	[32]	[33]
Pure mobile development environment	✗	✗	✗	✓	✓	✗	✗	✓	✓
Pure mobile web development environment	✓	✓	✗	✗	✗	✓	✗	✗	✗
Produces pure mobile apps	✗	✓	✓	✓	✓	✓	✓	✓	✓
Produces pure mobile web apps	✓	✗	✗	✗	✗	✗	✗	✗	✗
End user programming technique	Form-based WYSIWYG	WYSIWYG	WYSIWYG	Puzzle met.	Puzzle met.	PBE	Form-based	Form-based	Mockups
For multiple domains	✓	✓	✓	✓	✓	✓	✗	✗	✓
For nontechnical users	✓	✓	✓	✓	✓	✓	✓	✓	✓
Users can request new requirements to the crowd	✓	✗	✓	✗	✗	✗	✗	✗	✗
Users can massively share their productions	✓	✓	✓	✓	✓	✓	✓	✓	✗
Users can specify the control flow	✓	✓	✓	✓	✓	✓	✗	✗	✓
Additional assistance mechanisms	✓	✗	✗	✓	✓	✗	✗	✓	✓
Authoring process at client-side	✓	✓	✗	✗	✓	✗	✗	✗	✓
Evaluation with nontechnical users	✓	✓	✗	✓	✓	✗	✓	✓	✗
Evaluation without prior training	✓	✓	✗	✓	✗	✗	✓	✓	✓

existing data components and UI templates in the repositories and then perform an association between data items and visual elements. Finally, a platform-independent schema is generated and saved in the platform repository, so the user can download it for the supported platforms and execute it through a native engine. Diverse multipurpose combinations can be created and the design environment is a Web application, but they recommend not to execute it in mobile devices but in larger-screen ones. The experiment was conducted with a 10-minute demonstration and 36 participants, 17 of them with programming skills.

Other approaches empower the end users with the capability of creating mobile Web applications from desktop environments. For instance, [43] allows the user to create widgets, in the meaning of simple applications, that represent a specific Web interaction. These artefacts, called Tasklets, are created using Programming By Example (PBE). Users need to install a plug-in in their desktop Web browser and use it to record the sequence of steps required to perform the task. This tool saves the need for representing these steps, builds a Tasklet template, detects and defines potential parameters, and finally makes the script accessible for multiple platforms through their repository. A wide spectrum of Tasklets could be created and shared for both personal and public consumption.

Another related approach for EUD from desktop environments is presented in [31], where the authors present a cloud-based development platform of context-aware mobile services to be consumed as native applications. The platform is accessible through a Web-based application, where the producer can associate a set of context values (specific locations, areas, times, dates, etc.) with concrete information to be delivered to the clients meeting such conditions. The authors conducted an experiment with 10 tourism domain experts with no technical skills and no prior training, and they did it from a preinstalled native application. The resultant applications are bounded to the information delivery purpose.

We present in Table 5 a summary of the presented approaches through 14 outstanding features. It should be clarified that it does not take into account those works that do not contemplate mobile features, like [4], because mobility is a fundamental aspect of our approach, to support either the process of creating an application or the execution of the resulting applications. Analysed works are disposed as single rows, and features are arranged as columns. Applicant cases are ticked fields and unchecked values mean a nonsupported or nonmentioned feature in the corresponding writing.

As we can see, our approach faces the less common characteristics covered by existing works. The following ones are present in less than 50% of the works. 37.5% of the solutions perform the authoring process at client-side; 25% provide a pure mobile Web development environment; 12.5% offer end users the possibility of requesting new functionality to the crowd and there is no one allowing the creation of mobile Web applications that do not rely on native components.

We also found interesting that the control flow definition feature is not contemplated by other approaches implemented through the same end user programming technique than ours: the form-based one. In regard to such technique, we have overcome multidomain problems that the other form-based approaches do not. This is due to the domain-specific authoring tools inclusion based on building blocks combinations, so the difference is not achieved by the technique itself but through a combination of both techniques in the whole development process.

Among the analysed approaches supporting Web augmentation by means of end users, it does exist a relationship between complexity and expressivity. In order to address this issue, we propose a usable toolkit for end users, based on a framework with a set of default modules. This framework is extensible by developers, allowing them to create new modules for improving or adding new functionality to the end user's toolkit. In contrast to other approaches, our tool is expressive enough to support solutions in multiple domains and benefits from the possibility of combining augmenters of different levels of complexity to express even the control flow of the augmentations. Users can combine those augmenters of a low level of granularity for supporting a wide variety of purposes, for example, the ones in charge of displaying a simple text or just a video. However, building complex augmentations by combining simple augmenters increases their development effort, since it requires additional design time and a greater complexity for the user; he must have a deeper understanding of the use of the available augmenters and a clear logic behind their combination for achieving the kind of result he expects. In this sense, this can also mean a cost in the learning time, and it is critical in EUD since programming is not often the main activity of the user. It is also possible that the application that the end user needs to create requires a greater level of expressiveness. In both cases, here is where the use or extension of more specialized augmenters comes into play. Users with programming skills can extend and share augmenters as well as sensors, context types, context type's representations, and builders. In this setting, the restrictions that an end user may experience due to the lack of concrete constructs (e.g., augmenters) are mitigated by the possibility of outsourcing new requirements to the crowd, in conjunction with the higher level of expressivity that can be achieved by developers using the framework. MoWA framework is purely JavaScript-based and it benefits from the privileges of the Firefox APIs (https://developer.mozilla.org/en-US/Add-ons/SDK). Moreover, as a framework, it abstracts lower level details of implementation and offers a set of classes ready to instantiate or extend, reducing the time and energy in developing an application. Thus, our approach has the flexibility to respond to new requirements by end users in two ways: letting them do it by themselves and allowing developers to provide them with the required features through the development of specialized artefacts. This kind of collaboration between end users and artefacts developers actually happens in today's communities (https://forum.userstyles.org/categories/style-requests, https://greasyfork.org/es/forum/categories/script-requests).

7. Concluding Remarks and Further Work

In this paper, we proposed an approach for empowering end users with the capability of creating mobile Web augmentation applications from their mobile devices. Our solution consisted in a tool oriented to satisfy the end users' needs in multiple ways. In previous works, MoWA just contemplated the idea of a developer extending the framework and sharing applications with end users through a crowdsourcing platform. Therefore, end users were limited to ask for new implementations for a concrete scenario and to download and install existing applications. Now, they can use our authoring tool for creating their own solutions for concrete scenarios.

Reusing artefacts of our previous framework also let us take advantage on the possibility of supporting an in-situ authoring process, from mobile devices themselves. The benefits of such feature, also in conjunction with live programming capabilities, are that the user is building his solution in the application's target execution context, whether virtual (a Web page) or real (a geographical position). He is dealing with the dimensions of the device, the modes of interaction, the layout of the elements in the DOM, the Internet connectivity, the physical presence of physical objects that could interfere when associating a geographic position with a digital resource, and so on.

MoWA builders are created by developers who took technical decisions about the best way of solving domain-specific problems. Such tools supported the authoring process through a form-based implementation, assisting the user through a series of steps that allow them to choose the best configuration for their scenario. It also made it possible to omit one step in the software development process where communication between people usually has some understanding problems, the requirements gathering phase. This step is now performed by the same person that will be on charge of the application creation. We are also enabling opportunistic development in-situ, as solutions are implemented for mobile platforms, being capable of accessing and using the mobile characteristics during the process.

By using our authoring tool, the end user's needs could be satisfied by themselves, and now they can play two roles in the crowdsourcing platform: by demanding for concrete solutions and by solving other less-experienced user's needs. The worst possible scenario is an end user not finding a concrete builder, or not understanding how to combine components for a specific scenario problem. In this case, we are evaluating the benefits of enabling developers to create domain-specific authoring tools, so producers can use them for creating their solutions with a lower level of technical knowledge, because this way they do not need to take decisions about the proper sensor or kind of component to use in their application.

We demonstrated the feasibility of our approach by creating concrete builders for our authoring tool and using them for conducting an experiment with 21 participants, with a varied spectrum of demographic characteristics. Results proved that end users can deal with an in-situ creation process of mobile Web augmentation applications with a high level of confidence and 4 times faster than developers through scripting development. It also showed the feasibility of a tool

overcoming client-side scripting from browsers extensions (with no native dependencies) and the devices' processing limitations.

Evaluation also allowed us to collect observations, ratings, and opinions from the end users, which gave us the chance to improve the usability of the tools and the authoring process in general. From the technical point of view, we are also working to achieve resources optimization and implementing new ways of UI interactions, like drag and drop for the augmenter configuration thumbnails. New builders are being composed and we are also improving the ones related to augmenters, for enabling the end users to specify new kind of interactions and execution order, so users can define a more specific flow control.

We are also designing a DSL for applications specification, targeting different levels of knowledge and properly abstracting the required features. Our aim is to reach some common point between developers and end users, trying to generate new ways of collaborative development. In addition, we are also generating new mechanisms for supporting the development process in a collaborative environment and delegating some tasks to the crowd, with different levels of expertise. For example, an advanced end user can start a project in the community, by defining the basic structure of the application. Then, developers can contribute by enhancing the code with their low level technical knowledge, and some users can collect information about the real world, like pictures, locations, and describing places in-situ. Another participant can contribute in the project by collecting data for Web for augmentation purposes. Users should be able to suggest changes and also to fork the whole project, if the visibility of the project is public, or private but shared with them. Having the crowdsourcing platform and an underlying a bigger volume of data to analyse, it will allow us to better understand how well users discover and use the augmentations provided by developers. This will also lead us to analyse the evolution of projects and particularly the user's needs, behaviours, and learning.

Competing Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by the «STIC-AMSUD» Project named «WAMAW-OUR: Web Augmentation Methods for Adapting Web Sites for Supporting Opportunistic User Requirements».

References

[1] T. Sohn, K. A. Li, W. G. Griswold, and J. D. Hollan, "A diary study of mobile information needs," in *Proceedings of the 26th Annual CHI Conference on Human Factors in Computing Systems (CHI '08)*, pp. 433–442, Florence, Italy, April 2008.

[2] K. Cheverst, H. Turner, T. Do, and D. Fitton, "Supporting the consumption and co-authoring of locative media experiences for a rural village community: design and field trial evaluation of the SHARC2.0 framework," *Multimedia Tools and Applications*, 2016.

[3] W. Schwinger, C. Grün, B. Pröll, W. Retschitzegger, and A. Schauerhuber, "Context-awareness in mobile tourism guides—a comprehensive survey," Rapport Technique, Johannes Kepler University Linz, 2005.

[4] O. Díaz, C. B. Arellano, and M. Azanza, "A language for end-user web augmentation: caring for producers and consumers alike," *ACM Transactions on the Web*, vol. 7, no. 2, article 9, 2013.

[5] H. Lieberman, F. Paternò, M. Klann, and V. Wulf, "End-user development: an emerging paradigm," in *End User Development*, pp. 1–8, Springer Netherlands, 2006.

[6] N. O. Bouvin, "Unifying strategies for Web augmentation," in *Proceedings of the 1999 10th ACM Conference on Hypertext and Hypermedia (Hypertext '99)*, pp. 91–100, Darmstadt, Germany, February 1999.

[7] O. Díaz and C. Arellano, "The augmented web: rationales, opportunities, and challenges on browser-side transcoding," *ACM Transactions on the Web*, vol. 9, no. 2, article 8, 2015.

[8] D. Carlson and L. Ruge, "Ambient Amp: an open framework for dynamically augmenting legacy Websites with context-awareness," in *Proceedings of the 9th IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP '14)*, pp. 1–6, IEEE, Singapore, April 2014.

[9] G. Ghiani, M. Manca, F. Paternò, and C. Porta, "Beyond responsive design: context-dependent multimodal augmentation of web applications," in *Mobile Web Information Systems*, pp. 71–85, Springer International Publishing, 2014.

[10] W. Van Woensel, S. Casteleyn, and O. De Troyer, "A generic approach for on-the-fly adding of context-aware features to existing websites," in *Proceedings of the 22nd ACM Conference on Hypertext and Hypermedia (HT '11)*, pp. 143–152, Eindhoven, The Netherlands, June 2011.

[11] G. A. Bosetti, S. Firmenich, S. E. Gordillo, and G. Rossi, "An approach for building Mobile Web applications through Web Augmentation," *Journal on Web Engineering*, vol. 16, no. 1-2, pp. 75–102, 2016.

[12] C. Challiol, S. Firmenich, G. A. Bosetti, S. E. Gordillo, and G. Rossi, "Crowdsourcing mobile web applications," in *Current Trends in Web Engineering*, pp. 223–237, Springer, 2013.

[13] C. Scaffidi, M. Shaw, and B. Myers, "Estimating the numbers of end users and end user programmers," in *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '05)*, pp. 207–214, IEEE, Dallas, Tex, USA, September 2005.

[14] K. T. Stolee, S. Elbaum, and A. Sarma, "End-user programmers and their communities: an artifact-based analysis," in *Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement (ESEM '11)*, pp. 147–156, Banff, Canada, September 2011.

[15] A. J. Ko, R. Abraham, L. Beckwith et al., "The state of the art in end-user software engineering," *ACM Computing Surveys*, vol. 43, no. 3, article no. 21, 2011.

[16] D. C. Halbert, *Programming by example [Ph.D. thesis]*, University of California, Berkeley, Calif, USA, 1984.

[17] S. K. Chang, Ed., *Visual Languages*, Springer Science & Business Media, Berlin, Germany, 2012.

[18] S. Firmenich, G. Bosetti, G. Rossi, M. Winckler, and T. Barbieri, "Abstracting and structuring web contents for supporting personal web experiences," *Lecture Notes in Computer Science*

(including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 9671, pp. 76–95, 2016.

[19] S. Li, T. Xie, and N. Tillmann, "A comprehensive field study of end-user programming on mobile devices," in *Proceedings of the IEEE Symposium on Visual Languages and Human Centric Computing (VL/HCC '13)*, pp. 43–50, San Jose, Calif, USA, September 2013.

[20] B. Fling, *Mobile Design and Development: Practical Concepts and Techniques for Creating Mobile Sites and Web Apps*, O'Reilly Media, Inc, 2009.

[21] W. N. Schilit, *A system architecture for context-aware mobile computing [Ph.D. thesis]*, Columbia University, New York, NY, USA, 1995.

[22] N. O. Bouvin, B. G. Christensen, K. Grønbæk, and F. A. Hansen, "HyCon: a framework for context-aware mobile hypermedia," *New Review of Hypermedia and Multimedia*, vol. 9, no. 1, pp. 59–88, 2003.

[23] T. Höllerer and S. Feiner, *Mobile Augmented Reality. Telegeoinformatics: Location-Based Computing and Services*, Taylor and Francis Books Ltd, London, UK, 2004.

[24] A. Charland and B. Leroux, "Mobile application development: web vs. native," *Communications of the ACM*, vol. 54, no. 5, pp. 49–53, 2011.

[25] J. P. Espada, R. G. Crespo, O. S. Martínez, B. Cristina Pelayo G-Bustelo, and J. M. C. Lovelle, "Extensible architecture for context-aware mobile web applications," *Expert Systems with Applications*, vol. 39, no. 10, pp. 9686–9694, 2012.

[26] C. Asakawa and H. Takagi, "Transcoding," in *Web Accessibility*, pp. 231–260, Springer, London, UK, 2008.

[27] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Pearson Education, Uttar Pradesh, India, 1994.

[28] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, 2007.

[29] M. E. Fayad and D. C. Schmidt, "Object-oriented application frameworks," *Communications of the ACM*, vol. 40, no. 10, pp. 32–38, 1997.

[30] J. Danado and F. Paternò, "Puzzle: a visual-based environment for end user development in touch-based mobile phones ," in *Human-Centered Software Engineering*, pp. 199–216, Springer, Berlin, Germany, 2012.

[31] D. Martín, C. Lamsfus, and A. Alzua-Sorzabal, "A cloud-based platform to develop context-aware mobile applications by domain experts," *Computer Standards & Interfaces*, vol. 44, pp. 177–184, 2016.

[32] F. Pittarello and L. Bertani, "CASTOR: learning to create context-sensitive and emotionally engaging narrations in-situ," in *Proceedings of the 11th International Conference on Interaction Design and Children (IDC '12)*, pp. 1–10, Bremen, Germany, June 2012.

[33] J. Seifert, B. Pfleging, E. Del Carmen Valderrama Bahamóndez, M. Hermes, E. Rukzio, and A. Schmidt, "Mobidev: a tool for creating apps on mobile phones," in *Proceedings of the 13th International Conference on Human-Computer Interaction with Mobile Devices and Services (Mobile HCI '11)*, pp. 109–112, Stockholm, Sweden, September 2011.

[34] A. J. Ko, B. A. Myers, and H. H. Aung, "Six learning barriers in end-user programming systems," in *Proceedings of the IEEE Symposium on Visual Languages and Human Centric Computing*, pp. 199–206, Rome, Italy, September 2004.

[35] A. Fortier, G. Rossi, S. E. Gordillo, and C. Challiol, "Dealing with variability in context-aware mobile software," *Journal of Systems and Software*, vol. 83, no. 6, pp. 915–936, 2010.

[36] D. E. Millard, D. C. De Roure, D. T. Michaelides, M. K. Thompson, and M. J. Weal, "Navigational hypertext models for physical hypermedia environments," in *Proceedings of the 15th ACM Conference on Hypertext and Hypermedia (Hypertext '04)*, pp. 110–111, Santa Cruz, Calif, USA, August 2004.

[37] A. Couthures, "JSON for XForms," XML Prague 2011, 13, 2011.

[38] S. Harper, C. Goble, and S. Pettitt, "proximity: walking the link," *Journal of Digital Information*, vol. 5, no. 1, article 236, 2006.

[39] S. Ceri, F. Daniel, F. M. Facca, and M. Matera, "Model-driven engineering of active context-awareness," *World Wide Web*, vol. 10, no. 4, pp. 387–413, 2007.

[40] F. Corvetta, M. Matera, R. Medana, E. Quintarelli, V. Rizzo, and L. Tanca, "Designing and developing context-aware mobile mashups: the CAMUS approach," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9114, pp. 651–654, 2015.

[41] R. Francese, M. Risi, G. Tortora, and M. Tucci, "Visual mobile computing for mobile end-users," *IEEE Transactions on Mobile Computing*, vol. 15, no. 4, pp. 1033–1046, 2016.

[42] C. Cappiello, M. Matera, and M. Picozzi, "A UI-centric approach for the end-user development of multidevice mashups," *ACM Transactions on the Web*, vol. 9, no. 3, article 11, 2015.

[43] G. Manjunath, M. N. Murty, and D. Sitaram, "Tasklets: enabling end user programming of web widgets," *International Journal of Web Engineering and Technology*, vol. 8, no. 3, pp. 264–290, 2013.

[44] L. Cherbakov, A. Bravery, B. D. Goodman, A. Pandya, and J. Baggett, "Changing the corporate IT development model: tapping the power of grassroots computing," *IBM Systems Journal*, vol. 46, no. 4, pp. 1–20, 2007.

[45] F. Daniel and M. Matera, *Mashups: Concepts, Models and Architectures*, Springer, Berlin, Germany, 1st edition, 2014.