Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 5 (2011) 641–644

Procedia Computer Science

The 8th International Conference on Mobile Web Information Systems (MobiWIS)

Developing once, deploying everywhere: A case study using JIL

Carlos Duarte*, Ana Paula Afonso

LaSIGE, University of Lisbon, Campo Grande, 1749-016 Lisboa, Portugal

Abstract

With the increasing number of mobile platforms available, developers aiming to bring their products to the larger number possible of consumers are required to program for different platforms, using different development environments, and often resulting in applications with different user experiences. A possible solution for this problem are development frameworks supporting develop once, deploy everywhere cycles. However, these frameworks still possesses serious limitations, like the small number of features they make available when compared with native application development. In this paper, we explore one of these frameworks, and report on the impact of its limitations on the development of a cooperative geo-referencing mobile application.

Keywords: Mobile application development, Development frameworks, JIL

1. Introduction

Mobile devices and services are ubiquitous nowadays. This impacts, not only the final user, but also developers. The business model brought forward by the different application stores currently available, allowed programmers and entrepreneurs worldwide to publish software, commercial or free, targeting a market of millions of potential users. Moreover, this commercialization and distribution process does not require great effort from the part of developers, who trade part of their profits for the possibility to reach such a market, benefiting also from an increased visibility.

However, the increasing number of available target platforms and associated application stores, makes it harder to chose which of to develop for. From a dissemination and exploitation point of view, it is desirable to widen the number of potential clients, not limit it. As a result, developers have been programming for multiple platforms, with increased effort, and, often, resulting in applications that fail to present the same user experience when executed in different platforms. Considering this, application developers must choose a suitable framework for implementing their applications that is compatible with the largest number of target devices and supports the development of critical application requirements. Although the choice of development platform is market-driven [1] it is also depend on their features and the specific requirements of applications.

The logical evolution we have been witnessing in the past couple of years, is the abstraction of the target platforms, leading to environments where it is possible to develop once for multiple platforms. Examples of such environments are PhoneGap[1] or WAC[2] (Wholesale Applications Community) which unites leading organisations within

*Corresponding author. Tel.: +351 217 500 519; fax: +351 217 500 084

Email addresses: cad@di.fc.ul.pt (Carlos Duarte), apa@di.fc.ul.pt (Ana Paula Afonso)

[1] http://www.phonegap.com/

[2] http://www.wacapps.net/

Selection and/or peer-review under responsibility of Prof. Elhadi Shakshuki and Prof. Muhammad Younas. doi:10.1016/j.procs.2011.07.083

the telecommunications sector. Such environments typically provide APIs that abstract mechanisms to access the mobile device hardware, namely system and phone functions, network services, mobile operator details, web browsing and the device's operating system. Ideally, this would allow an application developed once to operate in every device, existing or still to be developed, independent of its operating system, as long as it provides support for such API.

Some reviews [1, 2] have compared mobile application runtime environments, such as Java Mobile Edition (ME), .NET Compact Framework, Flash Lite and Android, among others. These studies offer a general guidance into the functionalities and deficiencies of these platforms and reveal that we are far from supporting the "write once, run anywhere" paradigm. In this paper we report on the development of a mobile application using the JIL (Joint Innovation Lab) framework, which is now merged with the WAC framework. We start we a brief overview of the JIL framework. We then proceed with the description of the development effort of a collaborative geo-referencing application, and identify the limitations that the JIL framework imposed on the intended features development. We conclude the paper with some suggestions for improving such frameworks.

2. Exploring the JIL API

The JIL framework [3] provides support for the execution of widgets on the mobile device. Widgets are compiled from a minimal set comprised of a configuration file and an HTML file. Optionally, developers can also include a Javascript code file, a CSS file and media files (audio, images, ...). The development environment compiles the widget which is then transferred to the mobile device, where it is executed inside the frameworks runtime environment (available by installing the *Apps* application in the Android Market, for instance).

The JIL API offers several classes for functions and features typical of mobile devices:

- *Widget.Device* – used for interacting with the mobile device components, including accessing information about the cell phone and operator, but also on the available applications and file structure
- *Widget.PIM* – used for interacting with the device's personal information management features
- *Widget.Multimedia* – used for interacting with the device's multimedia capabilities, namely camera (when available) and audio playback
- *Widget.Messaging* – used for handling messaging creating and sending

The version of the JIL API evaluated (version 1.2.2) [4] presented several limitations, some of which have already been addressed in more recent versions. Android's sandbox model prevent access to some of the device's features. For instance, even though it was possible to use the device's camera to take pictures, it was not possible to access the resulting file, since the access to the file system was restricted to the files installed by the widget. Similarly, the sandbox model prevented access to audio and video player made available by the operating system. On a more positive note, support for GPS features was available and accessible. One of the features lacking that is more glaring is the absence of support for even a simple data base.

3. Developing a cooperative geo-enabled application with JIL

In order to more fully explore the development process supported by the JIL API, we developed an application that requires cooperation between different devices in a geo-referencing context. The application operates similarly to a Rally paper, where the user needs to find places based on clues. The clues are provided by the application. They consist on text with images, audio or video. The position of the user is checked by GPS. When the user is in the correct place a new clue is unlocked. Some clues require the simultaneous positioning of more than one user in the same place, or in different places. This endows the application with cooperation features.

Besides developing the mobile application, we developed two additional web-based applications, that can be run both on a desktop system and a mobile device. The first of these applications allows the building of the puzzles to be solved with the mobile application (Figure 1). The clue builder allows specifying the location the player must be at to unlock the next clue, a textual description, an accompanying media file, the number of players required to unlock the clue and how close the players need to be to the exact location in order for the clue to be unlocked. The location

can be selected manually on the map, or, if the user is creating the clues with a mobile device, automatically using the device's GPS. The clues are stored online in a central location, which is then accessed by the mobile application to play the level. Given the previously mentioned limitation the prevents access to media files generated by the application (pictures and videos in this situation), we were forced to upload the media files (captured using the device or obtained from other sources) to a central repository, and the file is then accessed through its URL.

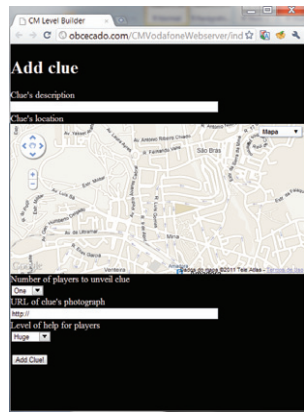

Figure 1: Creating a clue on the level builder application.

The mobile application assists players in finding the correct locations. It starts by creating teams for collaborative games (Figure 2, left) and selecting the level to play from the central repository. During regular game play, the mobile device's screen presents the clue and corresponding media file (Figure 2, right). The player has a "Check Clue" button available, that should be pressed whenever she wishes to check if she is in the correct location. If she is not yet in the correct location, feedback is presented letting the player know if she is near, far or very far from the location. Additionally, the direction she should head to reach the location is also presented. When the player gets to the desired location, within the margin of error defined in the level builder, and presses the "Check Clue" button, the next clue is unlocked (or the game ends it there are no more clues).

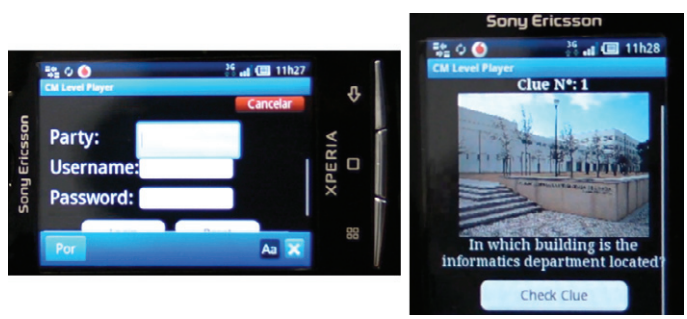

Figure 2: (left) Authentication and team selection. (right) The device's screen during regular game play.

When more than one player is required to unlock the next clue, the applications coordinate to make sure there are enough players at the required locations. Only then the next clue is unlocked. From then on, every member of the team that presses "Check Clue" (even the ones that may not have been present in the locations) receives the next clue.

When a player reaches the end of the game (presses the "Check Clue" button for the final time) the interface presents the total time it took the team to complete the game and the number of *checks* made by the player and team.

The final application is a visualizer that is fed by the GPS coordinates transmitted by the mobile devices to check their locations (Figure 3). This application can operate in real time, allowing a viewer to monitor the team's progress,

with the positions of each player being updated with every location check. Alternatively, the application can be used after the game has ended, allowing players to check their progress, as well as the progress of other team members.

Figure 3: Checking the team's progress in real time.

4. Conclusions

The idea behind mobile applications development frameworks that allow a develop once to deploy on several platforms has great potential. In practice, current frameworks, still suffer from limitations that prevent developers from exploiting the different platforms features in an efficient way. The big differences that exist between systems, limits the features that such APIs can provide, by having to search for the common features between them. When compared to what is possible to develop with a native application, we can see that applications developed resorting to these frameworks have to be limited in their scope. Additionally, by being executed in a middleware layer that is not native to the operating system, the performance of these applications suffers when compared to native applications.

In spite of these disadvantages, it is nevertheless possible to develop applications that go beyond a simple web application, relying on features specific to the mobile device, as is the case of its geo referencing capabilities. We developed such an application, and this paper describes its features and the limitations that had to be overcome in order to produce an acceptable prototype. The major limitation detracting from the user experience that we had to face, is related to limitations imposed by Android's sandbox model, which prevented access to the file system. This resulted in the need to store all media files in a central repository. While for game play the impact on the user experience is limited (taking longer to load the media file when compared to a local access), the impact on the clue building experience is more pronounced, since it is impossible to associate directly to a clue the media file captured by the mobile device. An upload of the file to the central repository is required. This operation has to be conducted outside the application, thus detracting from the user experience.

It is our belief that while the limitations pointed out in this paper are not overcome and the number of features made available when compared to what is available for native applications does not increase, the adoption of such development frameworks will be limited by independent developers.

References

[1] D. Gavalas, D. Economou, Development platforms for mobile applications: Status and trends, IEEE Softw. 28 (2011) 77–86. doi:http://dx.doi.org/10.1109/MS.2010.155.

[2] S. Blom, M. Book, V. Gruhn, R. Hrushchak, A. Köhler, Write once, run anywhere a survey of mobile runtime environments, in: Proceedings of the 2008 The 3rd International Conference on Grid and Pervasive Computing - Workshops, IEEE Computer Society, Washington, DC, USA, 2008, pp. 132–137. doi:10.1109/GPC.WORKSHOPS.2008.19.

[3] Vodafone Mobile Application Development, http://developer.vodafone.com/develop-apps/widgets/getting-started/.

[4] JIL 1.2.2 API Overview, http://developer.vodafone.com/develop-apps/widgets/getting-started/jil-122-api-overview/.