Magni - A Framework for Developing Context-aware Mobile Applications

Ricardo Queirós[1], Filipe Portela[2], and José Machado[2]

[1] ESMAD, Polytechnic of Porto, Portugal
[2] Algoritmi Research Centre, University of Minho, Portugal.

Abstract. The advent of Internet and ubiquitous technologies has been fostering the appearance of intelligent mobile applications aware of their environment and the objects nearby. Despite its popularity, mobile developers are often required to write large and disorganized amounts of code, mixing UI with business logic and interact, in a ad-hoc fashion, with sensor devices and services. These habits hinder the code maintenance, refactoring and testing, while negatively influencing the consistency and performance of mobile applications. In this paper we present Magni as an abstract framework for the design and implementation of personalized and context-aware mobile applications. The corner stone of the framework is its architectural pattern based on the Model-View-Presenter pattern in the UI layer relying in REST services the majority of the app features. This paradigm fosters the modular design, implementing the separation of concerns concept and allowing an easier implementation of unit tests. In order to validate the framework, we present a prototype for an healthcare automotive app. The main goal of the app is to facilitate the access to health related points of interest such as hospitals, clinics and pharmacies.

Keywords: Web services, Design patterns, Mobile frameworks, Geolocalization, Interoperability, Mobile healthcare, Automotive apps

1 Introduction

The increased use of mobile devices and their ubiquitous facet, fostered the design of context-aware applications that make use of data collected from the sensors' device and location services. Based on this data, a context-aware app can better understand users current situations, and use this information to provide optimized and customized experiences.

These type of mobile apps are difficult to implement for two reasons: 1) the challenges of their ubiquitous facet and 2) the absence of a widely accepted programming model.

In the former, developers must understand that these ubiquitous computing environments pose challenges regarding communication and interaction due to constraints such as dynamically changing network addresses and system configurations, susceptibility to disconnection and low bandwidth [1].

The later is even more problematic. Despite the existence of multiple context-aware mobile applications, they remain difficult to develop, with no widely accepted programming model available. In most cases, developers find themselves mixing UI aspects with logic and data layers, with no sense of design patterns. Other developers write a large amount of code locally to recreate existent services, increasing redundancy and harming code maintenance and testing.

In order to facilitate the development of such applications, several frameworks appeared in recent years to help developers to abstract, modularize and optimize their work. Some frameworks [2], [3], [4] focus on a specific platform (Android) rather being a multi-platform framework. Other approaches aim to provide location-specific information and services [5], [6], [7] through a central information system, which is responsible for storing, maintaining and communicating all location specific data. Finally, other frameworks [8], [9] are built as a layered architecture in order for portions of application components to be adapted based on current contextual information. None of these frameworks is concerned with software design patterns. In addition, none of them abstracts the use of ubiquitous platforms such as wearable and automotive platforms.

This paper presents Magni – a framework for developing context-aware mobile applications. Magni suggests the use of software design patterns for code organization in the UI layer relying on Web services for the majority of the app supported features. These patterns foster automated unit testing and the separation of concerns in presentation logic, improving the consistency of the application. Based on this framework, we created an automotive app prototype called MyHealth for a healthcare case study.

This work is organized as follows. Section 2 discusses some key concepts on context-aware frameworks such as the software patterns used, the mobile platforms supported and the domains adequacy. In Section 3, we present the Magni framework that was designed to help programmers in the development of context-aware mobile applications. Section 4 evaluates the proposed framework through the creation of a prototype for an healthcare case study. Finally, we enumerate the main contributions of this work and future directions.

2 Mobile framework facets

Mobile frameworks have multiple facets. In this section we detail their design patterns, supported platforms and domains.

2.1 Software Design Patterns

Nowadays, the majority of mobile applications use an ad-hoc Model-View architecture. In this context the components of the UI (e.g. Activities or Fragments in Android) implement logic, handle UI aspects and control the flow between data objects and UI. Using this approach, UI programmers end up with complex and massive monolithic code where everything is connected to everything and the work between the programmers team is chaotic.

In software engineering, a software design pattern is an abstract, general and reusable solution to a commonly problem. It cannot be transformed directly into source code, instead, it describes how to solve a problem. Thus, design patterns are formalized best practices that provide programmers with the tools needed to solve common problems when designing an application [10]. Other important advantage of the use of design patterns is that it decouples development allowing multiple developers to work simultaneously.

There are several software design patterns for implementing user interfaces on computers, the most popular are: Model-View-Controller (MVC), Model-View-Presenter (MVP) and Model-View-Viewmodel (MVVM).

MVC was introduced in the 1970s. Although originally developed for the UI layer, MVC has been widely adopted as an architecture for WWW applications through several Web frameworks. MVC pattern divides an application into three major components: Model, View, and Controller. The model component manages the business and the data model. It also defines the business rules for data defining how the data can be manipulated. The View represents the user interface components (e.g. Activity/Fragments in Android, HTML/CSS in Web) responsible for the data visualization received from the controller. The controller is the mediator between the View and the Model. It gets the input from users via the View, then processes the user's data through the Model, passing back the results to View.

MVP is an evolution of MVC, wherein the controller is replaced by the presenter. The Presenter is responsible for addressing all user interface events on behalf of the view. The View and the Presenter are completely separated, unlike View and Controller, and communicate to each other by an interface. The Presenter also doesn't handle the incoming request traffic like controller.

MVVM pattern supports two-way data binding between View and View-Model. This fosters automatic propagation of changes between ViewModel and the View. Generally, the ViewModel uses the observer pattern to inform changes in the ViewModel to the Model.

Regardless of the pattern choice, the most important is to use a software design pattern during the design and implementation of a mobile application.

2.2 Platforms

Mobile apps are not confined only to smartphones and tablets. In fact, other mobile platforms appeared in recent years, synchronized with the technology and economy evolution of societies. Some important examples are multimedia devices (e.g. TVs), automotive devices (e.g. cars, plains) and wearable devices (e.g. watches, fitness trackers).

One emergent industry is the automotive sector. In the United States, 17.5 million vehicles were sold in 2015, and expect to reach 20 million in 2019. Sales have also improved in the European Union since the financial downturn. In 2015, new car registrations in the E.U. hits the 12.6 million units.

In fact, the ubiquity of the car, plus the simultaneous convergence of information and communication technology with both the automobile and some

industries (e.g. healthcare, tourism), now provide some potentially promising opportunities for linkups. Based on these facts, automotive companies are investing in the mobile experience seeking to create a more rich connected car experience which means including 4G/LTE hotspots in a car so mobile devices can connect to the Internet or creating a closed integration through mirror frameworks (e.g. Android Auto for Android and Apple CarPlay for Apple) allowing drivers interact with their mobile devices from the car's Head Unit.

2.3 Domains

There are several domains where context-aware mobile apps can act. The predictable choice would be tourism where apps can help tourists to easily locate and access points of interest based on the user's location. Other important sector is healthcare. The health care industry incorporates several sectors that are dedicated to providing health care services and products.

Mobile Health (ou mHealth) as been defined as a reference to using mobile communication devices, such as mobile phones and tablet computers, for health services. A growing percentage of health-related smartphone apps are available, and some estimates predict 500 million patients will be using such apps by the year 2015 [11]. Merging with the previous section, the health and automobile sectors account for an annual 2.7 trillion and 1 trillion dollars respectively in the United States. Both industries are looking at various ICT-oriented solutions toward a "smart-health-oriented" car. With this in mind, the first Smart Seating applications were mainly related with improving in-car sitting posture. Beyond that, the automotive industry is now transforming the car into a connected component to healthcare and wellness services, with apps covering such areas of health as monitoring diabetes and the drivers heart rate.

3 The Magni Framework

This section presents a proposal for a context-aware mobile framework called Magni. The Magni framework is an abstract framework that can be used by developers to create well organized and sophisticated code based on software design patterns. We believe that the use of these patterns will improve code maintenance and test and create consistent apps. Magni relies all typical features of these kind of apps to a network of services (e.g. authentication, location, storage). In the following subsections the framework is described based on two models: architectural and integration models.

3.1 Architectural model

The Magni framework is the basis for the design and implementation of Magni instances as realizations of the framework for specific domains (e.g. healthcare, tourism, computer programming learning).

The Magni framework suggests the use of a client-server architecture in which presentation, application processing, and data management functions are physically separated as a multitier architecture (e.g. three-tier architecture). In this architecture, Magni emphasizes the UI layer, decoupling tasks through the Model-View-Presenter (MVP) design pattern (Figure 1). MVP is a user interface architectural pattern created to facilitate automated unit testing and improve the separation of concerns in presentation logic. Thus, a Magni instance (mobile app) will be organized in

- The **View** is a passive interface that displays data (the model) and routes user commands (events) to the presenter to act upon that data.
- The **Presenter** acts upon the model and the view. It retrieves data from repositories (the model), and formats it for display in the view. The Presenter is responsible for addressing all user interface events on behalf of the view. It receives input from users via the View, then process the user's data through the Model that passes the results back to the View.
- The **Model** is an interface defining the data.

Unlike the MVC pattern, in this model the View and the Presenter are completely separated, from each other and communicate to each other by an interface. This design pattern allows also the systematic use of abstraction mechanisms supporting a variety of implementation options.

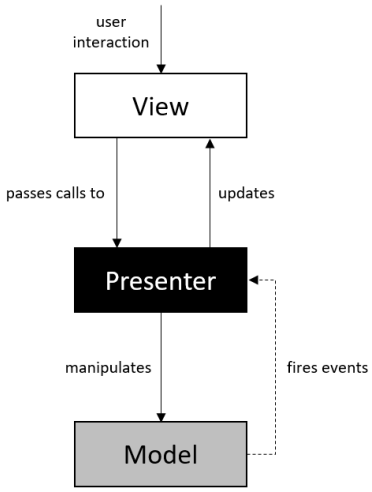

Fig. 1. Magni framework pattern UI Model

3.2 Integration Model

The Magni specification also comprises an integration model. This model recommends specifications for the communication between the Magni instance and the

services. A Magni instance can include several features related with authentication, location awareness, places discovering, sensing, rating, gamification, and many others. Instead of implementing all theses features, the Magni framework suggest that instances should stay small (thin clients) and rely all the "hard work" on services in the cloud. Services in the framework are organized in two groups based on its importance: **core and secondary**.

Core services are mandatory services that a context-aware app should have. For instance, one of the unique features of mobile applications is location awareness. Mobile users take their devices with them everywhere, and adding location awareness to a mobile app offers users a more contextual experience, with automated location tracking, geofencing, and activity recognition.

Secondary services are complementary services that complement the core services in a specific task, although its absence does not alter the execution flow of a feature process. Usually these services do not have graphical interfaces and are more specialized than the core services. An example of this kind of services is an adaptation service. Taking the previous example, an adaptation service could adjust the presentation order in accordance with the effective proximity of the points of interest. Another example of a secondary service is a social media service that resides on the cloud and can be used to integrate social features from a Social Media Platform (SMP) such as Facebook or Twitter in the Magni framework. In this context, a social service could set/get information to/from social networks.

This integration model relies on web services for communication among systems. Web services can be used mostly in two flavors: SOAP and REST. SOAP web services are usually action oriented, mainly when used in Remote Procedure Call (RPC) mode and implemented by an off-the-shelf SOAP engine such as Axis. Web services based on the REST style are object (resource) oriented and implemented directly over the HTTP protocol mostly to put and get resources. Both specifications have matured in distinct periods and they coexist nowadays: SOAP started earlier and now the trend is REST as stated from a directory of 3200 web APIs listed at ProgrammableWeb (http://www.programmableweb.com/). Regardless of these trends, the Magni specification does not encourage the use of any flavor in the communication specifications detailed in the following subsections. As far as possible, Magni tries to keep an equidistant position from both flavors.

This section analyses the communication specifications for the interaction with three core services typically found in context-aware apps: **authentication/storage services, location services and social services**.

Authentication/storage services User authentication is an important requirement for most mobile apps today. By being able to securely authenticate users and identify them your app can offer users a customized experience based on their interests and preferences. At the same time, data persistence is crucial. Despite some data can be stored in the device itself (e.g. user preferences, binary files and small databases emulating caching features), other data should

be stored remotely and centrally. In order to not reinvent the wheel, many apps rely on cloud services for this bureaucratic work.

A Backend-as-a-service (BaaS) is a cloud computing service model acting as a middleware component that allows developers to connect their Web and mobile applications to cloud services via application programming interfaces (API) and software developers kits (SDK). BaaS features include cloud storage, push notifications, server code, user and file management and user authentication and management as well as many other backend services. These services have their own API, allowing them to be integrated into applications in fairly simple way. One impressive example is the Firebase BaaS that offers free services such as analytics, crash reporting, user authentication, and cloud messaging.

Location services One of the unique features of mobile applications is location awareness. Mobile users take their devices with them everywhere, and adding location awareness to a mobile app offers users a more contextual experience. For instance, the location APIs available in Google Play services facilitate adding location awareness to mobile apps with automated location tracking through the Location API. Other features are supported by this API, such as the geofencing and activity recognition. The former combines the awareness of the user's current location with awareness of the user's proximity to locations that may be of interest. In this case, it is possible to define and adjust the proximity for the location, through coordinates (latitude and longitude) and a radius. These data define a geofence, creating a circular area, or fence, around the location of interest. For each geofence, the Location Services send entrance and exit events, or it can be specified a duration within the geofence area to wait, or dwell, before triggering an event. The later offers developers a powerful tool to augment the user experience. By getting information about the user's activity, apps can make intelligent decisions catering the application experience. For example, by asking if the user is starting to exercise so you can keep track of it with a fitness app (e.g. Google Fit), or preventing notifications from being sent when the user is driving.

Social media services OpenSocial is a set of three programming interfaces (APIs) for web-based social network applications created by Google. The OpenSocial API covers a broad range of capabilities such as, profile information (user data), relationships information (social graph) and activities (e.g. news feed). The main goal of OpenSocial is to provide a common framework to be used by developers to guarantee interoperability across several social networks on the Internet, which act as containers for each OpenSocial-compliant application. This specification provides a REST and RPC API communication flavours through which OpenSocial-compliant applications and containers interact with each other, transmitting user data, friend lists and activities. These protocols support various data exchange formats such as XML, JSON and ATOM. In order to authorise access to data stored in social networks, these APIs rely on the OAuth specification. The lack of adoption by major players such as Facebook

affects negatively the OpenSocial adoption. In order to get around with this issue other alternatives can be used. A well known approach is to build API wrappers that map the OpenSocial API to the native APIs. In 2015, W3C and the OpenSocial Foundation announced that OpenSocial standards will take place in the W3C Social Web Working Group, of which the OpenSocial Foundation is a founding member.

4 Evaluation

In order to validate the framework, we instantiate it through the creation of a prototype for a health-care case study called **MyHealth**.

MyHealth is a healthcare automotive app whose main goal is to facilitate the access to points of interest related to health (e.g. hospitals, clinics, pharmacies). Through a map and GPS, the user can see, in the car, available POIs nearby (sorted by various filters) and make the choice through voice recognition. The application will suggest the best directions and, using a rating functionality, the patient can post and access the feedback from other patients, ensuring quality and community building, while helping to improve the service.

Based on the framework, MyHealth relies all the authentication and storage stuff in the Baas Firabase. In order to add location awareness it uses the Location API from Google Play Services. Finally, the integration with Facebook and Twitter is ensured with the OpenSocial API. Figure 2 illustrates the integration model of the automotive app:

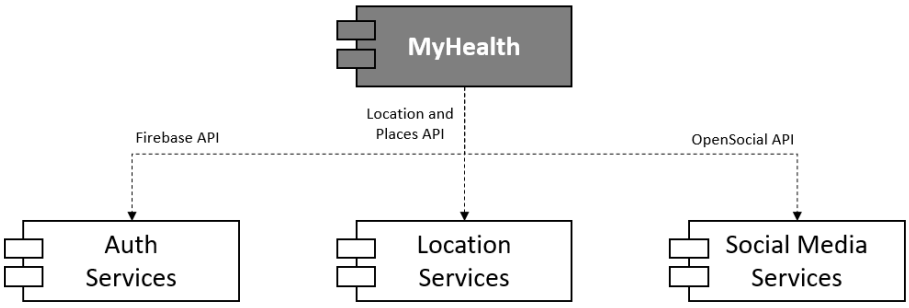

Fig. 2. MyHealth integration Model

Figure 3 illustrates the mockup of the app's main view:

5 Conclusion

This paper presents Magni as a framework for the development of context-aware mobile applications. Magni differs from current frameworks by its emphasis on

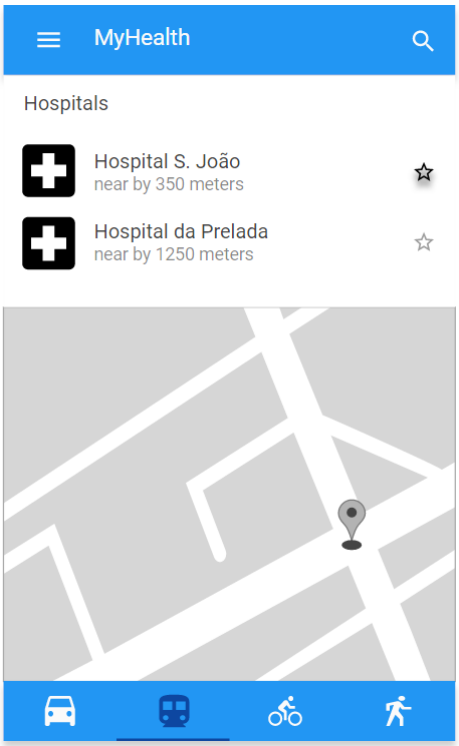

Fig. 3. Mockup on the main view of the MyHealth app

software design patterns and abstraction to different platforms. The architectural model of Magni is based on the MVP pattern in the UI layer and the remain components are accessed via REST services. This type of model will potentiate the maintenance and testing of the app's code. In order to validate the framework, an instance of Magni was created for the healthcare domain. The validation is still unfinished since only the mockups were created. Thus, the future work will be, to finish the prototype. In addition, there are other projects, in the tourism domain, that can be used to test and tune the framework.

References

1. G. H. Forman and J. Zahorjan: The challenges of mobile computing Computer 4, 38–47 (1994)
2. Lillian B. R. de Oliveira and Antonio A. F. Loureiro: CodeDroid: A Framework to Develop Context-Aware Applications MOBILITY – The First International Conference on Mobile Services, Resources, and Users (2011)
3. Bart van Wissen, Nicholas Palmer, Roelof Kemp, Thilo Kielmann, and Henri Bal: ContextDroid: An Expression-Based Context Framework for Android In PhoneSense (2010)
4. Alf Inge Wang and Qadeer Khan Ahmad: IASTED – International Conf. on Software Engineering and Applications (2010)
5. Tummala, H., Jones, J.: Developing Spatially-Aware Content Management Systems for Dynamic, Location-Specific Information in Mobile Environments 3rd ACM international workshop on Wireless mobile applications and services on WLAN hotspots, Mobility support and location awareness pp. 14–22, ACM, Cologne, Germany (2005).
6. Lpez-de-Ipia, D., Vazquez, J.I., Abaitua, J.: A Context-aware Mobile Mash-up Platform For Ubiquitous Web 3rd IET International Conference on Intelligent Environments pp. 116–123, IEEE, Ulm, Germany (2007)
7. Challiol, C., Rossi, G., Gordillo, S., De Cristfolo, V.: Designing and Implementing Physical Hypermedia applications ICCSA 2006, UWSI 2006 pp. 148–157, Springer Berlin / Heidelberg (2006)
8. William Van Woensel, Sven Casteleyn and Olga De Troyer: SCOUT: A Framework for Personalized ContextAware Mobile Applications ICWE 2009 Doctoral Consortium (2009)
9. Williams, Elizabeth and Gray, Jeff: Contextion: A Framework for Developing Context-aware Mobile Applications Proceedings of the 2Nd International Workshop on Mobile Development Lifecycle 27–31 (2014)
10. Martin, Robert C.: Design Principles and Design Patterns (2000)
11. Adibi, Sasan, ed.: Mobile Health: A Technology Road Map Springer (2015)