Towards a Collaborative Framework for the Design and Development of Data-Intensive Mobile Applications

Mirco Franzago, Henry Muccini
DISIM Department, University of L’Aquila, Italy
{mirco.franzago | henry.muccini}@univaq.it

Ivano Malavolta
Gran Sasso Science Institute, L’Aquila, Italy
ivano.malavolta@univaq.it

ABSTRACT

Guidelines and best practices on how to design and develop mobile applications are being periodically released by mobile OS vendors, mobile developers, and researchers. Still, a framework that collects and integrates them in a simple, holistic, and automated approach is missing.

This work proposes a modelling framework supporting the collaborative design and development of data-intensive mobile applications. By using Model-Driven Engineering techniques, we define four modelling languages covering the main concerns coming from the mobile app development domain; the framework supports the analysis of models and the automated synthesis of executable mobile applications for multiple platforms. This paper provides an overall view of the modelling framework, and highlights its main features for both technical and non-technical stakeholders.

Categories and Subject Descriptors
D.2.2 [Design Tools and Techniques]; D.3.4 [Processors]

General Terms
Design, Languages

Keywords
Mobile Applications, MDE, Code Generation

1. INTRODUCTION

As we all know, the use of mobile devices and mobile applications is growing at an exponential rate. Still, mobile applications are developed with ad-hoc development processes, and on-paper best practices [8]. Best practices for app design and development have been released by mobile platform vendors, the W3C consortium, and various practitioners and researchers. Still, what the community is missing is a systematic approach, and related automated tools, that may translate the known guidelines into automated tasks that developers can run in order to develop high-quality apps.

In this paper we propose a collaborative framework for the design and development of data-intensive mobile applications. With this scope in mind, our framework makes use of Model-Driven Engineering technologies for supporting the multi-view modelling and development of mobile applications. The framework proposes four modelling languages, specific to the domain of mobile applications development. Thus, in contrast with generic languages such as UML, they provide mobile-specific concepts as first class entities (such as, mobile app views, specific UI patterns, sensorial actions and events, etc.), making the modelling activity more intuitive and accessible to all involved stakeholders; also, our languages have been designed as to be more concise as possible, opening for a light-weight real-time collaboration among stakeholders throughout the whole mobile application lifecycle. To improve the design experience, our modelling framework is implemented as a web application, thus avoiding the use of heavy weight desktop environments.

This paper describes how we engineered the framework and the long-term features we are planning to realize. The rest of the paper is organized as follows. In Section 2 we describe the data-intensive mobile applications domain and some pre-conditions for our research plan. Section 3 discusses the proposed framework. Finally, Sections 4 and 5 present related work and conclusions, respectively.

2. BACKGROUND

2.1 Data-intensive mobile applications

This research is concerned with enhancing the design and development activities of data-intensive mobile applications, i.e., those applications whose primary purpose is to present a large amount of content to a variety of possible users. To further scope data-intensive apps, and by building on the definition of data-intensive web-sites [1], a data-intensive mobile app differs from other mobile apps because of their:

- support for delivering content to multiple devices;
- focus on browsing collections of data and basic interactions with data items, with simpler functional requirements;
- focus on information organization and navigation design where users can directly understand the structure of the mobile app;
- support of one-to-one content delivery, where each user must have the impression of interacting with an interface specifically tailored to her needs and preferences;
- simpler transactional requirements, in most cases limited to high-performance read-only access and standard write operations of a well-delimited part of data.
We focus our research on data-intensive mobile applications since: (i) by following the characterization provided above, data-intensive mobile applications represent the majority among the apps in the main app stores, including, e-commerce, content sharing, financial and accounting services, business to business services, etc.; (ii) all applications belonging to this family share the same traits and requirements, thus enabling our solutions (e.g., components and UI patterns) to be effectively reused across projects and organizations.

2.2 Design Issues and Challenges

Current practices in data-intensive mobile applications development are still plagued by a number of recurring issues and challenges [8]. In order to have a clear understanding of those issues, we are working closely with industry partners and we are continuously performing informal interviews with developers\(^1\), projects managers and other involved stakeholders for completing our knowledge about this application domain. Many of the identified issues have a technical nature however, from our experience and interactions with professionals in the field, we actually noted that issues and challenges pertaining to the design of a mobile application can have a much bigger and disastrous impact on its success [4]. In the following we elaborate on this by reporting the following main issues and challenges (constituting a re-organized subset of those reported in Section 1).

**Limited reasoning in the context.** During the design and development of a mobile app it is fundamental to reason on how it will behave in the real context of use. Nevertheless, important aspects about usage profile and specific environmental constraints are often ignored. This problem leads to poorly designed applications, or even to applications that do not add any relevant value to the end-user (who may have paid for obtaining them).

**Limited information architecture engineering.** Information architecture engineering can be defined as the process of reasoning about information structure, its visual layout, how the user can participate and understand presented information, and how navigation and access are facilitated [4]. It is evident that information architecture is a key factor of data-intensive mobile apps. Many stakeholders still tend to borrow concepts and best practices about information architecture and user experience of traditional (web) applications. However, working on a mobile app is totally different from working on a desktop program or website: the smaller display, different styles of user interaction, and contextual dependencies have a major impact on interaction design for mobile applications, which in turn has a strong influence on application development [8].

**Stakeholders diversity.** Many stakeholders of a mobile project have very different backgrounds and technical knowledge. For example, typical stakeholders include designers, developers, users, customers, etc. Stakeholders diversity leads to mismatches either in the used language, understanding of the application domain, or technical knowledge. Those differences, specially between technical and non-technical stakeholders, lead to poorly inspired discussions and additional design and development iterations, thus resulting in an unforeseen waste of resources, budget and time.

In order to overcome the above mentioned problems, we carry out a top-down reasoning process with the chief aim of specifying a design and development framework that can satisfy the needs of data-intensive mobile applications stakeholders. In the next section we will present our framework.

3. THE FRAMEWORK

We propose to build our modelling framework by adhering to the Model-Driven Engineering (MDE) paradigm. By building on MDE, our framework allows mobile apps stakeholders to: (i) focus on models rather than on the source code of the application being developed, thus abstracting from the complexity of the source code of the app (this is especially useful for non-technical stakeholders), (ii) make design decisions independently from implementation-specific concerns; (iii) easily communicate with each other since models are specific to the domain of data-intensive mobile applications; (iv) exploit the well-known principle of multi-viewpoint specification, in which the mobile app is represented by a set of independent models and correspondences among them. The latter is an elegant solution for managing complexity since it allows each stakeholder to focus on those aspects of the app in which he is expert, abstracting from the rest of the concerns.

In order to reason in an organized manner on our framework, we identified the main viewpoints of a data-intensive mobile project: navigation refers to the navigation flow and the logical structure of the mobile application, content refers to the structure and operations of the data managed by the application, UI refers to the presentation, layout and graphical style of the application, and business logic mainly refers to the internal behaviour of the application and the interaction between the user and the app user interface. In the following we describe the main principles of our framework.

**Multi-viewpoint modelling.** We defined a dedicated modelling language for each viewpoint prescribed in the previous section. Figure 1 shows how models, their inter-model correspondences, and the application source code relate to each other in our framework. More specifically, the navigation model represents the structure of a data-intensive mobile application in terms of its views and the navigation flow between them, the data model focuses on the structure of the various data types of the mobile app in a platform-independent manner, the UI model describes the layout of each view of the mobile application, together with the UI elements composing it, and the business logic model describes the internal operations of the application and the interaction between the user and the mobile application interface; this language is based on the event-condition-action paradigm, with mobile-specific concepts like GPS sensor or camera, a gestures on the UI, and so on. In this context, the main driver is to manage complexity by the well-known principle of “separation of concerns”; the intuition is that if a stakeholder has a concern framed by viewpoint \(v_x\) (e.g., user interface design), then he needs to work only on the corresponding model \(m_x\) (e.g., UI model), thus focusing on the details of a few things at a time.

**Platform independence.** The modelling languages are platform-independent, they have been designed so that models are valid for each kind of concrete platform the to-be mobile application will be deployed in (e.g., Android, iOS, Windows Phone, etc.). This is important for abstracting implementation-specific details to non-technical stakeholders, and for the validity of the proposed framework even when new mobile platforms will be available in the future.

---

\(^1\)One of the authors is a mobile apps developer with more than twenty projects in his portfolio.
Reuse. Reusing already implemented components helps customers in reasoning on already-known solutions, designers in exploiting UI layouts and user experiences with an already known user feedback, developers in producing new systems by building on already-developed and tested parts of the mobile application, etc. In this context, it is fundamental for us to allow all stakeholders to reuse the model of a specific part of a mobile app across projects and even organizations. In order to support this principle, our modelling languages are (loosely) coupled by means of typed correspondences (represented as dotted links in Figure 1) linking the various concepts across models in a non-intrusive manner. In so doing, each model is self-representative, it does not directly depend on the other models of the application, and can be re-used across projects and organizations. Examples of correspondences include those linking: a view in the navigation model to its corresponding main container in the UI model, an action in the interaction model to the corresponding CRUD operation in the data model to be performed when the action is executed, etc.

Figure 1: Collaborative design

Integration with source code. In our framework, models can be used to produce the source code of the mobile application being designed (see the central part of Figure 1). More specifically, code generation is performed by a pool of synthesizers. We define a pool of synthesizers as a quadruple \( <s_{\text{nav}}, s_{\text{data}}, s_{\text{int}}, s_{\text{ext}} > \) targeting a specific platform (e.g., Cordova 2.x, Cordova 3.x, Android, iOS). Each synthesizer takes as input a model representing a specific view of the application and produces the corresponding source code of the mobile application according to the target platform. The modelling framework is independent from the synthesizer pools used for generating the source code of the application, and it does not constrain stakeholders to target any specific platform. Rather, it will be up to a stakeholder with administrator privileges to decide which synthesizer pool must be used for the current project.

Our framework comes with a default synthesizer pool targeting Apache Cordova (http://cordova.apache.org), an open-source platform for building hybrid mobile applications using HTML, CSS and JavaScript. It gives us numerous advantages, among all it helps in managing one of the most recognised issues in mobile apps development: portability [8]. Finally, the source code produced by the execution of a synthesizer is automatically committed to a shared software versioning system (currently, SVN) so that app developers and UI designers can get the most updated version of the code just by performing an update operation from their machines.

Extensibility. Even if the proposed modelling languages have been designed to cover the stakeholder concerns of data-intensive mobile applications development, we are aware that in some situations stakeholders may need to adapt the framework to new requirements, technological platforms, or constraints coming from the application domain. Thus, we defined the proposed framework (and its underlying implementation) with extensibility in mind. Our framework provides three extension points for allowing third-party plugin developers to extend the framework with either (i) a new modelling language for describing a novel aspect of the mobile application, (ii) a new synthesizer for allowing the generation of code compliant with a new target platform (e.g., Tizen or Firefox OS), or (iii) new concepts within each modelling language, such as login action in the business logic model, a new type of UI pattern in the design model, etc.

The implementation of the framework knows at run-time which plugins are available and automatically adapts its user interface and the set of facilities available to stakeholders.

Real-time collaboration. Design is a team sport, especially in the field of mobile applications development where each stakeholder in the team has very different technical knowledge and background; thus each stakeholders’ perspective can surely help other team members see things they would not ordinarily see [3]. In light of these considerations, our framework allows all stakeholders to work on the models of the mobile application in a collaborative fashion. Figure 2 shows the collaboration mechanisms of the framework, they work at two different levels, namely: Intra-model collaboration: when working on any model of the app, each stakeholder sees in real-time all the modifications performed by other stakeholders on the same model. Basically, stakeholders can perform live concurrent editing on the various models of the application. This is achieved by exploiting a particular class of algorithms for multi-site real-time concurrency, called operational transformation [7]. Intuitively, operational transformations can be seen as a real-time SVN where users’ operations are very light-weight and support eventual consistency with any amount of lag. According to our framework, different stakeholders can work on a common version of the models stored on a central server. Over time, atomic editing operations are recorded and then propagated through a bidirectional communication channel to the pool of stakeholders working on the same model. Basically, an atomic editing operation represents an addition, removal, or update of any element within the model and it is sent to the central server as a model ”difference” to be applied on all the other local models currently accessed by all the other stakeholders.

Inter-model collaboration: when a model has reached a certain stability, a stakeholder can perform a “push” operation of the model. A push operation has two main consequences: (i) the synthesizer corresponding to the language of the model is executed in order to suitably update the source code of the mobile application, and (ii) a notification is sent to all the other stakeholders working on other models of the application. Executing the synthesizer allows the continuous alignment of the models with the source code of the application, so that (i) developers and UI designers always work on the latest version of the app with respect
to its models, and (ii) stakeholders working on the models can immediately see the impact of their change towards the implementation of the application. The main motivation for sending the notification to all stakeholders is that other models of the application can become disaligned with respect to the model being pushed (e.g., deleting a certain entity in the data model may result in a label in the UI model expecting a value of an attribute belonging to the deleted entity). In light of this, stakeholders working on other models of the application must know that they may need to adapt either the model or the correspondences between the pushed model and the one they are working on.

Figure 2: Collaborative development process

Tangibility. When reasoning on the design of a mobile app, it is of paramount importance to develop prototypes of the application to allow all stakeholders to try it via a tangible instrument. In our framework, we support this by means of automatic on-device prototype deployment (see Figure 2). At any time, each stakeholder can trigger the deployment of the source code of the modelled application into a dedicated container application running on their device. By doing this, stakeholders can try out the mobile application they are working on in a more tangible manner, they can learn from the direct feedback from users using the application, and they can exercise the mobile application in its real context of use (this is fundamental for mobile applications). The container app is generic and it is totally independent from the application being deployed. This is a peculiarity of hybrid applications since, being JavaScript interpreted, the framework needs just to send the new resources and integrate them in a suitable way in the container app.

5. CONCLUSIONS

This paper presents the main principles and modelling languages of our modelling framework for mobile applications. The main features of our modelling framework are: i) it enables the design of apps through well-known concepts and facilities at the right level of abstraction, ii) it supports multi-site, real-time modelling across the different stakeholders involved in a mobile project, iii) it abstracts from specific technologies so to maximize reuse, while supporting the mobile application deployment into various platforms, and iv) it enables an early validation of design decisions through incremental prototypes and analysis techniques.

6. REFERENCES


