The Journal of Systems and Software 119 (2016) 31–44

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Software architectural principles in contemporary mobile software: from conception to practice

Hamid Bagheri[a,*], Joshua Garcia[a], Alireza Sadeghi[a], Sam Malek[a], Nenad Medvidovic[b]

[a] School of Information and Computer Sciences, University of California, Irvine, United States
[b] Computer Science Department, University of Southern California, United States

ARTICLE INFO

Article history:
Received 3 May 2015
Revised 23 May 2016
Accepted 25 May 2016
Available online 1 June 2016

Keywords:
Software architecture
Android
Architectural styles

ABSTRACT

The meteoric rise of mobile software that we have witnessed in the past decade parallels a paradigm shift in its design, construction, and deployment. In particular, we argue that today's mobile software, with its rich ecosystem of apps, would have not been possible without the pioneering advances in software architecture research in the decade that preceded it. We describe the drivers that elevated software architecture to the centerpiece of contemporary mobile software. We distill the architectural principles found in Android, the predominant mobile platform with the largest market share, and trace those principles to their conception at the turn of century in software architecture literature. Finally, to better understand the extent to which Android's ecosystem of apps employs architectural concepts, we mine the reverse-engineered architecture of hundreds of Android apps in several app markets and report on those results.

1. Introduction

Mobile computing has come a long way from a decade ago. Development of mobile software used to be an art exercised by a few, savvy, experienced developers, capable of hacking low-level C code—the *lingua franca* of mobile software at the time. The resulting software systems were often monolithic, rigid, one-off programs, which were hard to construct, understand, and maintain (Picco et al., 2014). Although software architectural principles had found widespread use in structuring the traditional desktop software at the turn of century (Taylor et al., 2009), mobile software was often devoid of such structures (Picco et al., 2014; Medvidovic et al., 2003).

The dominant preconception was that for developing efficient software, suitable for deployment on resource-constrained mobile platforms, it is necessary to compromise on flexibility and decoupling achieved through architectural principles, such as decomposition of a software system into components, separation of communication links in the form of connectors, and so on Medvidovic et al. (2003); Malek et al. (2007). In particular, programming-language abstractions needed for the realization of those architectural concepts were deemed unsuitable for use in mobile software.

Today's mobile software, however, differs greatly from that of a decade ago (Wasserman, 2010). Our empirical investigation—the details of which are described in Section 5—shows that software architecture plays a significant role in the development of modern mobile software. Many of the ideas devised in pioneering software architecture work, developed around the turn of this century, have found a home in the contemporary mobile software. In particular, Android, which is the predominant mobile platform, realizes many of the architectural principles previously advocated by the software-engineering community.

At first blush, one may conjecture that the increasing prominence of software architectural principles is a natural progression of software-engineering practices in any computing domain. But when we look at other closely related areas of computing, such as embedded software, we do not find a similar adoption of software architectures. It is, thus, important to understand the drivers behind the rapid adoption of software architectures in mobile computing, as well as the nature of the adopted architectural concepts and principles, and how their use has impacted the development of mobile software.

To that end, we first describe several requirements that drove the adoption of many of the architectural principles advocated in the literature in modern mobile software development. We also trace back those principles to their conception in the pioneering software-architecture research, in particular the research on the

* Corresponding author.
E-mail addresses: hamidb@uci.edu (H. Bagheri), joshug4@uci.edu (J. Garcia), alirezs1@uci.edu (A. Sadeghi), malek@uci.edu (S. Malek), neno@usc.edu (N. Medvidovic).

http://dx.doi.org/10.1016/j.jss.2016.05.039
0164-1212/© 2016 Elsevier Inc. All rights reserved.

applicability and benefits of architecture-based design and development in a mobile setting. Afterwards, we present some of the key architectural concepts found in Android, often codified in its *application development framework*, which provides programming-language constructs for architecture-based development of mobile software, including support for the realization of software components, connectors, events, configurations, and architectural styles.

We argue that software architectural support in Android has played a key role in its meteoric rise and success. Our empirical observation corroborates the notion that architectural building blocks in Android are supported in the Android platform's programming constructs, which promises to dramatically improve an app developer's productivity, and also makes it much easier to develop complex apps without a formal education in programming, or previous programming experience. This, in turn, supports even novice programmers in developing sophisticated apps with the potential of becoming popular in app markets, such as Google play. Codification of an architectural family and separation of communication from the application logic, using asynchronous connectors, have facilitated the integration of third-party code in a mobile device, thereby directly spawning a vibrant ecosystem of apps.

The contributions of this paper can be summarized as follows:

- Identifies the drivers behind the rapid adoption of software architecture concepts and principles in contemporary mobile software, specifically Android.
- Distills the architectural principles found in Android and illustrates them using a popular mobile app.
- Traces back those principles to their conception in software-architecture research.
- Reports on the characteristics of architectures found in the Android ecosystem of apps by mining hundreds of Android apps in several app markets.
- Reflects on deviations from how architectural concepts have been prescribed in architecture literature and the manner in which Android has realized some of those concepts, thereby concluding with lessons that could be of interest to both the mobile-computing industry as well as software-architecture researchers.

The remainder of the paper is organized as follows. Section 2 outlines the mobile-computing requirements that drove the adoption of software architectures. Section 3 describes a popular mobile app that we use to illustrate the architectural concepts in Android. Section 4 presents the key architectural principles followed by Android as well as their conception in the literature that predates it. Section 5 reports on the architectural properties of hundreds of reverse-engineered apps. Section 6 discusses the salient outcomes of our study. Finally, the paper concludes with an outline of the related research in Section 7 and an overview of our contributions in Section 8.

2. Mobile computing drivers

Before describing the architectural concepts found in contemporary mobile software, it is important to understand the key challenges that the mobile-computing industry has had to overcome over the past decade. The need to overcome these challenges is the root cause of the drastic shift toward the adoption of software architectures in today's mobile software.

(D1) App ecosystem. Perhaps the most striking difference between today's mobile platforms and those of a decade ago is the notion of *app ecosystem*. An app ecosystem is the interaction of a set of independently developed software elements (apps) on top of a common computing platform that results in a number of software solutions or services (Manikas and Hansen, 2013; Bosch, 2009). App stores and apps have changed the landscape of mobile computing: entrepreneurs are able to reach a large consumer market, consumers can choose from thousands of apps at a nominal cost, and app advertising has created a lucrative form of revenue for the developers. Apps extend the capabilities available on a platform, making the platform more attractive to the users. Therefore, a vibrant app ecosystem is crucial to the success of a mobile platform, such as Android. A key challenge in conceiving such platforms, however, was encoding constraints and rules to enable a properly functioning ecosystem with certain norms of structure and behavior, yet remaining sufficiently flexible to allow the developers to fully exploit the capabilities available on modern mobile devices (Eklund and Bosch, 2014).

(D2) Developer productivity. As alluded to earlier, the development of mobile software previously involved low-level programming, often against the various device drivers, akin to the kind of practices still followed in the embedded-computing domain (Malek et al., 2007). At the same time, the success of an app ecosystem, and thus the corresponding mobile platform, hinges on the availability of a large number of apps for end users to choose from. Such an app ecosystem requires a large pool of qualified developers capable of creating apps without highly specialized skills. Thus, the awareness grew that mobile platforms vying for a vibrant app ecosystem need to provide the developers with high-level implementation abstractions, and properly-enforced rules and constraints on how those abstractions can be composed, to ease the construction of apps.

(D3) Interoperability. A particular challenge in conceiving the modern mobile-computing platforms lied in providing a rich user experience, where a mobile device's native capabilities (e.g., phone, camera, and GPS) as well as third-party apps are able to integrate and interact with one another. Achieving this objective requires interoperability between third-party apps that are developed independently, and possibly without knowledge of one another, as well as software and hardware services that are available on a multitude of proprietary devices (Ebert and Jones, 2009). This challenge called for explicit specification of exposed interfaces of apps, as well as standards, rules, and architectural styles that regulate the interactions of apps and system services.

(D4) Security and privacy. Seamless interoperability between apps, together with the various private user data collected on modern mobile devices, gave prominence to security and privacy issues (La Polla et al., 2013). In addition, the app-store model of provisioning apps proved convenient not only for the end users, but also for the malware writers that exploited it for delivering malicious code into the users' devices (Zhang et al., 2013). To combat these threats, proper abstractions were needed for specification, assessment, and enforcement of security properties (e.g., information flow and access control) at a higher level of granularity than code.

(D5) Resource constraints. Finally, as the apps deployed on mobile platforms continued to grow in size and complexity, resource constraints (e.g., energy and memory) continued to pose an ever-present challenge. Specifically, there was a need to manage and coordinate the resources consumed by third-party apps (Nikzad et al., 2014). As an example, consider that many apps may require access to GPS information, but an uncoordinated access to such information rapidly drains the battery of a mobile device. Similarly, multiple apps may be running on a mobile device, but at any point in time only parts of those apps are actively used; without dynamically offloading the unused elements at runtime, a device would run out of resources rapidly.[1] To address these chal-

[1] Note that in this paper the term "offload" is used to mean temporarily removing a task from processor/memory to make them available for other tasks. It should not