Cloud Twin: Interactive Cross-Platform Replay for Mobile Applications

Ethan Holder
Software Innovations Lab, Virginia Tech
eholder0@vt.edu

Categories and Subject Descriptors D.2.2 [*Software Engineering*]: Design Tools and Techniques—Evolutionary prototyping Computer-aided software engineering (CASE)

Keywords Android; Automated; Cloud; Emulate; Mobile; Multi-Platform; Replay; Windows Phone

1. Research Problem and Motivation

To successfully compete in the software marketplace, modern mobile applications must run on multiple competing platforms, such as Android, iOS, and Windows Phone. Producers of mobile applications spend substantial amounts of time, effort, and money to port applications across platforms, so as to maximize the potential customer base. Creating individual program versions for different platforms further exacerbates the maintenance burden, as each bug fix and feature enhancement must be applied to all platforms.

Presented in this paper is a solution to the heterogeneity problem of the mobile application market that does not require manual porting of applications nor shifting development into cross-platform frameworks. The solution, called *Cloud Twin*, makes it possible to execute mobile applications written for one platform natively on another platform. The basic idea behind Cloud Twin is that a mobile application has two isomorphic versions: *the source*, executed on a cloud-based edge server, and *the target*, executed on a local mobile device. Initial case studies with third-party applications indicate that Cloud Twin can become a viable solution to the heterogeneity of the mobile application market.

2. Background and Related Work

Recognizing the need for heterogeneity, mobile application designers have created frameworks for cross-platform mobile development, such as PhoneGap [1]. These platforms typically leverage the mobile web browser that executes applications written in JavaScript and CSS. Despite the widespread use of cross-platform mobile frameworks, developing native applications remains the preferred practice in the mobile software market. Native applications (i.e., written for a specific platform using the platform's API) have a unique look-and-feel expected by the customers; they also take advantage of platform-specific features such as the platform's native maps (Google Maps for Android [2], Apple Maps for iOS [3], and Bing Maps for Windows Phone [4]).

Cloud Twin builds on prior work utilizing aspect-oriented programming and reflection to reverse-engineer UIs at runtime with the purpose of subsequently translating them to other platforms [7]. Cloud Twin employs the same strategy for extracting UI elements. Specifically, this mechanism is used to produce the initial UI screen of the target application. While in prior work, focus was placed on extracting UIs and statically translating them to multiple additional platforms, Cloud Twin translates and updates UIs across platforms continuously at runtime.

Cloud Twin conceptually relates to the work performed to map various platform APIs to one another. Mobile platform vendors commonly provide publicly accessible mappings that show which APIs of the target platform can be used to emulate the functionality of the source platform. For example, Microsoft provides such mappings between Android and the Windows Phone [8]. These mappings specifically relate API calls from one language to equivalent API calls in the other language in a dictionary-like fashion. Cloud Twin differs by using an intermediate form that abstracts away the logic of either language. Thus, Cloud Twin differs by lending itself to being easily extended to other platforms and languages. As long as the source language can be captured and output in the form of the Cloud Twin intermediate language, the source platform application can be supported on other target platforms.

The intermediate UI form of Cloud Twin resembles the universal UI representations of independent UI models, such as those used in UIML [9] and the aforementioned PhoneGap [1]. UIML and PhoneGap enable platform independent

SPLASH '13, October 26–31, 2013, Indianapolis, Indiana, USA.

ACM 978-1-4503-1995-9/13/10.
http://dx.doi.org/10.1145/2508075.2514879

design and development of user interfaces. UIML employs an XML base language to subsequently generate user interfaces in a desired language. However, these and other platform independent approaches require that mobile applications be constructed using a particular language and the accompanying framework. By contrast, Cloud Twin supports mobile applications that have been constructed using their native platform APIs. Thus, Cloud Twin enables the execution of such applications natively on other mobile platforms.

3. Approach and Uniqueness

Cloud Twin presents an approach to natively executing the functionality of a mobile application written for another platform. The functionality is accessed by means of interactive cross-platform replay, in which the source application's execution in the cloud is mimicked natively on the target platform. The reference implementation of Cloud Twin natively emulates the behavior of Android applications on a Windows Phone. Specifically, Cloud Twin transmits, via web sockets, the UI actions performed on the Windows Phone to the cloud server, which then mimics the received actions on the Android emulator. The UI updates on the emulator are efficiently captured by means of Aspect Oriented Programming and sent back to be replayed on the Windows Phone. In addition to its basic services, Cloud Twin also specially handles sensor input as well as time and location services. In other words, it ensures the target's environment is used by both versions of the application.

4. Results and Contributions

The central insight derived from experimenting with the prototype implementation is that the mimicking functionality of Cloud Twin is quite efficient, with the resulting latencies not adversely affecting the user experience. With the edge server running within the same administrative domain and connected to by a Wi-Fi network, the latencies of executing common UI actions in a typical application never surpassed the one second threshold [5] [6], thus making the Cloud Twin approach feasible and useful. In particular, Cloud Twin is able to natively execute several small but real Android applications natively on the Windows Phone, with the users not suspecting that they were natively interacting with applications written for a different platform. These initial experiences indicate that Cloud Twin has the potential to become a practical solution to the problem of making a mobile application available on a variety of platforms.

The best measurement to present such results thus far is the latency of a complete UI update. This was determined by measuring the total time it took between pressing a button on the target application and updating a text label in response. This measurement encompasses the following sequence of events: (1) the button pressed, (2) the resulting event is captured and transmitted to the source application, (3) the press is replayed on the source, (4) the text label update is intercepted, (5) the update is sent back to the target, (6) the target's label is updated with the received data. Typical for modern user interfaces, the measured UI scenario demonstrates a realistic response time a user would encounter when interacting with Cloud Twin.

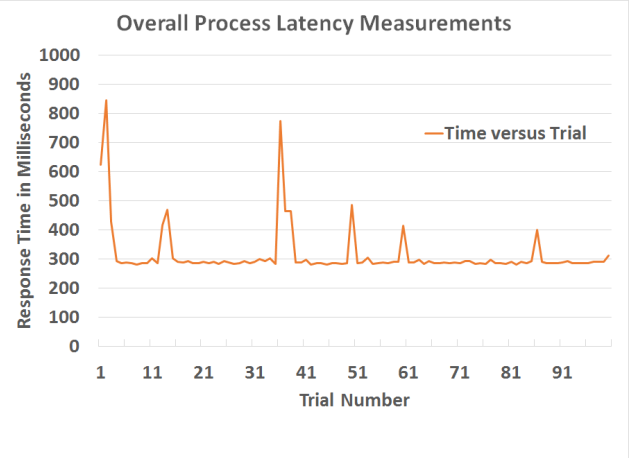

Figure 1. Measurements of the latency of the overall UI update process.

Figure 1 shows the results from repeating the measured operation 100 times. The overall average latency was 314 milliseconds, with a maximum of 846 milliseconds and minimum of 281 milliseconds. The important insight is that the response time never exceeded the one second threshold, thus not compromising the user experience [5] [6]. Future work will assess whether Cloud Twin can achieve comparable efficiency when processing more complex UI scenarios.

References

[1] R. Ghatol and Y. Patel. Beginning PhoneGap: Mobile Web Framework for JavaScript and HTML5. Apress, 2012.

[2] Google. Google Maps Android API. `https://developers.google.com/maps/documentation/android/`.

[3] Apple. Map Kit Framework Reference. `http://developer.apple.com/library/ios/documentation/MapKit/Reference/MapKit_Framework_Reference/`.

[4] Microsoft. Bing Maps APIs. `http://msdn.microsoft.com/en-us/library/dd877180.aspx`.

[5] R. B. Miller. Response Time in Man-Computer Conversational Transactions. In *AFIPS*, 1968.

[6] J. Nielson. Usability Engineering. Morgan Kaufmann, 1968.

[7] E. Shah and E. Tilevich. Reverse-engineering user interfaces to facilitate porting to and across mobile devices and platforms. In *Workshop on Next-generation Applications of Smartphones*, 2011.

[8] Microsoft Technologies. Windows Phone Interoperability. `http://windowsphone.interoperabilitybridges.com`.

[9] M. F. Ali and M. Abrams. Simplifying construction of multiplatform user interfaces using UIML. In *UIML Europe 2001 Conference*, 2001.