Aspect-Oriented Approach for User Interaction Logging of iOS Applications

Ilka Kokemor and Hans-Peter Hutter(✉)

Institute of Applied Information Technology (InIT),
Zurich University of Applied Sciences (ZHAW), Winterthur, Switzerland
hans-peter.hutter@zhaw.ch

Abstract. Mobile applications that don't meet the users' expectations regarding usability risk bad user feedbacks that lower the application's download rate or the app will just be quietly ignored. In order to identify and improve major usability issues of a mobile application, data about the real user interaction in the actual mobile usage context needs to be gathered without disturbing the interaction environment. This paper presents an aspect-oriented approach for gathering user interaction data for iOS applications without any additional hardware. To that end, two libraries have been developed that track the user and system behavior, take screenshots, record user videos and collect GPS coordinates. The data can later be downloaded from the mobile device to third party tools for further analysis. A usability test with a sample application shows the possibilities of the implemented framework.

Keywords: Mobile human computer interaction · iOS app design and development · Mobile app usability

1 Introduction

This paper shows an approach to gather detailed data about the real user interaction with an iOS application and how to analyze it to improve its usability issues.

1.1 Problem Description

Todays mobile devices not only allow to use an application anywhere anytime but also offer a variety of interaction modalities (touch, gestures, speech) the user can choose from. For usability researchers a lot of new questions therefore arise: How does a user interact with a mobile application? Which combination of modalities does a user use to interact and how does he actually interact in detail with the mobile application? When and where does a user interact with the application and what is the usage context? Which gestures does he use or try to use? Are the basic usability requirements described in ISO 9241-110 [14] fulfilled? For example, the principle about conformity with user expectations:

A. Marcus (Ed.): DUXU 2016, Part III, LNCS 9748, pp. 45–56, 2016.
DOI: 10.1007/978-3-319-40406-6_5

Can the user's expectations about the user interaction with the application, e.g. regarding gesture usage, be met? To answer this kind of questions, it is necessary not only to scrutinize the server logs but also to analyze in detail the mobile user interaction with the application. To get this information one way is to shadow him. But this is very time consuming, and worse, interferes with the real usage context. So to unobtrusively find out, how a user interacts with the application and what he actually intends to do, detailed data about the interaction, the system and usage context, and other sensor data (e.g. accelerometers, gyros) have to be gathered by the device itself without additional external hardware. There are already tools for desktop applications, e.g. [13], but mobile applications need a different approach because they are used in a mobile environment and tests in a laboratory environment cannot simulate how the real world affects the interaction with an application. So there is only the possibility to use the device in use.

1.2 Contributions

In this paper a toolkit that has been developed in our lab is presented which allows the recording of detailed data about the user interactions with an iOS application. The toolkit comprises two static libraries written in Objective-C and in Swift, resp. It logs all user interactions, i.e. view changes, button clicks, and gestures. In addition, it makes screenshots of the application screens and uses the front camera to record a user video while the user is using the application. Additionally, environmental and system information are gathered, including GPS coordinates and internet connectivity. So the usage context in which the user interacts with the application can be analyzed and used for the interpretation of the interaction flow. Often the user changes his or her behavior, because the system environment forces him to (e.g. due to a lost internet connectivity). All log information is stored in a XML file that is saved in the application's directory on the iOS device together with the screenshots and the user videos. For recording the GPS coordinates an additional application called TrackApp has been developed which is to be installed on the iOS device. After the user quits the mobile application TrackApp generates a GPX file that is also stored on the iOS device. All stored data can be uploaded to a usability test server for further analysis and can be synchronized with other recorded data.

The toolkit is based on a general framework of method swizzling that can be used not only for Objective-C but also for Swift in order to gather interaction data for any kind of usability tests, especially for multimodal interactions comprising e.g. gestures.

2 Related Work

Most approaches so far focus on the usability of Android applications and have often implemented a supplement which can be used on mobile devices using Android as operating system.

Balgates-Fernandez et al. presented in [8] the EvaHelper, a framework which logs user interactions with an Android application and stores the data in CSV files. The collected data in the CSV file can then be transformed to a format which can be imported in GraphML [15]. There the user interaction can be visualized as nested directed graphs.

Lettner et al. described in [16,17] their project AUToMAte (Automated Usability Testing of Mobile Applications). For their toolbox they implemented an Android mobile framework, an IDE Wizard and a cloud-based web server. The Android mobile framework can be added to Android applications with aspect-oriented programming measures. After the framework has been integrated into an Android application user interactions can be logged, e.g. changes between views and interactions on the screens. With the also provided web server metrics and statistics are calculated to locate usability issues within the application.

Holzinger et al. [10] presented an approach with an aspect-oriented addition to a sample iOS application which records interface events of keyboard input, context menus and drag and drop actions. The tracked data is logged in a text format. With this approach the quality of the auto-advancing short-key tagging of the sample application was measured by identifying user corrections. Holzinger uses an aspect-oriented approach of method swizzling and categories in Objective-C to gather the described data.

Google Analytics [12] is widely known and used for analyzing web activities. Also frameworks for Android and iOS are available. For that the Google Analytics Framework needs to be downloaded and integrated into the application's source code. This framework does not use an aspect-oriented approach like the one described in this paper. The recorded data cannot be downloaded but Google Analytics reports can be exported in different file types (e.g. CSV, Excel).

The companies Lookback [18] and Appsee [7] provide SDKs which can be integrated in iOS applications. While using the application a user video and a screen video are recorded. Also a timeline is logged showing which views are selected during the application's usage. On the screen video it is shown where the finger is moved. The data is uploaded to the company's servers as soon as there is an internet connection available. Additionally, Appsee provides a platform for further analysis of the interaction data. It is possible to follow the detailed interactions which are logged as well and draw heat maps that show which parts of the screen are used the most on a view. So both companies provide frameworks which can be used for gathering interaction data about applications. However, the data is automatically uploaded to their servers and only the videos can be downloaded after that. All other information about the user interaction cannot be downloaded. With the approach described in this paper, the logged data is stored in XML files which can be downloaded from the iOS device and then used in a tool that can interpret the XML format for further analysis. Also the screenshots and the user videos are stored on the iOS device and can be downloaded as well.

3 Requirements

There are several requirements for collecting information about the usage of an application to improve it's usability. All requirements are build on the prerequisite, that no third party device is necessary. Also all iOS applications should be supported, those written in Objective-C, those with a mix of Objective-C and Swift and those totally written in Swift.

User Activities. The user activities no only comprise the user interactions with the application but also his other activities, e.g. whether his attention is actually focused on the application or distracted by external events.

Screenshots or Screenvideos. Screenshots or screen videos help a lot when reconstructing the details of the user interaction flow. With that it is possible to dive into every detail of successful or unsuccessful interactions.

User Video. If user videos are recorded it is possible to get the user's reaction and emotion while he or she is using the application. Also the environment of the user is recorded. So it is possible to draw conclusions about the usage context as well.

Audio Recording. Audio recording can be added to user video recording. With that is possible not only to get a picture about the user's environment but also about the background noises. Additionally it is possible for the user to comment spontaneously.

Device and System Information. If the application heavily depends on certain device or system features, it is useful to log this information, too. If they are not available and prevent the user from actions he wants to perform, this is an important usability issue. Especially missing internet connectivity is often such an issue.

GPS Coordinates. Applications are normally used in a mobile environment, often while being on the move. Therefore the information where and when the user uses a certain application is important for analyzing it's usability. Recording the GPS coordinates is very helpful in this case.

4 Approach of Aspect-Oriented Programming

The approach for aspect-oriented programming in addition to an object-oriented design is mostly chosen because there are some issues in the system that occur allover the application, e.g. security, logging, or testing. Aspects are properties that tend to cut across functional components [11]. With adding aspects to an application it is possible to avoid restructuring it and just add the required aspect.

4.1 Aspect-Oriented Programming in AUToMAte

Lettner et al. [17] use aspect-oriented programming to add their framework to existing Android applications. They used AspectJ, an aspect-oriented programming framework for Java, to integrate the monitoring framework. With AspectJ the host application does not have and need any connection to the monitoring framework. The monitoring framework can be added easily and fast to the hosting application. Lettner et al. add aspects into applications with join point model (JPM). They use compile-time weaving, which means adding the framework during compilation time. They point out, that post-compile weaving and load-time weaving do not have any advantages to compile-time weaving and load-time weaving does not work with Android applications because of its virtual machine limitations.

4.2 Aspect-Oriented Programming with Objective-C

In the Objective-C Runtime Programming Guide [2] it is pointed out that many decisions are deferred to runtime. For that Objective-C programs interact with the runtime system at three levels: through Objective-C source code, through methods defined in the NSObject class of the Foundation framework, and through direct calls to runtime functions. So in contrast to Lettner et al. who chose compile-time weaving for aspect-oriented programming in Java, adding or changing functionality in an aspect-oriented way in Objective-C programs is done at runtime.

Method Swizzling. In Objective-C the messaging system is used to call methods. The compiler builds a structure for each class which includes the pointer to the superclass and a class dispatch table. While creating a new object, memory is allocated and its instance variables are initialized. The instance variables include a pointer (isa) to its class structure. With this pointer the object gets access to its class and indirectly to all superclasses as well. A class maintains a dispatch table to resolve messages sent at runtime: each entry in the table is a method, which keys a particular name, the selector, to an implementation, i.e. a pointer to an underlying C function. To swizzle a method is to change a class's dispatch table in order to resolve messages from an existing selector to a different implementation, while aliasing the original method implementation to a new selector [20]. Method swizzling used for method interception in Objective-C, since Apple provides the possibility to change the class dispatch table at runtime. For that the original selector shows to a new implementation and a new selector shows to the original implementation, which means swapping selectors. For that either the new implementation of the method is added directly to the original selector and the original implementation is replaced in the swizzled selector, or the methods are just swapped.

Categories. Holzinger et al. [10] use the concept of categories to add methods and functionality to existing classes. Categories are a runtime feature. Apple states,

"at runtime, there is no difference between a method added by a category and one that is implemented by the original class" [3]. In our case categories are also used to enhance implementation of certain classes to collect data about the user's interactions. The interface needs not be changed, since the methods are swizzled and no additional methods are added nor their interface is changed.

Holzinger et al. use a property list to configure aspects for certain classes. In our approach classes are extended with categories to add functionality to existing methods which does not require changing or adding property configurations. Since all this is implemented in a static library, we only need to consider that the categories are loaded properly into the used applications by setting the `Other Linker Flags` correspondingly.

4.3 Aspect-Oriented Programming with Swift

Using the Objective-C Approach. Since Swift is based on Objective-C and uses the same runtime environment, all Objective-C features are also available in Swift. That means we can use method swizzling and categories in Swift as well.

Approach of Compile-Time Weaving. In addition to Objective-C, Swift uses virtual method tables (vtables) for method runtime binding. While compiling the Swift Front End translates the code into Swift Intermediate Language (SIL). Then the SIL Optimizer optimizes SIL and generates code in Intermediate Representation (IR) format. Finally, the IR Optimizer transfers the IR format into machine code. In Objective-C the Clang Front End transfers the code directly into IR [9,19]. SIL generates the vtables for the classes, by mapping the method names to their implementations. Final methods are not listed here. They do not need to be mapped, since their implementation always stays the same. So for aspect-oriented programming, static weaving could be used to replace non-final methods in the vtables.

5 Implementation

To log data about user interactions with an application, the above described method swizzling was chosen. The reason for this decision was that the approach and implementation for Objective-C and Swift would be similar and final methods in Swift would be supported as well. To that end, two static libraries have been build to support apps written in Objective-C, Swift, or even a mix of both.

5.1 Methods to Be Swizzled

In our approach specific classes are enhanced that collect data whenever they are called. Whenever the user is moving his fingers over the touch screen, the class

`UITouch` is involved. The method `locationInView` of `UITouch` is central for that. It returns the location of the touch in the receiving view [5]. So whenever the user interacts with the screen, even when the application does not respond to the user's touch, the method `locationInView` is called. Therefore also unrecognized gestures, e.g. the user unsuccessfully tries to zoom in, can be recognized. The property `gestureRecognizers` in class `UITouch` holds an array of instances of class `UIGestureRecognizer`, which is the super class of all gesture recognizers. The array holds all gesture recognizers who would respond to the given touch of the user, even if it triggers no action. This feature can be used to find out if there are user gestures the app hasn't implemented.

All visual control elements are derived from `UIControl`. It includes the method `sendAction` which is called when there is an action triggered because the visual control element is activated. The method `endTrackingWithTouch` is called when the recording of a touch gesture is stopped [4]. Especially with the method `sendAction` it is possible to log control elements. However, in some cases the method `sendAction` does not include any implementation and is therefore not called, even if there are visual control elements.

All view controllers are derived from class `UIViewController`. This class is responsible for the view management. So when the user changes the view, e.g. through forward navigation, the `UIViewController` changes as well and another view will be displayed. The methods `viewDidAppear` and `viewWillDisappear` can be used to log this information. The first one notifies when a new view is shown, the other one triggers when the view is about to be shut down [6].

Normally the class `CLLocationManager` is used to collect GPS coordinates. There the method `didUpdateLocations` is called regularly to get new GPS coordinates [1].

5.2 Static Libraries

Method swizzling works a little different in Objective-C than in Swift. In Objective-C methods are normally overwritten in the function `load`. In Swift this must happen in the function `init`. Also it is necessary that the classes are swizzled in the according language. So classes in Objective-C need to be swizzled in Objective-C, Swift classes have to be swizzled in Swift. However, it is not possible to swizzle the same class twice in one library. Therefore two libraries were developed, one swizzling in Swift and one swizzling the same classes in Objective-C.

5.3 Functionality

To enhance an existing iOS application in order to log interaction data, one of the libraries need to be integrated into it. The following functionalities are currently implemented in both static libraries:

– For each user interaction a screenshot with a timestamp is taken and stored in the iOS device's file system.

- Start and end of the usage of the application is marked. For that the appropriate functions needs to be included in methods of the class `AppDelegate` of the monitored application.
- When the usage or observation is finished an XML file is created and stored in the iOS device's file system.
- GPS coordinates are tracked.
- System data are recorded, especially internet connectivity.
- A user video is recorded and stored in the iOS device's file system while the user uses the application.
- Easy integration into the existing application: for that the Apple standard of static libraries has been chosen.

6 Usability Analysis of a Sample Mobile Application

In ISO 9241-110 [14] seven general usability principles are listed which are important guidelines also for analyzing the usability of mobile applications. Conformity with user expectation and self descriptiveness e.g. is important for mobile applications because users will try to use common gestures to control the application and use hints within the application for its usage but they rarely read a handbook. The application must also be error tolerant, which means the system must help the users if there is a problem with the system or with the usage of the application. The user also wants to know why an error has occurred and how to avoid or solve the problem. As mobile users often try to perform a specific task, the principle "suitability for the task" is specially important for mobile applications.

In this case study a usability test of the ZHAW Engineering CampusInfo App [21] was conducted. The main functionality of this iOS application includes timetables for students, professors, classes, rooms and courses. The application also shows the actual menu and opening times of the canteens, the administration contact information, the campuses schematic maps and social media links. It also lists events and news of the ZHAW School of Engineering department. The application only exists in German language, therefore all given screenshots show German labels. For the usability test, the application was enhanced with the described Objective-C library. Users were recruited do use the application in their daily routine. After the tests the data was downloaded from their iOS phones and analyzed.

6.1 Evaluation Example

With the gained logs and videos, different evaluations are possible. A general statistic of the usage of the different functionalities for 3 users is shown in Fig.1. The overall time ratio spent in a functionality, however, does not tell us whether the functionality was actually used. If a user spends much time within one functionality, there could be times of inactivity which then can be analyzed using user videos and GPS logs. Or the user could spend much time within one functionality because he or she searches for something and cannot find it.

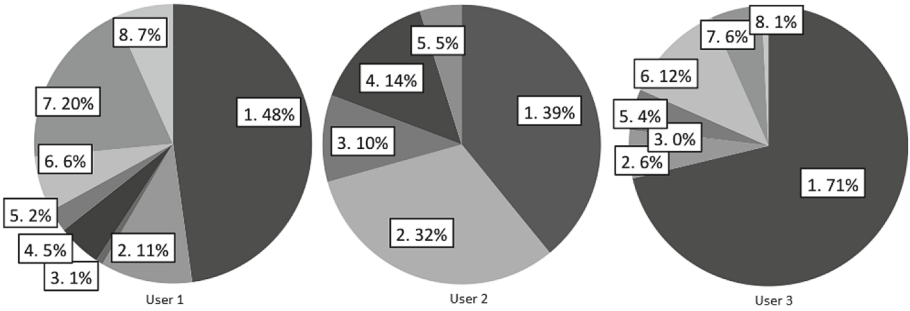

Fig. 1. Percent of time of functionality usage of the example application with three users (1. Timetable, 2. Cafeteria, 3. News and Events, 4. Settings, 5. Public Transport, 6. Location, 7. Contacts, 8. Social Media)

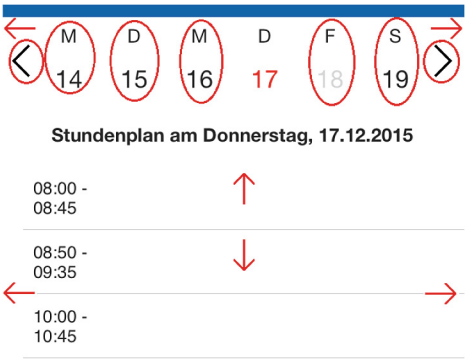

Fig. 2. Possible gestures and buttons to change the day or week, marked with red arrows and circles (Color figure online)

Figure 1 shows how much time three users spend within the functionalities of our example application. Users were given the task to use the example application during some days in their daily routine. In Fig. 1 the percentage of usage time within the functionalities of the application of different users is shown. As expected all users spent most of their time with the timetable functionality: User 1 48 %, User 2 39 %, and User 3 71 %. In the next important features the users start to defer: while User 2 spent 32 % of his time in Cafeteria, User 1 switched between Contacts and Cafeteria, and User 3 spent it in Locations. The overall time ratio spent in a functionality, however, does not tell us whether the functionality was actually used. The recorded user videos revealed, e.g., that User 3 put his mobile device into his pocket while the mobile application was still running in the timetable functionality.

Since all users spent most of their time within the timetable functionality, we continue our investigations there. In Fig. 2 the time table overview is shown. There are different possibilities to use gestures and buttons there. If the swiping

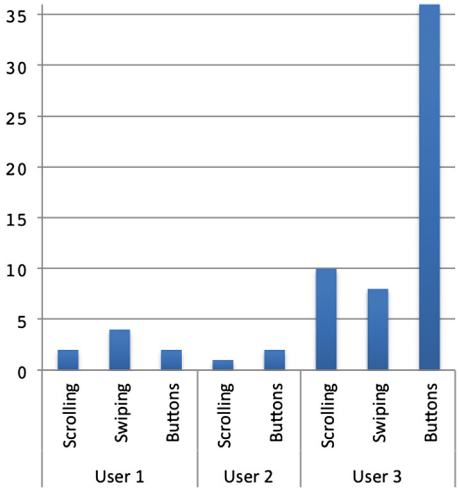

Fig. 3. Gestures and buttons used in timetable overview, focussing on changing the day or week

Fig. 4. Scrolling gesture in timetable overview

gesture is used in the table, the day is changed forward or backward depending on a left or right swipe. Swiping in the list of days view results in changing the week, forward or backward depending on a left or right swipe. But also buttons can be used here, either one of the day buttons to change the day of the current week, or the arrow buttons on the lift and right side to change the week. The possible gestures and buttons are marked with red arrows and red circles, resp.

With the interaction log it is possible to figure out, if the users used the swiping gestures or used the buttons to change the day or week. Figure 3 shows which gestures the users used in the given view and how many times they used the buttons for changing the day or week. User 1 changed the day or week using swiping gestures and buttons while User 2 used buttons only. User 3 largely

used the buttons but also the swiping gestures from time to time. All users used scrolling gestures as this is the only way to scroll through the time slots of a day.

The interaction log also provides the information, where the user touched the display during a gesture interaction. The coordinates of the gesture movement are logged and can be exported later to draw the gesture into the screenshot. In Fig. 4 a scrolling gesture is shown where first the user scrolled down the timetable and then up again.

7 Conclusion

In this paper we have shown how detailed interaction logging of an iOS application can be performed with method swizzling. To that end, two libraries for iOS applications were developed. We chose method swizzling and categories as an aspect-oriented approach to gather those data. The two libraries provided support applications written in Objective-C as well as in Swift. The interaction data is available in form of user videos and screenshots as well as different XML files recorded on the mobile device. These can be downloaded and used in third party usability research tools for further analysis and synchronization with other external recordings. With this setup detailed analysis of mobile usability issues can be performed in order to optimize the usability and user experience of a given mobile application.

References

1. Apple: CLLocationManagerDelegate, 11 November 2015. Website: https://developer.apple.com/library/prerelease/ios/documentation/CoreLocation/Reference/CLLocationManagerDelegate_Protocol
2. Apple: Objective-c runtime programming guide, 11 November 2015. Website: https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ObjCRuntimeGuide/Introduction/Introduction.html#apple_ref/doc/uid/TP40008048-CH1-SW1
3. Apple: Programming with objective-c - customizing existing classes, 11 November 2015. Website: https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/CustomizingExistingClasses/CustomizingExistingClasses.html
4. Apple: Uicontrol, 11 November 2015. Website: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIControl_Class
5. Apple: Uitouch, 11 November 2015. Website: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITouch_Class/#apple_ref/occ/instm/UITouch
6. Apple: UIViewController, 11 November 2015. Website: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIViewController_Class
7. Appsee: Appsee, 11 November 2015. Website: https://www.appsee.com/
8. Balagtas-Fernandez, F., Hussmann, H.: A methodology and framework to simplify usability analysis of mobile applications. In: IEEE/ACM International Conference on Automated Software Engineering, pp. 520–524 (2009)

9. Borzym, K.: Swift method dispatching a summary of my talk at swift warsaw, 11 November 2015. Website: http://allegro.tech/2014/12/swift-method-dispatching.html
10. Brugger, A.H.M., Slany, W.: Applying aspect oriented programming in usability engineering processes: on the example of tracking usage information for remote usability testing. In: 2011 Proceedings of the International Conference on e-Business (ICE-B), pp. 1–4, Jul 2011
11. Fayad, C.: Designing an aspect-oriented framework in an object-oriented environment. ACM Comput. Surv. (CSUR) **32**(41), 1–12 (2000)
12. Google.org: Google analytics, 11 November 2015. Website: https://developers.google.com/analytics/
13. Hilbert, D.M., Redmiles, D.F.: Extracting usability information from user interface events. ACM Comput. Surv. **32**(4), 384–421 (2000)
14. Ergonomics of human-system interaction Part 110: Dialogue principles, 11 November 2015. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=38009
15. Konstanz, U.: graphdrawing.org, 11 November 2015. Website: http://graphdrawing.org/
16. Lettner, F., Holzmann, C.: Automated and unsupervised user interaction logging as basis for usability evaluation of mobile applications. In: Proceedings of the 10th International Conference on Advances in Mobile Computing and Multimedia, pp. 118–127 (2012)
17. Lettner, F., Holzmann, C.: Sensing mobile phone interaction in the field. In: 2012 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), pp. 877–882 (2012)
18. Lookback: Lookback, 11 November 2015. Website: https://lookback.io/
19. Siracusa, J.: OS X 10.10 Yosemite: The ars technica review, 11 November 2015. Website: http://arstechnica.com/apple/2014/10/os-x-10-10/22/
20. Thompson, M.: Method swizzling, 11 November 2015. Website: http://nshipster.com/method-swizzling/
21. ZHAW School of Engineering: ZHAW Engineering CampusInfo, 11 November 2015. Website: https://itunes.apple.com/ch/app/zhaw-engineering-campusinfo/id715684381?mt=8