Extending aspect-oriented programming for dynamic user's activity detection in mobile app analytics

By Francisco Moreno, Silvia Uribe, Federico Álvarez and José Manuel Menéndez

Abstract— **Mobile apps analytics represent a core set in the mobile industry to extract relevant data with the aim of modeling user's behavior. Current solutions to detect user's activity in the mobile apps are usually oriented to analyze the resulting information rather than improving the way the information is obtained and tracked within the app. They are based on a continuous app code modification schema, which implies high development efforts and a clear problem to implement changes without compromising the time to come back to the market or cause problems with dependencies in the user's app updates. In this article we analyze the suitability of Aspect Oriented Programming for providing a more efficient way to detect user's activity inside apps, which may lead to obtain user analytics. We propose an innovative approach that relies on an in-app solution based on the embedding of a specific library and a configuration file in charge of setting up the events to be tracked in real time, without additional code changes in the app. Thus, this new schema will reduce the time and effort costs derived from the integration of 3rdparty trackers.**

I. INN-APP USER'S ACTIVITY DETECTION OVERVIEW

With the raise of the use in mobile devices and apps[1], the study of the user path within these is a must for the mobile market since it provides essential information to identify trends and understand behaviors. Nevertheless, the central challenge in this scenario has always been focused on the analysis of the obtained data[2], leaving aside the way they are collected. This is usually considered as a programming issue since it depends on how to integrate third party libraries, also known as trackers, with the app.

In the current scenario the obtained process is clear: apps owners analyze which are the characteristics points to be detected inside the app, and then the developers include marks in the code to track users' actions. According to this schema, the inclusion of these marks involves code changes that lead to new releases of the application that should be updated in the client side to take effect. Moreover, it has to be repeated every time a new point has to be detected, increasing the total number of app updates which are usually related to modifications for errors correction or new features integration.

This process implies an increase of costs in terms of time and money, thus the dynamic track of the user behavior inside mobile apps can only be assumed by those stakeholders who can afford this extra effort. It can also imply a loss of data if the user skips the app update, so the total track in real time becomes an almost impossible aim. In this article, we propose the use of Aspect Oriented Programming (AOP) as a new approach for improving the user mobile application path detection by easing the integration of the measured points which implies a costs reduction in terms of time to market and efforts and an increase of the points detection control and management.

II. RELATED WORK

A. *Mobile app analytics: initial considerations*

Mobile apps analytics usually refer to data collected while the app is being used. Through analysis, these data provide lots of insights into user behavior together with information about overall app performance. When a user's action detection is done, a lot of information can be derived, mainly depending on the specific application.

From generic usage statistics to particular information, such as the queries users enter into the e-shop search bar, can be provided if the measured points are correctly set.

Several trackers are typically used when talking about mobile analytics, which can be classified among three main groups [3]: firstly, the advertising trackers, such as *Google AdWords and Inmobi*, which represent almost the 65% of the trackers used, are mainly serve within in-app advertisements and they collect personal data from the users.

Secondly, the analytics trackers (24% of use), such as *Google Analytics(GA)* and *AppsFlyer(AF)*, which mainly track users' actions inside and across apps for client attribution and other marketing purposes. Finally, the utility trackers (11% of use), as *Bugsense* and *Crashlytics*, focused on assisting developers to track bugs.

According to this, our article is focused on the integration of analytics trackers, which obtain different inputs for improving the application overall performance and increase the user's experience by providing essential feedback to different schema recommendation [4].

There are several maintenance activities that must be done to maintain the high quality and stable performance. These activities are divided in four groups; adapt apps to operating system updates (adaptive maintenance), adapt apps to new technologies (preventive maintenance), fix errors and modify functionalities (corrective maintenance) and use the user feedback to obtain more appealing solutions (perfective maintenance)

Most of them are needed to assure continuous improvement and new features to be recognized by current and potential users but, due to the high associated cost of these modifications (representing about 40%-70% of the total cost of software life cycle [6], having the corrective maintenance a 20% of this effort), an efficient management and planning of them is needed.

Considering software maintenance as "the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment" [7], current solutions for user path analysis within mobile applications can be seen as one of these tasks. In fact, these activities can be classified as corrective tasks since they comprise code changes for detecting different user's actions but they do not really correct any malfunction of the application. For this reason, finding another way to include the associated measure points that does not imply a continuous and iterative code modification process could be an effective way to avoid overloading the maintenance scenario, allowing an efficient effort allocation for the appropriate application adjustment.

B. Current approaches and need for a new dynamic solution for user path analytics

Trackers, including the analytics ones, usually provide their SDKs as libraries that can be easily embedded into apps. Nevertheless, this integration implies direct code changes in the application. In particular, in the case of users' tracking, the changes can be due to the different actions that can be done inside the app that reveal important information for marketing analysts. These changes must be done immediately in order to obtain real time information. These two requirements are essential for an efficient user tracking, and they represent a real challenge that must be taken into consideration.

As it is explain in [8], current solutions require an initial analysis of the events to be tracked at the very beginning of the design phase in order to include the related code in the app. Moreover, the modification of these events implies a refactorization of the app each time a new event needs to be tracked [9], following a cycle schema that repeats itself. Nevertheless, this approach does not guarantee the success of the process since there is also a critical point: an app update from the user is still needed. In this context, a more efficient solution for user tracking is needed.

On the other hand, it is also important to consider the development and deployment time that every update involves. In this context, a quick code modification can imply a late new version due to different factors. A change in the development team, an error in the specification or even the time needed for market publication can cause an important delay in releasing the app, reducing the window opportunity of the change.

Finally, frequent modifications needed by current solutions for allowing a dynamic mobile app analytics also present another important problem: user's behavior in relation to app updates. In this respect, although users are aware of mobile app updates, almost half of them do not enable automatic updates (due to several reasons: lack of memory space, phone and app crashes, security and privacy concerns, and feature and functionality loss) [10], resulting in a loss of clients. Moreover, frequent updates may be discouraging, making that almost 35% of the users uninstall apps because of too many releases. Therefore, a solution for dynamic tracking of the user's behavior, which does not involve any update of the code, will be an efficient answer for mobile apps analytics. In this regard, this solution should not only reduce the related efforts, but also remove the user's dependence in order to provide an agile and real time response that makes the user tracking more affordable for different mobile stakeholders.

III. MANTRA: A NEW IN-APP USER ACTIVITY TRACKING APPROACH

A. Aspect-oriented programming

According to its own definition [11], AOP is an additional programming paradigm that extends the traditional object-oriented programming (OOP) model to improve code reuse across different object hierarchies. AOP modularizes the crosscutting concerns into units, called aspects, and then separates them from the modules that implements the system basic functionality, that is, the primary business logic, allowing to clearly express programs by including appropriate isolation, composition and reuse of the aspect code.

TABLE 1. AOP MAIN CONCEPTS

Concept	Description
Crosscutting concern	An aim that a program wants to achieve, it should be scattered among different classes and methods.
Aspect	A modularization of a concern that cuts across multiple objects.
Join point	A well-defined position in a program, such as the execution of a method, the handling of an exception, etc.
Advice	A class of functions that can modify other functions and that can be applied at a given join point. There are different types of advice: "around", "before" and "after".
Pointcut	A set of join points whenever reached the corresponding advices will be executed.
Weaving	The process in which an aspect is added into an object. It can be executed during the compilation time or during the running of the program.

Although in 2001 the MIT announced AOP as a key technology for the 10 future years, it was not widely adopted during that period, mainly because of the maintenance, validation and evolution difficulties [12]. Nevertheless, this trend has changed recently, expanding its use to many different fields, such as software testing [13] and web applications [14] among others, due to a set of important advantages, listed below [15]:

- More accuracy in software development, especially in changing and upgrading: AOP provides and efficient way to modularize the code by gathering what deals with the same aspect, avoiding redundancy and making that each part has a specific aim.
- Steady implementation by handling each aspect once.
- Reusability enhancement, since AOP allows isolating core concerns from the crosscutting ones, enabling more mixing and matching between them.
- Skill transfer enhancement: AOP concepts are reusable and transferable, reducing developers training and implementing time and cost.

For doing this, AOP introduces a set of concepts that enables its application, as can be seen in **¡Error! No se encuentra el origen de la referencia.**. As it is going to be explained, they provide an encouraging context for defining an innovative way of embedding the marks needed for user's track. In this regard, AOP allows facilitating this integration by removing the code changes and determining a seamless client-server communication for real-time track, which results in a less effort-consuming solution.

B. Dynamic user behavior tracking within mobile apps: our approach

Our approach is mainly based on a two-element combination: a server-client communication based on JSON file exchange, and an in-app library for tracking management, called MANTRA (Mobile ANalytics TRAcking).

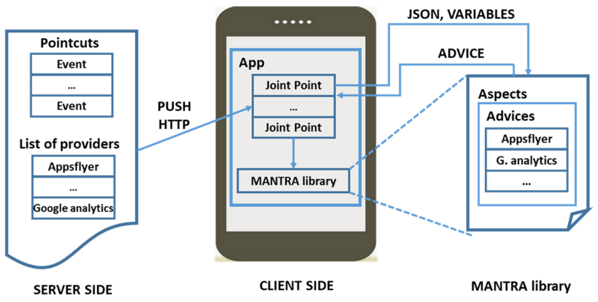

Fig. 1. Approach for in-app user tracking based on AOP

1) *Architecture of the solution*

Based on the AOP concepts from **¡Error! No se encuentra el origen de la referencia.**, our approach architecture can be seen in Fig. 1 and it is composed by the following elements:

- **Aspect**: each users' action delivered from the client side to the trackers. It is not an intrinsic app function but a common activity for them.
- **Advice**: the code that sends the info to the trackers, that is, the third party scripts for analytic trackers, such as GA or AF. The advice is injected in run time into the code, as defined in AOP.
- **Jointpoints**: the entire set of methods from the app where an advice can be executed.
- **Pointcuts**: the list of events to be detected inside the app where the library is included. These elements will be dynamically defined and they will be sent to the app within a JSON file by means of a push notification or via HTTP. At the same time, the app will send this JSON to the library in order to configure the pointcuts.

2) *Workflow*

The aforementioned elements comprise the core of this innovative solution and can be properly combined in order to define a new workflow. This workflow is composed by three main phases. The first one is related to the integration of the MANTRA library within the app and to the JSON initial configuration (which contains the events to be detected), together with the upload of the app to the market and its download to the user's device. The second phase is related to the acquisition of the data from the user's activity inside the app and the reconfiguration of the JSON in order to change the events to be detected. This updated JSON can be accessed by the app through different options: push notification, via a direct delivery within a static frequency model, etc. The final phase is related to the data compilation and delivery to the analytics trackers, which will be in charge of analyzing them and obtaining the corresponding overall results about user's behavior.

Considering this workflow, the need for direct changes in the code app is reduced to only once at the beginning of the cycle. This implies that there is only one compilation of the app and one publication in the market, reducing the time and effort to obtain it. Thereafter, the dynamic change of the events to be detected is done via a JSON file delivery from the server to the app, processing it inside the app thanks to the AOP capabilities. This JSON can be updated as many times as needed, and the changes are processed in real time, without recoding the app.

3) *Tracking parametrization and integration*

To configure the app, the MANTRA library provides different methods to set up the trackers credentials and the JSON url (methods *init(GA/AF,cred.)* and *setAspects(JSON)*).

As mentioned above, the exchanged JSON file is one of the two main pillars of this approach. This JSON mainly includes the set of events to be detected, but it can also comprise two more elements:

- A list with the different analytics providers that may be enabled in the application.

- A set of static and dynamic variables for the tracking execution that has to be sent from the app to the library before the pointcut execution starts to take effect.

Concerning the integration, the configuration of an event requires specifying the name of the class and the function where the event is located, as well as its type and other additional data in order to be well-processed. In this context, the user id can be sent as an additional variable and can be used to link the information between different trackers or different mobile platforms (iOS, Android, etc.). Finally, regarding the development of the MANTRA library, we have made use of two specific open code solutions for AOP paradigm implementation: MOAspects for iOS, and AspectJ for Android.

IV. Solution Analysis and Discussion

A. *Experimental Setup*

We have designed a simple experimental mobile app for applying the proposed solution. This app comprises two main screens. The first one has two buttons and one label that shows the number of times the first button is clicked and the second button shows the second screen when clicked. The second screen has one button to go back to the first screen. This app must track the following events with GA and AF:

1. First screen is loaded
2. First button is clicked
3. Second button is clicked
4. Second screen is loaded
5. Button in the second screen is clicked

Then we defined two main variables: a global one for all the events that contain a string, and a dynamic one to be sent when the first button is clicked specifying the number of times it has been clicked.

For testing purposes we have designed an app life cycle where the tracking needs change. Thus, these changes are:

1. Change the global variable sent with all the events;
2. Stop sending the dynamic value when the first button in the first screen is clicked.
3. Stop tracking with AF.
4. Start sending the dynamic value again.
5. Start tracking with AF again.

A skilled developer implemented two different versions of the app, the first one using the standard implementation, and the second one using AOP and MANTRA library. A comparison between both methods is shown in Fig. 2 The development of the app was divided into a list of tasks, and the developer was timed while doing the tasks and implementing the changes. The time invested in tasks that needed technical skills and the tasks that didn't were also measured in order to provide a qualitative and quantitative estimation about the related costs during the life cycle of the app.

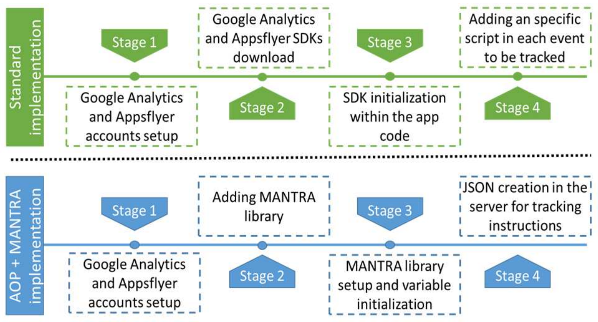
Standard implementation
Stage 1
Google Analytics and Appsflyer accounts setup
Google Analytics and Appsflyer SDKs download
Stage 2
Stage 3
SDK initialization within the app code
Adding an specific script in each event to be tracked
Stage 4
AOP + MANTRA implementation
Stage 1
Google Analytics and Appsflyer accounts setup
Adding MANTRA library
Stage 2
Stage 3
MANTRA library setup and variable initialization
JSON creation in the server for tracking instructions
Stage 4

Fig. 2. Standard vs. AOP+MANTRA implementations

B. *Results: time and effort comparison*

Fig. 3 right shows the time spent developing and setting up the apps during its life cycle. The Y axis represents the time in minutes, and the X axis represents the life cycle, where 0 is the initial development of the app and 1-5 are the changes defined in the previous section.

As seen in the figure, even when the time invested in the stage 0 is bigger in our approach, this difference is compensated while applying the changes, due to the fact that the time spent in them become significant smaller. At this point it is important to note that this comparison is done only with the development time and not considering the time to market (in such case, the difference would be even higher), and considering only 5 changes during the entire life cycle while, in a real case, the number of modifications could be bigger.

Thus we can confirm that the proposed solution provides a new method which requires less effort in terms of development time for managing the changes needed to dynamically track user's activity inside mobile apps.

On the other hand, Fig. 3 left represents the time, in percentage, spent developing the app and implementing the changes. As seen in the figure, the time spent in tasks that need technical skills is significantly smaller in the AOP+MANTRA solution than in the standard one, which also means a total cost reduction.

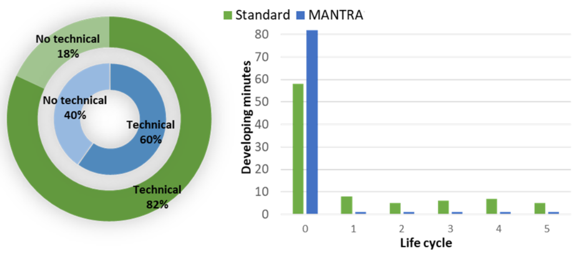

Fig. 3. Development performance comparison: technical/no technical skills needs (left), time effort along life cycle (right).

To estimate the impact of the library in the performance of the app we used the Android Studio profiler to compare the CPU and Memory usage. As seen in Fig. 4, the cost of using the libraries is minimal. Concerning disk occupancy, the app only takes 3.44 extra MB.

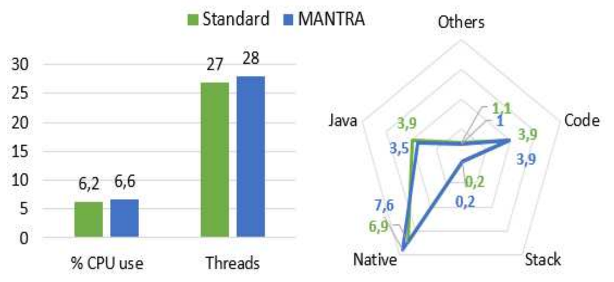

Fig. 4. Android app performance comparison: CPU use (left) and memory use (right)

V. Conclusions

According to the obtained results, the difference on development and maintenance time (regarding user activity tracking) is significant, due to the fact that our approach decreases the time needed to make the changes on the events to be tracked within the app. Another advantage is that the app resubmission to the app is no longer needed, which usually adds time on top of the development time. Considering this, our solution may save 2-5 hours for android apps and around 3 days in Apple store in every single modification.

Other consideration is that tracking changes can be done by users with no technical skills by easily and manually editing the JSON, or even with a graphic interface. In this regard, Fig. 5 shows a cost estimation of each step of the life cycle if all changes would have been done by a qualified developer (considering an average salary of a US developer in 2018). As we can see, using the AOP+MANTRA solution for tracking the events within the app may save development efforts.

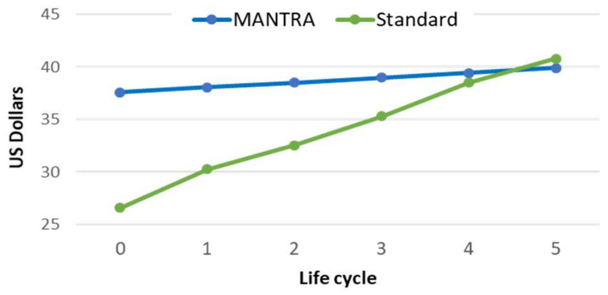

Fig. 5. Accumulated developing cost

Finally, and considering the previous results, it can be stated that one of the main innovations of this work is the application of the AOP paradigm to a new field, such as mobile app tracking, allowing a more efficient mobile apps development and maintenance cycle.

Acknowledgment

This work has been partially funded by the EC H2020-ICT-19-2016-2 76199 EasyTV project and the Spanish CDTI-Ministry for Science and Innovation, within the framework of the project LPS-Bigger (EXP 00064563/ITC-20133062).

About the Authors

Francisco Moreno (fmg@gatv.ssr.upm.es) is a predoctoral researcher at Universidad Politécnica de Madrid (UPM). His research interest includes mobile apps and machine learning.

Silvia Uribe (sum@gatv.ssr.upm.es) is a post-doctoral researcher at UPM. Her research interests include big data and UX/UI technologies.

Federico Álvarez (fag@gatv.ssr.upm.es) is an Assistant Professor at UPM. During the last 10 years he has coordinated UPM participation in several EU-funded projects. He is author and co-author of (70+) international papers.

José Manuel Menéndez (jmm@gatv.ssr.upm.es) is Full Professor at UPM and GATV Research Group Director. He has participated and coordinated more than 150 national and European research projects. He authored more than 200 research papers and 3 international patents.

References

[1] KAUR, Satwant. The revolution of tablet computers and apps: A look at emerging trends. IEEE Consumer Electronics Magazine, 2013, vol. 2, no 1, p. 36-41

[2] S. Van Canneyt, M. Bron, A. Haines, and M. Lalmas, "Describing Patterns and Disruptions in Large Scale Mobile App Usage Data". In *Proceedings of the 26th International Conference on World Wide Web Companion* (WWW '17 Companion). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 1579-1584, 2017.

[3] S. Seneviratne, H. Kolamunna, and A. Seneviratne, "A measurement study of tracking in paid mobile applications". In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks* (WiSec '15). ACM, New York, NY, USA, Article 7, 6 pages. 2015.

[4] D. Contreras, M. Salamo, I. Rodriguez and A. Puig, "Shopping Decisions Made in a Virtual World: Defining a State-Based Model of Collaborative and Conversational User-Recommender Interactions," in *IEEE Consumer Electronics Magazine*, vol. 7, no. 4, pp. 26-35, July 2018.

[5] X. Li, Z. Zhang and J. Nummenmaa, "Models for mobile application maintenance based on update history" *2014 9th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, Lisbon, Portugal, 2014, pp. 1-6.

[6] B. Hunt, B. Turner and K. McRitchie, "Software Maintenance Implications on Cost and Schedule," *2008 IEEE Aerospace Conference*, Big Sky, MT, 2008, pp. 1-6.

[7] IEEE. "IEEE standard for software maintenance". IEEE STD. 1219-1998, pages 1-3.

[8] Ferre, X., Villalba, E., Julio, H., & Zhu, H. (2017, September). Extending Mobile App Analytics for Usability Test Logging. In *IFIP Conference on Human-Computer Interaction* (pp. 114-131). Springer, Cham.

[9] Ravindranath, L., Padhye, J., Agarwal, S., Mahajan, R., Obermiller, I., & Shayandeh, S. (2012, October). AppInsight: Mobile App Performance Monitoring in the Wild. In *OSDI* (Vol. 12, pp. 107-120).

[10] M. Nayebi, B. Adams and G. Ruhe, "Release Practices for Mobile Apps -- What do Users and Developers Think?," *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2016, pp. 552-562.

[11] G. Kiczales et al, "Aspect-oriented programming". *ECOOP'97 — Object-Oriented Programming. ECOOP 1997.* Lecture Notes in

Computer Science, vol 1241. Springer, Berlin, Heidelberg. Akşit M., Matsuoka S.

[12] F. Munoz, B. Baudry, R. Delamare and Y. Le Traon, "Inquiring the usage of aspect-oriented programming: An empirical study," *2009 IEEE International Conference on Software Maintenance*, Edmonton, AB, 2009, pp. 137-146.

[13] ADINATA, Muhammad; LIEM, Inggriani. A/B test tools of native mobile application. En 2014 International Conference on Data and Software Engineering (ICODSE). IEEE, 2014. p. 1-6..

[14] Y. C. Yu, S. c. D. You and D. R. Tsai, "An intelligent Aspect-Oriented framework for web application," *INC2010: 6th International Conference on Networked Computing*, Gyeongju, Korea (South), 2010, pp. 1-5.

[15] R. Laddad, "Aspect-oriented programming will improve quality", IEEE Software, 2003, vol. 20, no.6, pp 90-91