Diagnosis of Privacy and Performance Problems in the Context of Mobile Applications

David Monschein
Karlsruhe Institute of Technology
Karlsruhe, Germany
ugdzy@student.kit.edu

Robert Heinrich
Karlsruhe Institute of Technology
Karlsruhe, Germany
robert.heinrich@kit.edu

Christoph Heger
NovaTec Consulting GmbH
Leinfelden-Echterdingen, Germany
christoph.heger@novatec-gmbh.de

ABSTRACT

Complex systems need to run flawless and protecting the users privacy is mandatory to stay competitive. The consideration of privacy aspects is even more important for mobile applications, because a lot of sensitive data is transferred over a bunch of different networks. Especially after the espionage scandals in the last years the users care about their privacy more than ever. There are several approaches which enable the monitoring and analysis of performance issues.

However, there is no approach which supports both privacy and performance diagnostics. Therefore we propose an approach for filling this gap, a monitoring and analysis strategy for applications which is capable of identifying privacy and performance problems. The monitoring is designed to collect performance metrics together with privacy related information. The resulting monitoring data is used to derive an architectural run-time model which takes part in the performance analysis and in the detection of privacy threats.

During our evaluation we analyzed the accuracy and the overhead to confirm that our approach can be used in production.

KEYWORDS

Mobile Application Monitoring, Application Privacy, Usage Profile Extraction, Privacy Analysis, Performance Analysis

ACM Reference Format:
David Monschein, Robert Heinrich, and Christoph Heger. 2018. Diagnosis of Privacy and Performance Problems in the Context of Mobile Applications. In *ICPE '18: ACM/SPEC International Conference on Performance Engineering Companion , April 9–13, 2018, Berlin, Germany*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3185768.3186283

1 INTRODUCTION

The performance of applications continues to be crucial for companies. Especially mobile applications with large target groups need to be optimized in terms of performance. Users of mobile applications expect fast response times and there are a lot of different mobile devices with different specifications, which makes it hard to perform well on each of them.

ICPE '18, April 9–13, 2018, Berlin, Germany

ACM ISBN 978-1-4503-5629-9/18/04…$15.00
https://doi.org/10.1145/3185768.3186283

Besides performance also other quality properties, such as privacy, are important for modern software systems. Privacy is a rough term, in most definitions it is related to data privacy, which deals with the ability to decide what data can be sent to third parties [16]. Today, when a lot of sensitive data is transferred and people care about their privacy, the protection of data flows is essential.

An actual problem is the lack of toolings that support the monitoring and analysis of privacy aspects. A big difference between mobile applications and conventional software is that the geographical location of the user can change while an application is in use. Changing the goegraphical location can result in a change of the network connection and may trigger a privacy problem if the new connection is insecure (e.g. weak encryption). For example a user accesses an application for social communication and while he is interacting with the mobile device he leaves his house and therefore the phone has no longer a connection to his own Wireless Local Area Network (WLAN). The device automatically connects to the mobile internet or to another WLAN and sensitive data of the user is transferred over a network, whose trustworthiness is not guaranteed. For instance, the encrpytion of the GSM protocol, which is used by mobile network connections, is breakable [2, 18].

Within this paper we present a monitoring and analysis approach with a particular focus on applications that are intended to run on mobile devices. This approach is capable of identifying both privacy and performance problems of the monitored application. Our monitoring is based on inspectIT [13] in combination with Kieker [19] and the analysis strategy uses iObserve [7] together with the Palladio Component Model (PCM) [15].

The main contributions of this paper are as follows:

- We propose a monitoring approach for observing both performance and privacy. Our monitoring collects relevant privacy information and uses it to detect privacy threats during the analysis.
- We describe an analysis method that builds upon iObserve and the PCM to get knowledge about the performance and the privacy characteristics of the application under observation.
- The evaluation of the proof-of-concept implementation, based on the community case study CoCoME [8], is summarized and presented within this paper.

2 STATE OF THE ART

There are several approaches for monitoring mobile applications. One of them is AppInsight [14], which is limited to the monitoring of performance aspects. Another familiar monitoring approach

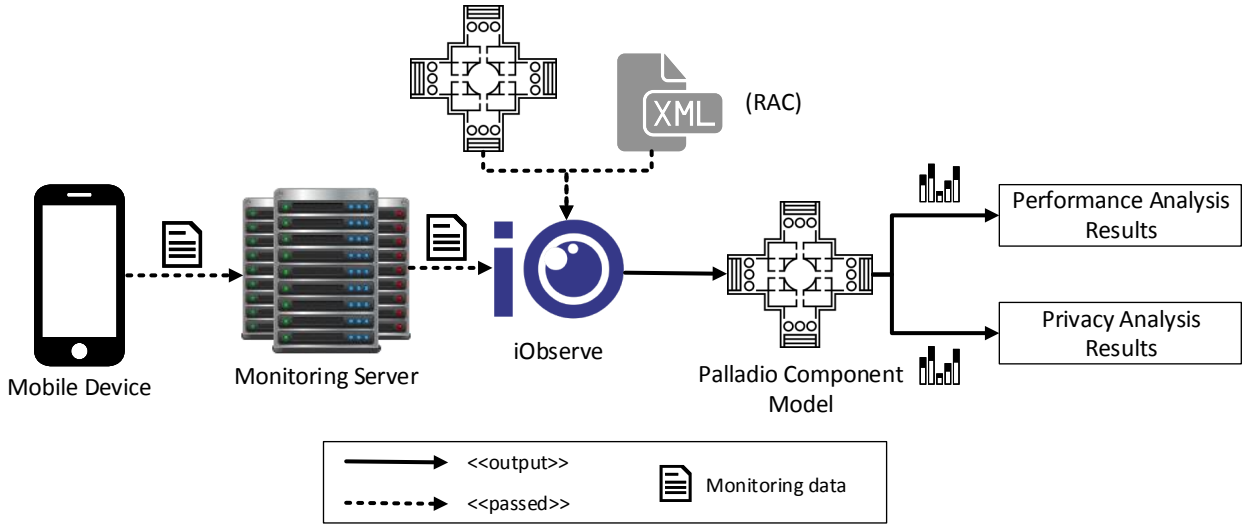

Figure 1: Visualization of the approach structure

is the SPASS-meter [3] monitoring framework. It provides an extension for monitoring Android applications and the goal of the SPASS-meter framework is to collect performance metrics. A monitoring approach which addresses the detection of privacy problems is TaintDroid [4]. TaintDroid is a modification of the Android operating system and is able to observe the usage of sensitive information. Accordingly, it is mandatory that the application to be monitored runs on a device which uses TaintDroid. This reveals the main handicap of TaintDroid: it is not possible to monitor existing applications in production because the user of the application needs to have the TaintDroid operating system installed. Our monitoring concept tries to combine the benefits of the mentioned approaches.

The contribution of this paper includes the analysis of privacy aspects, therefore we also introduce existing approaches in this dimension. An approach for modeling data flows and dealing with data flow analysis was presented by Stephan Seifermann. This approach suggests the realization of data flow analysis by seeing them as "first class entities" [17]. To realize this, it is suggested to extend the Palladio Component Model [17]. A well known approach for designing security critical software systems is UMLsec [11]. UMLsec makes it possible to express information about the security in the Unified Modelling Language (UML). For behaviour and structure descriptions UMLsec provides mathematical funded semantics [11]. Therefore, UMLsec models can be examined using model checkers. The big drawback of UMLsec is the requirement of a detailed design, which is in most cases not available at early development stages [17].

3 MONITORING AND ANALYSIS APPROACH

In this section we introduce the design of our monitoring and analysis approach.

3.1 Structure

Figure 1 outlines the important steps from collecting monitoring data to the generation of analysis results. First of all, it is necessary to collect performance and privacy information while the application under observation is running. This is done by adding code to the application. The purpose of this code is to collect monitoring data. The main goals which were targeted in the conception of the monitoring are as follows:

(i) Minimization of the effort needed for the integration into a mobile application
(ii) Compatibility to existing approaches (Kieker [19], inspectIT [13])
(iii) Minimization of overhead

The main artefact of the monitoring is the **Monitoring Agent**, this is the code which runs directly on the device and performs measurements. To weave this code into an existing application, there exist different strategies. In our proof-of-concept implementation we used bytecode modification to support the fully automated instrumentation of Android applications. Because we can not store monitoring data on the mobile device it is necessary to have a server component which receives measurements from the mobile agent and persists them. The server provides a simple RESTful [5] interface. In our case, the server component transforms the monitoring data into Kieker records [19]. This step is mandatory, because iObserve only supports the Kieker format at the moment. After transforming the monitoring data to Kieker records and writing them to a monitoring log, the resulting file can be passed to iObserve. Additionally iObserve requires an initial Palladio Component Model instance and a run-time architecture correspondence model (RAC) [10] as input. The RAC is used to associate monitoring data, which relates to implementation artefacts, with architectural run-time model elements [10]. The main idea of iObserve is to monitor the changes of the system environment and predict necessary adjustments. iObserve iterates over all monitoring records and applies different transformations to the initial architectural run-time model depending on the type and the content of the records. The updated model takes part in the analysis of performance and privacy [7].

The idea of our approach is to extend iObserve so that it is able to process privacy related monitoring data. A major benefit of this approach is the fact, that existing parts of iObserve and the Palladio Component Model stay untouched and can still be used to get a clue about the performance of the monitored application. To be able to store privacy aspects it is necessary to extend the Palladio Component Model.

3.2 Monitoring

Just like Kieker and inspectIT, we also gather method call traces to detect performance issues. This part is not an innovation as it uses existing parts of inspectIT and Kieker. Additionally, the monitoring keeps track of privacy relevant actions. In the following, we describe the most important privacy characteristics which are addressed by our monitoring approach.

The privacy threat detection is based on the analysis of network connections. Therefore it is necessary to observe all network requests performed by the monitored application. Important properties of a network request are the protocols used on the application layer (e.g. HTTP, HTTPS) and the ones that are used to transmit the data (e.g. LTE, GSM). Using this information it is possible to derive details about the encryption and the confidentiality of the connection. Furthermore, if the device uses a Wi-Fi connection, the encryption type and the ID of the access point are recorded. Especially unsecured Wi-Fi networks are prone to traffic sniffing attacks. Additionally we capture the end point of the network request and a checksum of the transferred data.

Section 3.5 explains how we use the monitoring data within the analysis.

3.3 Palladio Component Model Extension

As already mentioned before, iObserve adjusts an initial Palladio Component Model (PCM) with the information stored in Kieker monitoring logs. The current version of the PCM is focused on making performance predictions and there is no way to store privacy related data, like the connection protocol, in an instance of the PCM. Therefore it is necessary to extend the Palladio Component Model. We focused on extending the **Resource Environment Model** of the PCM [15], because we want to analyze network data flows in terms of privacy. The resource environment model consists of **Resource Containers**. They can be linked with so-called **Linking Resources**, which represent a network connection between two containers. For example, a mobile phone and a server are Resource Containers and if the mobile phone has a connection to the server they are connected with a Linking Resource. A Linking Resource can specify a **Communication Link Resource Specification** which contains information about the connection. We propose to introduce new sub-types that are able to store information about the privacy aspects of the connection (for example the protocol which is used to communicate).

3.4 iObserve Integration

iObserve is an approach to "cloud-based system adaptation and evolution through run-time observation and continuous quality analysis" [7]. In the current version iObserve supports the processing of the following monitoring record types:

- Flow records (Method call traces)
- Deployment records (Indicate the deployment of a component)

The flow records are used to derive a PCM usage model which describes the behaviour of the users. This is the job of the transformations $T_{EntryCall}$, $T_{EntryCallSequence}$ and $T_{EntryEventSequence}$ [7]. These transformations extract an usage profile from the collected method call traces. Therefore, it is necessary that we also collect flow records on the mobile device, so iObserve is able to create an accurate usage model. The deployment records indicate, as the name suggests, the deployment of a servlet, server or application component and are used to update the Resource Environment Model and the System Model [15] of the PCM instance.

In order to pass privacy related data to iObserve, it is necessary to add new monitoring records and transformations to iObserve. The transformations are responsible for processing the introduced monitoring records and use the included privacy information to adjust an instance of the extended PCM. The resulting PCM instance contains everything that is necessary to perform performance and privacy analysis tasks.

3.5 Analysis

The PCM, which arises from the execution of iObserve, can be analyzed in various ways. In our case we focused on the examination of network connections. Therefore we build a directed graph where clients and servers are represented by vertices and the edges indicate a data flow between two vertices. This is done by extracting the necessary data out of the Resource Environment Model [15] of the PCM instance. The edges also hold additional information about the connection, for example the connection type and the used protocol. Using this graph, we can use several analysis approaches. Our prototypical implementation iterates over all network connections (edges) and recognizes the usage of deprecated or insecure protocols and connection types (e.g. GSM protocol mentioned earlier or unsecured Wi-Fi connection). All network connections that are classified as insecure need to be protected with additional measures (e.g. end-to-end encryption). This analysis helps to identify vulnerable data flows. We are also able to elaborate on the network flow. For instance, A sends data to B and B forwards the data to C. Imagine the connection between B and C is insecure. As a result, all data that has been transferred from A to B is also at risk, even if the connection is provably secure. Using the network graph in combination with algorithms for analyzing flows within graphs makes it possible to determine dangerous routes in the network. In the mentioned scenario this strategy is able to notice that the traffic between A and B is exposed if the data flow between B and C is compromised. The more nodes are monitored, the more accuracte is the resulting graph. In the ideal case, the monitoring runs on all computers that are involved in the network communication of the application (servers and clients). A bit more complex version of this approach assigns integer values to the edges (e.g. 0 - 10), where 0 means that the connection is exposed and 10 represents a well-secured connection. The first step is to investigate on the used protocols and on the additional knowledge about the connection. Depending on the security characteristics, we apply a suitable value to the belonging edge. Afterwards we perform a flow analysis and

decrease the values of all edges that are reachable from a vulnerable connection. This results in a graph where the edges (which represent network connections) are classfied with different "security levels". We can also use different strategies and include additional information of the PCM to adjust the analysis so it fits a certain use-case.

The performance analysis is straightforward and relies on existing features of iObserve and the Palladio Component Model. After executing the iObserve analysis, all analysis tasks provided by the Palladio Component Model can be used to inspect the performance of the application. For example, the PCM supports the discovery of performance bottlenecks and scalability issues [15].

3.6 Proof-of-Concept

Based on Android [6], which is currently one of the most used operating systems worldwide, we built a proof-of-concept implementation. We used bytecode instrumentation, based on inspectIT [13], to automatically add code to the application to be monitored. A fine-grained description of the Android monitoring implementation is included in our previous work [12].

Additionally we implemented the extension for iObserve and the Palladio Component Model to be able to deal with privacy related monitoring data. Finally, we created an exemplary privacy-analysis which takes an instance of the extended Palladio Component Model as input, detects possible privacy threats and prints the results.

4 EVALUATION

We evaluated our approach using the proof-of-concept implementation mentioned in Section 3.6. In this section we summarize the results of the evaluation.

4.1 Goals

The evaluation investigates two quality dimensions of our approach:

- The first goal is to determine the accuracy of the components. This includes especially the validation of the correctness for the monitoring and analysis results.
- The second goal concerns the overhead of the mobile monitoring approach. To be applicable in production it is mandatory that the mobile monitoring has no significant impact on the performance of the application. Therefore we examine the performance and the network overhead that is caused by the monitoring.

4.2 Experiment Environment

To realize the evaluation described in the previous section we focused on the monitoring and analysis of the mobile shop client for CoCoME. The mobile shop is basically an extension which is connected to the conventional CoCoME [9] over an adapter. The evaluation environment consists of an Android device and a monitoring server. We used both a physical and a virtual Android device to collect monitoring data. To generate monitoring data, we need to interact with the application and it is important that the interaction is reproducible, otherwise we can not compare different monitoring logs. Therefore, we specified a set of use-cases which can be reproduced. These use-cases are executed automatically using *monkeyrunner*[1]. Monkeyrunner provides an API for simulating user-actions on an Android device. The considered use-cases are:

- **UC1 -** User selects a store and searches through the offered products.
- **UC2 -** The user of the application uses his credentials to sign in.
- **UC3 -** The customer selects the products he wants to buy and pays by adding his credit card.

4.3 Accuracy Evaluation

To measure the accuracy of our approach we generate monitoring data with the experiment environment described in the previous section. Furthermore, we also monitored the experiment environment with established tools. The main idea is to automatically simulate several use-cases and afterwards we compare the results of our approach to the ones produced by other toolings. After verifying that the monitoring is accurate, we use the collected monitoring data to test the iObserve and the Palladio Component Model analysis against different inputs.

First of all, we checked whether the observed network requests are equal to the network requests which tcpdump [1] recognized. Tcpdump is a utility for observing network requests. Furthermore, we gathered method call traces using *Android Monitor (AM)* [2] and compared them with the traces in our monitoring logs. The results showed that our monitoring approach is as accurate as similar tools in terms of network monitoring and method trace collection. **Table 1** and **Table 2** summarize the measurement results which arise from the simulation of the use-cases mentioned in 4.2. We have to notice that we ignored network requests and method calls issued by the underlying operating system, because it is impossible to gather them by monitoring the application only.

Use-Case	Action	#Traced Method Calls	
		Our monitoring	AM
UC-1	Browse store	31	31
UC-2	User Authentication	20	20
UC-3	Process sale	52	52

Table 1: Collected method call traces while simulating different use-cases

Use-Case	Action	#Network requests	
		Our monitoring	tcpdump
UC-1	Browse store	4	4
UC-2	User Authentication	2	2
UC-3	Process sale	7	7

Table 2: Recognized network requests while simulating different use-cases

Next, we used the generated monitoring logs to review the analysis tasks. First we inspected the accuracy of the iObserve extension by passing different monitoring logs. Afterwards we showed that the privacy analysis works appropriate. We modeled all conceivable privacy threats (deprecated protocols, insecure encryption,

[1] https://developer.android.com/studio/test/monkeyrunner/index.html
[2] https://developer.android.com/studio/profile/android-monitor.html

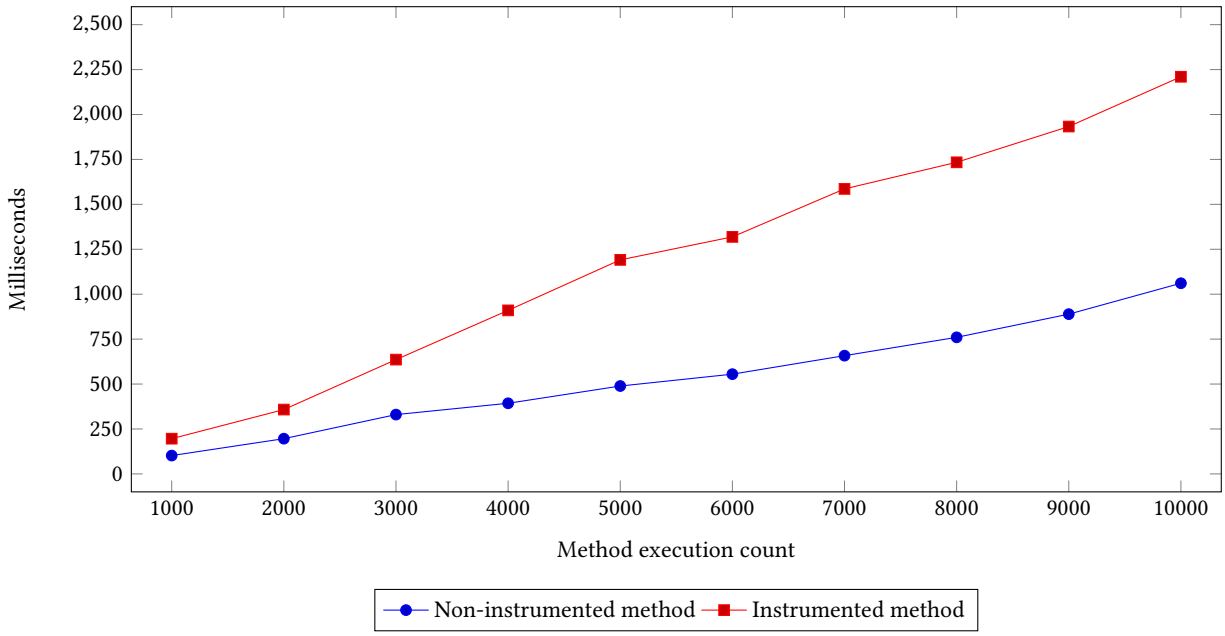

Figure 2: Average method execution times of the instrumented and the non-instrumented method

untrusted endpoints and the ones outlined in Section 3.5) to test the analysis against instances of these models. The intention is to ensure that all considered privacy problems are detected.

4.4 Performance Overhead Evaluation

First, we introduce the procedure to measure the performance overhead caused by the monitoring. Therefore we instrumented a single method. The test-application executes the method a predefined number of times and afterwards logs the time the execution needed. Next, we performed the measurement one-hundred times for both the instrumented method and the non-instrumented one and compared the results. Figure 2 visualizes the observed timings.

It shows the average amount of time that is needed to execute the measurement. The count of method executions within the measurement is plotted on the x-axis and the time needed for the execution is applied to the y-axis. At first sight, the overhead seems to be very high for a increasing number of method executions. In the following, we elaborate on the results shown in the line graph and explain why our approach nevertheless performs very well. First of all, the method, which is used to test the overhead, only performs two simple operations and therefore only needs less than 100μs for one execution. Therefore, the overhead caused by the instrumentation is even more predominant. If we use a method that runs for 1ms or longer, the gap between the lines would be much smaller, because the instrumentation causes a fixed overhead for every monitored method which is executed (overhead scales linearly with the number of instrumented methods).

Figure 3 inspects the monitoring overhead per monitored operation execution in detail. The Figure shows that the Trace registry is responsible for more than 50 percent of the overhead. This is due to the fact that we use a modified version of the Kieker tracing system

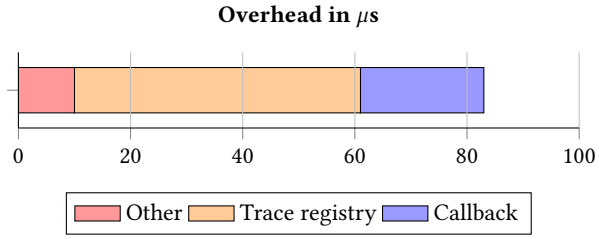

Figure 3: Monitoring overhead per execution of a monitored operation in μs

[20], which is not optimized for the usage on mobile platforms. The effort for sending the data **asynchronously** back to the monitoring server amounts to an average of 22μs. With only 22μs, the callback is really fast, regarding the fact that it is necessary to pool monitoring records and send them back to the monitoring server, because we can not store the data on the device itself. The remaining part of the overhead is caused by simple operations which are not part of the Kieker trace registry or the callback. All in all, a overhead of 80μs per execution of a monitored operation is not much and only has an impact on the performance if we monitor methods that are executed very often (>10000 times) or are intended to run very fast (<1ms).

To limit the monitoring overhead, it is necessary to ensure that low-level methods are not instrumented. Our instrumentation supports the selection of classes and methods to be monitored. Therefore, it is necessary to restrict the instrumentation to methods, where the overhead of around 80μs per execution has no significant impact on the performance. Furthermore, in the most cases it does not make sense to monitor methods that are executed very

frequently. These methods are often triggered by higher-level operations and it makes much more sense to monitor the higher-level methods and model them with Service-Effect-Specifications in the Palladio Component Model [15]. This guarantees that we do not lose any information, the overhead is reduced and the method call traces are clearer.

4.5 Network Overhead Evaluation

Second, we determined the network traffic caused by the monitoring. The traffic arises from the connection between the agent and the monitoring server. We used the test environment introduced in Section 4.2 and observed the network traffic caused by the monitoring. The packet analyzer tcpdump [1] was used to capture all network requests. We only outline the main results of the network overhead evaluation here, details about the procedure and the measurements are described in our previous work [12].

We showed that the most mobile internet connection types need less than ten seconds to send one thousand monitoring records. Only the GSM and the EDGE protocol needed plenty of time and we have to keep in mind that a GSM or EDGE connection is very unlikely in most urban areas like cities. Other connection types, like Wi-Fi connections, needed significantly less than one second to transmit one thousand records. Furthermore the network strategy of the monitoring is exchangeable, therefore it is possible to implement a strategy which collects the monitoring data until the connection is fast enough to transfer them. This reduces the impact on the network performance of the device.

5 CONCLUSION

Our presented approach provides a transparent and extensible way for monitoring applications and searching the monitoring data for privacy and performance problems. Kieker and inspectIT are used for monitoring purposes and iObserve in combination with the Palladio Component Model is responsible for the analysis parts. The monitoring approach has been implemented for Android, but the conception is platform independent and it is possible to implement it for other operating systems. The analysis uses iObserve to enrich a Palladio Component Model instance with the collected monitoring data and afterwards the resulting PCM instance can be analyzed. The monitoring and the analysis can be extended without an outstanding effort. This provides a solid base for future developments. The popularity of mobile applications is still growing and the technology is developing rapidly. The proposed approach takes care of this evolution with flexibility and extensibility.

The evaluation showed that our approach is accurate and performs well. We analyzed the functionality and measured the overhead caused by the monitoring to make sure that our approach is applicable in production. Smart networking strategies and the reduction of monitored operations make it possible to limit the monitoring overhead, so it fits nearly every use-case.

6 ACKNOWLEDGMENTS

This work has been supported by the German Federal Ministry of Education and Research (grant no. 01IS17106A, Trust 4.0, and grant no. 01IS15004, diagnoseIT), the DFG (German Research Foundation) under the Priority Programme SPP1593, and the Research Group of the Standard Performance Evaluation Corporation (SPEC).

REFERENCES

[1] 2017. tcpdump for Android. (2017). http://www.androidtcpdump.com/ http://www.androidtcpdump.com/, accessed 27.03.17.

[2] Elad Barkan, Eli Biham, and Nathan Keller. 2008. Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. *Journal of Cryptology* 21, 3 (2008), 392–429. https://doi.org/10.1007/s00145-007-9001-y

[3] Holger Eichelberger et al. 2014. Flexible resource monitoring of Java programs. *Journal of Systems and Software* 93 (2014), 163 – 186. http://www.sciencedirect.com/science/article/pii/S0164121214000533

[4] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. 2010. TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*. USENIX Association, Berkeley, CA, USA, 393–407. http://dl.acm.org/citation.cfm?id=1924943.1924971

[5] Otávio Freitas Ferreira Filho and Maria Alice Grigas Varella Ferreira. 2009. Semantic Web Services: A RESTful Approach. In *IADIS International Conference WWWInternet 2009*. IADIS, 169–180.

[6] Google Inc. 2017. Android. (2017). https://www.android.com/intl/en_en/, accessed 11.04.17.

[7] Robert Heinrich. 2016. Architectural Run-time Models for Performance and Privacy Analysis in Dynamic Cloud Applications. *SIGMETRICS Perform. Eval. Rev.* 43, 4 (2016), 13–22. https://doi.org/10.1145/2897356.2897359

[8] Robert Heinrich, Stefan Gärtner, Tom-Michael Hesse, Thomas Ruhroth, Ralf Reussner, Kurt Schneider, Barbara Paech, and Jan Jürjens. 2015. A Platform for Empirical Research on Information System Evolution. In *27th International Conference on Software Engineering and Knowledge Engineering*. 415–420.

[9] Robert Heinrich, Kiana Rostami, and Ralf Reussner. 2016. *The CoCoME Platform for Collaborative Empirical Research on Information System Evolution*. Technical Report 2016,2; Karlsruhe Reports in Informatics. Karlsruhe Institute of Technology. http://digbib.ubka.uni-karlsruhe.de/volltexte/1000052688

[10] Robert Heinrich, Eric Schmieders, Reiner Jung, Kiana Rostami, Andreas Metzger, Wilhelm Hasselbring, Ralf H. Reussner, and Klaus Pohl. 2014. Integrating Run-time Observations and Design Component Models for Cloud System Analysis. In *Proceedings of the 9th Workshop on Models@run.time co-located with 17th International Conference on Model Driven Engineering Languages and Systems (MODELS 2014), Valencia, Spain, September 30, 2014*. 41–46. http://ceur-ws.org/Vol-1270/mrt14_submission_8.pdf

[11] Jan Jürjens. 2002. *UMLsec: Extending UML for Secure Systems Development*. Springer Berlin Heidelberg, Berlin, Heidelberg, 412–425. https://doi.org/10.1007/3-540-45800-X_32

[12] David Monschein. 2017. *Mobile Application Monitoring - Privacy and Performance Diagnosis*. B. Sc. Thesis. Karlsruhe Institute of Technology.

[13] NovaTec Consulting GmbH. 2017. inspectIT: Manage your Java Application's Performance. (2017). http://www.inspectit.rocks/, accessed 11.04.17.

[14] Lenin Ravindranath, Jitendra Padhye, Sharad Agarwal, Ratul Mahajan, Ian Obermiller, and Shahin Shayandeh. 2012. AppInsight: Mobile App Performance Monitoring in the Wild. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. USENIX, 107–120.

[15] Ralf H. Reussner, Steffen Becker, Jens Happe, Robert Heinrich, Anne Koziolek, Heiko Koziolek, Max Kramer, and Klaus Krogmann. 2016. *Modeling and Simulating Software Architectures – The Palladio Approach*. MIT Press, Cambridge, MA. http://mitpress.mit.edu/books/modeling-and-simulating-software-architectures

[16] Margaret Rouse. 2013. Data privacy (information privacy). (2013). http://searchcio.techtarget.com/definition/data-privacy-information-privacy, accessed April 10, 2017.

[17] Stephan Seifermann. 2016. Architectural Data Flow Analysis. In *Proceedings of the 13th Working IEEE/IFIP Conference on Software Architecture (WICSA'16)*. IEEE, 270–271. https://doi.org/10.1109/WICSA.2016.49

[18] M. Toorani and A. Beheshti. 2008. Solutions to the GSM Security Weaknesses. In *2008 The Second International Conference on Next Generation Mobile Applications, Services, and Technologies*. 576–581. https://doi.org/10.1109/NGMAST.2008.88

[19] André van Hoorn et al. 2012. Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis. In *3rd ACM/SPEC International Conference on Performance Engineering*. ACM, 247–248. https://doi.org/10.1145/2188286.2188326

[20] André van Hoorn, Matthias Rohr, Wilhelm Hasselbring, Jan Waller, Jens Ehlers, Sören Frey, and Dennis Kieselhorst. 2009. *Continuous Monitoring of Software Services: Design and Application of the Kieker Framework*. Research Report. Kiel University. http://eprints.uni-kiel.de/14459/