2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)

An Observational Study on the State of REST API Uses in Android Mobile Applications

Abdelkarim Belkhir
Université du Québéc à Montréal
Montréal, Canada
belkhir.abdelkarim@courrier.uqam.ca

Manel Abdellatif
Polytechnique Montreal
Montreal, Canada
manel.abdellatif@polymtl.ca

Rafik Tighilt
Université du Québéc à Montréal
Montreal, Canada
tighilt.rafik@courrier.uqam.ca

Naouel Moha
Université du Québéc à Montréal
Montreal, Canada
moha.naouel@uqam.ca

Yann-Gaël Guéhéneuc
Concordia University
Montreal, Canada
yann-gael.gueheneuc@concordia.ca

Éric Beaudry
Université du Québéc à Montréal
Montreal, Canada
beaudry.eric@uqam.ca

***Abstract*—REST is by far the most commonly-used style for designing APIs, especially for mobile platforms. Indeed, REST APIs are well suited for providing content to apps running on small devices, like smart-phones and tablets. Several research works studied REST APIs development practices for mobile apps. However, little is known about how Android apps use/consume these APIs in practice through HTTP client libraries. Consequently, we propose an observational study on the state of the practice of REST APIs use in Android mobile apps. We (1) build a catalogue of Android REST mobile clients practices; (2) define each of these practices through a number of heuristics based on their potential implementations in Android apps, and (3) propose an automatic approach to detect these practices. We analyze 1,595 REST mobile apps downloaded from the Google Play Store and mine thousands of StackOverflow posts to study REST APIs uses in Android apps. We observe that developers have always used HttpURLConnection class for REST APIs implementation in Android apps. However, since the apparition of REST third-party libraries such as Okhttp, Retrofit and Google Volley, Android REST clients have been increasingly relying on the facilities offered by these libraries. Also, we observe that developers used to ignore some good practices of REST APIs uses in Android apps. Such practices are the use of HTTP third-party libraries, caching responses, timeout management, and error handling. Moreover, we report that only two good practices are widely considered by Android developers when implementing their mobile apps. These practices are network connectivity awareness and JSON vs. XML response parsing. We also find that Retrofit is the most targeted third-party HTTP client library by Android developers because of its ease of use and provided features. Thus, we conclude that service providers must strive to make their libraries as simple as possible while mobile-service consumers should consider existing libraries to benefit from their features, such as asynchronous requests, awareness to connectivity, timeout management, and cached responses.**

I. Introduction

The global market has experienced a tremendous increase in the number of mobile users over the past decades. In 2018, Google PlayStore boasted close to 2.2 millions of mobile applications (apps) while the Apple AppStore has over 2 millions[1]. Most of these mobile apps access remote data, business rules, and business processes of vital information-systems, which are remote services for architectural, efficiency, and security reasons.

REST (Representational State Transfer) APIs (Application Programming Interfaces) have been a mainstream form of services for some years now. They have been well suited for providing content to mobile devices, like smart-phones and tablets, because they offer a lightweight and flexible implementation for end users and mobile apps developers. Several research works studied best practices in the implementation of REST APIs [1], [2], [3], [4]. However, little is known on how Android apps use/consume these APIs in practice through dedicated HTTP client libraries.

These libraries were first implemented for mobile apps with "low-level" APIs provided by programming languages (sockets) or class libraries (HttpURLConnection). They can be implemented now using dedicated, third-party libraries, such as Google Volley or Retrofit. We ask in this paper how such dedicated HTTP client libraries are being used in practice by developers. Such knowledge is important for service providers because (1) mobile apps run on mobile devices that have many constraints in terms of memory, battery, computational power, and competition for resources, (2) it would help them to offer more features that ease the Android developers' work, and (3) it would help them improve the usability and performance of REST client libraries [5].

Consequently, we propose an observational study of the state of the practice of the use of REST APIs by Android apps. We answer the following research questions:

- **RQ1:** What is the state of the practices in the use of REST APIs by Android apps? We want to observe the use of practices by developers and their prevalence.
- **RQ2:** What is the state of the implementation of HTTP client libraries in Android REST clients? We want to observe the implementations used by developers for REST APIs, which will inform the choice of HTTP client libraries.

By answering these research questions, we want to recommend to developers the libraries and practices to adopt based

[1]https://www.statista.com/markets/424/topic/538/mobile-internet-apps

978-1-7281-3395-9/19/$31.00 ©2019 IEEE
DOI 10.1109/MOBILESoft.2019.00020

on their prevalence in implementations and their benefits for developers.

Therefore, we provide four contributions. We review the literature extensively and compile a catalogue of seven practices related to the development of mobile apps using REST APIs. These practices pertain to the use of dedicated, third-party libraries and help to understand how Android apps use/consume REST APIs. Then, we propose a framework, PIRAC, that automatically detects occurrences of the practices using detection rules. Then, we conduct an observational study on over 1,595 REST mobile apps out of 9,173 apps downloaded from the Google PlayStore to report how they use/consume REST APIs. Finally, we mine 12,478 StackOverflow posts to assess the importance of the identified practices from the point of view of Android developers.

The remainder of this paper is as follows. Section II describes related works. Section III details the design of our study. Section IV reports our observations. Section V answers our research questions based on our observations while Section VI describes the threats to validity of our observational study. Finally, Section VII concludes with future work.

II. Related Work

Several research works have been proposed in the literature on bad and good practices in REST APIs. However, few are the works that study how Android apps use/consume REST APIs.

In the context of mobile apps, Rodriguez *et al.* [1] were the first to study the traffic of HTTP requests from mobile clients. They evaluated the conformance of some state-of-the-art design best practices of REST APIs from the perspective of mobile clients. They analyzed these practices on a large data-set of 78 GB of HTTP requests collected from a mobile-Internet traffic-monitoring site. However, the best practices analyzed are common to any kinds of REST APIs and they focused specifically on HTTP requests.

Oumaziz *et al.* [5] conducted an empirical study on 500 popular Android apps and 15 popular services to identify best practices when using/consuming REST APIs for Android mobile clients. They showed that Android clients generally favour invoking REST APIs by using official dedicated service libraries instead of invoking services with a generic HTTP client like HttpURLConnection. They also presented which good practices service libraries should be implemented following an online survey and manual analyses of the apps. In this paper, (1) we go more in details to identify how dedicated service libraries are used by Android clients. (2) We propose a tool to automate the detection of these practices in Android mobile apps. (3) We empirically analyze more than 9,000 Android mobile clients to study the usage of REST APIs in Android mobile apps.

Several works were carried out for the detection of service anti-patterns [1], [2], [3], [4], [6], [7]. For example, in the context of REST APIs implementation, Palma *et al.* evaluated the design of several REST APIs and proposed different approaches to detect automatically REST (anti)patterns. They proposed SODA-R (Service Oriented Detection for Anti-patterns in REST) [4], a heuristics-based approach to detect (anti)patterns in REST systems. They relied on heuristics and detection rules for eight REST anti-patterns and five patterns. They applied their tool on a set of 12 widely-used REST APIs including BestBuy, DropBox, and Facebook. Then, Palma *et al.* proposed a syntactic and semantic approach to detect REST linguistic (anti-)patterns, which they define as poor/good practices in the naming, documentation, and choice of identifiers in REST APIs [7]. Finally, Palma *et al.* proposed UniDoSA [6], a unified approach that (1) embeds a unified meta-model for the three main service technologies REST, SCA, and SOAP and (2) detects the presence of anti-patterns in service-based systems.

A set of automatic approaches for the detection of REST (anti)patterns are proposed in these works. However, they specifically evaluated APIs without considering any interaction with clients, in particular mobile clients, as we do here. Other works proposed similar (anti)patterns detection approaches in service applications. They implemented other techniques, such as bi-level optimization problems [8] or ontologies [9].

In contrast, in this paper, we consider and automatically detect practices related to the development of REST mobile clients. We take also into account the interactions among clients and REST APIs, not on the service side but on the client one. We study in details the use of REST APIs libraries for mobile clients and how Android mobile apps use/consume REST APIs.

III. Study Design

This section presents the design of our study, which aims to answer our research questions. Answering these questions led us to conduct an observational study on hundreds of Android apps from the Google PlayStore. First, we reviewed the literature and developers' forums to build a catalogue of Android REST clients practices. Second, we developed a tool, PIRAC[2], to detect each of the identified practices. Third, we validated the detection precision of PIRAC on 80 Android apps collected from the F-Droid repository. Fourth, we applied PIRAC on 9,173 mobile apps downloaded from Google PlayStore to study the state of the practices in Android clients. Finally, we conducted an observational study on StackOverflow to assess the importance of the identified practices and the use of Android HTTP libraries from the developers' point of view. We describe here the first three steps and the last one in the next section.

A. Step 1: Cataloguing Android REST Mobile Clients Practices

To answer our first research question, we performed a domain analysis of development practices related to Android REST clients by studying their definitions and specifications in the literature as well as in online resources and articles. This domain analysis allowed us to identify seven practices

[2]http://git.sofa.uqam.ca/mabdellatif/pirac/tree/master

that we classified into two sets: (1) good and bad practices for Android REST clients and (2) other practices that are neither good nor bad practices.

List of Good and Bad Practices for Android REST Clients

1) ✓ **Use of third-party HTTP client vs.** ✗ **HttpURLConnection:** This practice concerns the use of third-party libraries to manage REST requests. It is recommended that mobile HTTP queries should be encapsulated in a method proposed by the interface of official third-party libraries, such as OkHttp, Retrofit, Google Volley, etc. A *Non-encapsulated HTTP Query* must be manually built by the developer with all the needed parameters using HttpURLConnection. This process could be long and complicated in some cases and could make the code difficult to maintain.
2) ✓ **Caching vs.** ✗ **Non caching:** Caching is the ability to keep copies of frequently accessed data in several places along the request–response path. Some third-party Android REST libraries offer facilities to manage response caching, such as removing a single cached response, clearing the entire cache, retrieving the date of a cached response, so that developers can accurately decide when an update should be made. It is recommended to cache frequent REST requests to reduce bandwidth usage, network latency, and battery consumption.
3) ✓ **Network connectivity aware vs.** ✗ **Unaware REST service invocation:** This practice pertains to the validation of the network connectivity before sending REST request. It is recommended to check network connectivity (1) to offload heavy REST queries when the device is connected to WiFi, (2) to increase device battery life, (3) to avoid charges related to limited mobile data, and (4) to detect network changes and resume incomplete REST requests.
4) ✓ **JSON vs.** ✗ **XML:** This practice pertains to REST responses parsing by mobile clients. It is recommended to parse REST responses with JSON as it is more human-readable than XML. Also, JSON is more CPU-friendly to parse as it is more compact than XML [10].
5) ✓ **Timeouts vs.** ✗ **Perpetual requests:** This practice is related to setting or not timeouts for REST requests. There are several types of timeouts: connection timeout, read timeout, write timeout, etc. If a mobile client fails to establish a connection to the server within the set connection timeout, it will consider that the request failed. It is recommended to set proper timeouts values to make mobile apps more responsive and user friendly.
6) ✓ **Specification vs.** ✗ **Non specification of a behaviour for failed requests:** The specification of a behaviour when REST requests fail is highly recommended to increase usability and responsiveness of the mobile client apps. Possible behaviours include to drop the requests until some change to the network connectivity or to retry the requests in some chosen time-intervals until successfully, etc.

List of Other Practices for Android REST Clients

1) **Synchronous vs. Asynchronous requests:** REST APIs requests can be synchronous or asynchronous. For synchronous requests, the code execution will block until the API call returns. For asynchronous requests, calls to remote APIs are made while the execution continues. Android developers should carefully choose whether to invoke REST APIs synchronously or asynchronously based on their needs to increase the responsiveness of their apps.

B. Step 2: Detection of the Practices

We developed a framework, PIRAC, to detect the seven identified practices. As depicted in Figure 1, our framework takes as input Android APKs, their corresponding meta-data, and a list of HTTP client libraries, which we use to filter the code to analyze. Our tool uses the SOOT framework [11] to parse the byte-code of mobile apps and extract all the information needed for our analyses, such as classes and methods. PIRAC creates models for Android mobile apps based on the information extracted by SOOT,and those extracted from the reconstructed manifest file. Then, we apply the detection algorithms for the identified practices to detect their uses.

An Android APK contains the compiled source code of the app as well as that of third-party and Android libraries. Running our analyses on the entire packaged code would (1) produce misleading results and (2) affect the execution time of our analyses. Thus, we filter the application code to differentiate the code of the app currently under analysis from the code belonging to Android SDK and third-party libraries. We rely on a *list* of third-party Android libraries, which contains 1,353 package names of the most used libraries identified by Li *et al.* [12]. This list has not been updated since 2016 so we updated it by adding 1,176 package names of the libraries that we manually collected from Android community Web sites[3].

After filtering the app code, we construct models of the APKs that embed all the required information to apply our detection heuristics. Afterwards, we identify classes of interest that are related to REST APIs services calls. Finally, we analyze these classes and identify the practices based on our detection rules.

In the following, we describe some of the detection rules that we use in our framework.

a) Android REST clients identification: Identifying automatically Android apps that make REST calls can be a very complicated task, especially with the use of static analyses methods. Indeed, we can only rely on some used practices in the apps source code to verify whether a given app is potentially using a REST API. We rely primarily on these rules:

[3]https://android-arsenal.com/

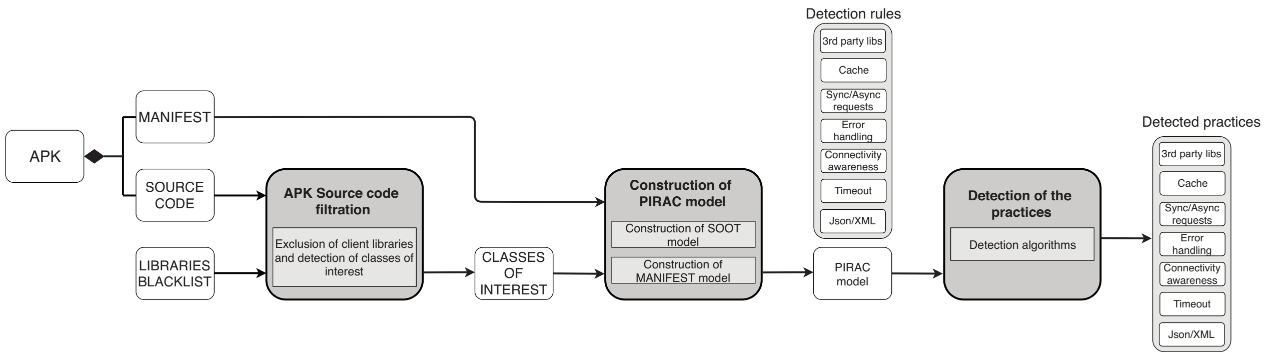

Figure 1. Detection of REST APIs practices usage in Android apps with PIRAC

1) *The use of Android INTERNET permission.* Android apps require Internet permission to access the mobile network. This information is explicitly defined in the Android manifest file.
2) *The referencing of an HTTP client library.* The communication with REST APIs is primarily based on the HTTP protocol. REST apps must use an HTTP library to communicate through this protocol. We rely on a list of 75 HTTP client libraries collected from a Maven repository[4].

When we detect these practices in an app, it is automatically marked as *"Potentially Using a REST API"*. For a better accuracy of the detection of our targeted practices, we ensured that the HTTP client library is executing REST calls and that it is not just referenced for other purposes/uselessly (dead code). Indeed, the simple presence of an HTTP client library in the APK file does not guarantee that this app uses/consumes REST APIs. Also, some Android apps may reference an HTTP client library to use only some of its classes without executing any REST calls.

b) Use of Third-party HTTP Library vs. HttpURLConnection: When it comes to executing HTTP calls, developers can rely on the native HttpURLConnection API or choose to use an external HTTP library. Our tool detects the usage of HttpURLConnection or an external library by analyzing the instantiation of objects and calls made with the specific Java methods of each of the libraries.

c) Cached vs. Non-cached Responses: Developers can use the caching capabilities offered by the HTTP client libraries or develop their own caching strategy. To detect response caching, PIRAC detects the use of relevant methods and classes provided by the libraries, which allow such operations. PIRAC detects also the creation and use of caching folders dedicated to Android apps.

d) Network-connectivity Aware vs. Unaware REST Requests: The Android SDK provides a class named `ConnectivityManager`, which provides information about the network connectivity of a device (network type, availability, etc.). Developers can use this class to adapt their use of REST API requests. To detect such behaviour, our tool detects the invocations of methods that provide information about network connectivity from the *ConnectivityManager* class.

e) JSON vs. XML Response Parsing: The responses from REST APIs come in multiple formats, mainly JSON and XML. The use of JSON is recommended due to its size relatively to XML, its readability by developers, and its ease of use. To detect the usage of one of these two data formats, our framework detects the usage of the most common JSON and XML libraries as well as their instantiation in a code executing an HTTP call.

f) Timeout vs. Perpetual REST Requests: Each one of the studied client libraries provide classes or methods to configure a timeout for their HTTP requests. To detect a timeout configuration, PIRAC detects invocations of these methods or instantiation of classes with a timeout value in the constructor.

g) Specification vs. Non-specification of a Behaviour Upon Failure: When a request fails, regardless of the reason, developers should implement custom logic and show an adapted message to the end user. To detect this behaviour, in the case of `HttpURLConnection`, we detect the retrieval of HTTP status codes after a request. For external libraries, we detect the usage of specific library error-handling methods (e.g., `onFailure()` or `response.isSuccessful()`).

h) Synchronous vs. Asynchronous Requests: When using third-party libraries, it is simple to perform asynchronous requests because these libraries offer specific methods with callbacks. To detect synchronous/asynchronous requests from third-party libraries, we implemented a detection approach specific to each kind of libraries. We rely on the detection of third-party REST clients methods dedicated to synchronous/asynchronous REST requests. The detection of asynchronous requests for Java `HttpURLConnection` is more challenging. When using this library, developers must customize and hard-code asynchronous requests, most commonly using the `AsyncTask` class, which provides methods with callbacks. We analyze the bodies of the methods running in the background to build a method-invocation call graph. Finally,

[4]https://mvnrepository.com/open-source/http-clients

we search for a REST call in each of these methods.

C. Step 3: Validating Harissa for the Detection of REST Mobile Clients Practices

For our validation, we analysed 1448 Android apps from F-Droid repository[5]. We applied our tool on this dataset. We manually validated each of our practices in 80 Android REST clients chosen randomly from the dataset by checking the source code of each app. We chose 80 apps to reach a confidence level of 95%. Table I summarises the detection precision and recall of each targeted practice. The precision detection for each targeted practice by our tool is satisfactory as it varies between 81.91% and 100%. The recall of our tool is also satisfactory as it varies between 80% and 95.33%. We reached an average detection precision of 93.70% while we had an average detection recall of 87.66%. These detection results confirm the reliability of our tool to detect our targeted practices.

Practices	Precision	Recall
Use of third-party HTTP Library vs. HttpURLConnection	100%	80%
Cache usage	99.38%	92.85%
Connectivity aware clients	95.62%	95.33%
JSON vs. XML	89.05%	88.69%
Timeout setting	90.00%	86.36%
Specification of a behavior at request failure	81.91%	85.39%
Synchronous vs. asynchronous calls	100%	85%
Average	**93.70%**	**87.66%**

Table I
OVERVIEW OF THE DETECTION PRECISION OF OUR TOOL

IV. OBSERVATIONS

A. Dataset

To conduct our observational study, we downloaded randomly 9,173 Android apps from the Google PlayStore. We used Androzoo[6] to collect APKs and meta-data for these apps. As depicted in Figure 2, the apps belong to varied categories, such as games, communication, weather, etc. We applied PIRAC on the data-set to extract all REST clients for a total number of 1,595 Android REST clients. These apps are also of different sizes, as shown in Figure 3.

Observation 1: We observe that the three main categories that use the most Android REST clients are *Lifestyle*, *Business& Finance*, and *Video& Media.*

B. Observations about the Identified Practices

In this section, we present our observations about the distribution of good and bad practices of REST APIs in Android clients.

[5]https://f-droid.org/en/
[6]https://androzoo.uni.lu/

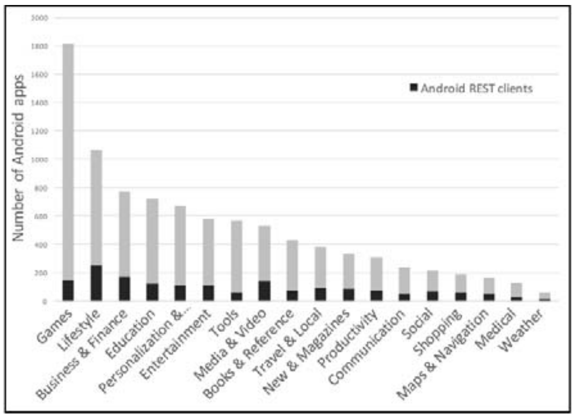

Figure 2. Distribution of Android apps by category

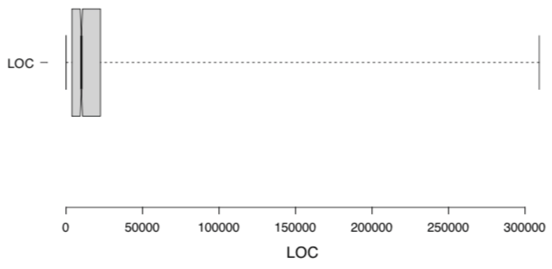

Figure 3. Android REST clients LOCs

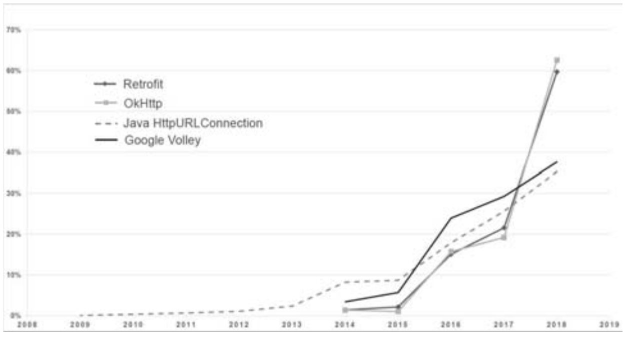

Figure 4. Evolution of the use of HTTP libraries in time

a) Use vs. Non-use of Third-party Libraries for HTTP Requests: Based on our previous work [5], the most used Android libraries to execute HTTP requests are OkHttp, Retrofit, Google Volley, and Java HttpURLConnection. We focused on these libraries and studied their usage evolution in REST mobile clients. We noticed that Java HttpURLConnection is the oldestlibrary to execute HTTP requests in REST Android clients. With Figure 4, we noticed that HttpURLConnection is getting less used by developers compared to newer third-party libraries (i.e., OkHttp, Retrofit, and Google Volley).

In 2014, OkHttp, Retrofit, and Google Volley have been released to ease and simplify the management of HTTP requests by Android clients. We observe that between 2014

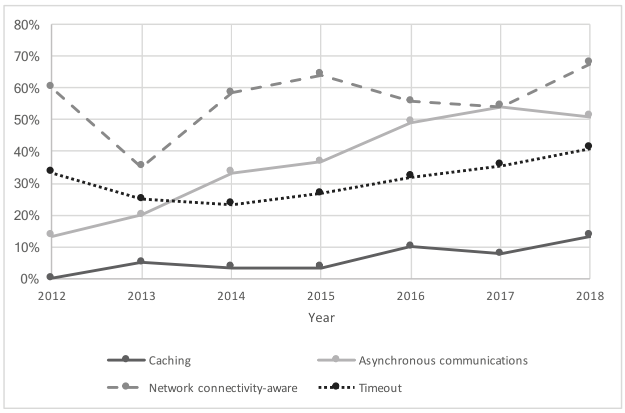

Figure 5. Evolution of the practices in time

and 2018, the usage evolution of OkHttp and Retrofit is almost the same because Retrofit uses the OkHttp library for HTTP requests. The usage evolution of these two libraries increased rapidly (60% and 64%, respectively) between 2017 and 2018 compared to Google Volley (37%) and Java HttpURLConnection (35%).

Observation 2: Recently, Android developers have been using OkHttp, Retrofit, and Google Volley to manage REST requests because they offer more interesting features compared to Java HttpURLConnection. For example, implementing asynchronous requests is easier when using third-party libraries because developers do not need to use Android `AsyncTasks` to run network operations in a separate thread.

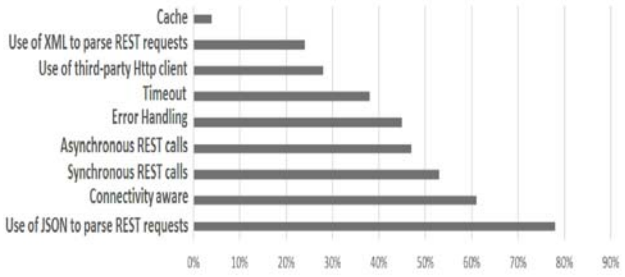

Figure 6. Distribution of Android REST client practices

b) Cached vs. Non-cached Responses: Although caching helps developers implement highly capable and scalable REST clients and services by limiting repetitive interactions, REST Android developers widely ignore the caching capability for their REST requests. Based on Figure 6, we observe that only 4% of REST Android clients cache REST responses, which forces the apps to retrieve duplicate responses from servers. This bad practice is known as *Ignoring Caching* anti-pattern [4], [13]. Also, based on Figure 5, we observe that the use of

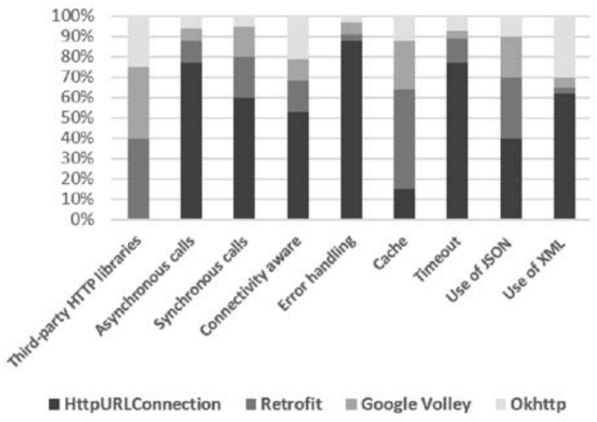

Figure 7. Distribution of Android HTTP libraries by practices

response caching has been rapidly increasing during the past five years.

As depicted in Figure 8, we observe that Android apps using third-party libraries tend to consider more frequently the use of caching (85% of Android apps considering caching are using third-party libraries).

Observation 3: Android developers tend to ignore the use of caching to manage REST requests. However, Android apps using third-party libraries tend to consider the usage of caching more.

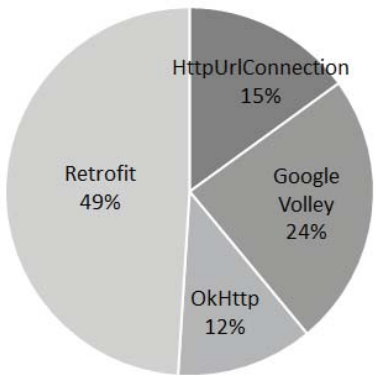

Figure 8. Cache usage by Android REST client library

c) Network-connectivity Aware vs. Unaware REST requests: Based on Figure 6, we observe that 62% of REST Android clients are aware of/check network connectivity before performing REST requests, which is a good practice with several advantages. Checking network connectivity first lets mobile clients improve battery life by offloading heavy network requests when the device is connected to WiFi, for example. Also, for most users, mobile data is limited and can be quite expensive.

Observation 4: The check of network connectivity before performing REST requests is a good, common practice for Android REST clients.

d) JSON vs. XML Response Parsing: Using JSON to communicate via REST APIs is highly recommended in comparison to XML due to its ease of use and of parsing. Based on Figure 6, we observe that this good practice is widely adopted by Android developers with 78% of Android REST clients using JSON to parse REST APIs responses while only 24% use XML.

Observation 5: The good practice of using JSON to communicate REST APIs responses is widely adopted by Android developers.

e) Timeout vs. Perpetual REST Requests: Based on Figure 6, we observe that only 36% of Android REST clients consider setting timeouts when performing HTTP requests. As depicted in Figure 9, we also observe that 77% of Android REST clients that consider timeouts use HttpURLConnection. This high percentage is due to HttpURLConnection being the only studied HTTP library that does not specify a default value for timeouts. When using such a library, developers must specify a value of timeout.

Observation 6: Although it is recommended to set proper timeouts values to make the mobile apps more responsive and user friendly, Android developers tend to ignore this good practice.

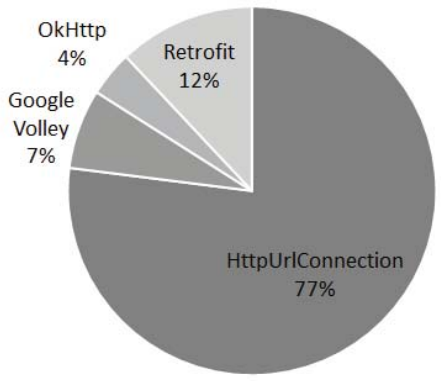

Figure 9. Timeout usage by client library

f) Specification vs. Non-specification of a Behaviour Upon Failure: Based on Figures 6 and 10, we observe that only 45% of Android REST clients handle HTTP requests failures when calling REST APIs. We also observe that Android developers do not take full benefit from the error-handling features provided by third-party libraries as 88% of error handling in Android apps are implemented with HttpURLConnection. The wide use of HttpURLConnection may hinder the adoption of such a good practice as developers must implement themselves how to catch the requests failures and specify what should happen in case of failures.

Observation 7: The specification of a behaviour in case of request failures is widely ignored by Android developers as they poorly use third-party HTTP libraries that facilitate errors handling.

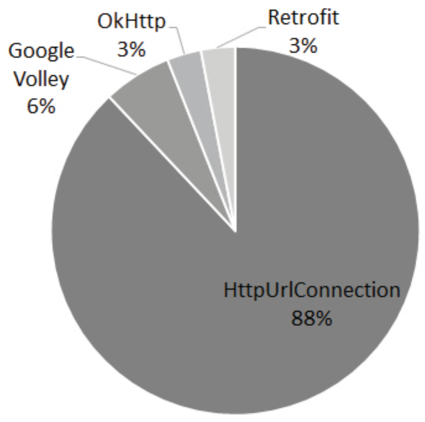

Figure 10. Error-handling usage by Android HTTP client library

g) Synchronous vs. Asynchronous REST Requests: Based on Figure 6, we observe that synchronous and asynchronous calls to REST APIs are almost equally used by Android clients. We also observe that asynchronous communications increased by almost 40% in the past five years thanks to the features offered by third-party libraries: Android developers do not have to manage asynchronous calls by themselves anymore. However, Android developers still poorly rely on third-party HTTP libraries to make asynchronous calls as 77% of asynchronous communications are made with HttpURL-Connection.

Observation 8: Synchronous and asynchronous calls to REST APIs are almost equally used by Android clients. Also, we observe that Android developers are still relying on third-party HTTP libraries to make asynchronous calls.

C. Mining Android REST API Usage Practices on Stack Overflow

We mined thousands of StackOverflow posts to assess the importance of the identified practices from the developers' point of view. We executed several search queries related to our identified practices for Android apps and manually filtered irrelevant posts. We obtained a total number of 12,478 posts related to REST API usage practices in Android apps.

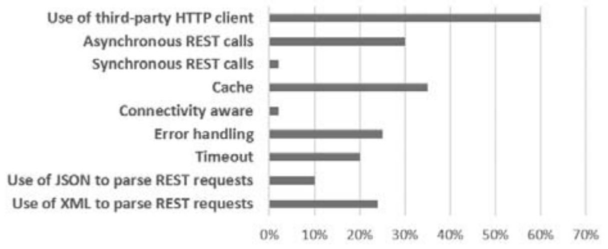

Figure 11. Distribution of the questions on StackOverflow about the identified practices

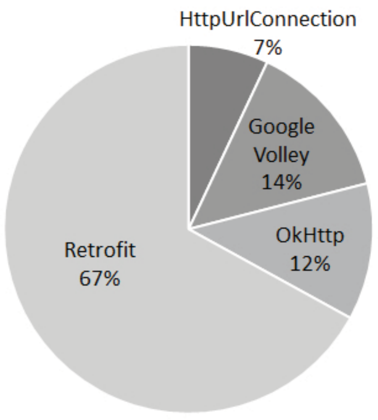

Figure 12. Distribution of the questions on StackOverflow about Android HTTP libraries

Based on Figure 11, we conclude that the most discussed practice on StackOverflow by Android developers is the use of third-party HTTP libraries. Android developers seem more interested about using third-party HTTP libraries to ease the management of HTTP requests.

As depicted in Figures 12 and 13, we also report the distributions of the questions in StackOverflow per Android HTTP library. Based on Figure 13, we report that, since the apparition of Android HTTP dedicated libraries in 2013, the number of questions about HttpURLConnection has been significantly decreasing. Also, based on Figure 12, we observe that there is a high interest in using Android HTTP client libraries by Android REST clients developers to implement the identified practices. Indeed, we found that Android developers ask most questions about REST requests, when using Retrofit (67%). Also, we found that 14% of the questions were about Google Volley, 12% were about OkHttp, and only 7% were about HttpURLConnection.

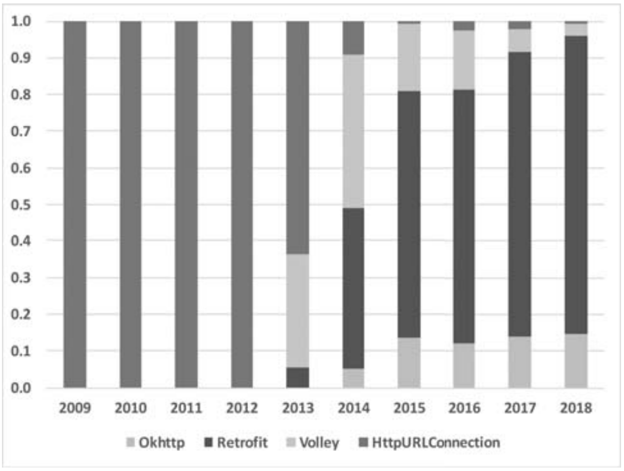

Figure 13. Evolution of the questions on StackOverflow about Android HTTP libraries over the years

We explain the high interest by Android developers in Retrofit by the poor documentation of this library in comparison to its features and its high performance in comparison to OkHttp, Google Volley, and HttpURLConnection[7].

Observation 9: Since the apparition of third-party HTTP libraries, Android developers ask more questions about them in comparison to HttpURLConnection.

There is also a concern by Android developers on StackOverflow about asynchronous (30%) vs. synchronous (2%) HTTP requests to REST APIs because (1) asynchronous communications with REST APIs have more complicated implementation than synchronous ones and (2) the multiple ways provided by HTTP libraries to handle such kind of requests are confusing to novice developers. Android developers seem more interested in managing such calls with third-party HTTP libraries (mainly Retrofit as we can see in Figure 14) as they offer more straightforward ways to manage such kind of requests in comparison to HttpURLConnection.

We report that Android developers do not ask many questions about the “connectivity aware" request practice (only 2%), probably because it is very simple to implement.

Also, we observe that Android developers do not ask many questions about parsing REST APIs responses with JSON. The more problematic parsing format is XML with 25% of the questions about XML parsing and only 9% are about JSON.

[7]https://bit.ly/2ypkEl9

Observation 10: There is a high interest on StackOverflow in using third-party HTTP libraries for Android REST clients. Also, making asynchronous calls, caching responses, error handling, and XML parsing are the most discussed practices.

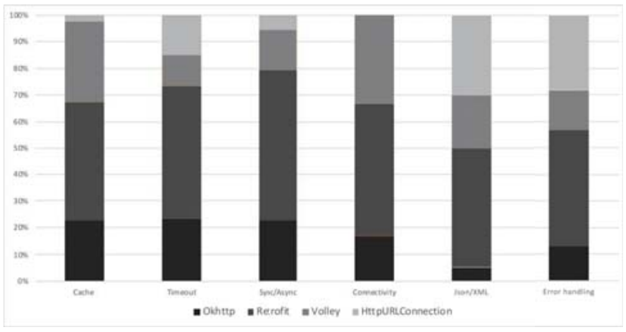

Figure 14. Distribution of the questions on StackOverflow about Android REST client libraries by practice

V. Answers to Our Research Questions

We now answer our research questions using our previous observations.

A. *RQ1: What is the state of the practices in the use of REST APIs by Android apps?*

We identified several practices of REST APIs uses in Android clients that we classified into two sets: (1) good and bad practices for Android REST clients and (2) other practices that are neither good nor bad.

We observed that only two good practices are widely considered by Android developers when implementing their mobile clients: network connectivity awareness and JSON response parsing. Moreover, based on our observations, we found that Android developers widely ignore some good practices of REST APIs uses in Android apps: the use of HTTP third-party libraries, caching responses, setting timeouts, and error handling.

Based on our observations we also found that:

- Caching REST responses is the most problematic practice for Android developers. This is reflected by the high number of questions about caching REST requests vs. the low percentage of Android apps using cache as shown by Figures 6 and 11).
- Specifying a behavior when requests fail is also very problematic for Android developers as they widely ignore this good practice and ask many questions about it on StackOverflow.
- Asynchronous REST request practice seems to be difficult for developers as reflected by the high number of questions in this topic vs. the high number of apps relying on asynchronous calls.
- Connectivity aware REST requests and JSON responses parsing are the most evident/trivial practices for Android developers as reflected by the low number of questions about these practices on StackOverflow vs. their high usage in Android apps.

B. *RQ2: What is the state of the implementation of HTTP client libraries in Android REST clients?*

Choosing an efficient mobile HTTP library to communicate with a REST API can be difficult for developers because they must handle many aspects, such as making connections, caching, retrying failed requests, parse responses, handle errors, etc. Although only 28% of Android REST clients use third-party HTTP libraries, since the apparition of such libraries, developers have been using them increasingly. The evolution of their usage by developers is rapidly growing (almost 98% of apps released in 2018 are using third-party libraries). Also, since the apparition of third-party libraries, Android apps tend to use more and more good practices when performing REST requests. Such practices are caching requests, timeout, connectivity awareness, JSON vsẊML responses parsing, and error handling.

After comparing the state of implementation of HTTP libraries in Android apps and the distribution of questions about these libraries in StackOverflow, we found that Retrofit is the most used and discussed third-party HTTP library because of its ease of use, high performance and its caching features.

Finally, we found that apps using third-party libraries tend to use more "good" practices: for example, Figure 7 shows that 85% of the apps with cache are using third-party libraries and 60% of the apps with JSON responses parsing are also using third-party libraries. The non-use of HTTP third-party libraries may hinder the adoption of good practices, which is the case for timeouts and error handling.

Thus, we answer that developers should use third-party libraries to benefit from their implementations of good practices of REST API uses in Android mobile apps.

VI. Threats to Validity

In this section, we discuss the threats to validity of our observational study and the measures that we took to limit them.

Threats to internal validity. Although we did not carry any statistical tests, we assume that the identified practices are representative characteristics of the Android REST clients. There may be other characteristics that describe more accurately these Android REST API clients. We will study more practices to cover possible other characteristics. We also related the practices manually thanks to the information provided in the literature. Yet, other researchers should perform similar analyses to confirm/infirm ours.

The source code of some of the apps in our data-set is obfuscated. Even a simple obfuscation affects our detection precision because names of packages, classes, and methods change. Also, we do not detect custom caching strategies as we only detect cache-related and asynchronous calls using third-party libraries. Custom caching and synchronisation strategies are challenging to cover because they vary among apps. More research should be done to cover such custom implementations.

Another threat arises from the data gathered from StackOverflow. Although we used several search queries to collect our data, we may have missed some interesting posts about REST APIs uses in Android apps. To mitigate this risk, we used a combination and keywords to gather the most significant posts on Stack Overflow. Also, we do not consider any filtering process for duplicate posts as they are useful for our study: they show the developers' interest in using/adopting a particular REST practice.

Threats to external validity concern the generalizability of our results. Although we presented the largest study on the practices and implementations of Android REST apps, we cannot generalise our results to all mobile apps. Future work is necessary to analyze more mobile REST clients, from other mobile platforms to confirm and—or infirm our observations on their design quality characteristics. Also to generalize our results, we should consider other developers' forums, such as GitHub, Quora, or Reddit.

VII. Conclusion and Future Work

Several research works studied REST APIs development practices for mobile apps. However, little is known on how Android mobile apps use/consume REST APIs.

We described in this paper an observational study about the state of the practices of REST APIs uses in Android mobile apps. We provided four contributions: (1) we reviewed the literature extensively and compiled a catalogue of seven practices related to the development of apps using REST APIs. These practices pertained to the use of dedicated, third-party libraries and helped to understand how Android apps use/consume REST APIs, (2) we proposed a framework, PIRAC, to detect automatically occurrences of the identified practices using detection rules, (3) we conducted an observational study on over 1,595 REST mobile apps out of 9,173 apps downloaded from the Google PlayStore to report how they use/consume REST APIs, (4) we mined 12,478 StackOverflow posts to assess the importance of the identified practices and the use of Android HTTP libraries from the developers' point of view.

Based on our observational study, we found that developers used to ignore some good practices of REST APIs uses in Android clients. Such practices are the use of HTTP third-party libraries, caching responses, timeout management, and error handling. We found that only two good practices are widely considered by Android developers when implementing their mobile clients: network connectivity awareness and JSON vs. XML responses parsing. We also found that, although only 28% of Android REST clients use third-party HTTP libraries, since the apparition of such libraries, developers have been using them increasingly.

Thus, we concluded that service providers must strive to make their libraries as simple as possible and that mobile apps developers should consider existing HTTP libraries to make use of good practices and get benefit from their features, such as asynchronous requests, timeout management, caching responses, and error handling.

Future work includes selecting and–or defining more practices and analyzing their prevalence in Android apps as well as other apps for other mobile operating systems, in particular iOS. Thus, we could further recommend to developers best practices when developing apps that use REST APIs. We also want to study source-code transformations that would allow developers to migrate their apps from one library, in particular HttpURLConnection, to other libraries, like Retrofit or Google Volley, to benefit from their features.

References

[1] C. Rodríguez, M. Báez, F. Daniel, F. Casati, J. C. Trabucco, L. Canali, and G. Percannella, "REST apis: A large-scale analysis of compliance with principles and best practices," in *16th International Conference Web Engineering*, ser. Lecture Notes in Computer Science, vol. 9671. Springer, 2016, pp. 21–39.

[2] F. Petrillo, P. Merle, N. Moha, and Y. Guéhéneuc, "Are REST apis for cloud computing well-designed? an exploratory study," in *14th International Conference on Service-Oriented Computing*, ser. Lecture Notes in Computer Science, vol. 9936. Springer, 2016, pp. 157–170.

[3] H. Brabra, A. Mtibaa, L. Sliman, W. Gaaloul, B. Benatallah, and F. Gargouri, "Detecting cloud (anti) patterns: Occi perspective," in *International Conference on Service-Oriented Computing*. Springer, 2016, pp. 202–218.

[4] F. Palma, J. Dubois, N. Moha, and Y. Guéhéneuc, "Detection of REST patterns and antipatterns: A heuristics-based approach," in *12th International Conference on Service-Oriented Computing*, ser. Lecture Notes in Computer Science, vol. 8831. Springer, 2014, pp. 230–244.

[5] M. A. Oumaziz, A. Belkhir, T. Vacher, E. Beaudry, X. Blanc, J.-R. Falleri, and N. Moha, "Empirical study on rest apis usage in android mobile applications," in *International Conference on Service-Oriented Computing*. Springer, 2017, pp. 614–622.

[6] F. Palma, N. Moha, and Y.-G. Guéhéneuc, "Unidosa: The unified specification and detection of service antipatterns," *IEEE Transactions on Software Engineering*, 2018.

[7] F. Palma, J. Gonzalez-Huerta, N. Moha, Y. Guéhéneuc, and G. Tremblay, "Are restful apis well-designed? detection of their linguistic (anti)patterns," in *13th International Conference on Service-Oriented Computing*, ser. Lecture Notes in Computer Science, vol. 9435. Springer, 2015, pp. 171–187.

[8] H. Wang, M. Kessentini, and A. Ouni, "Bi-level identification of web service defects," in *14th International Conference on Service-Oriented Computing*, ser. Lecture Notes in Computer Science, vol. 9936. Springer, 2016, pp. 352–368.

[9] H. Brabra, A. Mtibaa, L. Sliman, W. Gaaloul, B. Benatallah, and F. Gargouri, "Detecting cloud (anti)patterns: OCCI perspective," in *14th International Conference on Service-Oriented Computing*, ser. Lecture Notes in Computer Science, vol. 9936. Springer, 2016, pp. 202–218.

[10] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, "Comparison of JSON and XML data interchange formats: A case study," in *22nd International Conference on Computer Applications in Industry and Engineering*, 2009, pp. 157–162.

[11] R. Vallée-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan, "Soot: A java bytecode optimization framework," in *CASCON First Decade High Impact Papers*. IBM Corp., 2010, pp. 214–224.

[12] L. Li, T. F. Bissyandé, J. Klein, and Y. Le Traon, "An investigation into the use of common libraries in android apps," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 403–414.

[13] S. Tilkov, "Rest anti-patterns," *InfoQ Article (July 2008)*, 2008.