Bachelor Degree Project

Systematic literature review on REST antipatterns

Author: Andrei Neagu
Supervisor: Francis Palma
Semester: VT 2021
Subject: Computer Science

Abstract

Context: API growth is accelerating. RESTful APIs are gaining traction and are backed by major players. Extending the APIs commonly introduce antipatterns, which are bad solutions to problems.
Objective: The purpose of this review is to identify and analyze the current, state-of-the-art approaches in detecting antipatterns in RESTful APIs.
Method: Six research questions are clearly defined. Search strings are used in digital libraries to identify studies in the field of antipatterns in RESTful APIs. The studies must come from reputable sources. Studies are subjected to inclusion-exclusion and quality assessment.
Results: Eight studies were selected, four were from a Conference and four were from a Journal, an even distribution.
Conclusion: Various techniques were discovered, each selected study presented a single technique. Classifications for the techniques and antipatterns were made. The research field is young with future work planned.

Keywords: REST, Antipatterns, Detection, API

Contents

1 Introduction **1**

2 Related work **2**

3 Research method **3**
3.1 The Research Questions . . . 3
3.1.1 Question Structure . . . 4
3.2 Search strategy . . . 4
3.2.1 Search strings . . . 4
3.2.2 Literature resources . . . 4
3.3 Study selection . . . 4
3.3.1 Scrutiny . . . 5
3.3.2 Quality assessment of selected studies . . . 6
3.4 Data synthesis . . . 7

4 Threats to validity **8**

5 Results **9**
5.1 Overview of the selected studies . . . 9
5.2 Frequency analysis of words in the selected studies . . . 10
5.3 The state-of-the-art techniques employed in the detection of antipatterns (RQ1) . . . 10
5.3.1 Detection of REST Patterns and Antipatterns: A Heuristics-Based Approach . . . 10
5.3.2 Are RESTful APIs Well-Designed? Detection of their Linguistic (Anti)Patterns . . . 10
5.3.3 Detecting Cloud (Anti)Patterns: OCCI Perspective . . . 11
5.3.4 Semantic Analysis of RESTful APIs for the Detection of Linguistic Patterns and Antipatterns . . . 11
5.3.5 An Observational Study on the State of REST API Uses in Android Mobile Applications . . . 11
5.3.6 On semantic detection of cloud API (anti)patterns . . . 12
5.4 UniDoSA: The Unified Specification and Detection of Service Antipatterns 12
5.4.1 A URI parsing technique and algorithm for anti-pattern detection in RESTful Web services . . . 12

6 The classifications of the state-of-the-art approaches (RQ2) **13**

7 The processes involved in the state-of-the-art approaches (RQ3) **14**

8 The accuracy of the state-of-the-art approaches (RQ4) **15**
8.1 Observations . . . 15

9 The pros and cons of the state-of-the-art approaches (RQ5) **15**
9.1 Types of antipatterns . . . 15
9.2 Results . . . 16

10 The subjects and objects in the relevant literature (RQ6) **16**

11 Discussion and results **17**

12 Conclusion **18**

References **19**

1 Introduction

Representational State Transfer (REST) is a standard for developing applications that use Web services. The term representational state transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation [1]. A RESTful service must provide its Web resources in a textual form and allow them to be read and modified with a stateless protocol and a clearly defined set of operations. For a Web service to be considered RESTful it must adhere to the architectural constraints of REST.

The difference between emerging APIs and traditional SOAP is targeting third-party consumers and are designed in a HTTP friendly way. Salesforce and eBay took the first steps, major players such as Facebook, Twitter, Google, LinkedIn, Microsoft and Amazon followed. API growth is accelerating. Web APIs are here to stay [2].

Designing and developing Web-based applications confronts common software engineering challenges [3]. Those challenges represent the introduction of antipatterns.

Antipatterns are poor design practices that degrade the design quality and reduce maintainability, extensibility and reusability of software. Design patterns propose good solutions to recurring software problems, the absence of design patterns often results in the presence of antipatterns. Detecting and correcting antipatterns in APIs improves quality.

Numerous methods and techniques are proposed in the literature to detect antipatterns in RESTful APIs however the research field is not organized. There are several studies that summarized detection methods in object-oriented systems or SOAP-based Web services. However, to date, there is no consolidated study that reviews the methods and techniques applied in detecting antipatterns in RESTful APIs or RESTful Web services.

We plan to perform a literature review to study the current state of the art methods and techniques that are applied to the detection of antipatterns in RESTful APIs or Web services. The major objective to this SLR is to learn about the relevant approaches in detecting antipatterns in RESTful APIs and their characteristics.

A systematic literature review is a means of evaluating and interpreting all available research relevant to a particular research question, topic area, or phenomenon of interest. Systematic reviews aim to present a fair evaluation of a research topic by using a trustworthy, rigorous, and auditable methodology [4].

Taking into consideration the advice of my supervisor, the primary studies that will be selected for reviewing need to come from reputable sources.

The intended target group is the author and other students in the field of Computer Science.

The number of selected studies for review is eight. Using the data collected we try to answer the research questions defined in the research methodology.

2 Related work

There are no other systematic literature reviews in this research domain. A scoping review is a relatively new approach to evidence synthesis and differs from systematic reviews in its purpose and aims [5]. The purpose of a scoping review is to provide an overview of the available research evidence without producing a summary answer to a discrete research question [6]. Alshraiedeh and Katuk [7] talk about SOAP and REST antipatterns in the scoping review. A relevant research question is RQ2: What are the types of antipatterns found in RESTful web service? The review also lists all selected studies in Section 2. No other related work was found.

3 Research method

The main goal of this SLR is to report the detection methods for antipatterns in RESTful APIs, more details can be found written in Section 1. The evidence-based research was initially introduced in the medical domain because expert opinion is not reliable. Evidence-based software engineering (EBSE) is the analogous concept. The concept tries to bring evidence to decisions made in the software engineering. Keele et al. [4] suggested software engineers apply EBSE.

This section describes the protocol (Fig. 3.1) we will follow to perform the SLR. The protocol contains the research questions, the search strategy, the study selection and the data extraction.

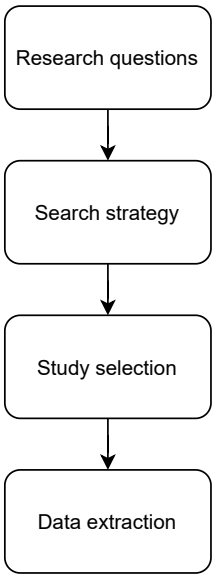

Figure 3.1: Phases of the research method.

3.1 The Research Questions

ID	Research Question	Motivation/Clarification
RQ1	What are the state-of-the-art techniques employed in the detection of antipatterns?	Identification of antipattern detection techniques
RQ2	What are the classifications of the state-of-the-art approaches?	Classification of techniques identified
RQ3	What are the processes involved in the state-of-the-art approaches?	Documenting the processes involved
RQ4	What is the accuracy of the state-of-the-art approaches?	Identification of the accuracy of the techniques
RQ5	What are the pros and cons of the state-of-the-art approaches?	Important to know usefulness of the approaches and drawbacks
RQ6	What are the subjects and objects in the relevant literature?	The subjects are the antipatterns and the objects are the RESTful APIs

Table 3.1: Research questions

3.1.1 Question Structure

The research questions (Table 3.1) are based on the following PICOC(Population, Intervention, Comparison, Outcome, Context) criteria.

- Population: API development, REST
- Interventions: antipatterns, anti-pattern, anti-patterns
- Comparison: the analysis of the impact in the relevant literature between the population and the interventions
- Outcomes: a classification of the techniques that are used to identify antipatterns in REST
- Context: an exclusive focus on evidence collected from the relevant techniques on antipatterns

Through this SLR, we try to answer the six research questions written above.

3.2 Search strategy

In this section we discuss the search strategy we used. An effective search string is essential to search relevant literature. The details of the search process are explained bellow.

3.2.1 Search strings

[("antipattern" **OR** "anti-pattern" **OR** "antipatterns")
OR ("detecting" **OR** "detection")
OR ("mobile" **OR** "android")]
AND ("REST" **OR** "RESTful")

In this SLR we focus on the methods and techniques applied for detection of antipatterns in RESTful APIs. We derive the major terms as being " antipatterns" , "detection", "mobile" and "REST". We identify alternate spellings and synonyms. We identify relevant keywords in the literature. We rely on Boolean operators such as "AND", "OR" to narrow or broaden the search appropriately. The search string will search for studies ranging from the year 2000 until 2021.

3.2.2 Literature resources

Seven electronic databases were used to extract data in this research. These are: IEEE Xplore, ACM Digital Library, ScienceDirect, Springer, Web of Science and Google Scholar. The title, abstract and keywords are used for searching of the relevant literature in the digital libraries mentioned in this section.

3.3 Study selection

This section is focused on identifying the primary studies that provide direct evidence about the research questions. Table 3.2 reports the result of each term associated with antipatterns.

ID	Term search	IEEE	ACM	ScienceDirect	Springer	Web of Science	Google Scholar	Total
1	antipattern/anti-pattern/antipatterns AND REST/RESTful	2	4	1	7	7	3830	3851
2	detecting/detection AND REST/RESTful	1087	228	242	406	992	943	3898
3	mobile/android AND REST/RESTful	679	580	476	386	691	614	3426
Total		1089	232	243	413	999	5387	11175

Table 3.2: Number of selected studies

3.3.1 Scrutiny

For a SLR to be systematic the search process has to search all relevant literature. To select the relevant literature we apply a three stage process described bellow.

Stage 1: We extracted 11175 studies from a thorough search in the digital literature resources. The extracted studies are narrowed down to the domain of Computer Science.

Stage 2: We manually examine the title, abstract and keywords of the studies. In this stage we also remove duplicated studies and we only present the most up-to-date findings to avoid published duplicated data or out-of-date data. This resulted in 9 studies out of 11175. We also pursue the reference lists of relevant papers and if they result in additional papers we add them to our list. As a result 11 more studies were added after pursuing the reference lists of the relevant papers however after removing duplicated studies, studies that report duplicated data and out-of-date studies only 20 studies were selected.

To be mentioned is the fact that the search string with ID number 2, the one that yielded the most hits on the digital libraries, resulted in no additional relevant studies.

Stage 3: Furthermore, the filtered studies go through the inclusion and exclusion criteria detailed bellow. The inclusion criteria:

- Studies published in English
- Studies published between 2000 and 2021
- Studies that focus on antipatterns in RESTful web services
- All studies that have the potential of answering one research question

The exclusion criteria:

- Studies not published in English
- Short articles(less than five pages)
- No focus on RESTful antipatterns explicitly
- No detection techniques of antipatterns
- Gray literature

Consequently, 9 studies are left that satisfy the inclusion and exclusion criteria in Stage 3.

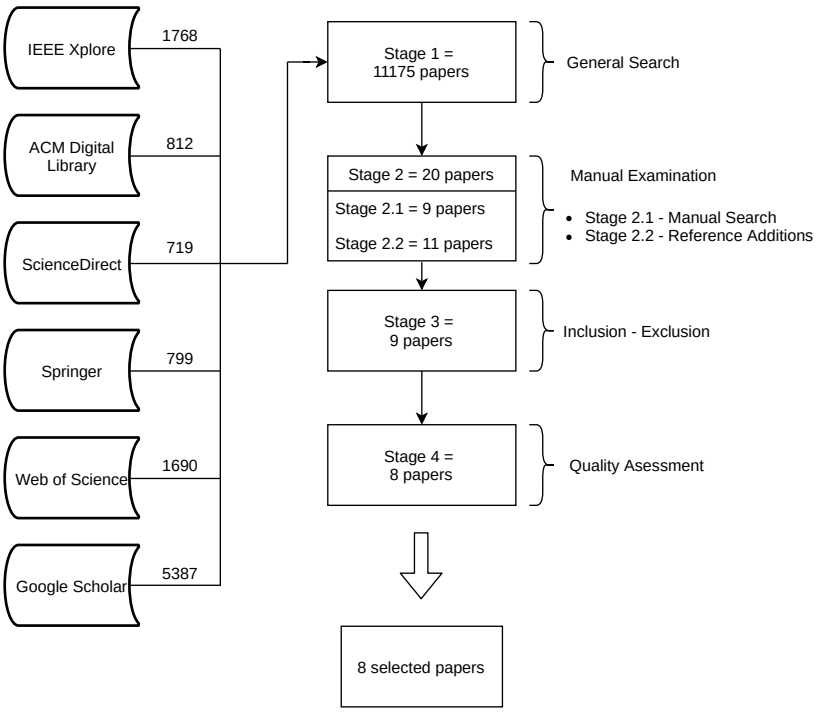

Figure 3.2: Study selection process

3.3.2 Quality assessment of selected studies

Aspects	Checklist questions
Design	Are the aims of the research clearly articulated?
Conduct	Did the study evaluate the proposed method and are the results explained well?
Analysis	Did the study apply any accuracy measures for evaluation?
Conclusion	Did the primary study explain well all mentioned research questions?

Table 3.3: Quality assessment checklist questions

There are a number of ways to assess the quality of studies in an SLR. We formulated a checklist (Table 3.3) to check the quality of the studies based on the guidelines set in Keele et al. [4]. Each question has three answers: Yes, No and Partly. These values equal to 1.0, 0.0 and 0.5 . The quality score is determined by the sum of the values. The studies that will be considered are studies with a score greater than 2.

Reference	Design	Conduct	Analysis	Conclusion	Score
Palma et al.[3]	1	1	1	0.5	3.5
Palma et al.[8]	1	1	1	0.5	3.5
Brabra et al.[9]	1	1	1	0.5	3.5
Rodriguez et al.[10]	0.5	0.5	0	0.5	1.5
Palma et al.[11]	1	1	1	1	4
Belkhir et al.[12]	1	1	1	1	4
Brabra et al.[13]	1	1	1	1	4
Palma et al.[14]	1	1	1	1	4
Alshraiedeh et al.[15]	1	0.5	1	0.5	3

Table 3.4: Quality assessment results

At the end of the exercise, in Table 3.4, we wrote the quality assessment results. Brabra et al.[9] has been included with one mention, the antipatterns detected are significant only in an OCCI context, we name them OCCI REST antipatterns, however the approach is relevant. Rodriguez et al.[10] was found to be of a significantly lesser quality than the rest so it was not included. After the end of Stage 4, the Quality assessment, 8 studies that will help us in providing answers to the research questions satisfy our criteria. The number of selected studies is one less than after Stage 3. Figure 3.2 shows the representation of study selection criteria.

3.4 Data synthesis

Data synthesis involves collating and summarising the results of the included primary studies [4]. In this step we summarize proofs, synchronize and precisely document clearly defined data. The data extraction forms must be designed to collect all the information needed to address the review questions and the study quality criteria [4]. We extracted the data in an Excel and published it online where we present the results.

Search Criteria	Data Item	Motivation
General	Title Year Type of article	Needed for visualizing the interest in the domain
RQ1	Approach name Approach implementation Frequency analysis	Details needed for answering RQ1 Relevant exercise
RQ2	Type of approach	Classifications of the approaches
RQ3	Steps	Important to list the steps of the approaches
RQ4	Precision Recall F-measure	Quantitative data needed to answer RQ4
RQ5	Applicability Threats to validity info	Qualitative data needed to answer RQ5
RQ6	Antipatterns APIs tested	The subjects and the objects of the studies

Table 3.5: Data extraction criteria

Qualitative data is data describing the attributes or properties that an object possesses. The properties are categorized into classes that may be assigned numeric values. However, there is no significance to the data values themselves, they simply represent attributes of the object concerned.

Quantitative data is data expressing a certain quantity, amount or range. Usually, there are measurement units associated with the data, e.g. metres, in the case of the height of a person.

Some studies document their findings differently. The qualitative data related to RQ1 was organized in a coherent manner. In RQ2, we focused on the type of the approaches, we explained them and we defined classifications for the approaches used in the studies. In RQ3, the collected data was presented as bullet points in a list. In RQ4, we collected the quantitative data and synchronized it. To be mentioned is the fact that the studies validated the approaches differently and measured and documented the findings differently. In RQ5

we documented the pros and cons using metrics such as the domain of applicability, age, extensibility and whether or not the data necessary to replicate the study was published. In RQ6 we documented the APIs tested and the antipatterns detected. To be mentions is that some studies document the antipatterns as the opposite of patterns while others focus only on antipatterns.

4 Threats to validity

One critical element in applying the research method is to mitigate threats to validity. The number of studies included in the review is low. We have consulted the relevant literature[16] and we identified three major validity threats along with mitigation strategies.

- Bias in data extraction
- Subjective interpretation about the extracted data
- Subjective quality assessment

The first and second threat exist due to the fact that this review is done at a bachelor's level with limited resources. To address this threats Zhou et al.[16] suggests establishing a review method and data reviews by external reviewers. We minimized the threat by consulting with the supervisor which provided us with qualified opinions.

The third threat exists due to the fact that that the quantitative and qualitative data extraction was done by a single researcher at a bachelor's level. We minimized the threat by applying the review method and reviewing the data collected multiple times, at every stage.

5 Results

In this section, the results we achieved using the above mentioned research methodology will be presented. In the beginning we will present an overview about the studies included in the review and lastly we will answer our research questions.

5.1 Overview of the selected studies

The review contains 8 selected papers. In Figure 5.3 we can see the year of publication for the selected studies. Since the creation of REST by Roy Fielding in his doctoral thesis, we can see that only starting with the year 2013 there was an interest in detecting antipatterns from academia. In Figure 5.4 we can see that the distribution of the papers is even, 4 studies are published in a Conference, the other 4 are published in a Journal.

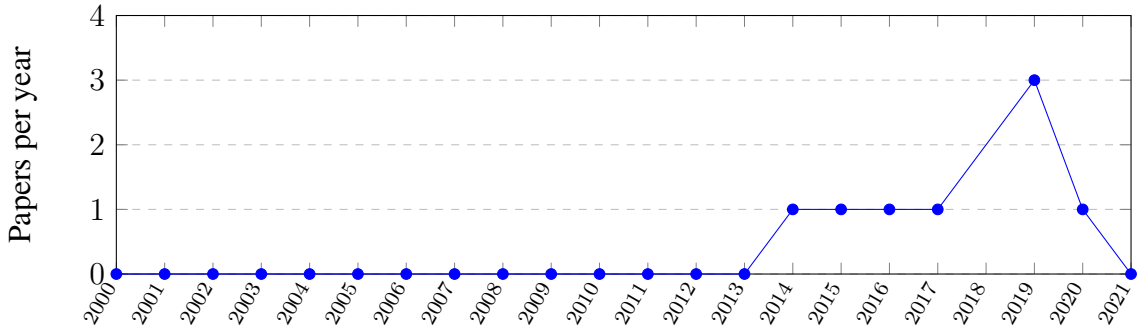

Figure 5.3: Year-wise distribution of studies

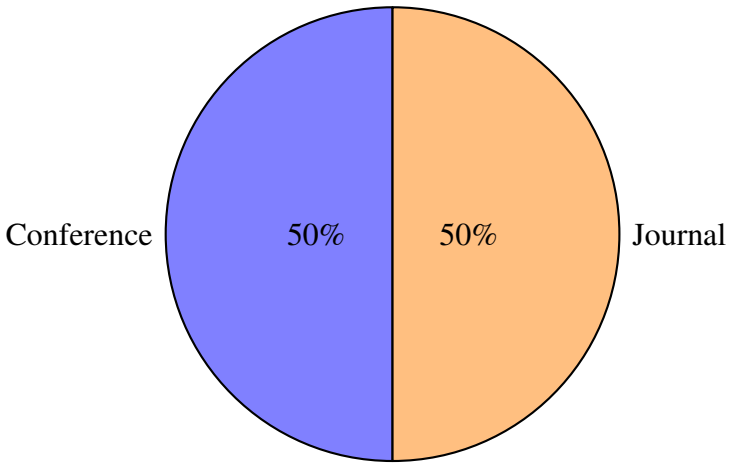

Figure 5.4: Publication types

5.2 Frequency analysis of words in the selected studies

In this section we present our findings after doing a frequency analysis on the introduction and conclusion in the selected studies for this review. Here we can see that the studies have, as the most used words, terminology that is found in the titles of the studies, as a general rule. The results are presented in Table 5.6.

Source	First word	Second word
Palma et al.[3]	REST	Patterns
Palma et al.[8]	RESTful	APIs
Brabra et al.[9]	REST	Cloud
Palma et al.[11]	RESTful	APIs
Belkhir et al.[12]	Practices	Apps
Brabra et al.[13]	REST	Cloud
Palma et al.[14]	Service	Antipatterns
Alshraiedeh et al.[15]	RESTful	Antipaterns

Table 5.6: Most used words in the introduction and conclusion

5.3 The state-of-the-art techniques employed in the detection of antipatterns (RQ1)

The techniques used can be presented each in it's own section per the number of selected studies. Every study has one main approach.

5.3.1 Detection of REST Patterns and Antipatterns: A Heuristics-Based Approach

The technique is named SODA-R (Service Oriented Detection for Antipatterns in REST). The approach is heuristics-based in detection of (anti)patterns in RESTful systems. It's underlying framework is SOFA.

The approach defines detection heuristics with identified static and dynamic properties relevant to each antipattern. A static property is a property that can be obtained before invoking the methods. On the other hand a dynamic property is obtained after making a service call to access a resource. From the heuristics defined previously the detection algorithms are implemented along with the service interfaces used for invoking the REST services. The methods from the interfaces are dynamically invoked. Lastly, the application, based on SOFA, applies the heuristics in the form of detection algorithms on the requests with the result being the detected antipatterns.

5.3.2 Are RESTful APIs Well-Designed? Detection of their Linguistic (Anti)Patterns

The technique is named DOLAR (Detection Of Linguistic Antipatterns in REST). It applies syntactic and semantic analyses for the detection of linguistic antipatterns in RESTful APIs. It's underlying framework is SOFA.

The approach defines algorithmic rules for antipatterns to identify their linguistic aspects. Using the algorithmic rules come the implementation of services' interfaces and the implementation of detection algorithms of linguistic (anti)patterns. Each of the interface methods is mapped to a HTTP method that are later executed. The detection alrorithms for linguistic antipatterns are written by transforming the alorithmic rules defined previously. For each RESTful API there are clients that call the methods in the interfaces,

authentication is done in this step. The SOFA framework applies the rules in form of algorithms and returns the detected antipatterns.

DOLAR looks at URIs and compares every node or resource using WordNet(widely-used lexical database) to find similarities between words or contexts. Furthermore Stanford's CoreNLP checks if the nodes are verbs or nouns to detect some antipatterns.

5.3.3 Detecting Cloud (Anti)Patterns: OCCI Perspective

The technique is a semantic-based approach, relying on an Antipattern Ontology built in OWL 2 (Web Ontology Language) which is used by the detection rules written in SWRL (Semantic Web Rule Language).

Ontology engineering aims at making explicit the knowledge contained within software applications. The paper uses 3 Ontoligies: Antipattern Ontology, REST API Ontology and OCCI Ontology. The Antipattern Ontology's main concept is Ptt:Pattern property followd by Ptt:name, Ptt:description and Ptt:required. The REST API ontology's main concept is the Rest:API property following three other properites such as Rest:AuthorizationProtocol, Rest:Element(HTTP properties) and Rest:Operation. The OCCI Ontology: The heart of the OCCI Ontology is the Occi:Resource which has three sub-concepts Occi:Storage, Occi:Compute and Occi:Network. A Resource can be a virtual machine, a virtual switch, a Storage Link or Network Interface. The reources's classifications are built into the OCCI Core Model.

The SWRL rules consist in two parts which are called the Antecedent and Consequent. An Antecedent defines the conditions which have to be met and the Consequent define the results of the fulfillment of the conditions defined in the Antecedent. The rules can result in a defined predicate, or a SQWRL query, which contains SQL like operation that use the knowledge inferred by SWRL rules.

The detection rules are applied by a developed detection tool automatically. Firstly the ontologies must be instantiated with the information from various APIs and together with the detection rules the knowledge base is created to finally apply the reasoning process to return the detected antipatterns.

5.3.4 Semantic Analysis of RESTful APIs for the Detection of Linguistic Patterns and Antipatterns

The approach is named SARA (Semantic Analysis of REST-ful APIs). It improves upon DOLAR (Detection of Linguistic Antipatterns in REST). SARA is supported by the SOFA framework.

SARA looks at URIs and compares every node or resource using WordNet,a widely-used lexical database. Furthermore Stanford's CoreNLP checks if the nodes are verbs or nouns to detect some antipatterns. To overcome the limits of DOLAR, SARA adds a combinaton of two more steps. The Latent Dirichlet Allocation(LDA) topic modeling technique and a second-order semantic-similarity metric. LDA is a popular technique in the natural-language processing domain. The second-order semantic-similarity metric is based on the distributional similarity between terms to decide their semantic similarities.

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

The approach is named PIRAC. It uses the SOOT framework for static analysis of the byte-code of Android APKs. PIRAC takes as an imput an Android APK, metadata and

their HTTP cleint libraries. It uses the SOOT framework to extract information needed for analysis such as classes and methods. PIRAC filters the application code and constructs models which contain the information necessary to apply the detection heuristics. PIRAC applies rules on APKs to check if the application uses a REST API, this can be a difficult task. To detect the antipatterns listed in the study, PIRAC looks at, for example, the HTTP classes used, ConnectivityManager, which provides information about the network connectivity, JSON or XML respones, timeout related methods, failure/succes related methods.

5.3.6 On semantic detection of cloud API (anti)patterns

This approach is a continuation of the approach in Section 5.3.3 expanded with an updated Antipattern Ontology and further detection algorithms based on SPARQL (Simple Protocol and RDF Query Language) queries. The Antipattern Ontology is built in OWL. The detection rules are built in SWRL.

The proposed antipattern detection algorithm takes as input the 3 Ontologies, the list of SWRL rules to detect REST antipatterns and the list of SPARQL queries to return relevant details related to each REST pattern. As output, it returns the list of patterns with relevant details and recommendations. Another step is applied during the detection stage of the antipatterns based on SPARQL queries which returns details on antipatterns as well as recommendations. The detection tool furthermore checks complicance with OCCI patterns and REST patterns. Extensibility is improved as the new antipatterns defined can be easily integrated within the already established knowledge base.

5.4 UniDoSA: The Unified Specification and Detection of Service Antipatterns

The approach is named UniDoSA (Unified Specification and Detection of Service Antipatterns). It's underlying framework SOFA (Service Oriented Framework for Antipatterns). It supports the specification and detection of service antipatterns. The Detection component in the SOFA framework returns the occurrences of service antipatterns.

UniDoSA is a unified approach in detecting antipatterns in REST, SOAP and SCA. It identifies the antipatterns in the services mentioned above and creates a meta-model. The meta-model is derived from indivdual models form previous works. Using the elements in the meta-model, a DSL (domain specific language) is created to formalize the antipatterns as rule cards. Rule cards are human readable and machine processable high level abstractions. Depending on the antipattern, to abstract them we need static or dynamic properties. The generation of the detection algorithms is template based by parsing the rule cards and feeding the templates with their values. It leverages EMF (Eclipse Modeling Framework). The generative process is fully automated. The detection algorithms generated in Java are applied by SOFA where the concrete detection happens. SOFA framework can automatically compute metrics, perform structural analyses on service interfaces, and apply the rules on these services to decide if a service can be considered as an antipattern. In the case of RESTful services it converts the services into SCA services and then it invokes them.

5.4.1 A URI parsing technique and algorithm for anti-pattern detection in RESTful Web services

The approach is an algorithm based approach.

This study uses techniques such as checking the length of the names of the nodes in URIs, checking the meaning of the nodes, via WordNet, and checks the symbols used in the nodes. It uses 3 algorithms. The principal researcher implemented the algorithms for the detection of antipatterns. The algorithms are written in C# code and are applied. This process results in the detection of antipatterns with a list of the antipatterns returned.

6 The classifications of the state-of-the-art approaches (RQ2)

The relevant approaches can be classified in three groups. In Table 6.7 we can see the classification of the approaches in each selected study. In Figure 6.5 we can see the types of classifications we have identified for each approach and further down bellow we have explained them.

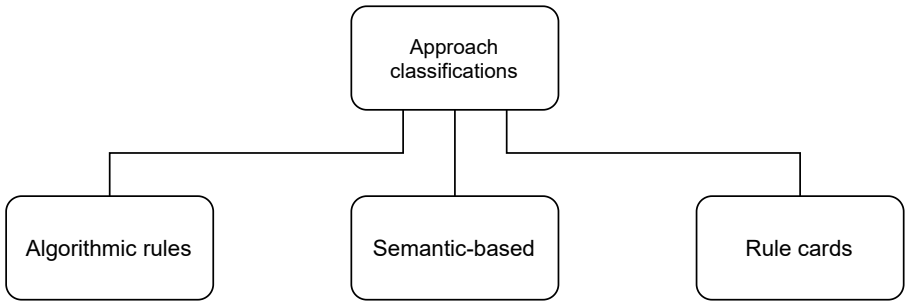

Figure 6.5: Types of approaches

Algorithmic rules are approaches that use predefined algorithms which need to be implemented manually. In the selected studies they are named Detection Heuristics, Algorithmic Rules and Detection Rules. This is the dominant, majority approach, found in most studies.

Semantic-based is an approach which uses predefined rules that interrogate a populated knowledge base which contains information about APIs. The rules are SQL like. This approach is seen in Brabra et al.[9] and Brabra et al.[13].

Rule cards is an approach which a higher level of abstraction. It formalizes the antipatterns as rule cards in a DSL. The detection algorithms are generated automatically. This approach is seen in the last study done by Palma et al.[14]

Source	Classification
Palma et al.[3]	Algorithmic rules
Palma et al.[8]	Algorithmic rules
Brabra et al.[9]	Semantic-based
Palma et al. [11]	Algorithmic rules
Belkhir et al.[12]	Algorithmic rules
Brabra et al.[13]	Semantic-based
Palma et al.[14]	Rule cards
Alshraiedeh et al.[15]	Algorithmic rules

Table 6.7: Classification of the approaches

7 The processes involved in the state-of-the-art approaches (RQ3)

In this section we list the steps followed by the selected studies in detecting RESTful antipatterns. We can see the processes in Table 7.8.

All approaches begin, as their first step, with the definition of the relevant antipatterns.

All approaches have an automatic detection step as all approaches follow one of the three classification of approaches defined in Section 5 which are automatic in nature.

The approach followed in Brabra et al.[13] offers, as Step 4, a list of recommendations.

The approach followed in Palma et al.[14] offers, as Step 2, also an automatic generation of the algorithms stage due to it's generative type of approach, which is unique.

Palma et al.[3] does not have a validation step explicitly or a detection step defined however, the approach, does have validation of results and detection as steps.

Source	Processes
Palma et al.[3]	Step 1: Analysis of Patterns and Antipatterns Step 2: Detection of Patterns and Antipatterns
Palma et al.[8]	Step 1: Analysis of Linguistic (Anti)Patterns Step 2: Implementation of Interfaces and Detection Algorithms Step 3: Detection of Linguistic (Anti)Patterns
Brabra et al.[9]	Step 1: Definition of OCCI REST (Anti)Patterns Step 2: Analysis and Definition of Detection Rules Step 3: Detection of OCCI REST (Anti)Patterns
Palma et al.[11]	Step 1: Analysis of linguistic patterns and Antipatterns Step 2: Implementation of interfaces and detection algorithms Step 3: REST methods invocation Step 4: Detection of linguistic patterns and Antipatterns
Belkhir et al.[12]	Step 1: Cataloguing Android REST Mobile Clients Practices Step 2: Detection of the Practices Step 3: Validating Harissa for the Detection of RESTMobile Clients Practices
Brabra et al.[13]	Step 1: Definition of OCCI/REST (Anti)Patterns Step 2: Analysis and definition of detection rules Step 3: Detections of (Anti)Patterns Step 4: Cloud REST API correction
Palma et al.[14]	Step 1: Specification Step 2: Generation Step 3: Detection Step 4: Validation
Alshraiedeh et al.[15]	Step 1: Extract RESTful URIs Step 2: Implement the detection algorithm Step 3: Detect the anti-patterns Step 4: Validate the results

Table 7.8: Processes

8 The accuracy of the state-of-the-art approaches (RQ4)

We extracted as much relevant data about the approaches, normalized the data and presented it in Table 0.9. The recall represents the ratio between the true-detected anti-patterns and all existing true anti-patterns, whereas precision represents the ratio between the true-detected anti-patterns and all detected anti-patterns [15]. The data we collected is represented by Equations 1, 2 and 3.

$$F - \text{measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{1}$$

$$\text{Precision} = \frac{\text{True detected antipatterns}}{\text{All detected antipatterns}} \tag{2}$$

$$\text{Recall} = \frac{\text{True detected antipatterns}}{\text{All exsisting true antipatterns}} \tag{3}$$

The approaches in [9] and [13] have published validation data for approaches which detect Cloud related antipatterns. We have not included data related to Cloud antipatterns.

The approach in [14] detects antipatterns in REST, SOAP and SCA. We included data in our review as averages combined for the different types of antipatterns.

8.1 Observations

The approach in [11] is an improved form of [8]. Validation 1.2 shows that [11] is a better approach, however, Validation 1.1 does not show any large difference.

The approach in [14] is build on knowledge from [3], [8] and [11]. We can see higher averages in this approach compared to earlier approaches however it is due to the fact that the data is combined and not a direct comparison with earlier works.

The approach in [13] is built on [9]. There is no large difference in accuracy in the updated version. This is to be expected as the approach did improve in areas other than the detection stage.

9 The pros and cons of the state-of-the-art approaches (RQ5)

In this section we will discuss the differences between the approaches and we will present the pros and cons to each approach in Table 0.10.

9.1 Types of antipatterns

We have defined different types of antipatterns which the approaches detect.

HTTP properties antipatterns is a type of antipatterns which focus on HTTP headers and methods. For instance, the HTTP request headers Accept and Cache-Control and their corresponding values. Similarly Similarly, the HTTP response headers Location and Status and their corresponding values[3]. The use of methods such as GET and PUT must also be appropriate.

URI antipatterns is a type of antipatterns which focus on the linguistic aspects of URIs used in RESTful apis. RESTful APIs are written for consumers. The name and structure of URIs should convey meaning to those consumers. A linguistic aspect for the

detection of the Contextless Resource Names antipattern is, for example, to check if a URI nodes belong to the same semantic context [8].

OCCI REST antipatterns is a type of antipatterns which live in Open Cloud Computing Interface. Open Cloud Computing Interface (OCCI) is the open cloud standard addressing heterogeneity, interoperability, integration and portability in Cloud Computing [13]. The OCCI specification defines general-purpose models, network protocols and other types of protocols for management tasks of cloud resources. OCCI antipatterns represent the good and bad practices of the presented guidelines in the OCCI RESTful Protocol.

Android REST design antiatterns is a type of practices seen in how Android apps use/consume REST APIs. The practices are related to the development of REST mobile clients and interactions between clients [12].

9.2 Results

Up-to-date, there is no study that compares all approaches. No subjective conclusions were drawn by the researcher.

The best approach which is applicable to both HTTP properties antipatterns, URI design antipatterns and other technologies is [14].

The worst approach, due to it's limited scope, is [15]. It detects only two antipatterns using a technique inferior to [11].

[13] is a specialized approach which detects both OCCI REST antipatterns, URI antipatterns and HTTP properties antipatterns. It uses an original, extensible technique.

[12] is a specialized approach which introduces good and bad practices related to Android mobile apps which consume REST APIs.

[3], [8], [11], [9] are approaches which are out-of-date.

10 The subjects and objects in the relevant literature (RQ6)

In this section we list the subjects and objects of the selected approaches. In our case, the subjects are the antipatterns and the objects are the APIs used for validation. The results can be found in Table 0.11.

We can see that, in the approaches, every study defines the antipatterns using different terminology.

The study which detects the most antipatterns is [13], with 18 in total. This is an increase of 6, the previous approach [9], which is not up-to-date can detect 12 antipatterns. The number of APIs on which the validation was done is 5, one less than 6, the number of APIs tested in the previous approach. An important fact is that the focus of this approach is OCCI, however the approach is relevant to general purpose RESTful APIs.

The approach which we previously recommended as the best one [14], is detecting 8 antipatterns. If we compare it with [13] and we exclude the OCCI antipatterns, both approaches detect the same number of antipatterns.

The worst approach is [15]. It detects two antipatterns, less than the approach in [11], which detects 6 antipatterns. The number of APIs tested in [15] is 16 and the number of APIs tested in [11] is 18 respectively.

The approach in [12] detects 6 antipatterns. The approach does it's validation on 9173 unnamed Android apps from F-droid. This is the only approach of it's kind, dedicated to the development of mobile applications which consume RESTful APIs.

The approaches in [3], [8] and [9] are outdated approaches. The antipatterns defined and detected in the approaches mentioned above are out-of-date.

11 Discussion and results

This part of the work discusses the research methodology used in the review and the challenges in applying it. Further improvements that can be made are discussed. Furthermore, social and ethical considerations are made. Finally, future work that can be done in detection of antipatterns in RESTful APIs is discussed.

RESTful APIs are simpler and more efficient than traditional SOAP APIs. Today, many companies leverage REST. It is important to detect antipatterns introduced in RESTful APIs as antipatterns are bad design solutions which are opposed to patterns, good solutions to problems.

A systematic literature review is a nonlinear process in which initial decisions are overwritten during the execution of the project. The final research method is different from the initial version after gaining knowledge in the field.

The project began with four research questions but two more were added resulting in six research questions. The reason for the expansion of the scope is the low number of available primary studies.

The search strategy took a considerable amount of time, more than anticipated. The reason for this was the systematic nature of the project. To collect quality data, a thorough search in the digital databases is needed, with as much data collected at every step.

The major challenge of the scrutiny stage, in which we apply our subsequent multiple filtering stages, was applying inclusion-exclusion criteria and quality assessment. A single study [10], proved to be difficult to assess. Ultimately , the study was rejected due to a low quality assessment score.

The data synthesis, the part in which the data needed to answer the research questions is collected went fast and was presented in the report. The data for each research question was saved in an Excel file.

One improvement to the review that can be made is further discussion of the data collected.

In this review, a lot of ethical considerations are nonexistent as this is not primary research. There are, however, some things to me mentioned. Quoting out of context is an informal fallacy in which a passage is removed from its surrounding matter in such a way as to distort its intended meaning [17]. Therefore, it is important to preserve the context of the cited sentences. The data we present in this review is obtained exclusively from high quality databases, the primary studies come only from publishers that offer peer reviewed papers. The data presented here was done so in an objective and impartial way, in no way promoting one source over the other.

This review is relevant to any researcher or software engineer who wishes to see the current state of detection of RESTful antipatterns.

FUTURE WORK TODO

12 Conclusion

There are several studies that summarized detection methods in object-oriented systems or SOAP-based Web services.However, to date, there is no consolidated study that reviews the methods and techniques applied in detecting antipatterns in RESTful APIs or RESTful Web services. The review aims to provide a view of the field of detection techniques of RESTful antipatterns and examine them.

The selected number of studies is eight. The first paper was published in 2013 with four papers published in a conference and for papers in a journal, an even ratio.

Regarding RQ1, each primary study yielded one technique in detecting REST antipatterns. The details and description of the techniques are listed in a digestible way.

Regarding RQ2, the approaches were classified with three originally defined classifications. The approaches mainly use Algorithmic rules as their type, however [9] and [13] uniquely use Semantic-based approaches. Lastly, [14] uses Rule cards as the approach type. This is the only study which uses a DSL and a high level approach which generates detection algorithms. The study was found to be the best approach. The conclusion drawn is that the main focus currently in the research field is creating new approaches.

Regarding RQ3, the processes used in the approaches were presented in a table. The main steps are analysis and definition of the antipatterns, implementation of the detection method based on class, detection and lastly, validation of the results.

Regarding RQ4, the accuracy was collected with three metrics in focus: Precision, Recall and F-measure. The results were presented in a table. The average low value for all three metrics is 80% while the high value is at least 90%. There was no identified trend in accuracy. In conclusion, the approaches are useful but not 100% accurate, manual validation is necessary.

Regarding RQ5, to discuss the pros and cons of the approaches four types of antipatterns were classified. We took into consideration what type of antipatterns the approaches can detect, the relevant field of applicability and whether or not the approaches are up-to-date. The results are presented in a table and discussed.

Regarding RQ6, the subjects and objects were presented in a table. The results shows the detection capability of the methods. Further analysis can be made on the antipatterns and the APIs.

References

[1] R. T. Fielding, "Architectural styles and the design of network-based software architectures. 2000," *University of California, Irvine*, p. 162, 2000.

[2] G. Block, P. Cibraro, P. Felix, H. Dierking, and D. Miller, *Designing Evolvable Web APIs with ASP. NET: Harnessing the Power of the Web*. " O'Reilly Media, Inc.", 2014.

[3] F. Palma, J. Dubois, N. Moha, and Y.-G. Guéhéneuc, "Detection of rest patterns and antipatterns: a heuristics-based approach," in *International Conference on Service-Oriented Computing*. Springer, 2014, pp. 230–244.

[4] S. Keele *et al.*, "Guidelines for performing systematic literature reviews in software engineering," Citeseer, Tech. Rep., 2007.

[5] Z. Munn, M. D. Peters, C. Stern, C. Tufanaru, A. McArthur, and E. Aromataris, "Systematic review or scoping review? guidance for authors when choosing between a systematic or scoping review approach," *BMC medical research methodology*, vol. 18, no. 1, pp. 1–7, 2018.

[6] H. Arksey and L. O'Malley, "Scoping studies: towards a methodological framework," *International journal of social research methodology*, vol. 8, no. 1, pp. 19–32, 2005.

[7] F. Alshraiedeh and N. Katuk, "Soap and restful web service anti-patterns: a scoping review," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 8, no. 5, pp. 1831–1840, 2019.

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

[9] 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.

[10] C. Rodríguez, M. Baez, 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 *International conference on web engineering*. Springer, 2016, pp. 21–39.

[11] F. Palma, J. Gonzalez-Huerta, M. Founi, N. Moha, G. Tremblay, and Y.-G. Guéhéneuc, "Semantic analysis of restful apis for the detection of linguistic patterns and antipatterns," *International Journal of Cooperative Information Systems*, vol. 26, no. 02, p. 1742001, 2017.

[12] A. Belkhir, M. Abdellatif, R. Tighilt, N. Moha, Y.-G. Guéhéneuc, and É. Beaudry, "An observational study on the state of rest api uses in android mobile applications," in *2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2019, pp. 66–75.

[13] H. Brabra, A. Mtibaa, F. Petrillo, P. Merle, L. Sliman, N. Moha, W. Gaaloul, Y.-G. Guéhéneuc, B. Benatallah, and F. Gargouri, "On semantic detection of cloud api (anti) patterns," *Information and Software Technology*, vol. 107, pp. 65–82, 2019.

[14] F. Palma, N. Moha, and Y.-G. Guéhéneuc, "Unidosa: The unified specification and detection of service antipatterns," *IEEE Transactions on Software Engineering*, vol. 45, no. 10, pp. 1024–1053, 2018.

[15] F. S. Alshraiedeh and N. Katuk, "A uri parsing technique and algorithm for anti-pattern detection in restful web services," *International Journal of Web Information Systems*, 2020.

[16] X. Zhou, Y. Jin, H. Zhang, S. Li, and X. Huang, "A map of threats to validity of systematic literature reviews in software engineering," in *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2016, pp. 153–160.

[17] S. M. Engel, "With good reason an introduction to informal fallacies," 1982.

Source	Validation (Average)	Details
Palma et al.[3]	Precision = 89.42% Recall = 94% F-measure = 91.65%	All APIs tested
Palma et al.[8]	**Validation 1** Precision= 81.4% Recall = 78% F-measure = 79.66% **Validation 2** Precision = 79.7%	Validation 1 - All URIs in Dropbox Validation 2 - Partial set of URIs in Facebook, Dropbox, Twitter and YouTube No Recall or F-measure calculated due to partial validation
Brabra et al.[9]	Precision = 98 % Recall = 100 % F1-measure = 99 %	OpenStack, Microsoft and Rackspace
Palma et al.[11]	**Validation 1.1** DOLAR Precision = 81.3% Recall = 78.0% VS SARA Precision= 80.9% Recall = 81.0% **Validation 1.2** DOLAR Precision = 79.7% VS SARA Precision = 87.3% **Validation 2** Precision = 73.2%	Validation 1.1 and 1.2 - Same dataset in Palma et al.[8] Validation 2 - All antipatterns, URIs in all APIs
Belkhir et al.[12]	Precision = 93.70% Recall = 87.66%	Validation 1448 Android apps from F-Droid
Brabra et al.[13]	**Validation 1** Precision = 100% Recall = 95.1% F-measure = 97,4% **Validation 2** Precision = 100% Recall = 91.2% F-measure = 95.3%	Validation 1 - OOi Validation 2 - Rackspace REST antipatterns
Palma et al.[14]	Precision = 89.78% Recall = 96.67% F-measure = 91.3%	Validation averages for REST, SOAP and SCA antipatterns combined
Alshraiedeh et al.[15]	**Ambiguous names** Precision = 87.56% Recall = 81.19% F-measure = 84.18% **Amorphous URIs** Precision = 88.17% Recall = 83.42% F-measure = 85.68%	Validation results per antipattern

Table 0.9: Validation results of the approaches

Source	Pros	Cons
Palma et al.[3]	Initial approach detecting HTTP properties antipatterns	No URI design antipatterns detection Outdated approach
Palma et al.[8]	Initial approach detecting URI design antipatterns	No HTTP properties antipatterns detection Outdated approach
Brabra et al.[9]	Initial approach of OCCI REST design antipatterns	Relevant just to OCCI Cloud Outdated approach
Palma et al.[11]	Up-to-date approach in detecting URI design antipatterns.	No HTTP properties antipatterns detection
Belkhir et al.[12]	Initial definition of Android REST design patterns.	Relevant just to Android applications Does not detect caching mechanisms
Brabra et al.[13]	Detection of URI design antipatterns Detection of HTTP properties antipatterns Detection of OCCI REST antipatterns Up-to-date approach in detecting OCCI REST antipatterns Extensible	Not best in class
Palma et al. [14]	Detection of URI design and HTTP properties REST antipatterns Detection of SOAP and SCA and REST antipatterns though the unified approach Up-to-date approach in detecting service antipatterns Extensible	None
Alshraiedeh et al.[15]	None	Only two antipatterns are detected SARA is more sophisticated

Table 0.10: Pros and cons of the approaches

Table 0.11: Subjects and objects of the approaches

Source	Subjects	Objects
Palma et al.[3]	**patterns VS antipatterns:** - **VS** Breaking Self-descriptiveness Entity Linking **VS** Forgetting HypermediaResponse Caching **VS** Ignoring Caching Content Negotiation **VS** Ignoring MIME Types - **VS** Ignoring Status Code - **VS** Misusing Cookies - **VS** Tunnelling Through GET - **VS** Tunnelling Through POST End-point Redirection **VS** - Entity Endpoint **VS** -	Alchemy BestBuy Bitly CharlieHarvey DropBox Facebook Musicgraph Ohloh TeamViewer Twitter YouTube Zappos
Palma et al.[8]	**patterns VS antipatterns:** Contextualised **VS** Contextless Resource Names Hierarchical **VS** Non-hierarchical Nodes Tidy **VS** Amorphous URIs Verbless **VS** CRUDy URIs Singularised **VS** Pluralised Nodes	Alchemy BestBuy Bitly CharlieHarvey Dropbox Externalip Facebook Instagram Musicgraph Ohloh

Table 0.11: Continued

Brabra et al.[9]	**patterns VS antipatterns:** **Management related (anti)patterns** Query Interface Support **VS** Missing Query Interface Compliant Create **VS** Non-Compliant Create Compliant Update **VS** Non-Compliant Update Compliant Delete **VS** Non-Compliant Delete Compliant Retrieve **VS** Non-Compliant Retrieve Compliant Trigger Action **VS** Non-Compliant Trigger Action **Cloud structure (anti)patterns** Compliant Link between Resources **VS** Non-Compliant Link between Resources Compliant Association of resource(s) with Mixin **VS** Non-compliant Association of resource(s) with Mixin Compliant Dissociation of resource(s) From Mixin **VS** Non-compliant dissociation of resource(s) From Mixin **REST related (anti)patterns** Compliant URL **VS** Non-Compliant URL Compliant Request Header **VS** Non-Compliant Request Header Compliant Response Header **VS** Non- Compliant Response Header	OpenStack COAPS OpenNebula Amazon S3 Microsoft Azure Rackspace

Table 0.11: Continued

Palma et al.[11]	**patterns VS antipatterns:** Contextualized **VS** Contextless resource names Hierarchical **VS** Nonhierarchical nodes Tidy **VS** Amorphous URIs Verbless **VS** CRUDy URIs Singularized **VS** pluralized nodes Pertinent **VS** Nonpertinent documentation	Alchemy BestBuy Bitly CharlieHarvey Dropbox Externalip Facebook GoogleBook Instagram LinkedIn MusicGraph Ohloh StackExchange
Belkhir et al.[12]	**patterns VS antipatterns:** Third-party HTTP client **VS** HttpURL-Connection Caching **VS** Non caching Network connectivity aware **VS** Unaware REST service invocation JSON **VS** XML Timeouts **VS** Perpetual requests Specification **VS** Non specification of a behaviour for failed requests	9173 Android apps fromF-Droid repository

Table 0.11: Continued

Brabra et al.[13]	**patterns VS antipatterns:** **REST antipatterns** Tidy URIs **VS** Amorphous URIs Verbless URIs **VS** CRUDy URIs Singularized nodes **VS** pluralized nodes Correct use of POST, GET, PUT, DELETE, HEAD **VS** Tunneling every things through GET and POST Supporting status code **VS** ignoring status code Supporting caching **VS** ignoring caching Supporting MIME Types VS Ignoring MIME Types Supporting hypermedia VS forgetting hypermedia **OCCI antipatterns** Compliant URL **VS** non-compliant URL Compliant Request Header **VS** Non-Compliant Request Header Compliant Response Header **VS** Non- Compliant Response Header Compliant Link between Resources **VS** Non-Compliant Link between Resources Compliant Association of Resource(s) with Mixin **VS** Non-compliant Association of Resource(s) with Mixin Compliant Dissociation of Resource(s) From Mixin **VS** Non-compliant dissociation of resource(s) From Mixin Query interface support **VS** Missing Query interface Compliant create **VS** Non-compliant create Compliant update **VS** Non-compliant update Compliant delete **VS** Non-compliant delete Compliant retrieve **VS** Non-compliant retrieve Compliant Trigger Action **VS** Non-Compliant Trigger Action	OOi RESTful API COAPS RESTful API OpenNebula OCCI RESTful API Amazon S3 RESTful API Rackspace RESTful API

Table 0.11: Continued

Palma et al.[14]	**antipatterns:** Ambiguous Name Bloated Service Nobody Home Deprecated Resources CRUDy Interface CRUDy URI Forgetting Hypermedia Ignoring MIME Types	Alchemy BestBuy Bitly CharlieHarvey DropBox Externalip Facebook GoogleBook Instagram Linkedin Musicgraph Ohloh StackExchange TeamViewer Twitter Walmart YouTube Zappos

Table 0.11: Continued

Alshraiedeh et al.[15]	**antipatterns:** Ambiguous names Amorphous URIs	BestBuy Bitly CharlieHarvey Dropbox Externalip Facebook GoogleBook Instagram LinkedIn Ohloh StackExchange TeamViewer Twitter Walmart YouTube Zappos