A Lexical and Semantical Analysis on REST Cloud Computing APIs

Fabio Petrillo[1(✉)], Philippe Merle[2], Francis Palma[3], Naouel Moha[4], and Yann-Gaël Guéhéneuc[1]

[1] Concordia University, Montréal, Canada
fabio@petrillo.com
[2] Inria Lille - Nord Europe, Villeneuve d'Ascq, France
[3] Linnaeus University, Växjö, Sweden
[4] Université du Québec à Montréal, Montréal, Canada

Abstract. Cloud computing is a popular Internet-based computing paradigm that provides on-demand computational services and resources, generally offered by Cloud providers' REpresentational State Transfer (REST) APIs. Developers use REST APIs by invoking these APIs by their names and, thus, the lexicons used in the APIs are important to ease the developers' comprehension. In this paper, we study the lexicons and the linguistic (anti)patterns from 16 providers of REST Cloud Computing APIs. We observe that, although the 16 REST APIs describe the same domain (Cloud computing), contrary to what one might expect, their lexicons do not share a large number of common terms and 90% of the terms (3,561/3,947) are just used by one provider. Thus, the APIs are lexically heterogeneous and there is not a consensus on which terms to use in Cloud computing. Further, we observe that the majority of the URIs, 54%, follow the *Contextualised Resource Names* pattern, which is considered a good practice in REST API design. However, a majority of the URIs, 62.82%, suffer from the *Non-pertinent Documentation* antipattern. Thus, we present three main contributions: (1) a tooled approach, called CloudLex, for extracting and analysing REST Cloud computing lexicons; (2) our analysis of the terms used in 16 REST APIs in 59,677 term occurrences; (3) our analysis of the linguistic (anti)patterns in more than 23,000 URIs of the 142 services of the 16 Cloud providers. We also show that CloudLex has an average precision of 84.82%, recall of 63.57%, and F1-measure of 71.03% on one complete API, `Docker Engine`, which confirms the accuracy of our semantic analyses for the detection of linguistic (anti)patterns.

1 Introduction

Cloud computing has transformed the information-technology industry [2] by hosting applications and providing resources (*e.g.,* CPU and storage) as services on-demand over the Internet [23]. Cloud providers, such as Google Cloud Platform (a commercial public Cloud) and OpenStack (an open-source stack for

D. Ferguson et al. (Eds.): CLOSER 2017, CCIS 864, pp. 308–332, 2018.
https://doi.org/10.1007/978-3-319-94959-8_16

building public/private Clouds), usually offer these services in the form of REST (REpresentational State Transfer) [7] APIs, the *de facto* standard adopted by many software organisations for publishing their services.

Most of Cloud providers, such as Google Cloud Platform or OpenStack, propose their own proprietary APIs. Conversely, open and standard Cloud APIs have also been proposed, such as the Open Cloud Computing Interface (OCCI) [14], which is a vendor-neutral cloud standard.

This observed variety of cloud APIs may decrease developers' comprehension, especially within such a complex and technical context as Cloud computing. Moreover, well-designed and well-named REST APIs may attract client developers to use them more than poorly designed and named ones, particularly in the current open market, where Web services are competing against one another [13]. Indeed, client developers must understand the providers' APIs while designing and developing applications that use these APIs.

Therefore, the understandability of REST APIs are two major quality characteristics, which are reachable when best practices for REST APIs design [13] and naming are followed. Because developers' comprehension is essential for Cloud computing adoption [21], we claim that this comprehension requires quality lexicons in the APIs and the URIs used to access these APIs.

Consequently, we study 16 different and well-known REST APIs to investigate and organise their lexicons. We also study the linguistic (anti)patterns on REST APIs of 16 cloud providers, extending and complementing our previous work [17]. Linguistic antipatterns represent poor solutions to recurring naming problems, which may hinder (1) the consumption of APIs by client developers and (2) the maintenance/evolution of APIs by the API developers. In contrast, linguistic patterns are good solutions to recurring naming problems—they facilitate the consumption and maintenance of APIs [15].

For the semantic analysis of Cloud REST APIs, we apply the SARA approach [15]. We rely on WordNet[1] and Stanford CoreNLP[2] as English dictionaries with a combination of Latent Dirichlet Allocation (LDA) topic modeling technique [3] and second-order semantic-similarity metric [11,12]. 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.

The remainder of the paper is organised as follows. Section 2 presents the main concepts about natural-language processing and the second-order semantic-similarity metric. Section 3 presents the key concepts of CloudLex, our approach to analyse lexically and semantically Cloud computing REST APIs. Sections 4 and 5 present our results, answer the research questions, and discuss threats to validity. Section 6 presents some related work. Finally, Sect. 7 concludes the paper with future work.

This paper is an extension of our previous conference paper [17]. We extended our paper with (1) a dataset sixteen Cloud computing providers (thirteen more

[1] wordnet.princeton.edu.
[2] nlp.stanford.edu/software/corenlp.shtml.

that the previous paper; (2) a larger analysis of the terms used in 16 REST APIs in 59,677 term occurrences; (3) an analysis of the linguistic (anti)patterns in more than 23,000 URIs of the 142 services of the 16 Cloud providers. Moreover, we addressed new quality dimensions and three new research questions. Finally, our analysis confirm and corroborate the previous results presented in our conference paper.

2 Background

The second-order semantic-similarity metric [11,12] and the Latent Dirichlet Allocation (LDA) algorithm [3] are applied in natural-language processing, *e.g.,* [15] for various purposes. We now present them briefly because we use them to analyse the linguistic quality of Cloud APIs.

2.1 Second Order Semantic Similarity

The second-order semantic-similarity metric helps finding *distributionally* the most similar terms among a set of terms and computes similarity scores for the terms based on the second-order terms vectors [11,12]. Two terms are distributionally similar if they have multiple co-occurring terms in the same syntactic relations.

Table 1. Window set up for the calculation of window-position triples (WPT).

−3	−2	−1	-	+1	2	3
social	media	and	newsportal	are	more	popular
online	social	and	newsportal	are	beyond	those

Table 1 shows an example of the analysis of the term *newsportal*. If we consider *window_size* as ±3, it gives us two occurrences of *newsportal*. When we consider the position, the term *newsportal* has eight unique features (without stop words, *e.g.,* "are", "beyond", "and") as shown in Table 2 in the WPT column. If we do not consider the window position, then the term *newsportal* has seven different features without stop words, as shown in the Co-occurrences column.

By moving the window over the corpus, we can get the terms vectors for each term. Using the terms vectors and normalising the counts [12], we can compute the distributionally similar terms used in similar contexts. For example, if *newsportal* co-occurs with three terms {*social, online, media*, and both the terms *print*} and *media* co-occur with those three terms, then the terms *newsportal*, *print*, and *media* are said to be distributionally similar.

We can obtain the list of the n most similar terms for a given input by using this technique. We can use this list as the *second order* term vector for a given term, which contains the terms that occur together in similar contexts. We can apply a similar technique to compare the *second order* terms vectors and

Table 2. Examples of the window-position triples and co-occurrences for the example in Table 1.

WPT		Co-occurrence	
$<newsportal, -3, social>$	1	$<newsportal, social>$	2
$<newsportal, -3, online>$	1	$<newsportal, online>$	1
$<newsportal, -2, media>$	1	$<newsportal, media>$	1
$<newsportal, -2, social>$	1	$<newsportal, are>$	1
$<newsportal, +1, are>$	2	$<newsportal, more>$	2
$<newsportal, +2, more>$	1	$<newsportal, popular>$	1
$<newsportal, +3, popular>$	1	$<newsportal, those>$	1
$<newsportal, +3, those>$	1		

compute the *second-order semantic-similarity* metric [11,12]. Compared to WordNet [4,12], this metric allows going beyond the *is-a* relationships between nouns and verbs because WordNet only contains synonyms (warm–hot), meronyms (car–wheel), and antonyms (hot–cold).

2.2 Latent Dirichlet Allocation

In natural-language processing, topic models describe documents as aggregations of latent topics. Latent topics are clustered set of terms [3]. The LDA algorithm extracts topic models from a corpus of terms built from documents. These topic models are low-dimensional representations of the contents of the documents. The cardinality of each topic model, its dimensionality or size, k is set beforehand as an input to building topic models using LDA. LDA allows binding multiple topic models to a single document, which gives flexibility in deciding if a document or part thereof belongs to a topic.

However, LDA is impeded by the sizes of the vocabulary, the numbers of terms, inherent to the majority of documents corpora [3]. Consequently, a new document may contain new and unobserved terms to be classified that were not initially present in the training corpus. This problem along with the *bag-of-words* assumptions motivate us to define an approach combining LDA and second-order semantic-similarity metric. The former allows obtaining a low-dimensional representation of a corpus and the later measures the semantic similarity between the terms in the corpus.

2.3 Linguistic Patterns and Antipatterns

We now describe four linguistic (anti)patterns: *Contextualised vs. Contextless Resource Names* [8] and *Pertinent vs. Non-pertinent Documentation* [15], which we will study in the rest of this paper.

Contextualised vs. Contextless Resource Names. URIs of Cloud resources defined by Cloud providers should be *contextual*, *i.e.,* it is a good practice for URIs to be in semantically-related or similar contexts. The *Contextless Resource Names* linguistic antipattern is introduced when API developers do not design URI nodes within the same semantic context. An example of *Contextless Resource Names* linguistic antipattern is `www.provider.com/server/research/stock?id=01` where the terms "server", "research", and "stock" are not from semantically-related contexts. In contrast, `www.provider.com/server/memory?size=1024` is an example of *Contextual Resource Names* pattern because "server" and "memory" belong to semantically-related contexts (assuming that the statement "server has memory" is true). The consequences of the *Contextless Resource Names* antipattern include not providing a clear context for a Cloud resource and misleading the cloud API clients, which reduce API understandability [8].

Pertinent vs. Non-pertinent Documentation. The *Non-pertinent Documentation* linguistic antipattern occurs when the documentation of a Cloud resource URI is not consistent with its set of nodes. Therefore, this antipattern involves both the URI and its documentation. Contrary to this antipattern, a well-documented URI describes its goals and functions using relevant semantic terms. An example of a *Non-pertinent Documentation* is `/v2/tenant_id/flavors/flavor_id/os-extra_specs/key_id` – Gets the value of the specified key from OpenStack, in which the URI and its documentation have no semantic similarity. In contrast, from the same Cloud provider, `/v2/software_deployments/` – Lists all available software deployments. is an example of *Pertinent Documentation* pattern as this URI–documentation pair shows a high semantic similarity. As a consequence of the *Non-pertinent Documentation* linguistic antipattern, Cloud API consumers may make incorrect assumptions on the URIs, which can hinder their comprehension. In addition, for Cloud API providers, this may hinder understandability during the maintenance and evolution [15].

3 Approach

We now present CloudLex, our approach to building the lexicon of Cloud computing REST APIs. First, we introduce a conceptual model of Cloud computing REST APIs. Second, we describe our approach to extract and analyse lexicons from Cloud APIs. Finally, we describe our semantic analysis of Cloud APIs.

3.1 Conceptual Model for Cloud Computing REST APIs

Cloud computing is the root concept of this model composed of the key concepts of **Provider**, **Service**, **Resource**, and **Action**, their main attributes and aggregation relationships. Figure 1 sketches our conceptual model. In our conceptual model, we abstract Cloud computing actors (companies, implementations,

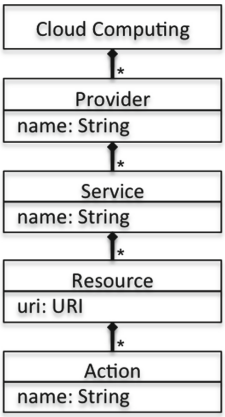

Fig. 1. Conceptual model of cloud computing REST APIs [17].

and standards) under the single concept of **Provider**. Each provider supplies a set of REST APIs. For example, in Google Cloud Platform, each API is in fact a commercial Google *product*, such as `compute` and `sql`.

Independently of the name used by Cloud providers (product, API, extension), each provider's REST API exposes conceptually useful services, *e.g.,* managing virtual machines, networks, databases, or applications, orchestrating their deployment, controlling their access, etc. The number and contents of these services are extremely heterogeneous for each provider: hundreds of services in Google Cloud Platform, more than one hundred in OpenStack, and five in OCCI. In our conceptual model, we abstract this diversity of functional services under the single concept of **Service**.

Each provider's service manages a set of computing resources implemented as REST resources. A service is characterised by a unique resource identifier, *e.g.,* URI, URL, etc., and usually a documentation to describe the service. For example, virtual machines are accessible through the URI /project/zones/zone_id/instances/instance_id in the `compute` service of Google Cloud Platform and the URI /tenant_id/servers/server_id in the `os-compute-2` service of OpenStack. Our conceptual model abstracts this diversity of computing resources under the single concept of **Resource**.

Each resource supports common CRUD operations (Create, Retrieve, Update, and Delete) and some specific business behaviours, like starting and stopping a virtual machine, attaching a disk to a virtual machine, etc. Our conceptual model abstracts this diversity of operations and behaviours under the single concept of **Action**.

To instantiate this conceptual model, we designed a tooled approach for identifying automatically **Service**, **Resource**, and **Action** from Cloud computing REST APIs of any **Provider** and then extracting and analysing the lexicons of these APIs.

3.2 CloudLex

The CloudLex approach divides in four steps:

Step 1. Collecting Documentation. The first step consists in collecting manually the documentation of a provider's Cloud computing REST API.

Step 2. Parsing Documentation. The second step parses all the provider's documentation to identify the **Service**, **Resource**, and **Action** of the conceptual model automatically to create the `Cloud Dataset` of our conceptual model.

Step 3. Extracting Lexicon. The third step extracts the lexicon of each provider from its associated `Cloud Dataset`. The lexicon of each provider contains the `name`s of all the services, the terms extracted from the path of the URIs of all provided resources, and the `name`s of all the actions defined on the resources

For example, the URI /project/zones/zone_id/instances/instance_id contains five segments: {`project`}, `zones`, {`zone_id`}, `instances`, and {`instance_id`}. We keep in the lexicon all segments not enclosed by braces, *e.g.,* `zones` and `instances`. Other segments enclosed by braces identify specific resources, usually as identifiers, such as {`project`} and all {`instance_id`}.

Step 4. Analysing Lexicon. The fourth step analyses automatically the lexicons. We use various analyses to count occurrences of each term in the lexicons, identify nouns and verbs, singular and plural terms, and lower/upper/camel cased terms.

The lexicons are encoded as CSV files (Comma-Separated Values). This implementation choice fosters the *reusability* of the lexicons by researchers and practitioners. Most of the CloudLex parsers, extractors, and analyses are implemented in Python, a dynamic scripting language providing simple libraries to get and parse HTML pages/Swagger files, and read/write CSV files. The implementation of the CloudLex approach is freely available at https://github.com/Spirals-Team/CloudLexicon.

3.3 Semantic Analysis of Cloud APIs

The semantic analysis of Cloud APIs requires four automatic steps shown in Fig. 2. The first step involves the collection of API documentation and performing a pre-processing phase, for example to remove stop words. The second step processes URI nodes to their base form (*a.k.a.*, lemmatisation) using the Stanford Core NLP. The third step involves the extraction of topic models using LDA. In the final step, we compute the second-order semantic-similarity metric between the obtained topic models and the nodes in a URI. We illustrate with a running example the semantic analysis, showing the detection of the *Contextless Resource Names* antipattern and *Contextual Resource Names* pattern.

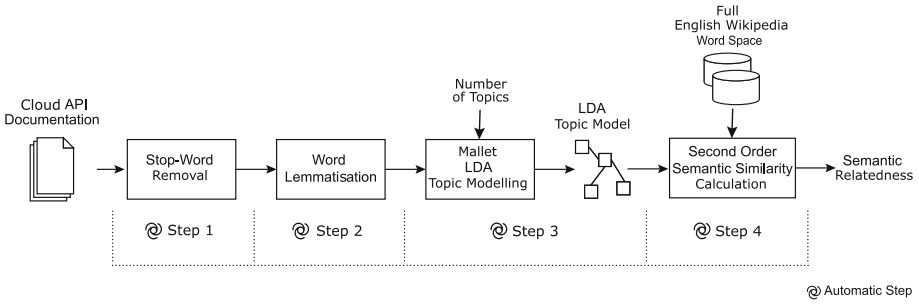

Fig. 2. The Semantic analysis method applied to cloud APIs from [15].

Semantic Analysis. To infer the contextual relationships between nodes in URIs, we rely on Mallet LDA topic modeling[3]. LDA forms a model for a given document, which represents a URI and its documentation. Our proposed approach applies Mallet topic modeling to build topic models by accepting resource identifiers and descriptions of Cloud resources as input.

Table 4 shows an excerpt of the LDA topic model created during our analysis of the Cloud API provider `Docker Engine`. The complete topic model consists of ten topic clusters and we consider the ten most relevant terms in each topic, *i.e.,* the top ten terms. We can use this set of topics to measure similarity between resource identifiers, if two URI nodes are semantically related. URI nodes are semantically related if they are from the same topic [19] (Table 3).

Table 3. List of extracted topics in `Docker Engine`.

List of topics
auth
build
commit
containers
events
exec
images
info
ping
version

To compute the semantic similarity between terms, we use the second-order semantic-similarity metric. We rely on the distributional second-order similarity

[3] http://mallet.cs.umass.edu.

Table 4. Top ten terms for `Docker Engine` topic model with k=10.

Topic0	Topic1	Topic2	Topic3	Topic4	Topic5	Topic6	Topic7	Topic8	Topic9
copy	resize	image	amp	container	create	search	ping	container	json
unpause	start	version	restart	info	attach	height	history	top	exec
commit	tag	event	wait	push	log	log	change	change	load
build	build	width	width	width	stop	stop	auth	auth	kill
		pause	pause	pause	pause	pause	pause	pause	export

because the nodes in the URI can be slightly different from the terms used for their description (structure and form). A pair of terms are distributionally similar if they have common co-occurring terms, *i.e.,* terms that appear frequently with the same set of terms as neighbors. As described in Sect. 2.1, the computation of this similarity is based on the corpus of terms extracted from the URI dataset, which we analyse to find the terms that occur together within a context of $\pm window\_size$ terms. We then process the resulting terms matrix to build terms vectors that represent the distribution of a term in the corpus and show the terms sharing a maximum number of co-occurrences. We use these to compare two terms by analysing the extent to which these two terms have similar second-order terms vectors [12]. We use DISCO [11] to compute such distributional similarity between terms.

Identifying Antipatterns. To compare the context of every pair of nodes in a URI, we measure the second-order semantic similarity between each node with the ten top terms of each topic. Based on the similarity value, we determine to which topics a node belongs. We consider that a node belongs to a topic if the average second-order semantic similarity value is greater than a predefined *threshold*, *i.e.,* 0.3, for any terms in each topic. We simulate over several threshold values and choose the minimum with reasonable results based on our simulation results and comprehension. And, finally, we choose the same threshold based on the work in [15] where the authors reasoned their detection accuracy over this threshold value.

If, for a given nodes pair of a URI, the intersection of topic sets to which each node belongs is empty (*i.e.,* there is no common topic for that pair of nodes in the URI), then we report the URI as a *Contextless Resource Names* antipattern. Otherwise, if each pair of nodes in the URI share at least one common topic, then we report the URI as a *Contextual Resource Names* pattern.

Table 5 shows the results for a resource URI from the `Docker Engine` API `https://docker.engine.com/images/search`. The base forms of each node (*i.e.,* `image` and `search`) appear in Topic2 and Topic6 shown in Table 4. Hence, we report the URI as a *Contextual Resource Names* pattern. As shown in Table 5, we have two nodes in base forms: `image` and `search`. We compare the second-order similarity for each node with each terms of the obtained topic model shown in Table 5. The first column shows the second-order similarity values between the first node `image` and the topic model, and the second column shows the values

Table 5. Example of analysis of a URI from the `Docker Engine` API.

https://docker.engine.com/images/search		
URI Nodes: /images/search		
Node: image	Node: search	Topic average
image *vs.* **Topic0**	**search** *vs.* **Topic0**	
copy: 0.45550877	copy: 0.31544656	
unpause: −2.0	unpause: −2.0	
commit: 0.0	commit: 0.098813675	
build: 0.32840174	build: 0.01179231	
Max Result Topic 0: 0.45550877	Max Result Topic 0: 0.31544656	0.385477665
image *vs.* **Topic1**	**search** *vs.* **Topic1**	
resize: 0.2891726	resize: 0.57163477	
start: 0.012896833	start: 0.018111441	
tag: 0.101503715	tag: 0.1996914	
Max Result Topic 1: 0.2891726	Max Result Topic 1: 0.57163477	0.430403685
image *vs.* **Topic2**	**search** *vs.* **Topic2**	
image: 2.0	image: 0.2916552	
version: 0.2870915	version: 0.24986088	
event: 0.0743737	event: 0.042658847	
width: 0.12692761	width: 0.011248095	
pause: 0.07810811	pause: 0.09917007	
Max Result Topic 2: 2.0	Max Result Topic 2: 0.2916552	1.1458276
image *vs.* **Topic3**	**search** *vs.* **Topic3**	
amp: 0.032073073	amp: 0.011314062	
restart: 0.011473239	restart: 0.031095618	
wait: 0.036101706	wait: 0.09236097	
Max Result Topic 3: 0.0	Max Result Topic 3: 0.0	0
image *vs.* **Topic4**	**search** *vs.* **Topic4**	
container: 0.1470323	container: 0.070620485	
info: 0.23985307	info: 0.5876746	
push: 0.04289078	push: 0.078571856	
Max Result Topic 4: 0.23985307	Max Result Topic 4: 0.5876746	0.413763835
image *vs.* **Topic5**	**search** *vs.* **Topic5**	
create: 0.16315852	create: 0.19966161	
attach: 0.21755742	attach: 0.015138787	
log: 0.16914968	log: 0.31177378	
stop: 0.017904773	stop: 0.100144744	
Max Result Topic 5: 0.21755742	Max Result Topic 5: 0.31177378	0.2646656

(*continued*)

Table 5. (*continued*)

https://docker.engine.com/images/search		
URI Nodes: /images/search		
Node: image	Node: search	Topic average
image *vs.* **Topic6**	**search** *vs.* **Topic6**	
search: 0.2916552	search: 2.0	
height: 0.15941465	height: 0.0106301615	
Max Result Topic 6: 0.2916552	Max Result Topic 6: 2.0	1.1458276
image *vs.* **Topic7**	**search** *vs.* **Topic7**	
ping: 0.0	ping: 0.0	
history: 0.06135501	history: 0.05444772	
change: 0.11254101	change: 0.13595133	
auth: 0.04527525	auth: 0.28564966	
Max Result Topic 7: 0.11254101	Max Result Topic 7: 0.28564966	0.199095335
image *vs.* **Topic8**	**search** *vs.* **Topic8**	
container: 0.1470323	container: 0.070620485	
top: 0.18044263	top: 0.0	
Max Result Topic 8: 0.18044263	Max Result Topic 8: 0.0	0.090221315
image *vs.* **Topic9**	**search** *vs.* **Topic9**	
json: 0.11969886	json: 0.4653473	
exec: 0.073554516	exec: 0.2546377	
load: 0.14074977	load: 0.13070202	
kill: 0.0	kill: 0.13537067	
export: 0.022680383	export: 0.051750746	
Max Result Topic 9: 0.14074977	Max Result Topic 9: 0.4653473	0.303048535
Maximum Average		**1.1458276**

between the second node `search` and the topic model. The third column shows the average second-order similarity for each topic for both nodes. We obtain the maximum average of 1.1458 as the second-order semantic-similarity metric. Topic6 has the maximum average vale for the nodes, which is higher than the predefined threshold. Thus, we report these two nodes as contextual. We repeat this process for all the URIs and Cloud APIs to decide if the nodes in URIs are from the same context.

4 Cloud Lexical Analysis

Using CloudLex presented in Sect. 3.2, we extract a total of **3,947 different terms** in the REST APIs of **16 Cloud computing providers**. We analyse the terms to answer two main research questions as follows.

4.1 RQ1: Do RESTful Cloud APIs Follow Good Practices?

We analyse automatically the quality of the terms along five dimensions. We group the terms based on these dimensions, counting how many terms belong to each dimension. Table 6 shows the result of the grouping from which we observe several findings.

1. **Parts of Speech:** we classified the parts of speech of each term as *Noun* or *Verb*.
2. **Number:** we classified the terms as *Plural* or *Singular*.
3. **Casing:** we classified the terms as *Camel*, *Lower*, or *Upper* cased.
4. **Hyphenation:** we classified the terms based on the presence or not of hyphens.
5. **Underscoring:** we classified the terms based on the presence or not of underscores.

First, we find that 69% of the terms are nouns (3,323/4,843) and 31% verbs (1,520/4,843) and that the majority of the APIs avoid using extensively verbs. Second, we observe that 56% of the terms are plural (2,127/4,843) and 44% are singular (2,716/4,843). Third, the analysis of the APIs shows that 52% of the terms use camel casing (2,517/4,843), close to 48% are lowercase (2,313/4,843), and only 0.27% are upper case (13/4,843). Finally, 98% of the terms do not include hyphens (4,735/4,843) or underscores (4,753/4,843).

Petrillo et al. [18] compiled a catalog of 73 best practices in the design of REST APIs. In the catalog, there are five best practices directly related to URI

Table 6. Number of terms by provider and quality dimensions.

Provider	Terms Ocurrences by Provider	Parts of Speech		Quantity		Case			Hyphens		Underscores	
		Noun	Verb	Plural	Singular	Camel	Lower	Upper	No	Yes	No	Yes
1and1	42	93%	7%	64%	36%	0%	95%	5%	60%	40%	100%	0%
Cloud Foundry	75	93%	7%	64%	36%	0%	100%	0%	52%	48%	100%	0%
CloudStack	565	2%	98%	28%	72%	99%	1%	0%	100%	0%	100%	0%
Digital Ocean	46	52%	48%	54%	46%	0%	100%	0%	72%	28%	100%	0%
Docker	32	34%	66%	16%	84%	0%	100%	0%	97%	3%	100%	0%
Google Cloud	349	47%	53%	48%	52%	51%	49%	0%	100%	0%	100%	0%
Heroku	14	79%	21%	64%	36%	0%	100%	0%	93%	7%	100%	0%
IBM Bluemix	52	67%	33%	52%	48%	0%	100%	0%	100%	0%	96%	4%
Kubernetes	16	94%	6%	75%	25%	0%	100%	0%	100%	0%	100%	0%
Microsoft Azure	622	67%	33%	63%	37%	58%	42%	0%	100%	0%	100%	0%
OCCI	46	70%	30%	0%	100%	0%	100%	0%	74%	26%	100%	0%
OpenStack	160	76%	24%	54%	46%	8%	89%	3%	92%	8%	68%	32%
Oracle Cloud	1,518	88%	12%	52%	48%	60%	40%	0%	100%	0%	99%	1%
OVH	1,014	79%	21%	23%	77%	48%	52%	0%	100%	0%	100%	0%
Rackspace	146	81%	19%	51%	49%	5%	94%	1%	95%	5%	92%	8%
VMWare	146	81%	19%	51%	49%	5%	94%	1%	95%	5%	92%	8%
Total	**4,843**	**69%**	**31%**	**44%**	**56%**	**52%**	**48%**	**0%**	**98%**	**2%**	**98%**	**2%**

lexicons: (1) *lowercase letters should be preferred in URI paths*; (2) *a singular noun should be used for document names*; (3) *a plural noun should be employed for collection names*; (4) *a verb or verb phrase should be used for controller names*; (5) *underscores should not be used*. Our results show that the APIs, in general, follow these five good practices.

We thus conclude that **the lexicon of the analysed Cloud computing REST APIs contains a majority of nouns, which are equally singular or plural, and are mainly in lower case, following REST API best practices.**

4.2 RQ2: Which Lexicon Is Adopted by Cloud Computing Providers?

We count the number of providers that adopt a same term, observing that although the 16 studied REST Cloud APIs describe the same domain, contrary to our expectation, they do not share a large number of common terms. In fact, 90% of the terms (3,561/3,947) are used by one provider only, 5% of the terms (198/3,947) are adopted by two providers, and 5% of the terms (198/3,947) are adopted by three providers or more. If we define **consensus** when all providers adopt a term, there is no term that is consensual in the 16 studied APIs.

Although a majority of the terms are adopted just by one provider, we can highlight 23 terms that are used by seven[4] or more providers: images (used by 11 providers), events (10), users (9), services (9), stop (9), resources (8), logs (8), roles (8), snapshots (8), restore (8), actions (8), restart (8), instances (7), domains (7), volumes (7), credentials (7), config (7), export (7), start (7), tags (7), validate (7), and resume (7).

Thus, **we conclude that the 16 Cloud APIs are lexically heterogeneous, with few common terms. There is not a consensus on which terms to use in Cloud computing REST APIs**.

5 Semantic Analysis of Cloud APIs for the Detection of Linguistic (Anti)patterns

In this section, we assess the effectiveness of our semantic analysis approach on Cloud APIs by (1) verifying if linguistic (anti)patterns do occur in Cloud APIs and (2) analysing the detection accuracy for the detected linguistic (anti)patterns. In this paper, we perform the validation study on more than 23,000 URIs from 16 Cloud API providers.

5.1 Subjects and Objects

The subjects of our semantic analysis are the four linguistic patterns and antipatterns described in Sect. 2.3 and the objects of our analysis are the 23,062 URIs

[4] We chose at least seven providers to show a short list of terms (around of 20 terms). The full list is available at https://github.com/Spirals-Team/CloudLexicon.

Table 7. Number of URIs tested from each Cloud API provider.

Cloud API providers	Test URIs
1and1 Cloud Server	161
Apache CloudStack	563
Cloud Foundry	233
Digital Ocean	131
Docker Engine	40
Google Cloud Platform	505
Heroku	30
IBM Bluemix	113
Kubernetes	114
Microsoft Azure	1,820
OCCI	204
OpenStack	588
Oracle Cloud	11,264
OVH	4,229
Rackspace	479
VMware	2,588
Total	**23,062**

from the 16 Cloud API providers. Table 7 summarises the numbers of URIs per provider.

5.2 Research Questions

We propose two research questions to assess the effectiveness of our semantic analysis for the detection of linguistic (anti)patterns:

RQ$_1$ *To what extent do the analysed Cloud APIs contain the linguistic patterns and antipatterns (defined in* Sect. 2.3*)?*
By answering RQ$_1$, we show the quality of the URIs.
RQ$_2$ *How accurate are the detected linguistic patterns and antipatterns?*
By answering RQ$_2$, we show whether our identification process is accurate.

5.3 Validation Process

For the validation, we collected URIs and their corresponding documentations for each Cloud APIs and subsequently applied the detection rules of linguistic patterns and antipatterns [15] in the form of detection algorithms using CloudLex. We validated detection results in two parts: (1) for *Contextless vs. Contextual Resource Names*, 50 randomly-selected URIs from the 16 Cloud APIs and for *Pertinent vs. Non-pertinent Documentation*, 25 randomly-selected

URI–documentation pairs to measure the overall precision and (2) to measure precision and recall for one Cloud API provider, we chose `Docker Engine` with its reasonable number of URIs, *i.e.,* other providers have high numbers of URIs up to 11,264.

We involved three professionals, who are expert in Android, iOS, and Cloud development, to identify the true positives and false negatives to define a gold standard for `Docker Engine`. They also assisted in our validation process where we calculate overall precision by randomly choosing 50 URIs and 25 URI–documentation pairs. For the validation purposes, we provided them with the online descriptions[5] of the linguistics patterns and antipatterns, the sets of 50 randomly selected URIs, the set of 25 randomly selected URI–documentation pairs, and all the URIs and URI–documentation pairs from `Docker Engine`. We involved odd number of professionals to resolve their conflicts with majority decision.

5.4 Detection Results

Table 8 shows detection results for the first 15 URIs from `Docker Engine`. As shown in Table 8, all the URIs from `Docker Engine` are detected as *Contextualised Resource Names* pattern.

Table 8. Detection results of *Contextless vs. Contextualised Resource Names* for the first 15 URIs from `Docker Engine`.

URI	Detected as
`https://docker.engine.com/auth`	Pattern
`https://docker.engine.com/build`	Pattern
`https://docker.engine.com/commit`	Pattern
`https://docker.engine.com/containers/create`	Pattern
`https://docker.engine.com/containers/{id}/start`	Pattern
`https://docker.engine.com/containers/{id}/stop`	Pattern
`https://docker.engine.com/containers/{id}/top`	Pattern
`https://docker.engine.com/containers/{id}/logs`	Pattern
`https://docker.engine.com/containers/{id}/attach`	Pattern
`https://docker.engine.com/containers/{id}/exec`	Pattern
`https://docker.engine.com/containers/{id}/json`	Pattern
`https://docker.engine.com/containers/{id}`	Pattern
`https://docker.engine.com/containers/{id}/unpause`	Pattern
`https://docker.engine.com/containers/{id}/export`	Pattern
`https://docker.engine.com/containers/{id}/wait`	Pattern

[5] http://sofa.uqam.ca/resources/antipatterns.php.

Table 9 shows the detection results for the first 20 URI–documentation pairs of the `Docker Engine` API. All these pairs are identified as *Pertinent Documentation* except two. Those two pairs: (1) `https://docker.engine.com/containers/id/json` – [inspect a container] and (2) `https://docker.engine.com/containers/json` – [List containers] are `Non-pertinent Documentation`. The rationale behind this outcome could be the absence of the specific term "json" in both their documentations and, in the first documentation, the term "inspect" seems important but the URI does refer to it. All our analyses results are available on our project Web site at https://github.com/Spirals-Team/CloudLexicon.

In the following sections, we answer our two research questions on the presence of patterns and antipatterns in Cloud APIs (RQ_1) and on the accuracy of our identification approach (RQ_2).

Table 9. Detection results of *Pertinent vs. Non-pertinent Documentation* for the first 20 URI–documentation pairs of `Docker Engine`.

URI	Description	Detected as
`https://docker.engine.com/auth`	Check auth configuration	Pattern
`https://docker.engine.com/build`	Build an image from Docker file via stdin	Pattern
`https://docker.engine.com/commit`	Create a new image from a containers changes	Pattern
`https://docker.engine.com/containers/create`	Create a container	Pattern
`https://docker.engine.com/containers/{id}/start`	Start a container	Pattern
`https://docker.engine.com/containers/{id}/stop`	Stop a container	Pattern
`https://docker.engine.com/containers/{id}/top`	List processes running inside a container	Pattern
`https://docker.engine.com/containers/{id}/logs`	Get container logs	Pattern
`https://docker.engine.com/containers/{id}/attach`	Attach to a container	Pattern
`https://docker.engine.com/containers/{id}/exec`	Image tarball format	Pattern
`https://docker.engine.com/containers/{id}/json`	Inspect a container	Antipattern
`https://docker.engine.com/containers/{id}`	Remove a container	Pattern
`https://docker.engine.com/containers/{id}/unpause`	Unpause a container	Pattern
`https://docker.engine.com/containers/{id}/export`	Export a container	Pattern
`https://docker.engine.com/containers/{id}/wait`	Wait a container	Pattern
`https://docker.engine.com/containers/{id}/pause`	Pause a container	Pattern
`https://docker.engine.com/containers/json`	List containers	Antipattern
`https://docker.engine.com/containers/{id}/changes`	Inspect changes on a containers file system	Pattern
`https://docker.engine.com/containers/{id}/restart`	Restart a container	Pattern
`https://docker.engine.com/containers/{id}/copy`	Copy files or folders from a container	Pattern

Table 10. Summary of the semantic analyses of the 16 Cloud APIs.

API providers	URIs tested	Contextless	Contextual	Pertinent	Non-pertinent
1and1 Cloud Server	161	2 (1.24%)	159 (98.76%)	72 (44.72%)	89 (55.28%)
Apache CloudStack	563	0 (0%)	563 (100%)	0 (0%)	563 (100%)
Cloud Foundry	233	5 (2.15%)	228 (97.85%)	-	-
Digital Ocean	131	35 (26.72%)	96 (73.28%)	46 (35.11%)	85 (64.89%)
Docker Engine	40	0 (0%)	40 (100%)	35 (87.50%)	5 (12.50%)
Google Cloud Platform	505	248 (49.11%)	257 (50.89%)	58 (11.49%)	447 (88.51%)
Heroku	30	5 (16.67%)	25 (83.33%)	14 (46.67%)	16 (53.33%)
IBM Bluemix	113	8 (7.08%)	105 (92.92%)	91 (80.53%)	22 (19.47%)
Kubernetes	114	3 (2.63%)	111 (97.37%)	-	-
Microsoft Azure	1,820	1,744 (95.82%)	76 (4.18%)	-	-
OCCI	204	4 (1.96%)	200 (98.04%)	-	-
OpenStack	588	57 (9.69%)	531 (90.31%)	476 (80.95%)	112 (19.05%)
Oracle Cloud	11,264	4,717 (41.88%)	6,547 (58.12%)	-	-
OVH	4,229	2,702 (63.89%)	1,527 (36.11%)	-	-
Rackspace	479	62 (12.94%)	417 (87.06%)	-	-
VMware	2,588	1,003 (38.76%)	1,585 (61.24%)	-	-
Total	23,062	10,595	12,467	792	1,339
Average for APIs		23.16%	76.84%	48.37%	51.63%

5.5 RQ_1: To What Extent Do the Analysed Cloud APIs Contain the Linguistic Patterns and Antipatterns?

Table 10 shows the summary of our detection results in the 16 Cloud APIs. Figures 3 and 4 show the total numbers of linguistic patterns and antipatterns for each Cloud API. All the Cloud APIs follow the *Contextualised Resource Names*

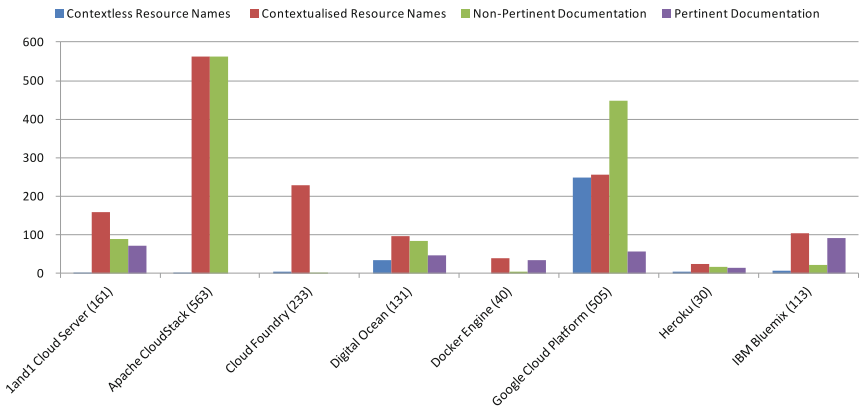

Fig. 3. Detection results of the four patterns and antipatterns in the 16 Cloud APIs (part-I).

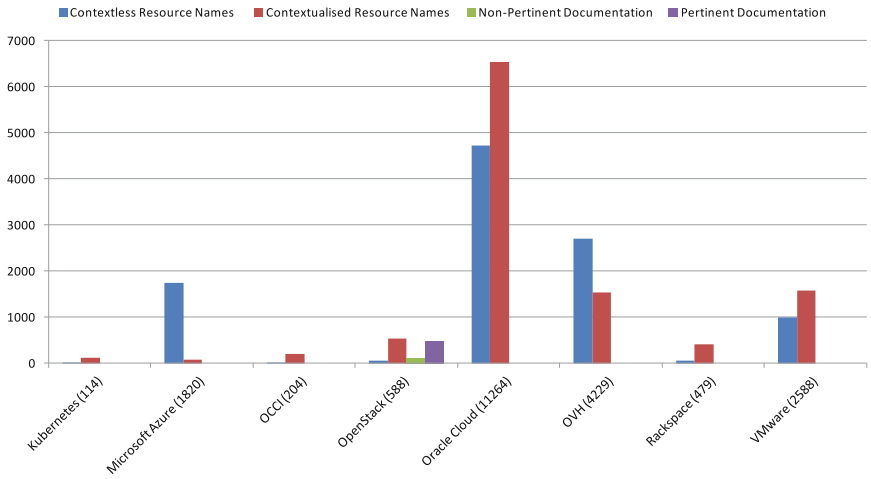

Fig. 4. Detection results of the four patterns and antipatterns in the 16 Cloud APIs (part-II).

pattern. We identified two APIs following the *Contextualised Resource Names* pattern for all of their URIs, *e.g.,* `Apache CloudStack` and `Docker Engine`. Around 50%–98% of the URIs from remaining 14 Cloud APIs follow this pattern. In addition, we identified the *Contextless Resource Names* antipattern in some large Cloud APIs, including `Google Cloud Platform` (248 occurrences, 49%), `IBM Bluemix` (8 occurrences, 7%), `Microsoft Azure` (1,744 occurrences, 95%), `Oracle Cloud` (4,717 occurrences, 41%), and `VMware` (1,003 occurrences, 38%). In summary, out of 23,062 URIs, 10,595 (45.94%) are *Contextless Resource Names* and 12,467 (54.06%) are *Contextualised Resource Names*.

Moreover, of the eight Cloud APIs in which we identified the *Pertinent vs. Non-pertinent Documentation* (anti)patterns, we report that they all have *Non-pertinent Documentation*. In particular, 89 occurrences (55.28%) of `1and1 Cloud Server`, 563 occurrences (99.98%) of `Apache CloudStack`, 85 occurrences (64.89%) of `Digital Ocean`, 5 occurrences (12.50%) of `Docker Engine`, 447 occurrences (88.51%) of `Google Cloud Platform`, 16 occurrences (53.33%) of `Heroku`, 22 occurrences (19.47%) of `IBM Bluemix`, and 112 (19.05%) occurrences of `OpenStack` have *Non-pertinent Documentation*. Thus, out of 2,131 pairs of URIs–documentations, 1,339 (62.82%) have *Non-pertinent Documentation*.

These findings suggest that majority, *i.e.,* 54%, of the analysed URIs follow *Contextualised Resource Names* pattern that is a good practice in API design. In contrast, a majority, *i.e.,* 62.82%, of the analysed URI–documentation pairs suffer of the *Non-pertinent Documentation*.

We can positively answer RQ_1: analysed Cloud APIs contain the linguistic patterns and antipatterns. The majority of the analysed URIs have *Contextualised Resource Names*—a good design practice. Our findings also show that the majority of the analysed URI–documentation pairs have *Non-pertinent Documentation*.

5.6 RQ_2: *How Accurate Are the Detected Linguistic Patterns and Antipatterns?*

Table 11 shows that our approach identified 22 URIs as antipatterns, 14 of them were true positive as validated manually. Therefore, we obtained an accuracy of 63.64%. Together with *Contextualised Resource Names* pattern (with 78.57% accuracy), we obtain an overall precision of 71.10% for *Contextualised vs. Contextless Resource Names*. Similarly, for *Pertinent vs. Non-pertinent Documentation*, we obtain an average precision of 77.27% with the manual validation for randomly selected 25 URI–documentation pairs from the 2,131 pairs. Thus, for the first validation, we obtain an average precision of 74.19%.

Table 11. Validation summary on 50 randomly-selected URIs and 25 randomly-selected URI–documentation pairs of the 16 Cloud APIs.

Antipatterns/patterns	P	TP	Validated	Precision	Average precision for anti(pattern)
Contextless Resource Names	22	14	20	63.64%	71.10%
Contextualised Resource Names	28	22	30	78.57%	
No detection	0	0	0	-	
Non-pertinent Documentation	11	6	6	54.55%	77.27%
Pertinent Documentation	14	14	19	100%	
No detection	0	0	0	-	
Average Precision					**74.19%**

While the goal of our first validation is to measure the overall accuracy of our approach for linguistic analysis, in this second validation, we want to show not only the precision but also the recall of our approach. Professional developers manually validated all 40 URIs from `Docker Engine` Cloud API for *Contextualised vs. Contextless Resource Names* and all 40 URI–documentation pairs from `Docker Engine` for *Pertinent vs. Non-pertinent Documentation*. Table 12 shows that we did not identify any occurrence of *Contextless Resource Name*s but the manual validation revealed instances of contextless nodes in three URIs. Therefore, our detection process missed all three true positives (*i.e.*, recall of 0%) for this antipattern. However, we obtained 100% of precision. On average, for the *Contextualised vs. Contextless Resource Name*, we obtain an average precision of 92.5% and an average recall of 50% with F1-measure of 64.91%. Similarly, for the *Pertinent vs. Non-pertinent Documentation*, we obtain an average precision and recall of 77.14% with F1-measure of 77.14%.

We conclude that, as shown in Table 12, our approach has a global average precision of 84.82% and a global average recall of 63.57% with an average F1-measure of 71.03%. Thus, we can positively answer RQ_2 on the detection accuracy of linguistic patterns and antipatterns.

Table 12. Validation summary on the URIs and URI–documentation pairs of `Docker Engine`.

Antipatterns/patterns	P	TP	Validated	Precision	Avg. precision	Recall	Avg. recall	F1-measure
Contextless Resource Names	0	0	3	-	92.5%	0%	50%	64.91%
Contextualised Resource Names	40	37	37	92.50%		100%		
No detection	0	0	0	-		-		
Non-pertinent Documentation	5	3	5	60%	77.14%	60%	77.14%	77.14%
Pertinent Documentation	35	33	35	94.29%		94.29%		
No detection	0	0	0	-		-		
Average Precision					**84.82%**			
Average Recall							**63.57%**	
Average F1-measure								**71.03%**

5.7 Threats to Validity

As with any empirical study, threats exist that reduce the validity of our results, which we attempted to mitigate or had to accept. We now discuss these threats and the measures that we took on them.

Threats to the Construct Validity. These threats concern the relationship between theory and observations. We assumed that good naming practices [18] improve the quality of the REST APIs of the Cloud providers [23]. Although these assumptions are legitimate and have been withheld by many researchers and works, for example that of Zhang and Budgen [22], future work should study whether these good naming practices apply universally to all Cloud services. Also, we argued that the lexicons should be homogeneous to help developers' comprehension but this argument should be validated experimentally.

Threats to Internal Validity. These threats concern confounding factors that can affect our dependent variables. Although we did not carry any statistical analysis on the characteristics of the studied REST APIs, we assumed that the lexicons were a feature of the REST APIs. However, there may be other terms that describe more accurately these REST APIs and that impact their comprehension, in particular their documentations. Future work includes analysing and contrasting more APIs with more terms and documentations.

Threats to External Validity. These threats concern the generalization of our results. Although we presented, to the best of our knowledge, the largest study on the lexicons of Cloud computing REST APIs, we cannot generalise our results to all Cloud computing REST APIs. Future work is necessary to analyze more REST APIs, from other Cloud providers, open-source implementations, and standards to confirm and–or infirm our observations.

6 Related Work

Recently, there is a growing interest in the design quality evaluation of REST APIs. However, to the best of our knowledge, few studies made specifically a lexical evaluation of REST APIs in general, and none in the domain of cloud computing.

In related work for the general design quality evaluation of REST APIs, we can cite the research work of Hausenblas [10], who studies some widely used REST Web APIs in terms of URI space design, resource representations, and hyperlinking support. Rodríguez et al. [5] evaluated also the conformance of good and bad design practices in REST APIs from the perspective of mobile applications. They analysed large data logs of HTTP calls collected from the Internet traffic of mobile applications, identified usage patterns from logs, and compared these patterns with best design practices. Zhou et al. [24] showed how to fix design problems related to the use of REST services in existing Northbound networking APIs in a Software Defined Network and how to design a REST Northbound API in the context of OpenStack. These previous work made contributions to the design evaluation of REST APIs for general or specific domains, mobile and networking, while we consider the domain of cloud services.

Palma et al. [16] evaluated the linguistic aspects of several REST APIs based on REST patterns and anti-patterns, which correspond to good and bad practices in the design of REST services. However, the APIs evaluated were selected from different and general domains. They included Facebook, Twitter, Dropbox, and Bestbuy. So, it was not possible to compare and discuss the results among the APIs. Moreover, the list of patterns and anti-patterns was really compared to this focused study.

Petrillo et al. [18] evaluated three cloud computing REST APIs using a catalog of 73 general best practices. However, this catalog was mainly dedicated to the design of REST APIs from a conceptual and syntactic point of view, but not necessarily lexical. The present paper specifically focuses on a lexical evaluation of cloud computing REST APIs.

Researchers have analysed Cloud APIs to verify if the developers properly use them [9,20]. However, no study were conducted in the literature to assess the linguistic quality of Cloud APIs. In the following, we discuss some relevant research done on assessing the structural correctness of APIs [9,20] or the use of Cloud ontology [1,21] for comprehension.

Developers repeatedly need to manually ensure that they are building HTTP requests using correct URIs while developing framework-based JavaScript Web applications, which is error-prone. Wittern *et al.* [20] proposed an approach for statically checking request URIs in JavaScript-based applications by extracting their URLs, HTTP methods, and the corresponding request data. The authors evaluated if request URIs in JavaScript files conform to their publicly available specifications. With analysing more than 6,000 request URIs, the approach achieved the detection accuracy of more than 95%. This study ensures developers use URI correctly, however, does not analyse the linguistic quality which we perform in our work.

Haupt *et al.* [9] proposed a framework for structural analysis of REST APIs based on their online specifications. The authors considered this proposed framework as a first catalog of REST APIs describing structural characteristics that comprises a set of metrics and graphical representation for each API. Moreover, this framework comes useful in identifying the non-conformity of REST APIs to the REST design principles and architectural style. Similar to our work in this paper, the proposed framework in [9] considers the API descriptions and the structure of REST APIs. The authors also proposed a meta-model for REST APIs describing the structure of REST APIs. However, this study only verifies structural correctness of APIs but does not verify their linguistic quality.

Androcec *et al.* [1] provided a global view of Cloud computing ontology after a systematic review. The authors classified the relevant studies into four main categories: cloud resources and services description, cloud security, cloud interoperability, and cloud services discovery and selection. The authors found that the studies from 'cloud resources and services descriptions' category applied Cloud ontologies to describe Cloud resources and services, classify the current services, and pricing models. The Cloud interoperability category consisted of the studies that use ontologies to achieve interoperability among different Cloud providers and their offered services. The authors concluded that Cloud Computing ontologies are primarily applied in the discovery and selection of the best candidate service in accordance with users' computing requirements and the specifications of Cloud resources and services.

In another study, Youseff *et al.* [21] proposed a detailed Cloud ontology to facilitate the comprehension of the Cloud technology as, the authors suggested, it would enable the community to design more efficient cloud applications. Based on the fact that the current state-of-the-art in Cloud computing research lacks the thorough understanding of the classification of the cloud systems, the authors presented a classification of Cloud components, their relationships, and their dependency on concepts from other service computing domains. According to the proposed ontology, the Cloud computing systems fall within *applications*, *software environments*, *software infrastructure*, *software kernel*, and *hardware* categories. However, both the studies [1,21] focused on ontology aspect of REST APIs. The linguistic aspect of the Cloud APIs is not considered in these studies.

Last but not least, Chalitta *et al.* [6] defined formal-based framework for semantic interoperability in multi-clouds, organizing a catalogue of formal models that mathematically describe cloud APIs, describing their concepts and semantic interoperability.

7 Conclusion and Future Work

Cloud computing is a popular Internet-based computing paradigm that provides on-demand computational services and resources, generally offered by Cloud providers' REpresentational State Transfer (REST) APIs. Developers use REST APIs by invoking these APIs by their names and, thus, the lexicons used in the APIs are important to ease the developers' comprehension. We claimed that

Cloud computing lexicons reflect the nature of their APIs and the automatic detection of "linguistic" antipatterns could further boost the adoption of Cloud computing.

We presented three contributions. First, we introduced CloudLex, an approach to build the lexicons of Cloud computing REST APIs, based on a conceptual model and providing a toolkit to extract and analyse these lexicons. Second, we extracted and studied the lexicons of 16 REST Cloud Computing APIs. Finally, we analysed semantically the (anti)patterns in 1,297 URIs of the 142 services of the 16 Cloud providers.

We showed that the 16 APIs form **a lexicon of 3,947 different terms** to express all provided services. We found that this lexicon contains a majority of nouns, which are equally singular or plural, and are mainly in lower case, following REST API best practices.

We observed that, although the 16 studied REST Cloud APIs describe the some domain (Cloud computing), contrary to what one might expect, they do not share a large number of common terms. In fact, 90% of the terms (3,561/3,947) are used by only one provider, 5% of the terms (198/3,947) are adopted by two providers, and the other 5% of the terms (198/3,947) are adopted by three providers or more. Thus, we conclude that the 16 APIs are lexically heterogeneous, which point that there is not a consensus on which terms to use in Cloud computing.

We also showed that, through our semantic analysis of the Cloud APIs, 54% of the URIs follow the *Contextualised Resource Names* pattern, which is considered a good practice in API design. However, 62.82% of the URIs suffer from the *Non-pertinent Documentation* antipattern. We also reported the detection accuracy on one complete API, `Docker Engine`, with a global average precision of 84.82% and a global average recall of 63.57% for an average F1-measure of 71.03%, which confirms the accuracy of our semantic analyses for the detection of linguistic patterns and antipatterns.

In future work, we plan to build an ontology of Cloud computing APIs, establishing semantic joins between services and resources from different providers to deal with semantic interoperability between Clouds. Further, future work is necessary to analyze more REST APIs, from other Cloud providers, open-source implementations, and standards to confirm and–or infirm our observations.

Acknowledgment. This work is partially supported by the OCCIware research and development project (http://www.occiware.org) funded by French Programme d'Investis sements d'Avenir (PIA). It is also co-funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

1. Androcec, D., Vrcek, N., Ševa, J.: Cloud Computing Ontologies: A Systematic Review, September 2017
2. Armbrust, M., Stoica, I., Zaharia, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A.: A view of cloud computing. Commun. ACM **53**(4), 50 (2010)

3. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent Dirichlet allocation. J. Mach. Learn. Res. **3**(4–5), 993–1022 (2003)
4. Budanitsky, A., Hirst, G.: Evaluating WordNet-based measures of lexical semantic relatedness. Comput. Linguist. **32**(1), 13–47 (2006)
5. Rodríguez, C., Baez, M., Daniel, F., Casati, F., Trabucco, J.C., Canali, L., Percannella, G.: REST APIs: a large-scale analysis of compliance with principles and best practices. In: Bozzon, A., Cudre-Maroux, P., Pautasso, C. (eds.) ICWE 2016. LNCS, vol. 9671, pp. 21–39. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-38791-8_2
6. Challita, S., Paraiso, F., Merle, P.: Towards formal-based semantic interoperability in multi-clouds. In: 10th IEEE International Conference on Cloud Computing (CLOUD), Honolulu, Hawaii, USA, June 2017. https://hal.inria.fr/hal-01519831
7. Fielding, R.T.: Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California, Irvine (2000)
8. Fredrich, T.: RESTful Service Best Practices: Recommendations for Creating Web Services, May 2012. http://www.restapitutorial.com/resources.html
9. Haupt, F., Leymann, F., Scherer, A., Vukojevic-Haupt, K.: A framework for the structural analysis of REST APIs. In: Proceedings of the 1st IEEE International Conference on Software Architecture, ICSA 2017, 3–7 April 2017, Gothenburg, Sweden, pp. 55–58. IEEE Computer Society (2017)
10. Hausenblas, M.: On entities in the web of data. In: Wilde, E., Pautasso, C. (eds.) REST: From Research to Practice, pp. 425–440. Springer, New York (2011). https://doi.org/10.1007/978-1-4419-8303-9_19
11. Kolb, P.: DISCO: a multilingual database of distributionally similar words. In: Proceedings of 9th Conference on Natural Language Processing KONVENS 2008, Berlin, Germany, no. 2003, pp. 37–44 (2008)
12. Kolb, P.: Experiments on the Difference between Semantic Similarity and Relatedness. In: 17th Nordic Conference of Computational Linguistics, Odense, Denmark, NODALIDA 2009 (2009)
13. Masse, M.: REST API Design Rulebook, vol. 53. O'Reilly Media, Sebastopol (2011)
14. Nyren, R., Edmonds, A., Papaspyrou, A., Metsch, T., Parak, B.: Open Cloud Computing Interface - Core. OCCI-WG Specification Document 1.2, Open Grid Forum (2016). https://redmine.ogf.org/attachments/217/core.pdf
15. Palma, F., Gonzalez-Huerta, J., Founi, M., Moha, N., Tremblay, G., Guéhéneuc, Y.G.: Semantic analysis of RESTful APIs for the detection of linguistic patterns and antipatterns. Int. J. Coop. Inf. Syst. **26**(02), 1742001 (2017). http://www.worldscientific.com/doi/abs/10.1142/S0218843017420011
16. Palma, F., Gonzalez-Huerta, J., Moha, N., Guéhéneuc, Y.-G., Tremblay, G.: Are RESTful APIs well-designed? Detection of their linguistic (anti)patterns. In: Barros, A., Grigori, D., Narendra, N.C., Dam, H.K. (eds.) ICSOC 2015. LNCS, vol. 9435, pp. 171–187. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48616-0_11
17. Petrillo, F., Merle, P., Moha, N., Guéhéneuc, Y.G.: Towards a REST cloud computing Lexicon. In: ScitePress (ed.) 7th International Conference on Cloud Computing and Services Science, CLOSER 2017, pp. 376–383. INSTICC, Porto, April 2017. https://hal.archives-ouvertes.fr/hal-01480593
18. Petrillo, F., Merle, P., Moha, N., Guéhéneuc, Y.G.: In: Proceedings of 14th International Conference on Service-Oriented Computing, ICSOC 2016, Banff, Canada, 10–13 October 2016 (2016)

19. Steyvers, M., Griffith, T.: Probabilistic topic models. In: Landauer, T., McNamara, D., Dennis, S., Kintsch, W. (eds.) Latent Semantic Analysis: A Road to Meaning. Laurence Erlbaum, Mahwah (2007)
20. Wittern, E., Ying, A.T.T., Zheng, Y., Dolby, J., Laredo, J.A.: Statically checking Web API requests in JavaScript. In: Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, pp. 244–254. IEEE Press, Piscataway (2017). https://doi.org/10.1109/ICSE.2017.30
21. Youseff, L., Butrico, M., Silva, D.D.: Toward a unified ontology of cloud computing. In: 2008 Grid Computing Environments Workshop, pp. 1–10, November 2008
22. Zhang, C., Budgen, D.: What do we know about the effectiveness of software design patterns? IEEE Trans. Softw. Eng. **38**(5), 1213–1231 (2012)
23. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state-of-the-art and research challenges. J. Internet Serv. Appl. **1**(1), 7–18 (2010)
24. Zhou, W., Li, L., Luo, M., Chou, W.: REST API design patterns for SDN northbound API. In: 2014 28th International Conference on Advanced Information Networking and Applications Workshops. pp. 358–365. IEEE, May 2014. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6844664