Detecting Cloud (Anti)Patterns: OCCI Perspective

Hayet Brabra[1,4](✉), Achraf Mtibaa[2], Layth Sliman[3], Walid Gaaloul[4], Boualem Benatallah[5], and Faiez Gargouri[1]

[1] ISIMS Sfax, Miracl Laboratory, Sfax, Tunisia
brabra.hayeet@gmail.com, faiez.gargouri@gmail.com
[2] ENETCOM Sfax, Miracl Laboratory, Sfax, Tunisia
achrafmtibaa@gmail.com
[3] Efrei, Paris, France
layth.sliman@efrei.fr
[4] TELECOM SudParis, CNRS UMR Samovar, Évry, France
walid.gaaloul@mines-telecom.fr
[5] UNSW, Sydney, Australia
boualem.benatallah@gmail.com

Abstract. Open Cloud Computing Interface (OCCI) follows a set of guidelines (i.e. best practices) to create interoperable APIs over Cloud resources. In this paper, we identify a set of patterns that must be followed and anti-patterns that should be avoided to comply with the OCCI guidelines. To automatically detect (anti)patterns, we propose a Semantic-based approach, relying on SWRL (Semantic Web Rule Language) rules and in SQWRL (Semantic Query-Enhanced Web Rule Language) queries to describe the (anti)patterns symptoms. An evaluation, conducted on real world Cloud service APIs, shows the feasibility of the proposed approach by assessing their compliance to OCCI standard.

Keywords: OCCI · Pattern · Anti-pattern · Ontology · SWRL · SQWRL

1 Introduction

Cloud Computing has emerged as a new technological paradigm that aims to offer a novel vision to deliver computing resources with significant cost reduction. However, its rapid evolution has reached a level of complexity due mainly to the vast and heterogeneous amount of services and resources. More precisely, the management of a potentially large number of Cloud services with heterogeneous interfaces is a challenge [13]. The lack of interoperability solutions may hinder the widespread adoption of Cloud Computing because organization fear of vendor lock-in [6,12]. The latter refers to a situation in which once an organization has selected a Cloud provider changing to another provider can be only very costly.

A common way for avoiding these issues is the use of open standards [6,9,12]. Actually, many Cloud projects are developing standards for the Cloud.

However, the most popular one is Open Cloud Computing Interface (OCCI) [1]. OCCI is an open standard defining a meta-model for Cloud resources and a RESTful API for management tasks. A well-designed management API hides the heterogeneity and evolution of the managed resources across various providers, while providing unified and efficient access to them. Among the OCCI specifications, OCCI HTTP Protocol [16] provides a set of guidelines (i.e. recommended best practices) to create unified APIs for managing Cloud resources. These best practices represent together a minimal set to achieve the interoperability and provide a uniform way to discover and manage Cloud resources across various providers. The non-compliance or poor adoption of such practices in current Cloud resource management APIs may negatively impact the interoperation of Cloud services. Currently, OCCI members provide a textual description of suggested guidelines in [16] as well as a compliance test tool [2] that does not provide a clear and detailed description. This tool can be used to show the presence of best practices, but never to show the absence of one of them. In this paper, we aim at providing developer both good and poor practices in Cloud RESTful APIs according to OCCI perspective and thus increasing its understandability. Some efforts have been realized in the past to deal with RESTful APIs. For example, Francis et al. [8] proposed a heuristic-based approach to detect (anti)patterns in RESTful systems to enhance their understandability and reusability. Such approach is tailored to deal with REST APIs like facebook and Twitter as it only focuses on REST aspects. Therefore, they cannot be applied to RESTful APIs developped for cloud services or ressources. Along with REST aspects, OCCI provides guidelines that relate to the structure and definition of Cloud resources that have not been so far considered in existing research work.

In this paper, we define non-compliance (respectively, compliance) to OCCI RESTful API guidelines as OCCI REST Anti-patterns (respectively, OCCI REST Patterns). We propose semantic-based detection of OCCI REST Patterns and Anti-patterns in Cloud RESTful APIs. More specifically, we propose (1) a semantic specification of 28 common OCCI REST(anti)patterns for Cloud RESTful APIs; (2) SWRL rules[1] in conjunction with SQWRL queries[2] for automatic detection of OCCI REST(anti)patterns (3) and a validation of our approach by analyzing the 28 OCCI REST (anti)patterns on real world Cloud RESTful APIs, including Openstack, COAPS, OpenNebula, Amazon S3, Microsoft Azure and Rackspace. The remainder of the paper is structured as follows: in Sect. 2 we discuss the related work. Section 3 presents the proposed approach. Section 4 presents a validation of our solution. Finally, we conclude the paper and provide insight for future works.

2 Related Work

Over the last years, several researches used patterns and anti-patterns to express architectural concerns and solutions in Object Oriented Systems (OO systems),

[1] https://www.w3.org/Submission/SWRL/.
[2] http://protege.cim3.net/cgi-bin/wiki.pl?SQWRL.

Service Oriented Architectures (SOAs) and recently in RESTful APIs. In the context of OO systems, Kessentini et al. [11] proposed an automated approach to detect various types of design defects in the source code. The proposed approach is based on detection rules that are defined as combinations of metrics and thresholds that better conform to known instances of design defects. Fourati et al. [7] proposed an approach that identifies anti-patterns in UML designs through the use of existing and newly defined quality metrics. The proposed approach examines the structural and behavioral information through the class and sequence diagrams. Another interesting effort has been done in [20] which proposed SPARSE, an OWL ontology based knowledge system that aims at assessing software project managers in the anti-pattern detection process.

Other related works have focused on the detection of anti-patterns and patterns in SOA. Dudney et al. [4] have defined a catalog of 53 anti-patterns related to the architecture, design, and implementation of J2EE-based systems. In [15], Moha et al. addressed the lack of methods and techniques for detecting SOA anti-patterns in service-based systems (SBSs). The aim is to provide an approach based on BNF grammar to detect the anti-patterns that may occur under SBSs in order to help the software engineer in assessing the design and QoS. This approach is also adapted to detect anti-patterns in Web Services [19].

However, we noted that both OO and SOA detection methods cannot be directly applied to RESTful APIs because OO focuses only on classes and SOA focuses on services and WSDL descriptions. In consequence, Francis et al. in [8] proposed a heuristics-based approach to detect (anti)patterns in RESTful systems. Additionally, the same authors have proposed DOLAR approach which applies syntactic and semantic analyses for the detection of linguistic (anti)patterns in RESTful APIs [18]. Both approaches only define a set of (anti)patterns focusing on properties related to REST architecture. Thus, they cannot be applied to RESTful APIs developped for cloud services or ressources. Motivated by these considerations, in this paper, we will focus on (anti)patterns that relate to the structure and definition of Cloud resources that have not been so far considered in previous research works. These (anti)patterns will be then used to assess the compliance of current Cloud RESTful APIs to OCCI guidelines.

3 Approach Overview

Our approach is based on semantics solutions to formally define (anti)patterns and ensure their automatic detection. The first reason to have such choice is the need for a technique that deals with the structure and semantic relations among resources, services, and parameters and able to resolve the ambiguity in terminologies used by developer to describe a cloud RESTful API. The second is to provide an automatic support to detect (anti)patterns through applying a reasoning process to draw inferences from details in a Cloud RESTful API with the assurance that the provided new knowledge is sound. In this section, we present an overall overview of our proposed approach. As shown in Fig. 1, our approach proceeds in three steps:

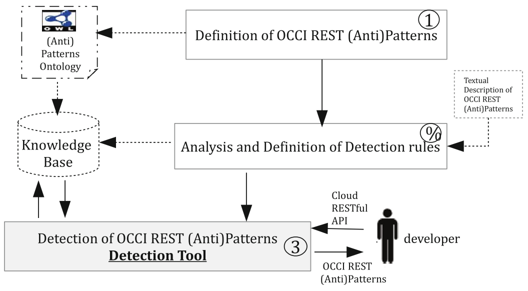
(Anti) Patterns Ontology
Definition of OCCI REST (Anti)Patterns
1
Textual Description of OCCI REST (Anti)Patterns
Knowledge Base
Analysis and Definition of Detection rules
2
Cloud RESTful API
Detection of OCCI REST (Anti)Patterns
Detection Tool
3
developer
OCCI REST (Anti)Patterns

Fig. 1. Approach overview

Step 1. Definition of OCCI REST (Anti)Patterns: This step allows to define the core ontology that we call *(Anti)Patterns Ontology*. The proposed ontology contains the relevant and necessary concepts for the detection purpose.
Step 2. Analysis and Definition of Detection Rules: This step consists of analyzing the textual description of OCCI REST(anti)patterns from the OCCI RESTful Protocol [16] to identify their relevant features. We use these relevant features to define semantic rules required for the detection of (anti)patterns. Both rules and (Anti)Patterns Ontology are then stored in a knowledge base which can be later interrogated with SQWRL queries for analysing the detection results.
Step 3. Detection of OCCI REST (Anti)Patterns: This step deals with the automatic application of detection rules defined in Step 2 using our detection tool to detect each (anti)pattern. This tool will return to the developer a set of detecting OCCI REST (anti)patterns in a given RESTful API.

3.1 Definition of OCCI REST (Anti)Patterns

In this step, we perform a domain analysis on both OCCI descriptions for Cloud resources and documentations of the RESTful API existing in the literature in order to build (Anti)Patterns Ontology, a model that provides a semantic definition of OCCI REST (anti)patterns using OWL 2 (OWL 2 Web Ontology Language) [3]. (Anti)Patterns Ontology is specified as a set of interrelated ontologies viz. *Pattern Ontology*, *Anti-Pattern Ontology*, *REST API Ontology* and *OCCI Ontology*.

Pattern Ontology: The Pattern Ontology, as depicted in Fig. 2, captures the necessary information defining an OCCI REST pattern in term of attributes that are linked to its main concept *Ptt:Pattern*. Those attributes (i.e. equivalent to data type properties in OWL language) are *Ptt:name*, *Ptt:description* and *Ptt:required* its value is a boolean that depicts whether the pattern is required or no. In addition, the *Ptt:Pattern* concept has two relationships which are *Ptt:Disjoint* and *Ptt:Concerns* and represents a range of *Rest:hasPattern*, which denotes that it can be an occurrence of the pattern either for a given API or its elements. The *Ptt:Disjoint* relationship shows the corresponding anti-pattern for

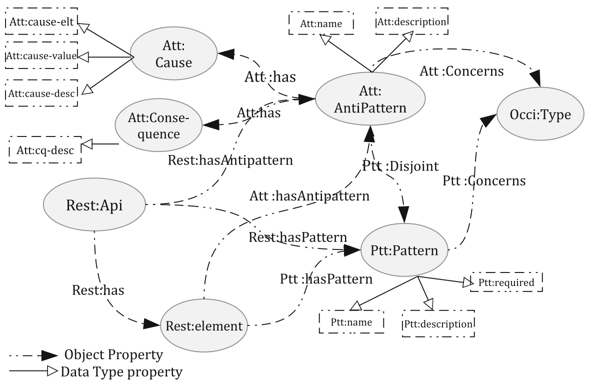

Fig. 2. (Anti)patterns ontology

a given pattern. The Ptt:Concerns relationship indicates that a pattern concerns a given OCCI Type (i.e. Resource, Mixin, etc.).

Anti-Pattern Ontology: As depicted in Fig. 2, the definition of Anti-Pattern Ontology is similar to the Pattern Ontology. However, as opposed to OCCI REST pattern, we use OCCI REST anti-pattern to capture a bad practice of such OCCI RESTful API guidelines. For that reason, we add along with the provided definition for a pattern except the attribute *Ptt:required*, the *Att:Cause* and *Att:Consequence* as concepts defining respectively the cause of OCCI RESTful anti-pattern and the consequence that may result from its occurrence. *Att:has* relationships links respectively the *Att:AntiPattern* concept with *Att:Cause* and *Att:Consequence* concepts. In that way, we provide developer enough knowledge overviewing the detected anti-pattern. *Rest:hasAntipattern* denotes that the given API or one of its elements can have an anti-pattern.

REST API Ontology: The REST API ontology aims at providing a semantic-based description of the structural and functional characteristics of the OCCI RESTful API. OCCI RESTful API follows a RESTful API design, meaning that applications use standard HTTP methods to retrieve and manipulate OCCI resources. To define this ontology, we exploit the various documentations of the RESTful API while taking OCCI RESTful API specificities into account. The main concept is *Rest:API* (a REST API) which is linked, as shown in Fig. 3(b), to the following three concepts: *Rest:AuthorizationProtocol* (an authorization protocol used to access the API), *Rest:Element* (an abstract type describing through its subclasses the possible components that we can find in a REST API, including request header, response header, code status, operation, URL, request, response, etc.) and *Rest:Operation* (a REST operation such as *Create a Server*)

OCCI Ontology: All RESTful API operations are applied on OCCI types (i.e. Category, Resource, etc.) that are already defined both in OCCI Core [17] and OCCI infrastructure [14]. To allow such capability, we define OCCI ontology

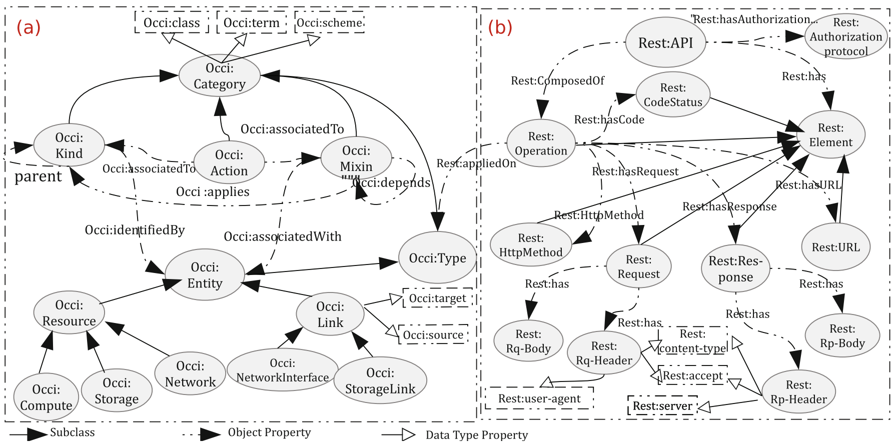

Fig. 3. (a) OCCI ontology; (b) REST API ontology

that provides a semantic description of the real-world resources abstraction provided in these two specifications while taking the OCCI rendering syntax of those resources [5] into account. The Fig. 3(a) gives an overview of the OCCI Ontology. The heart of the OCCI Ontology is the *Occi:Resource* concept which has three sub-concepts *Occi:Storage*, *Occi:Compute* and *Occi:Network*. In such way, a Resource can be a virtual machine, a virtual switch, etc. *Occi:Resource* is complemented by the *Occi:Link* concept which associates one resource instance with another. The Link type can be a *Storage Link* or *Network Interface* and contains a number of common attributes (e.g. *Occi:source*, *Occi:target*). Both *Occi:Resource* and *Occi:Link* inherit the *Occi:Entity* concept. The *Occi:Kind* is the core of the classification system built into the OCCI Core Model. *Occi:Kind* is a specialization of *Occi:Category* and introduces additional capabilities in terms of actions. Occi:Action identifies an invocable operation applicable to an entity instance. The last type defined by the OCCI Core Model is the Occi:Mixin type. An instance of Mixin can be associated with an entity instance to mix-in additional capabilities at run-time [17].

3.2 Analysis and Definition of Detection Rules

In this step, we analyses the textual definitions of the (anti)patterns listed in Tables 1, 2, and 3 to identify their relevant features. These features will then be used to define the semantic rules required to detect (anti)patterns. We rely on SWRL language to properly define these rules. A SWRL rule consists of two parts which are called the *Antecedent* and *Consequent*. An *Antecedent* represents a conjunction of one or more atom aiming together to define the conditions that must be met. Whereas, *the Consequent* specifies the fact that may be resulted in case of fulfillment of the conditions defined in *Antecedent*. SWRL rules can contain also SWRL built-ins (e.g. swrlb:matches) or SQWRL queries

Table 1. Management related (anti)patterns.

1.Query Interface Support vs Missing Query Interface
Description: To be compliant with OCCI, Query interface must be implemented. It allows the client to discover all capabilities he refers to [16]. It represents three operations applied on Actions, Kind, and Mixin, including retrieval of all registered Kinds, Actions and Mixins (HTTP verb GET must be used), adding (HTTP verb POST must be used) and removing a Mixin(HTTP verb DELETE must be used). It must be found at the path /-/ on the root of the implementation. The poor practice of query interface or the no support of it leads to the *Missing query interface anti-pattern*
2. Compliant Create vs. Non-Compliant Create
Description: The Create operation that can be applied to whatever entity (i.e. Mixin, Resource) should be compliant with the OCCI RESTful Protocol. This compliance requires the respect of the creation guidelines regarding the type of the entity that may be created. These constraints are summarized as follows:
- To create a Mixin definition, the HTTP verb POST must be used and HTTP Category term, scheme and location must be supported in this definition
- To create a Resource instance, the HTTP verb POST or PUT must be used and the request must contain one and only one HTTP Category rendering which refers to a specific Kind instance defining the type of a resource instance
The poor practice of one of those guidelines leads to *the Non-Compliant Create anti-pattern*
3. Compliant Update vs. Non-Compliant Update
Description: The Update operation, which can be applied to entities (i.e. Mixin, Resource) should be compliant with the OCCI RESTful Protocol. This compliance requires the respect of the update guidelines regarding the type of the entity that may be updated. These guidelines are summarized as follows:
- To full update a Mixin, HTTP verb PUT must be used and all URIs which are part of the collection must be provided along with the request
- To partial update a Resource, HTTP verb POST must be used and the information which are updated must be provided along with the request
- To full update a Resource, HTTP verb Put must be used in the request
The poor practice of one of those guidelines leads to *the Non-Compliant Update anti-pattern*.
4. Compliant Delete vs. Non-Compliant Delete
Description: The Delete (or Remove) operation that can be applied either to Mixin or Resource, should be compliant with the OCCI RESTful Protocol. This compliance requires the respect of the Delete guidelines regarding the type of the entity that may be deleted. These guidelines are summarized as follows:
- To remove a Mixin definition, HTTP verb DELETE must be used and the information about which Mixin should be deleted must be provided
- To delete all Resources below a given path or only one, i.e. The HTTP verb DELETE must be used and no other information should be added to the request
The poor practice of one of those guidelines leads to *the Non-Compliant Delete anti-pattern*
5. Compliant Retrieve vs. Non-Compliant Retrieve
Description: The Retrieve operation, which can be applied to Resource or Link, should be compliant with the OCCI RESTful Protocol. This compliance requires the respect of the Retrieve guidelines regarding the type of the entity that may be retrieved [16]. These guidelines are summarized as follows:
- To Retrieve a Resource or Link instance, the HTTP verb GET must be used in the request and the server response must return at least the HTTP Category which defines the Kind of the Resource or Link and associated attributes
- To Retrieve all Resources belonging to Mixin or Kind, HTTP verb GET must be used and a list containing all resource instances which belong to the requested Mixin or Kind must be returned
The poor practice of one of those guidelines leads to *the Non-Compliant Retrieve anti-pattern*
6. Compliant Trigger Action vs. Non-Compliant Trigger Action
Description: The Trigger action that can be applied to a Resource should be compliant with the OCCI RESTful Protocol, i.e. HTTP verb POST must be used and a query exposing the term of the Action must be added to the URI. Additionally, the HTTP Category defining the Action must be also provided [16]. The poor practice of one of those guidelines leads to the *Non-Compliant Trigger Action anti-pattern*

(e.g. sqwrl:select). SWRL built-ins are user-defined predicates, including basic mathematical operators and functions for string manipulations. SQWRL queries define a set of operators that can be seen as SQL-like operations used to exploit the knowledge inferred by SWRL rules.

Table 2. Cloud structure (anti)patterns.

1. Compliant Link between Resources vs. Non-Compliant Link between Resources
Description: To create a Link between two resources, the HTTP POST verb must be used and its kind as well as a "source" and "target" attributes must be provided. The non existence of one of them leads to the *Non-Compliant Link between Resources anti-pattern*
2. Compliant Association of resource(s) with Mixin vs. Non-compliant Association of resource(s) with Mixin
Description: Association of resource(s) with a Mixin should be compliant, i.e. HTTP POST verb must be used and the URIs which uniquely define the resources must be provided in the request. The poor practice of one of those guidelines leads to *the Non-compliant Association of resource(s) with Mixin anti-pattern*
3. Compliant Dissociation of resource(s) From Mixin vs. Non-compliant dissociation of resource(s) From Mixin
Description: Dissociation of resource(s) from a Mixin should be compliant, i.e. HTTP DELETE verb must be used and the URIs which uniquely define the resources must be provided in the request. The poor practice of one of those guidelines leads to *the Non-compliant Dissociation of resource(s) From Mixin anti-pattern*

Table 3. REST related (anti)patterns

1. Compliant URL vs. Non-Compliant URL
Description: A URL Path should be Compliant, i.e. Whenever the URL Path is rendered it must be either a String or as defined in RFC6570 [10]. The *non-Compliant URL anti-pattern* occurs as the consequence of the poor practice of such guidelines
2. Compliant Request Header vs. Non-Compliant Request Header
Description:A Request Header can be considered compliant, i.e. client (e.g. OCCI client) should specify the media types its implementation data formats (e.g. OCCI Data formats) support in the Accept header and the implementation (e.g. OCCI version) version number in the User-Agent header and must specify the media type its implementation data format (e.g. OCCI data format) support in the Content-type header [16]. The poor practices of those guidelines leads to the *Non-Compliant Request Header anti-patern*
3. Compliant Response Header vs. Non- Compliant Response Header
Description:A Response Header can be considered compliant, i.e. a server (e.g. OCCI server) should specify the media types its implementation data formats (e.g. OCCI Data formats) support in the Accept header, and must specify the media type its implementation data format (e.g. OCCI data format) used in an HTTP response in Content-type header and the implementation (e.g. OCCI version) version number in the Server header [16]. The poor practices of all guidelines leads to the *Non-Compliant Response Header anti-patern*

We distinguish three categories of (anti)patterns, two mainly focus on the Cloud service aspects and one addresses general aspects of the REST services according to OCCI perspective: Management Related (Anti)Patterns, Cloud Structure Related (Aanti)Patterns, and REST Related (Anti)Patterns. In the following, we define these categories, while explaining SWRL rules for detecting an example of (anti)pattern under each one.

Management Related (Anti)Patterns: They represent the poor and best practices in the main management operations applied on Cloud resources and services, with respect to OCCI perspective (see Table 1). We identify 6 patterns and its opposite anti-patterns respectively in Query interface, Create, Retrieve, Update, Delete operations and in Trigger actions. Figure 4 illustrates the SWRL rules we define for the *Query interface support pattern* and the *Missing query interface anti-pattern*. The SWRL rule for *Query interface support Pattern* aims to evaluate for each operation both the value of an URL and the used verb of an HTTP Method. We report that the API has this pattern if we find at least an URL value match"\- \" path and the verb of HTTP method is one of the common HTTP verbs. swrlb:matches(?urlval, "\-\") consists to check whether the URL value equal to"\-\". detection: matches (?verb, "POST", "PUT", "GET", "DELETE") is our custom built-in we create in order to check whether the verb of the HTTP method (i.e. ?verb) contains one of the common HTTP verb. The mechanism to extend the SWRL language in order to define new built-ins is detailed in our project site (http://www-inf.it-sudparis.eu/SIMBAD/tools/ORAP-DT/).

Figure 5 illustrates a partial instantiation of the (Anti)Pattern Ontology with knowledge extracted from the REST operation (GET /-/ HTTP/1.1: that is means what a Cloud provider can be provisioned) existing in an OpenStack RESTful API. After executing the above SWRL rule, the object property "*Rest: hasPattern*" in red color was added between the *Rest:OS-Query* (an instance of REST:Operation concept) and *Ptt:Query-interface-support* (an instance of Ptt:Pattern concept). Conversely, we report an occurrence of *Missing query interface Anti-pattern* if we haven't found the "\- \" path among all possible URLs existing in an API. This is carried out through sqwrl:makeSet(?s1,?urlset) that makes any URL its value equal to "\- \" in a set ?urlset and the built-in sqwrl:isEmpty(?urlset) that ensures that the resulted set is empty.

1.Rest:API(?ap) ^ Rest:ComposedOf(?ap,?operation) ^ Rest:hasURL(?operation, ?url) ^ Rest:value(?url, ?urlval) ^ swrlb:matches(?urlval,"\-\") ^ Rest:hasHttpMethod(?operation, ?httpmethod) ^ Rest:verb(?httpmethod, ?verb) ^ detection:matches(?verb, "POST", "PUT", "GET", "DELETE")->Rest:hasPattern(?ap, Ptt:Query_interface_support) **(a) Query interface support Pattern**	*1.Rest:API(?ap) ^ Rest:ComposedOf(?ap, ?operation) ^ Rest:hasURL(?operation,?url) ^ Rest:value(?url, "\-*) ^ sqwrl:makeSet(?s1, ?url) ^ sqwrl:isEmpty(?s1) ./ Rest:hasAntiPattern(?ap, Att:Missing_query_interface)* **(b) Missing query interface Anti-pattern**

Fig. 4. SWRL rule for query interface (anti)pattern

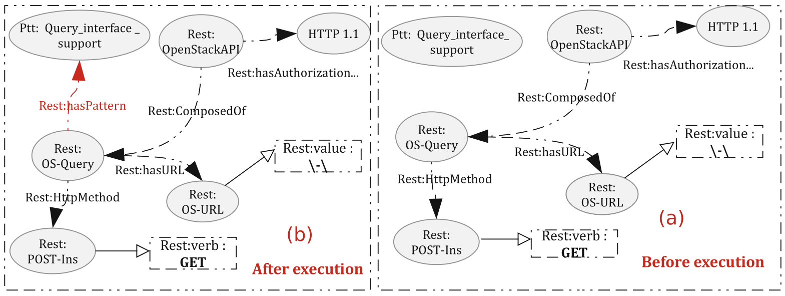

Fig. 5. SWRL rule for *Query interface support Pattern* (a) before execution and (b) after execution (Color figure online)

Cloud Structure Related (Anti)Patterns: They represent the poor and best practices to link Cloud resources between each others as well as to create a collection of resources using a Mixin, with respect to OCCI perspective (see Table 2). We identify 3 patterns and their opposite anti-patterns respectively in the creation of the Link between two Cloud resources, Association of resource(s) with a Mixin and Dissociation of resource(s) from a Mixin.

Figure 6 illustrates the SWRL rule that we define for the *Compliant Link between Resources Pattern.* The latter aims to evaluate for each operation applied on the link type Occi:Link(?link) the used verb in the HTTP Method, the Kind that identifies the link type and whether the link type contains both the source and target attributes. We report that the given operation Rest:Operation(?op) has the *Compliant Link between Resources Pattern* if we ensure firstly, that the verb of the HTTP method is "POST" using (swrlb:matches(?verb, "POST")), secondly the kind type (?kind) has ethier "Network Interface" or "Storage Link" as value for its attribute *?term* using (detetion:matches(?term, "Storage link", "Network Interface"), non-empty value for its attribute ?Occi:scheme(?kind, ?scheme) and a "kind" value for its attribute ?class and finally if the link type contains non-empty values for both source and target attributes. Conversely, we report an occurrence of *Non-Compliant Link between Resources Anti-pattern* if we detect poor practice of one of those constraints. The SWRL rules for this anti-pattern as well as an instantiation of the (Anti)Pattern Ontology showing the usefulness of those rules are available on our project Web site.

1. Rest:Operation(?op) ^ Rest:hasHttpMethod(?op, ?httpmd) ^ Rest:verb(?httpmd, ?verb) ^ swrlb:matches(?verb, "POST") ^ Rest:hasRequest(?op,?req)^ Rest:has(?req,?reqbody) ^ Rest:hasParameterDefinition(?reqbody, ?pradef) ^ Occi:Link(?link) ^ Rest:isComposedOf(?pradef, ?link) ^ Occi:identifiedBy(?link, ?kind) ^ Occi:term(?kind, ?term) ^ detetion:matches(?term, "Storage link", "Network Interface")^ Occi:scheme(?kind, ?schee) ^ Occi:class(?kind, "kind") ^ Occi:source(?link, ?source) ^ Occi:target(?link, ?target) -> Rest:hasPattern(?op, Ptt:Compliant_Link)

Compliant Link between Resources Pattern

Fig. 6. SWRL Rule for Compliant Link between Resources Pattern

REST Related (Anti)Patterns: They represent the poor and best practices in the main REST API components. Here, we note that some of the existing REST (anti)Patterns already defined in [8,18], can be used, particularly, Verbless URIs (respectively, CRUDy URIs) and Ignoring Status Code (respectively, Supported Status Code). Additionally, according to OCCI perspective, we identify 3 new REST (anti)patterns relating respectively to the URL, request header and response header (see Table 3). Like management related (anti)patterns and Cloud Structure (anti)patterns, the detection rules for the REST (anti)patterns are also specified through SWRL language in conjunction with SQWRL queries.

3.3 Detection of OCCI REST (Anti)Patterns

This step consists of applying the detection rules defined in step 2 to detect the possible OCCI (anti)patterns in Cloud RSTTful APIs. To do so, we have implemented a detection tool to verify those APIs and automatically detect the (anti)patterns may be occurred into them. The detection of (anti)patterns in a given Cloud RESTful API requires firstly the instantiation of the different proposed ontologies with the knowledge extracted from this API. The instantiated ontologies as well as the defined SWRL rules construct together the knowledge base that we apply in the reasoning process to detect such (anti)pattern.

4 Validation

This section discusses the validation of our approach. We have developed a proof of concept implementation detailed on our project site[3]. Then we have used real world Cloud REST APIs to conduct this evaluation. Our objective is twofold. Firstly, we aim to show the effectiveness of our approach in terms of accuracy (i.e. precision, recall and F-measure values). Then, we aim to explore whether the selected Cloud providers respect OCCI REST patterns. In the following subsections, we firstly describe our proof of concept and the used datasets. Secondly, we discuss and analyze the experiment results.

4.1 Proof of Concept and Experiment Setting

Proof of Concept. We evaluate our approach through a proof of concept which is a web-based application developed using J2EE integrating with a query and a reasoning engine developed using OWLAPI and SWRLAPI. These APIs are used to deal with SWRL rules and SQWRL queries. Our tool takes into account the semantic base knowledge provided in order to detect the possible OCCI REST (Anti)patterns. It is based on 65 SWRL rules, including 28 SWRL rules for the patterns and 37 for the anti-patterns.

[3] Description: http://www-inf.it-sudparis.eu/SIMBAD/tools/ORAP-DT/.

Experiment Setting. We built the experimental datasets by performing an analysis in the Cloud RESTful APIs of Cloud services. We choose only 6 candidates, including OpenStack, COAPS, OpenNebula, Amazon S3, Microsoft Azure and Rackspace, since its underlying REST operations are well explained. From those operations, we have collected the required knowledge in order to semantically describe each API. Then, we have involved an expert manually evaluated the REST operations in order to identify the true positives and false negatives required to compute precision, recall and F1-measure values. Precision is the ratio between the true detected (anti)patterns and all detected (anti)patterns [18]. Recall is the ratio between the true detected (anti)patterns and all existing true (anti)patterns [18]. Finally, the F1-measure represents the weighted harmonic mean of the precision and recall values.

4.2 Experiment Results Analysis

Herein, we present the detection results in all selected Cloud REST APIs, the validation of our development tool in terms of precision, recall, and F1-measure values and show whether Cloud providers respect the OCCI REST patterns by computing their compliance degrees to those patterns.

Table 4 shows detailed detection results for the 28 OCCI REST (anti)patterns on 6 Cloud RESTful APIs. The first column lists the identified (anti)patterns. The remaining columns present the analysed Cloud RESTful APIs. For each (anti)pattern in each Cloud RESTful API, we report the total number of its occurrences derived from our detection tool. The last column indicates the occurrence percentage (OP) of each (anti)pattern compared to the total number of operations that may contain such kind of (anti)pattern (i.e. the percentage of Query interface support is computed compared to all existing query operations in the selected API). As specified in Table 4, in the management related (anti)patterns category, the most frequent patterns are *Compliant Delete* and *Compliant Update* Patterns. This means that the majority of the analyzed APIs follows either explicitly or implicitly the OCCI guidelines in deleting and updating of a given Cloud resource. In contrast, the most frequent anti-patterns are *Non-Compliant Trigger Action* and *Non-Compliant Create*. A clear majority of Cloud RESTful APIs does not include the category rendering which refers to a specific Kind instance defining the type of resource instance that will be created. Likewise, for triggering an action on a resource, neither the query exposing the term of the action, nor the HTTP Category defines the action were included in the REST operation. With regard to Cloud Structure (Anti)Patterns, the most frequent pattern is the *Compliant Link between Resources*. However, this pattern is tested using a low number of operations related to a link type due to its lack in the selected Cloud RESTful APIs. Additionally, we do not report any occurrence of (anti)patterns related both to the association and dissociation of resources from Mixin. Finally, with regard to REST Related (Anti)patterns, the most frequent patterns are *Compliant URL* and *Verbless URIs*. The majority of the analysed APIs did not include any CRUDy terms or any of their synonyms and all used URIs were either string or follow the structure as defined in [10].

Table 4. Detection results of the 28 OCCI REST (anti)patterns

Cloud REST API	Open-Stack (28)	COAPS (18)	Open-Nebula (20)	Amazon S3 (56)	Microsoft Azure (119)	Rack-space (62)	O.P %
Management Related (Anti)patterns							
Query interface support (3/8)	3	0	0	0	0	0	3 %
Missing query interface (5/8)	0	1	1	1	1	1	60 %
Compliant Create (9/44)	5	1	0	1	2	0	20 %
Non-Compliant Create (35/44)	0	1	3	8	13	10	80 %
Compliant Update(34/50)	3	2	3	6	14	6	68 %
Non-Compliant Update(16/50)	0	0	0	0	16	0	32 %
Compliant Delete(41/44)	3	2	3	8	16	9	93 %
Non-Compliant Delete (3/44)	0	0	0	1	2	0	7 %
Compliant Retrieve(69/130)	10	5	5	19	30	0	54 %
Non-Compliant Retrieve(61/130)	0	0	6	2	31	22	46 %
Compliant Trigger Action(4/39)	4	0	0	0	0	0	10 %
Non-Compliant Trigger Action(35/39)	0	7	0	6	14	8	90 %
Cloud Structure (Anti)patterns							
Compliant Link between Resources(4/4)	2	0	2	0	0	0	100 %
Non-Compliant Link between Resources (0/4)	0	0	0	0	0	0	0 %
Compliant Association of resource(s) with Mixin	0	0	0	0	0	0	-
Non-Compliant Association of resource(s) with Mixin	0	0	0	0	0	0	-
Compliant Dissociation of resource(s) from Mixin	0	0	0	0	0	0	-
Non-Compliant Dissociation of resource(s) from Mixin	0	0	0	0	0	0	-
REST Related (Anti)patterns							
Compliant URL(303/303)	28	18	20	56	119	62	100 %
Non-Compliant URL (0/303)	0	0	0	0	0	0	0 %
Compliant Request Header(62/303)	0	0	0	0	0	62	20 %
Non-Compliant Request Header(241/303)	28	18	20	56	119	0	80 %
Compliant Response Header (118/303)	0	0	0	56	0	62	39 %
Non-Compliant Response Header(185/303)	28	18	20	0	119	0	61 %
Supported Status Code(303/303)	28	18	20	56	119	62	100 %
Ignoring Status Code(0/303)	0	0	0	0	0	0	0 %
Verbless URIs(297/303)	28	15	20	54	119	62	98 %
CRUDy URIs(6/303)	0	3	0	2	1	0	2 %

Table 5 shows the validation results of our detection tool on OpenStack, Microsoft and Rackspace RESTful APIs. The first column lists the identified (anti)patterns. The remaining columns list the three selected APIs for the validation followed by precision, recall and F1-measure values. For each (anti)pattern in each Cloud RESTful API, we report precision, recall and F1-measure values for the detection results. The last two rows show the average and total average

Table 5. Complete validation results on Openstack, Microsoft and Rackspace REST APIs

(Anti)Patterns	OpenStack			Microsoft Azure			Rackspace		
	P	R	F1	P	R	F1	P	R	F1
Query interface support	100 %	100 %	100 %	-	-	-	-	-	-
Missing query interface	-	-	-	100 %	100 %	100 %	100 %	100 %	100 %
Compliant Create	100 %	100 %	100 %	-	-	-	-	-	-
Non-Compliant Create	-	-	-	86 %	100 %	92 %	100 %	100 %	100 %
Compliant Update	100 %	100 %	100 %	87 %	100 %	93 %	100 %	100 %	100 %
Non-Compliant Update	-	-	-	94 %	100 %	96 %	-	-	-
Compliant Delete	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %
Non-Compliant Delete	-	-	-	66 %	100	79 %	-	-	-
Compliant Retrieve	100 %	100 %	100 %	96 %	100	98 %	-	-	-
Non-Compliant Retrieve	-	-	-	100 %	100 %	100 %	100 %	100 %	100 %
Compliant Trigger Action	100 %	100 %	100 %	-	-	-	-	-	-
Non-Compliant Trigger Action	-	-	-	100 %	100 %	100 %	100 %	100 %	100 %
Compliant Link between Resources	100 %	100 %	100 %	-	-	-	-	-	-
Non-Compliant Link between Resources	-	-	-	-	-	-	-	-	-
Compliant Association of resource(s) with Mixin	-	-	-	-	-	-	-	-	-
Non-Compliant Association of resource(s) with Mixin	-	-	-	-	-	-	-	-	-
Compliant dissociation of resource(s) from Mixin	-	-	-	-	-	-	-	-	-
Non-Compliant dissociation of resource(s) from Mixin	-	-	-	-	-	-	-	-	-
Compliant URL	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %
Non-Compliant URL	-	-	-	-	-	-	-	-	-
Compliant Request Header	-	-	-	-	-	-	100 %	100 %	100 %
Non-Compliant Request Header	100 %	100 %	100 %	100 %	100 %	100 %	-	-	-
Compliant Response Header	-	-	-	-	-	-	100 %	100 %	100 %
Non-Compliant Response Header	100 %	100 %	100 %	100 %	100 %	100 %	-	-	-
Supported Status Code	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %
Ignoring Status Code	-	-	-	-	-	-	-	-	-
Verbless URIs	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %
CRUDy URIs	-	-	-	100 %	100 %	100 %		-	-
Average	**100 %**	**100 %**	**100 %**	**95 %**	**100 %**	**97 %**	**100 %**	**100 %**	**100 %**
Total Average	**Precision = 98 %**			**Recall = 100 %**			**F1-measure = 99 %**		

results of those values respectively. For the Openstack RESTful API which is an OCCI-based API, we obtained the best results of the detection with a precision of 100 %, signifying that all the detected (anti)patterns are in the list that the expert determined manually. The recall in this sample is 100 %, signifying that all (anti)patterns, that may occur, have been successfully detected by our tool. The average of these values obtained for this API is 100 % for the precision, 100 % for the recall and 100 % for the F1-measure. We report the same values for the Rackspace RESFful API. However, we obtain the detection results for

Microsoft API with average values that are lower than the previous ones, viz. a precision of 95 %, a recall of 100 %, and an F1-measure of 97 %.

Finally, we aim at showing whether the Cloud providers respect the OCCI REST patterns. To do so, we have computed for each Cloud RESTful API its compliance degree. The compliance degree indicates the percentage of OCCI REST patterns that each API has over all its operations. The compliance degree is defined as follows:

$$\textbf{Compliance degree} = \frac{1}{14} * \sum_{i=1}^{14} \left(\frac{\sum P_i}{\sum OP_P i}\right)$$

where P_i is a pattern (e.g. P_1 denotes the Query interface support pattern), 14 is the number of patterns, $\sum OP_P i$ is the total number of operations that may contain the pattern P_i (e.g. three operations that may contain the Query interface support pattern in OpenStack RESTful API). As shows in Fig. 7, Openstack RESTful API represents the most compliant API with OCCI patterns. This is not surprisingly because this API is already based on OCCI standard. Additionally, Rackspace as well as Amason S3 has acceptable compliance degrees, this signifying that over 47 % of operations in those APIs follow implicitly the OCCI standard. In contrast, although OpenNebula and COAPS RESTful APIs had already based on OCCI, it seems that they did not carefully follow all OCCI guidelines. Microsoft Azure RESTful API uses its own model to describe Cloud resources, thus explaining the poor compliance degree it has.

Summarizing up, our detection approach performs better when dealing with Cloud RESTful APIs have based on OCCI standard than with non OCCI-based APIs. This is mainly due to the fact that our ontology defines the RESTful API based on OCCI descriptions. However, we can resolve this limitation by adding some semantic equivalence relations between the terminologies used to describe Cloud resources according to OCCI perspective and those according to a specific Cloud provider. Our detection tool has achieved a total average precision value of 98 %, a recall value of 100 % and an F1-measure of 99 % in detecting of 28 (anti)patterns on three Cloud RESTful APIs. Moreover, the obtained compliance degrees for the selected Cloud RESTful API shows their handful support of the OCCI REST patterns.

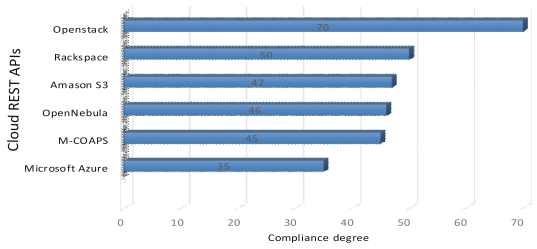

Fig. 7. OCCI Compliance degrees of Cloud RESTful APIs

5 Conclusion and Future Work

This paper identifies from OCCI Standard a set of (anti)patterns related to Cloud management APIs. It proposes a Semantic-based detection approach of OCCI REST (Anti)patterns in Cloud RESTful APIs. To validate this approach, we conducted an evaluation by analyzing 28 OCCI REST (anti)patterns on a real world Cloud RESTful APIs that invoking 303 operations. We showed that the proposed approach allows the detection of the OCCI REST (anti)patterns with good results in terms of precision, recall and F1-measure. We observed also through the obtained compliance degrees that there is no widespread adoption of the OCCI patterns in the selected Cloud RESTful APIs.

At short term, we want to apply our approach on other Cloud RESTful APIs for better understanding the OCCI REST patterns in the Cloud and their applications. At long term, we will propose our management API based on those identified patterns while avoiding their anti-patterns in order to resolve the heterogeneity and evolution problem of the managed Cloud resources.

References

1. Open Cloud Computing Interface. http://occi-wg.org/
2. OCCI Compliance Testing Tool (2011). http://occi-wg.org/2011/01/18/occi-compliance-testing-tool/
3. OWL 2 Web Ontology Language Document Overview, 2nd edn. (2012). https://www.w3.org/TR/owl2-overview/
4. Dudney, B., Asbury, S., Krozak, J.K., Wittkopf, K.: J2EE An-tiPatterns. Wiley, Hoboken (2003)
5. Edmonds, A., Metsch, T.: Open cloud computing interface - text rendering. Technical report, Open Grid Forum (2016)
6. Edmonds, A., Metsch, T., Papaspyrou, A., Richardson, A.: Toward an open cloud standard. IEEE Internet Comput. **16**(4), 15–25 (2012)
7. Fourati, R., Bouassida, N., Abdallah, H.B.: A metric-based approach for anti-pattern detection in UML designs. In: Lee, R. (ed.) Computer and Information Science 2011. Studies in Computational Intelligence, vol. 364, pp. 17–33. Springer, Heidelberg (2011)
8. Palma, F., Dubois, J., Moha, N., Guéhéneuc, Y.-G.: Detection of REST patterns and antipatterns: a heuristics-based approach. In: Franch, X., Ghose, A.K., Lewis, G.A., Bhiri, S. (eds.) ICSOC 2014. LNCS, vol. 8831, pp. 230–244. Springer, Heidelberg (2014)
9. Garcia, A.L., del Castillo, E.F., Fernandez, P.O.: ooi: Openstack occi interface. SoftwareX (2016, in press)
10. Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., Orchard, D.: URI Template. RFC 6570,423 Internet Engineering Task Force (2012). http://www.ietf.org/rfc/rfc6570.txt
11. Kessentini, M., Vaucher, S., Sahraoui, H.: Deviance from perfection is a better criterion than closeness to evil when identifying risky code. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, ASE 2010, pp. 113–122 (2010)

12. Lewis, G.A.: The role of standards in cloud-computing interoperability. Technical report, Software Engineering Institute, Carnegie Mellon University (2012)
13. Martino, B.D., Esposito, A., Cretella, G.: Semantic representation of cloud patterns and services with automated reasoning to support cloud application portability. IEEE Trans. Cloud Comput. **PP**(99), 1 (2015). doi:10.1109/TCC.2015.2433259
14. Metsch, T., Edmonds, A.: Open cloud computing interface - infrastructure. Technical report, Open Grid Forum (2016)
15. Moha, N., Palma, F., Nayrolles, M., Conseil, B.J., Guéhéneuc, Y.-G., Baudry, B., Jézéquel, J.-M.: Specification and detection of SOA antipatterns. In: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (eds.) Service Oriented Computing. LNCS, vol. 7636, pp. 1–16. Springer, Heidelberg (2012)
16. Nyren, R., Edmonds, A., Metsch, T., Parak, B.: Open cloud computing interface - http protocol. Technical report, Open Grid Forum (2016)
17. Nyren, R., Papaspyrou, A., Metsch, T., Parak, B.: Open cloud computing interface -core. Technical report, Open Grid Forum (2016)
18. Palma, F., Gonzalez-Huerta, J., Moha, N., Gueheneuc, Y.G., Guy, T.: Are RESTful APIs well-designed? Detection of their linguistic (anti)patterns. In: Barros, A., Grigori, D., Narendra, N.C., Dam, H.K. (eds.) Service-Oriented Computing. Lecture Notes in Computer Science, pp. 171–187. Springer, Heidelberg (2015)
19. Palma, F., Moha, N., Tremblay, G., Guéhéneuc, Y.-G.: Specification and detection of SOA antipatterns in web services. In: Avgeriou, P., Zdun, U. (eds.) ECSA 2014. LNCS, vol. 8627, pp. 58–73. Springer, Heidelberg (2014)
20. Settas, D.L., Meditskos, G., Stamelos, I.G., Bassiliades, N.: SPARSE: a symptom-based antipattern retrieval knowledge-based system using semantic web technologies. Expert Syst. Appl. **38**(6), 7633–7646 (2011)