

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/236891408>


A Framework for Self-descriptive RESTful Services

Conference Paper · May 2013
DOI: 10.1145/2487788.2488183

CITATIONS
10

READS
111

2 authors:



Luca Panziera
The Open University (UK)
13 PUBLICATIONS **61** CITATIONS


SEE PROFILE




Flavio De Paoli
Università degli Studi di Milano-Bicocca
122 PUBLICATIONS **946** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

- 

EW-Shopp: Supporting Event and Weather-based Data Analytics and Marketing along the Shopper Journey [View project](#)
- 

ESOCC 2012 [View project](#)

A Framework for Self-descriptive RESTful Services

Luca Panziera
University of Milan - Bicocca
Department of Computer Science, Systems and
Communication (DISCo)
Viale Sarca 336/14, 20126
Milan, Italy
panziera@disco.unimib.it

Flavio De Paoli
University of Milan - Bicocca
Department of Computer Science, Systems and
Communication (DISCo)
Viale Sarca 336/14, 20126
Milan, Italy
depaoli@disco.unimib.it

ABSTRACT

REST principles define services as resources that can be manipulated by a set of well-known methods. The same approach is suitable to define service descriptions as resources. In this paper, we try to unify the two concepts (services and their descriptions) by proposing a set of best practices to build self-descriptive RESTful services accessible by both humans and machines. Moreover, to make those practices usable with little manual effort, we provide a software framework that extracts compliant descriptions from documents published on the Web, and makes them available to clients as resources.

Categories and Subject Descriptors

H.3.5 [Information Storage and Retrieval]: Online Information Services—*Web-based services*; D.2.11 [Software Engineering]: Software Architectures—*Languages (e.g., description, interconnection, definition)*

Keywords

Web services; REST; Service description; Information extraction; Semantic Web

1. INTRODUCTION

The number of Web APIs that more or less follow the principles of REST increased dramatically in recent years [14]. The reason for this wide adoption can be seen in the direct use of HTTP [6], the native protocol for the Web, which makes RESTful services more Web-oriented than classical Web Services, which exploit SOAP as additional protocol [23, 28]. Moreover, the well-known semantics of HTTP methods and the use of URIs as endpoints to identify and access services as resources reduce the learning curve compared to WSDL-based services that define dedicated interfaces. However, REST principles include HATEOAS (*hypermedia as the engine of application state*) as control mechanism, which is often misused or neglected by available services.

A limit of the RESTful style is that the issue of describing services is not explicitly addressed. The definition of rich descriptions including non-functional properties (NFPs) is crucial to discover services that meets given requirements [22]. Examples of NFPs and their relevance are: licensing restrictions on data returned by a service is to be considered

to prevent illegal use; service usage limits, such as number of requests per day, should be specified to tell Web developers the correct use of a service; Quality of Service (QoS) properties should be advertised to support a correct design of composite services.

In the literature, some standards, such as WADL [9] and WSDL 2.0 [3], and Semantic Web models for RESTful services, such as hRESTS/MicroWSMO [11] and SA-REST [7], have been proposed to enable for rich descriptions, but very few descriptions compliant with these proposals are actually available on the Web. Moreover, few descriptions compliant to these models are available as resources according to the REST principles.

Currently, information about services is dispersed over several Web sources. Providers publish descriptions of their own services as HTML pages in which the information is published in natural language. More structured descriptions are available in Web API repositories, such as *ProgrammableWeb*¹, or specialized wikis for developers. In addition, third-party monitoring websites, such as *API status*², provide information about response time and availability of popular services. All information is provided in natural language, hence suitable for humans, but more difficult to understand by machine. Moreover, without tools, the manual discovery and collection of dispersed information might become very difficult.

The aim of this paper is to discuss a set of best practices and propose a framework to collect information on the Web to semi-automatically generate complete and up-to-date descriptions of services as RESTful services. Information on services are extracted from provider's documentation and trustworthy third-party sources. The resulting descriptions are published as resources associated with services to make them self-descriptive. The final goal is to provide descriptions that support service discovery by humans as well as by automatic tools. The proposal is based on PCM-lite, a meta-model for describing functional and non-functional service properties, and a REST-compliant interaction protocol to manage (accessing and modify) service descriptions.

The paper is structured as follows. In section 2, the best practices to make a RESTful service self-descriptive are provided. Then, the framework architecture and the techniques adopted to extract service information is described in section 3. Section 4 discusses the lesson learned by implementing self-descriptive services in practices through the proposed framework. Related work is discussed in section 5. Finally, we draw conclusions and discuss future work in section 6.

¹<http://www.programmableweb.com>

²<http://api-status.com/>

2. DESCRIPTIONS AS A SERVICE

The discovery of RESTful services that offer public functionalities on the Web is an important issue to be addressed. Despite the availability of several standard languages and models, information about services is dispersed and heterogeneous. Service providers publish documentation as common HTML Web pages. Also third-party Web sources, such as wikis, Web APIs repositories, and monitoring services provide additional descriptions as HTML or semi-structured documents, such as XML and JSON. To give an example, descriptions of *Twitter* REST APIs are available through provider documentation web pages³, in the *ProgrammableWeb* repository, as HTML page⁴ and Atom feeds provided according to REST⁵, and through *API status* monitoring service as JSON or XML document that contains real-time service performances. These documents provide information about service functionalities, as well as non-functional characteristics, such as data licensing, usage limits and response time, that need to be described to support users to choose among a number of similar services that are currently available on the Web (e.g., geolocation and mapping services).

Information is mostly in natural language that can be easily interpreted by humans, but cannot be easily evaluated by machines. Even if the information is provided by semi-structured documents there are problems due to: (i) heterogeneous data formats; (ii) different vocabularies that prevent common tools to identify synonyms and homonyms; and (iii) adoption of models that are not compliant with service description standards, such as WSDL 2.0 or WADL. Moreover, the manual discovery of such a dispersed knowledge through standard Web search engines might be prohibitive.

To address these issues, we introduce the concept of self-descriptive RESTful service, a service that represents itself according to REST principles, to enable effective discovery by humans and machines. A self-descriptive RESTful service offers a unique URI which identifies a root resource as a starting point for discovering all its resources by exploiting HATEOAS principle. To reach the goal we propose five best practices for representing functional and non-functional characteristics of a service, and provide them through browsable descriptions.

Best Practice 1: Information modelling

RESTful services need to be described according to a shared and formal model that specifies functional and non-functional properties.

The definition of a sound model for representing service properties is fundamental for allowing machines and humans to understand and manipulate descriptions exploiting the same syntax and semantics.

Best Practice 2: Semantic data model

Data that describes service properties need to be represented according to RDF.

RESTful services can return representations in any format, which is an advantage for developers and providers, because they can adopt the most suitable format according to specific domain, context and needs. However, this heterogeneity turns to be a disadvantage for machines and users, because it

reduces interoperability in service composition. This aspect becomes crucial when dealing with service discovery, since descriptions need to be compared and composed to evaluate and match a service with a user request. If descriptions are expressed in RDF, both syntax and semantic interoperability can be addressed. The former by adopting a shared semantic data model, the latter by enabling the use of Semantic Web tools, such as reasoners, which can make inference, and matchmakers [24], which are able to compare descriptions referring to different domain ontologies. It should be noted that we propose RDF as a constraint for representing service properties, but not for resources that implement service functionalities.

Best Practice 3: Common vocabulary

Property values should represent concepts that are linked to concepts available on the Linking Open Data Cloud.

In recent years, we have witnessed the spread of the phenomenon known as the *Linking Open Data Cloud*. The Linking Open Data Cloud is the result of the interconnection between datasets published according to *Linked Data* principles [10], which is a set of best practices that share some common characteristics with REST, namely: (i) identify concepts by URI, (ii) use HTTP to look up concepts descriptions and (iii) include links to other URIs in documents in order to discover other concepts. The latter is a way to implement the HATEOAS principle (see Best Practice 5) [18]. The most popular Linked Data datasets are DBpedia [2] and Yago [25], which are the semantic representation of, respectively, Wikipedia and Wordnet [5]. Therefore, they refer to general-purpose ontologies to represent a vast portion of human knowledge. These two datasets are central hubs for hundreds of other datasets that define concepts for specific domains (geography, music, etc.). If property values are defined as concepts available on the Linking Open Data Cloud, they represent well-known concepts, which means preventing ambiguity, synonymy and homonymy, and thus facilitating evaluations by automatic tools. Relations between datasets, defined as links, allow users and machines to discover additional information related to concepts that represent property values.

Best Practice 4: Human interpretability

A natural language description, or label, must be associated with each service property.

Property definitions in pure RDF, without natural language descriptions, allow machine to manage information, but reduce human readability. To address this issue, a `rdfs:label` attribute or a `rdfs:comment` attribute should be specified for each service property.

Best Practice 5: RESTful descriptions

Service must provide descriptions as aggregations of properties published as RESTful resources.

Descriptions can be designed and implemented as special resources, which means to reduce the difference between a service and the associated descriptions. The implementation of the REST architectural style via HTTP methods enables the use of the same models and tools to manage both services and descriptive data, with a clear advantage in simplicity and interoperability. In particular, the use of Linked Data in combination with RDF provides a standard format for

³<https://dev.twitter.com/docs/api/1.1>

⁴<http://www.programmableweb.com/api/twitter>

⁵<http://api.programmableweb.com/>

hyperlinks to support the HATEOAS principle. The rest of this section discusses some descriptions samples.

2.1 RESTful descriptions

According to the first practice, we need to adopt a model and a language to express properties and descriptions. A good candidate is the Policy Centred Meta-model (PCM) [4], which was designed by our research group to support semantic descriptions of NFPs. The high expressiveness of the PCM allows service providers to define NFPs that refer to complex business domains. However, PCM flexibility and completeness have requested the definition of several concepts that makes it difficult to master all the features to deliver complete descriptions, which, in addition, require large computational resources to be processed by reasoners [16, 21, 19].

In this paper, we introduce PCM-lite, whose features are: (i) full support to properties (both functional and non-functional); (ii) expressive, but simpler than PCM; and (iii) data format independence.

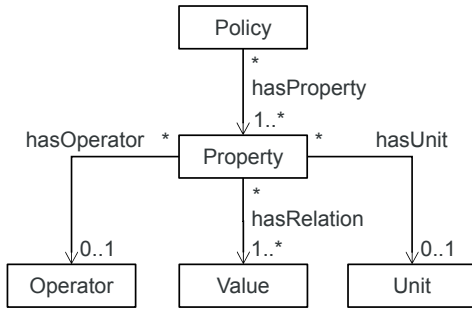


Figure 1: PCM-lite formalization

The semantics of the PCM-lite is shown in figure 1 as UML class diagram. The lightweight meta-model has the following characteristics. Each service can be described by one or more *policies* that collects a set of *properties* into a single entity. *Properties* represent either functional or non-functional characteristics. We do not make an explicit distinction because this classification is not explicitly defined in literature [4], and may change according to subjective evaluations or usage context. Anyway, for the purposes of the model, this classification is unnecessary.

A *property* can have a *relation* with one or more values. The *relation* can be a generic (e.g., “data format *has* value JSON”), refers to specific resource feature (e.g., “resource *has* method GET and PUT”) or customized by users. Moreover, optional unit can be associated with *properties* for defining numeric properties (e.g., “usage limit: 500 requests per day”). Finally, a operator that can be a logic quantifier (\forall , \exists), for a set of symbolic values, or defines single numeric values, with $=$, unbounded intervals, with \leq and \geq , or bounded intervals with *range* (e.g., “response time: \leq 200 ms”). Finally, PCM-lite is defined by an abstract model that can be implemented in different languages, such as RDF, XML and JSON. In this paper we adopt RDF.

For the lack of space, we illustrate PCM-lite by means of some examples. The next listings show policies and properties *as a resource* describing the *Foo* service by RDF documents

expressed in N-Triples [8]⁶. These descriptions should be returned by the *Foo* REST API if the service will be self-descriptive according to the proposed best practices and resemble the information available on the its real documentation.

Listing 1 shows a policy composed of four properties. Two of them are resources, which provides main service functionalities, and two NFPs, namely *data license* and *response time*, which describe the service. To satisfy the fourth practice, label and description in natural language are provided. According to the fifth practice, this policy is identified by the URI <https://api.Foo.com/twPolicy>.

Listing 1: A PCM-lite policy for *Foo* REST APIs

```

@prefix : <https://api.Foo.com/>
@prefix pl: <http://pcm.disco.unimib.it/pcm-lite/>
...
:twPolicy rdf:type pl:policy .
:twPolicy rdfs:comment
"The REST API provides simple interfaces for most
Foo functionality."@en .
:twPolicy rdfs:label "Foo REST API"@en .
:twPolicy pl:hasProperty :resource1 .
:twPolicy pl:hasProperty :resource2 .
...
:twPolicy pl:hasProperty :datalicense .
:twPolicy pl:hasProperty :responsetime .

```

In listing 2, a resource (resource1) is further specified by two values: `:statuses/mentions_timeline`, which represents the URI of the resource, and `dbpedia:HTTP_GET`, which states that the GET method is available for that resource. The GET method is a well-known concept linked to a DBpedia concept, according to the third practice. The resource description is accessible and modifiable at <https://api.Foo.com/resource1>, according to the fifth practice.

Listing 2: A property as a resource for *Foo* REST APIs

```

@prefix dbpedia: <http://dbpedia.org/resource/>
...
pl:hasValue rdf:type pl:hasRelation .
pl:hasMethod rdf:type pl:hasRelation .
...
:resource1 rdf:type pl:property .
:resource1 rdfs:comment
"Returns the 20 most recent mentions (tweets
containing a users's @screen_name) for the
authenticating user."@en .
:resource1 pl:hasValue
:statuses/mentions_timeline .
:resource1 pl:hasMethod dbpedia:HTTP_GET .

```

Finally, the listing 3 shows a NFP *as a resource* identified by <https://api.Foo.com/responsetime> that represents the service response time. The document specifies that the response time is less than 0.153 seconds. “Second” and “less than” are well-known concepts, therefore are represented as DBpedia concepts.

Listing 3: Response time property as a resource for *Foo* REST APIs

```

@prefix dbpedia: <http://dbpedia.org/resource/>
...
:responsetime rdf:type pl:property .
:responsetime rdfs:label

```

⁶N-ary relations are defined according to the third use case available in the W3C documentation, <http://www.w3.org/TR/swbp-n-aryRelations/>

```

"Service response time"@en .
:responsetime pl:hasOperator dbpedia:Less-than .
:responsetime pl:hasValue "0.153" .
:responsetime pl:hasUnit dbpedia:Second .

```

3. THE FRAMEWORK

In the previous section, we proposed a set of best practices that defines the model, data format and the vocabulary to be adopted for defining service descriptions, and the method for publishing and managing them as resources. In order to making feasible the practices, we propose a software framework that: (i) semi-automatically extracts service information from provider documentation and third-party service information disperse over the Web and (ii) makes the collected information available as a self-descriptive service.

3.1 Framework architecture

A generic architecture of a self-descriptive RESTful service that exploits our framework is shown in figure 2. The architecture is composed by a *REST interface* that manages HTTP messages. The business logic implements the service functionalities by accessing a database that contains that state of the entities represented as resources. Our framework is based on a module called *self-description wrapper* that can be considered as a plug-in for generic RESTful services.

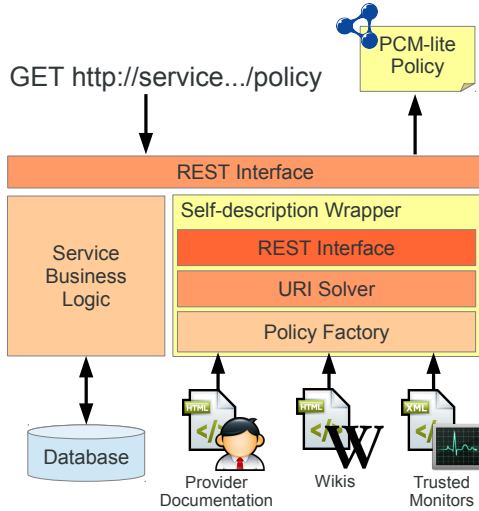


Figure 2: Self-descriptive REST service architecture

Since the wrapper is designed as RESTful service, there is a complete platform independence between the wrapper and the service implementation. Moreover, the additional module can be easily integrated by implementing a HTTP message forward through the main REST interface as shown in figure 3. The result is to hide transparency of the plug-in module behind the main service interface. Finally, the wrapper can be deployed on an additional host in order to preserve the performances of the main service.

A *URI solver* component aims to retrieve or modify the correct policy or property instance on the base of the requested policy URI. The policy extraction and construction is performed by the *policy factory*. In the following subsections will be explained the approach implemented by the policy factory, also introduced in [19].

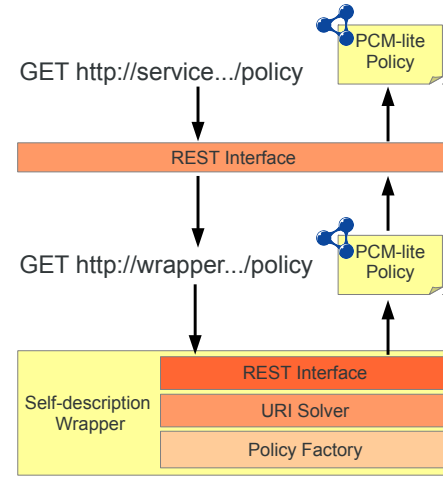


Figure 3: Message forwarding to wrappers

3.2 Property extraction

The service descriptions actually available on the Web are mainly textual descriptions, published as HTML Web pages or semi-structured data, such as JSON, XML or XML-based extensions (e.g., RSS or Atom), provided by Web API repositories (e.g., *ProgrammableWeb*) or external monitoring services (e.g., *API status*).

Actually, Web pages have a hidden structure that defines the page layout (e.g., paragraphs, titles, etc.) by HTML standard tags. Therefore, textual descriptions can be considered as semi-structured data and the same technique for property value extraction can be applied.

Such semi-structured documents can be exploited to generate PCM-lite descriptions by means of *Source-to-policy templates* (S2PTs). A S2PT, associated with a source specifies the properties that will be included in the extracted policies and, for each property, the process of extraction of its value. To extract service information a template defines several XPath⁷ expressions, that allow for the definition of paths that identifies portions of XML, HTML and JSON format.

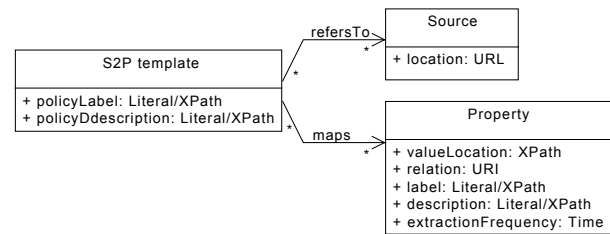


Figure 4: Formalization of Source-to-Policy Templates (S2PTs)

The S2PT formalization is represented by the UML class diagram in figure 4. A template refers to one or more sources that are identified by an URL and maps a set of properties that are represented by the following attributes: (i) a XPath expression that identifies the value location in the document;

⁷Formalized at: <http://www.w3.org/TR/xpath/>

(ii) a URI that defines the relation between the property and the value (e.g., *hasValue* or *hasMethod*); (iii) a natural language label that defines the property name (e.g., “data license” or “resource”); (iv) a natural language description of the property; (v) a time interval that defines the extraction frequency, in order to support dynamic properties such as response time. Moreover, a S2PT has a label and a comment that describe the PCM-lite policy. Labels and comments for policies and properties can be defined manually as literal or extracted from documents through the definition of XPath expressions.

According to the formalization in figure 4, a S2PT can be a description defined in a semi-structured data format, such as XML or JSON, or in RDF. An example of a XML-based S2PT for *Twitter* documentation, defined in HTML, is presented in listing 4. The XML code shows in detail the extraction configuration for the resource represented in the listing 2. The *policy factory*, according to the extraction frequency defined in S2PTs, periodically extracts and creates PCM-lite policies by executing the process shown in figure 5 and represented as activity diagram according to UML 2.

Listing 4: An example of S2PT for extracting *Twitter* service descriptions from provider documentation

```
<?xml version="1.0" encoding="UTF-8"?>
<s2pt:template
  xmlns:s2pt="http://pcm.disco.unimib.it/s2pt"
  ...
>
  <s2pt:source>
    <s2pt:location>
      https://dev.twitter.com/docs/api/1.1
    </s2pt:location>
  </s2pt:source>
  <s2pt:policyLabel>
    /body/div/h1
  </s2pt:policyLabel>

  <s2pt:Property>
    <s2pt:valueLocation>
      /table/tbody/tr/td[1]/a
    </s2pt:valueLocation>
    <s2pt:relation>
      pl:hasValue
    </s2pt:relation>
    <s2pt:label>Resource</s2pt:label>
    <s2pt:description>
      /table/tbody/tr/td[2]
    </s2pt:description>
    <s2pt:extractionFrequency>
      24 hours
    </s2pt:extractionFrequency>
  </s2pt:Property>

  <s2pt:Property>
    ...
  </s2pt:Property>
  ...
</s2pt:template>
```

The first phase is the *document retrieval*. By using the source location URL defined in the S2PT, the document that contains the value is retrieved by a HTTP GET. Then, the *document portion extraction* is performed by submitting the retrieved document and the XPath expression associated with the property to a XPath engine. The result returned by the engine is a textual portion of document that contains the property value.

The third process step is the *named-entity recognition* (NER) [17]. NER tools are able to identify textual terms that refers to specific semantic concepts in a domain ontology.

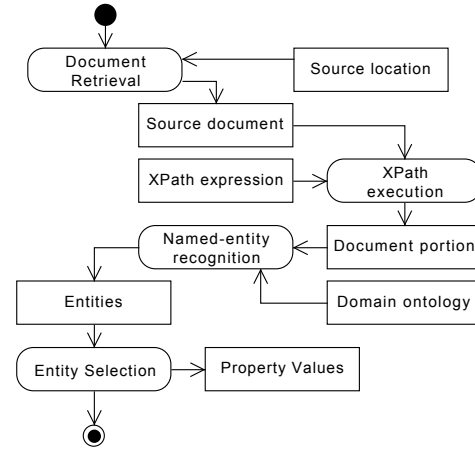


Figure 5: Property value extraction process from textual and semi-structured data

To give an example, a NER tool can recognize from the text “Extensible Markup Language” the concept *dbpedia:XML* defined in DBpedia. NER can exploit ontologies defined by domain experts or ontologies already available on the Web. In our framework, we exploit *DBpedia-spotlight* [15], a NER tool that identifies concepts defined by DBpedia in a text.

Finally, *entity selection* is required to associate concepts identified by NER with property values. More values can be extracted from a text, in this phase we need to identify which are to be associated with a given property. To give an example, in the text “You are expected to be able to parse the XML or JSON response payloads” the *DBpedia spotlight* can identify the concepts *payload* and *parsing* that do not represent a value of the property “data formats”.

The preliminary step of the *entity selection* is processing the property description provided by the S2PT with the NER tool. The technique adopted is to perform NER on the property description, defined in the S2PT, and the text portion that contains the value. Then, the concept that will be considered a value is the entity in the extracted text that has a lowest *semantic distance* (under a particular threshold) with the concepts identified by the property description. For semantic distance, the lowest number of relations between not equivalent concepts defined by the domain ontology graph is considered. For instance, by referring to the previous example, XML and JSON are selected because their distance between the concept *DataFormat* identified in the property description “data formats” is lower to the other identified concepts. The same approach is adopted also to identify units of measurements.

The architecture of the policy factory is shown in figure 6. The architecture is composed of three basic elements: *data source manager*, *property value evaluator* and *policy builder*.

The *data source manager* has the task of retrieving source documents. A second task of this module is to signal potential source faults due to temporary unavailability or removed documents. The *property value evaluator* provides functionalities to extract property values and represent them as semantic concepts. To accomplishing the task, this module includes a *XPath engine* and a *NER tool*. Finally, the *policy builder* enacts the process in figure 5 by orchestrating

data source manager and property value evaluator to build PCM-lite policies.

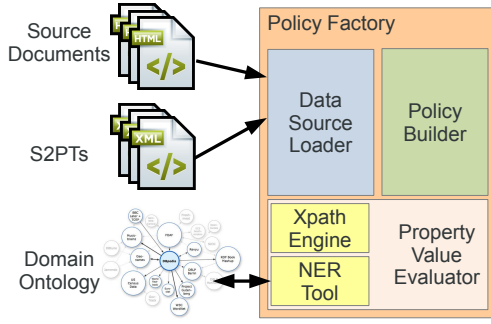


Figure 6: Policy factory architecture

4. LESSON LEARNED

PCM-lite descriptions and the presented framework have been tested over the last months to evaluate the proposal. In this section we discuss the use of self-descriptive RESTful services by humans and tools, and the (semi-)automatic generation of descriptions and management by the framework.

4.1 About the best practices

According to the REST principles, service requesters can access a PCM-lite policy through a HTTP GET. Then, values of properties that are relevant for requesters can be retrieved performing additional GETs by links to properties included in the policy. According to this protocol, the information can be discovered in the same way for any service and deployed by any provider. For example, *Bing Maps geocode*⁸ and *Google geocoding APIs*⁹, that provide similar geocoding functionalities, need to be compared to understand their publication policies by a user looking for data with free licensing to reuse the information provided. Possible data licensing descriptions associated with the two services are shown in listings 5 and 6. A common vocabulary allows users to compare properties of the two services by accessing to DBpedia descriptions.

Listing 5: Data licensing property as a resource for *Bing Maps geocode* service

```
:datalicense rdf:type pl:property .
:datalicense rdfs:label "Data licensing"@en .
:datalicense pl:hasOperator dbpedia:For_all .
:datalicense pl:hasValue dbpedia:EULA .
```

Listing 6: Data licensing property as a resource for *Google geocoding APIs*

```
:datalicense rdf:type pl:property .
:datalicense rdfs:label "Data license"@en .
:datalicense pl:hasOperator dbpedia:For_all .
:datalicense pl:hasValue dbpedia:Creative_Commons .
```

To prove that descriptions compliant with the presented best practices allows machines to perform evaluations on self-descriptive services, we could exploit the *Policy Matchmaker and Ranker for Web* (PoliMaR-Web) tool [21, 19, 20]. The PoliMaR-Web is a service matchmaker for supporting users to discovery the best service according to given requirements.

⁸<http://msdn.microsoft.com/en-us/library/ff701715.aspx>

⁹<https://developers.google.com/maps/documentation/>

The tool is able to match a set of user constraints on service properties, defined according to PCM-lite, by combining mathematical functions and semantic reasoning, therefore it is able to infer that `dbpedia:Creative_Commons` is a free license. In [21], we proved that PoliMaR-Web can be efficient (response time is aligned with an interactive use) and effective (in terms of precision and recall) to evaluate Web API descriptions, including RESTful services.

However, some limitations for humans persist despite the fourth practice constraints the definition of natural language descriptions associated with properties. The two main limitations concern (i) the description modification by humans and (ii) data format readability. According to the fifth practice, the information is made available via HTTP according to REST principles. Currently, the most common tools that allows users to access to information available through HTTP are Web browsers, which was designed to access resources with GET and POST methods, and therefore does not support fully management of descriptions. Moreover, data formats that are adopted to define RDF documents, such as XML, N3, N-Triples and Turtle, do not present descriptions that are very readable for humans. This issue directly emerge by reading listing 1, 2 and 3.

4.2 About the framework

The self-description wrapper is able to make self-descriptive every existing service by exploiting existing descriptions. However, to enable the information extraction, a manual analysis of sources to define templates that identifies property values is requested. Even if a template for a source can serve multiple services (e.g., the template for *ProgrammableWeb* supports about 6,000 services to date), such human effort needs to be reduced to make the approach effective. The automatic identification of properties is an issue still under investigation.

In addition, the precision of the NER tools depends on the domain and adopted techniques [17]. Therefore, the correct identification of property values is not always guaranteed. We performed a precision and recall evaluation to verify if *DBpedia spotlight* correctly identifies property values. The test involved the extraction of 1816 property values available in 500 descriptions provided by *ProgrammableWeb*. These precision p_e and recall r_e are evaluated as follow:

$$p_e = \frac{|C_c \cap I_c|}{|I_c|} \text{ and } r_e = \frac{|C_c \cap I_c|}{|C_c|},$$

where C_c is the set of correct concepts that represent the extracted values and I_c is the set of concepts extracted. The identification of correct concepts is implemented by comparing the NER result with a manual property annotation that represents the standard goal.

Table 1: Effectiveness of property value extraction through named entity recognition

Properties	p_e	r_e
Data formats	0.92	0.91
Licensing	0.79	0.78
Usage Limits	0.45	0.61
Average	0.72	0.77

Precision and recall have been computed for three properties belonging to different domains. For each property, the average of precision and recall for each value that refers to a specific properties have been computed (Table 1). On *data formats*, the extraction has a good precision and recall. Instead, the extraction of *usage limits* sometimes fails. This experiment has proven that effectiveness can vary significantly. Therefore, the definition of reliable techniques to perform named entity recognition is still an issue.

5. RELATED WORK

To the best of our knowledge, there are no other approaches that propose to deliver self-descriptive RESTful services by (i) collecting disperse information and (ii) providing descriptions that are usable by humans and machines. However, some of the aspects addressed in the paper have been treated in the literature.

In this paper, we suggest to use PCM-lite to represent service descriptions since semantic models in the literature show some limitations. The combination between *HTML for RESTful Services* (hRESTS) and MicroWSMO [11] permits providers to bind HTML pages and semantic descriptions. hRESTS is a micro format that identifies functional properties in HTML documents. The format defines additional attributes associated with HTML tags. The hRESTS attributes associate portions of text identified by tags with a service functionality that is represented as a concept of the MicroWSMO model.

The result is that hRESTS supports the identification of properties through tags that providers must specify in Web documents based on HTML. In addition, our approach through *templates* is not tied to HTML documents and supports other semi-structured data based on XML and JSON.

Despite the fact that MicroWSMO supports the definition of NFPs, the hRESTS limitation is mapping only functional properties. Moreover, MicroWSMO strictly separates functional properties and NFPs. As discussed in section 2, if a provider defines a property as non-functional that is considered functional by users (or viceversa), discovery tools could consider the property defined in a user request incomparable with the provider description, because the two property definitions are not instances of the same class.

The Resource Linking Language (ReLL) [1] is a very rich model that allows providers to represent RESTful services by combining the advantages of REST and Linked Data. As well as PCM-lite, ReLL is data format independent and provides a formal definition of resources and links in order to follow HATEOAS principle. Unfortunately, the meta-model does not allow providers to specify NFPs that are key elements to support users for choosing among similar services available on the Web (e.g., geolocation and mapping services).

RESTdesc has been proposed in [27] to combine REST and Linked Data [18], as we suggest with the third practice. This approach is based on an extension of RDF/N3 descriptions that specifies the service functionalities as a set of preconditions that, combined with a user request, generate a specific post condition. A common characteristic that RESTdesc shares with our framework is the adoption of vocabularies that can be provided by the Linking Open Data Cloud. RESTdesc is able to model the behaviour of the service according to HATEOAS. However, PCM-lite provides

a straightforward user interpretation of properties and does not require non-standard extensions of RDF/N3.

The second practice that we propose is the adoption of RDF as semantic data model. JSON-LD [12] is a promising semantic model that is able to implement truly RESTful services that support Linked Data. However, in the current scenario, we suggest RDF because technologies that are able to perform advanced semantic evaluations (e.g., reasoners) that support JSON-LD do not exist to the best of our knowledge.

Karma [26] is a tool proposed for integrating RESTful services with Linking Open Data Cloud. The Karma approach is to provide RESTful resources defined according to XML or JSON as Linked Data. Karma extracts information from structured datasets in order to construct RDF representations. However, this approach does not consider that information about services can be dispersed over the Web and NFPs are not provided directly by services as resources. Moreover, most of the information on services is provided as textual descriptions in partially structured HTML documents. Finally, Karma does not consider that Web documents can include information unrelated to services, which means that service properties should be extracted from portions of structured documents. We addressed this issue by S2PTs that support the identification of pertaining information. The manual work required by S2PT definition is rewarded by higher effectiveness.

Finally, [13] is the only work that proposes an approach for extracting service information from RESTful service documentation. The method composes HTML structure analysis with natural language processing to deliver a complete automatic technique for extracting information, but it takes into account only service functionalities, neglecting NFPs.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a set of best practices to build self-descriptive RESTful services for enabling an effective service discovery by humans and machines. Moreover, to make practices usable with little manual effort, we provide a software framework that extracts compliant descriptions from disperse service information on the Web, and makes them available to clients as resources. In such a way, NFPs are accessible by standard RESTful methods.

PCM-lite descriptions proved to be effective to support users to discover services by means of automatic tools. However, the automatic generation of description is still an issue since important manual effort is necessary to extract properties and property values from existing descriptions on the Web, and name identity recognition requires expert supervision. We are now working to address these issues.

To become fully compliant with the REST principles we need to include all the possible methods, so to be able to completely manage descriptions and services. Moreover, we should be able to access resources (services and their descriptions) through regular browsers and HTML pages. The current Web browsers do not support the full set of HTTP methods. We will develop plugins to overcome this limitation and improve human involvement.

7. REFERENCES

- [1] R. Alarcón and E. Wilde. From RESTful services to RDF: Connecting the Web and the Semantic Web. *CoRR*, abs/1006.2718, 2010.
- [2] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - A crystallization point for the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, 2009.
- [3] R. Chinnici, J. Moreau, A. Ryman, and S. Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. Technical report, W3C, June 2007. Available at <http://www.w3.org/TR/wsd120/>.
- [4] F. De Paoli, M. Palmonari, M. Comerio, and A. Maurino. A Meta-model for Non-functional Property Descriptions of Web Services. In *Proceedings of 6th IEEE International Conference on Web Services, ICWS 2008*, pages 393–400, 2008.
- [5] C. Fellbaum. Wordnet. *Theory and Applications of Ontology: Computer Applications*, pages 231–243, 2010.
- [6] R. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California - Irvine, 2000.
- [7] K. Gomadam, A. Ranabahu, and A. Sheth. SA-REST: Semantic Annotation of Web Resources. Technical report, W3C, April 2010. Available at <http://www.w3.org/Submission/SA-REST/>.
- [8] J. Grant and D. Beckett. RDF Test Cases. Technical report, W3C, February 2004. Available at <http://www.w3.org/TR/rdf-testcases/#ntriples>.
- [9] M. Hadley. Web application description language (WADL). Technical report, W3C, August 2009. Available at <http://www.w3.org/Submission/2009/SUBM-wadl-20090831/>.
- [10] T. Heath and C. Bizer. Linked data: Evolving the web into a global data space. *Synthesis Lectures on the Semantic Web: Theory and Technology*, 1(1):1–136, 2011.
- [11] J. Kopecký, K. Gomadam, and T. Vitvar. hRESTS: An HTML microformat for describing RESTful Web services. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, WI-IAT 2008*, pages 619–625. IEEE, 2008.
- [12] M. Lanthaler and C. Gütl. On using json-ld to create evolvable restful services. In *Proceedings of the Third International Workshop on RESTful Design, WS-REST 2012*, pages 25–32, 2012.
- [13] P. Ly, C. Pedrinaci, and J. Domingue. Automated information extraction from Web APIs documentation. In *Proceedings of 13th International Conference on Web Information Systems Engineering, WISE 2012*, pages 497–511, 2012.
- [14] M. Maleshkova, C. Pedrinaci, and J. Domingue. Investigating Web APIs on the World Wide Web. In *Proceedings of the IEEE 8th European Conference on Web Services, ECOWS 2010*, pages 107–114, 2010.
- [15] P. N. Mendes, M. Jakob, A. García-Silva, and C. Bizer. DBpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th International Conference on Semantic Systems, I-Semantics 2011*, pages 1–8, 2011.
- [16] S. Mokhtar, A. Kaul, N. Georgantas, and V. Issarny. Towards efficient matching of semantic Web service capabilities. In *Proceedings of the International Workshop on Web Services Modeling and Testing, WS-MATE 2006*, 2006.
- [17] D. Nadeau and S. Sekine. A survey of named entity recognition and classification. *Linguisticae Investigationes*, 30(1):3–26, 2007.
- [18] K. Page, D. De Roure, and K. Martinez. REST and Linked Data: a match made for domain driven development? In *Proceedings of the Second International Workshop on RESTful Design, WS-REST 2011*, pages 22–25. ACM, 2011.
- [19] L. Panziera, M. Comerio, F. Palmonari, M. De Paoli, and C. Batini. Quality-driven Extraction, Fusion and Matchmaking of Semantic Web API Descriptions. *Journal of Web Engineering*, 11(3):247–268, 2012.
- [20] L. Panziera, M. Comerio, M. Palmonari, C. Batini, and F. De Paoli. PoliMaR-Web: multi-source semantic matchmaking of Web APIs. In *Proceedings of 13th International Conference on Web Information Systems Engineering, WISE 2012*, pages 812–814, 2012.
- [21] L. Panziera, M. Comerio, M. Palmonari, and F. De Paoli. Distributed matchmaking and ranking of web apis exploiting descriptions from web sources. In *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2011*, 2011.
- [22] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: a Research Roadmap. *International Journal of Cooperative Information Systems*, 17(2):223–255, 2008.
- [23] C. Pautasso, O. Zimmermann, and F. Leymann. RESTful Web Services vs. “big” Web Services: Making the Right Architectural Decision. In *Proceedings of the 17th international conference on World Wide Web, WWW 2008*, pages 805–814, 2008.
- [24] P. Shvaiko and J. Euzenat. Ontology matching: State of the art and future challenges. *IEEE Transactions on Knowledge and Data Engineering*, 25(1):158–176, January 2013.
- [25] F. Suchanek, G. Kasneci, and G. Weikum. YAGO: A large ontology from wikipedia and wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217, 2008.
- [26] M. Taheriyan, C. Knoblock, P. Szekely, and J. Ambite. Rapidly Integrating Services into the Linked Data Cloud. In *Proceedings of 11th International Semantic Web Conference, ISWC 2012*, pages 559–574, 2012.
- [27] R. Verborgh, S. Coppens, T. Steiner, J. Vallés, D. Van Deursen, and R. Van de Walle. Functional descriptions as the bridge between hypermedia apis and the semantic web. In *Proceedings of the Third International Workshop on RESTful Design, WS-REST 2012*, pages 33–40. ACM, 2012.
- [28] S. Vinoski. Putting the “Web” into Web services: interaction models, part 2. *Internet Computing, IEEE*, 6(4):90–92, 2002.