Toward Automatic Semantic API Descriptions to Support Services Composition

Marco Cremaschi, Flavio De Paoli

Toward automatic semantic API descriptions to support services composition

Marco Cremaschi and Flavio De Paoli

Department of Informatics, Systems and Communication
University of Milan - Bicocca, Viale Sarca 336/14, Milan, Italy
{cremaschi,depaoli}@disco.unimib.it

Abstract. The ability to provide appropriate and complete API descriptions to let users discover services that satisfy a set of requirements and compose them to fulfil more complex users' needs is critical for the success of any modern ICT solution. Composition suffers from the lack of semantic matching between properties included in published API descriptions. The work presented in this paper addresses this issue by discussing the current formats and tools to build API descriptions, and presenting a method for extracting and associating semantic to properties. Such method relies on a revised version of Table Interpretation techniques to support semantic annotations of API properties. The objectives are to enrich the popular OpenAPI Specification format with semantic annotations, and add the functionality of semantic annotation and composition to the associated editor.

1 Introduction

The ability to provide appropriate and complete API descriptions to let users discover services that satisfy a set of requirements and compose them to fulfil more complex users' needs is critical for the success of any modern ICT solution. Extensive researches have been conducted with the vision to create automatic integration of Web Services and APIs. Most of these approaches face the problem to make candidate APIs communicate each others due to the lack of semantic matching between input and output data. Although implementing APIs has become common practice, meta-level API definition and implementation have yet to be settled to widely-accepted standards [14]. To automate the interactions between APIs a semantics description of the exchanged data is needed. Approaches to achieve the goal are: creating API descriptions in a logic-based language (e.g., RDF), or linking existing descriptions to shared domain vocabularies or ontologies (e.g., DBpedia). As the former needs expertise in logic-based languages, its adoption has demonstrated to be curtailed; the latter is more approachable, and enriching existing descriptions reduces the effort required.

The work presented in this paper has been partially supported by the EU H2020 project EW-Shopp - Supporting Event and Weather-based Data Analytics and Marketing along the Shopper Journey - Grant n. 732590.

There are many active initiatives to promote the creation and publication of descriptions associated with APIs (see Section 2). A shortcoming is the lack of support to add detailed information that qualifies the properties of an API (e.g., classification of input and response data). As a result, these formats are suitable to complete simple tasks, but inefficient in automatic API discovery and composition due to the lack of machine processable semantics [16]. A critical aspect is the capability of including metadata, which can be interpreted by machine agents in a bottom up way (i.e., information structure should be in pieces to whole) [17]. In the real world, a developer may need to compose APIs that refer, for example, to location information. He or she may search directories such as Programmable Web[1], collect descriptions, and understand the meaning of involved terms, e.g., understand that *address* refers to *city* and *street*, and *latitude/longitude* refer to a geographic *area*; but a machine agent is unable to understand those links without a shared representation of property semantics. The use of links to concepts in shared vocabularies allows machine agents to address the issue.

The goal of our project is to (semi)automatically create semantic descriptions that correlate properties at semantic level to enhance interoperability and composition by machine. The adopted methodology is: (i) evaluate the current approaches to create API descriptions to identify a reference format; (ii) develop a Table Interpretation method to collect sample data from existing APIs and associate them to appropriate concepts from shared vocabularies; and finally (iii) develop methods to support automatic composition. In this paper we concentrate on the first two steps to describe the approach and outline the tools under development. This work roots and extends the one presented in [10] by proposing a more effective Table Interpretation technique, and an initial set of composition rules.

Section 2 discusses the different approaches to API descriptions and motivate the choice of addressing OpenAPI Specification as the reference standard. Section 3 illustrates the methods to extract information and associate them with semantic concepts. Section 4 outlines composition techniques and shortly describe the ongoing works on tools development and testing, and finally Section 5 illustrates conclusions and future work.

2 Service descriptions: state of the art

Descriptions have been classified into functional, dealing with provided APIs and exchanged parameters to state what a service provide and how to access it, and non-functional, dealing with meta information that allow potential users to understand how a given service provides its service [9]. A further classification splits descriptions in syntactic and semantic. The former dealing with the format of calls and exchanged messages, and the latter adding a meaning to the description terms.

[1] http://www.programmableweb.com

The most popular syntactic description model is WSDL 2.0 (Web Services Description Language) [3], which defines an XML format for describing Web services by separating the abstract functionality offered by a service from concrete details such as how and where that functionality is offered. Although it supports descriptions of both SOAP-based services, and REST/API services, it is the de-facto standard for the former, but is rarely adopted for the latter. The Web Application Description Language (WADL) [6] is a machine-readable XML format that was explicitly proposed for API services. WADL was also proposed for standardisation, but there was no follow-up.

More recently, *user-friendly* and *easy-to-use* metadata formats have been introduced, along with editors to support developers in the creation of descriptions for REST APIs. Among others, popular description formats are the Open API Specification (OAS)[2] (also known as Swagger specification), which provides human-readable API descriptions based on YAML and JSON. RAML is a YAML-based language for describing RESTful APIs. API Blueprint is a documentation-oriented web API description language, which provides a set of semantic assumptions laid on top of the Markdown syntax. The Hydra specification, which is currently under heavy development, tries to enrich current web APIs with tools and techniques from the semantic web area.

The OAS is the most promising choice at the moment [15], since (i) a simple format to specify descriptions, and (ii) a large set of vendor-neutral API tools, supported by a very large community of active users, are provided. Such tools provide great support to almost every modern programming languages to create and test APIs. Moreover, the Open API Initiative is an open source project sustained by relevant stakeholders, such as Google, IBM, Microsoft and PayPal[3].

The description formats discussed so far are mainly syntactic, which means that little support to automate operations such as services discovery and composition, and verification of coherence to given interaction and building patterns is provided. Although there are many approaches proposed to enrich services descriptions with semantics, the manual work required to create descriptions, and the lack of interoperability standards limited their adoption. The initial approach proposed by the semantic web community was to define a global ontology to include model, definitions and descriptions in a coherent system that can be used to make discovery and automatic composition. The most popular proposals are OWL-S (Ontology Web Language for Services) [11] and WSMO (Web Service Modelling Ontology) [13]. The major problem with these approaches is the expertise required to build and manage such descriptions. The result is that nobody actually use them. Anyway, the knowledge gained with these semantic studies has led to the definition of simpler and easier models that marries the annotation approach introduced by hRESTS and RDFa.

Table 1 illustrates the characteristics of API description models with respect to the supported type of services (SOAP and/or REST), the capability of hosting semantic annotations, the serialisation language to publish the descriptions,

[2] https://www.openapis.org/specification/repo
[3] https://www.openapis.org/membership/members

the availability of supporting tools, and finally the human readability of the descriptions. Table 2 is an adapted and updated version of the one presented in [15] to compare the number of questions posed in Stack Overflow and the number of stars (showing appreciation to a project) received by the four description models under study. The numbers give evidence of increasing interests in the use of description models. The presence of a comprehensive set of tools that support the creation, publication, use and maintenance of service descriptions is one of the most relevant elements that state the success of a description model. The most popular model is OAS, which we consider as reference format for our research that aims at delivering semantic-enabled tools for describing and discovering first, and then compose API services.

Table 1. Comparison of API description standards.

Description	***Service type***	***Semantics***		***Serialization***	***Tool***	***Human Readable***
		Yes/No	*Format*			
WSDL [3]	v1.1 SOAP v2.0 REST	No	-	XML	Yes	No
WADL [6]	REST	No	-	XML	Yes	No
hREST [7]	REST	No	-	Microformat	No	Yes
RDFa [1]	REST	No	-	HTML+RDF	No	Yes
OpenAPI Specification	REST	No	-	YAML, JSON	Yes	Yes
RAML	REST	No	-	YAML	Yes	Yes
API Blueprint	REST	No	-	Markdown	Yes	Yes
OWL-S [11]	SOAP REST	Yes	OWL	OWL	No	No
WSMO [13]	SOAP REST	Yes	MOF[a]	MOF	No	No
SA-WSDL [8]	v1.1 SOAP v2.0 REST	Yes	RDF	XML	No	No
Micro WSMO [7]	REST	Yes	RDF	RDF	No	Yes
SA-REST [5]	REST	Yes	RDF, OWL	RDF	No	Yes

[a] Meta-Object Facility

3 An approach to semantic description building

The task of building descriptions has been recognised as a critical activity mainly for the effort needed to actually write such descriptions, and the expertise required to deliver semantic enriched descriptions. The use of tools that (semi)automatically extract information to enrich existing descriptions should be the right approach to incrementally build effective descriptions. In this project we adopt the best practices proposed by the OAS model, which have been already implemented in the Swagger editor[4], and extend them to add semantic annotations. The extension consists in the definition of new elements in the description format to host semantics, and a technique to identify such annotations by collecting actual responses of services. The process of annotating an API description consists of three steps: (i) building a table with the results collected

[4] https://swagger.io/swagger-editor/

from actual executions of the service; (ii) annotate the table by a Table Interpretation technique; and finally (iii) include the annotations in the API description.

The execution of a set of calls on the bases of the *input parameters*[5] in the existing descriptions allows for collecting responses to create a table with *properties*[6] populating the header row and responses data populating the columns. The Table Interpretation technique [18] allows for extracting semantic information from a table, which means give an interpretation to the values in structured data sources.

Table 2. Comparison of API description models.

Detail/Model		***API Blueprint***	***RAML***	***WADL***	***OpenAPI Spec***
Format		Markdown	YAML	XML	YAML, JSON
Licence		MIT	ASL2.0	Sun	ASL 2.0
Available		Github	Github	www.w3c.org	Github
Sponsored by		Apiary	Mulesoft	Sun	Reverb
Version		Format 1A revision 7	1.0	31 August 2009	2.0
Initial commit		Apr 2013	Sep 2013	Nov 2006	Jul 2011
Pricing plan		Yes	Yes	No	No
StackOverflow	2015	75	37	156	732
Questions	2017	921	644	1,075	8,954
Github	2015	1,819	1,058	N/A	2,459
Stars	2017	5,390	2,735		6,360

An algorithm analyses the table content and associates the semantic concepts (or classes, types) extracted from ontologies in the Linked Open Data Cloud (LOD), which represents the knowledge in a certain domain. In this way API's properties and values can be "understood" by a computer. Based on the state of the art [18, 12], given a well-formed relational table and reference sets of concepts (e.g., DBpedia classes), datatypes (e.g., DBpedia datatypes), named entities (e.g., DBpedia resources) and relations (e.g., DBpedia objectProperty and datatypeProperty), a Table Interpretation process is composed of these tasks:

1. classify columns as a "literal column" (Literal column) if contains generic data (e.g., strings, numbers, dates) or as a "named entities columns" (NE-column) if contains instances of a concept (e.g., dbr:Milan is a dbo:City);
2. annotate column headers with concepts if they contain entity mentions (NE-column) (e.g., the header *city* can be mapped to dbo:City), or properties of concepts if they contain literals (Literal column) (e.g., the header *latLng* can be mapped to *geo:location*);
3. disambiguate entity mentions in "content cells" (or simply cells) by linking them to the existing reference entities (e.g., *Milan* and *London* can be mapped to *dbr:Milan* and *dbr:London*);

[5] https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md#parameters-definitions-object

[6] https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md#schema-object

4. identify the relations between columns (e.g., set a relation between columns *city* and *country* using *dbo:country*). The type of relationship can be an object property if it connects two semantic concepts (from NE-column to a NE-column), or a data-type property if it links a concept to its specific property (from the NE-column to a literal column).

Once the annotation has been identified, the API description we propose to enrich a OpenAPI Specification adding two new properties: (i) *classAnnotation* to hold the annotations relating to the type of the columns, (ii) *propertyAnnotation* to hold the annotations that represent the relationships between columns. Semantic annotations included in the description take the form of URIs that uniquely identifies the concepts and relations in the reference ontologies.

Inputs need a different approach since the input parameters cannot populate a table. Natural Language Processing (NLP) techniques [4] can help to extract entities from the textual description associated with the API. Such entities will be sought after in reference ontologies, and the user needs to validate or modify the candidate annotations.

Listing 1.1 show an example of an OpenAPI description augmented with semantic annotation. This API provides a list of spots (places to practice surf) in the specified city. Listing shows how the input parameter "city" has been annotated with the class *City* and "name" with the class *Place* of DBpedia (*classAnnotation*). Similarly, classes have been identified for the other properties. In addition, "address" and "country" have been annotated with *propertyAnnotation*s to qualify them as related to "name", which has been identified as a main property, through the relations *dbo:address* and *dbo:country*, respectively.

4 Composition rules

As noted above, the annotations can enable the composition of services, which mainly takes the form of "mashup" of API responses. Let's proceed with an example to clarify what we mean by API composition. Assume that a professional surfer wants to find the best location (spot) to practise. The sportsman want to choose the spot, based on personal preferences and/or the context (e.g., weather and sea conditions, spot facilities, accessibility, etc.). Unfortunately, he has to invoke different services (e.g., weather forecast, spot list) to collect data before making an informed decision. The surfer saves time and effort if all data are available in an aggregated way; for example the list of spots returned by the previous API can be composed with an API that provide information about weather[7] or sea condition[8], or with a list of surf schools[9].

Two kind of composition patterns can be identified: *flow composition*, which means that all or part of the output of an API is used as input of another API; and *parallel composition* (or mashup of outputs).

[7] https://www.wunderground.com/weather/api
[8] https://developer.worldweatheronline.com/api/marine-weather-api.aspx
[9] http://www.surfline.com/home/index.cfm

In the former, if inputs and outputs are not of the same type, an additional API that allows conversion or integration of data is needed. In the example, to compose the API regarding sea condition and spot list, a third API that convert the address of a spot into latitude and longitude (e.g., Google Maps API) is required. This two new parameters can be used to invoke sea-condition API. The second pattern foresees that the responses from an API will be filtered out with the responses from another API. The user can define what are the discriminating properties for the composition. The user can also define the metrics that will be used in the composition of the responses. These metrics are: strings similarity metrics that are used for text fields; and, definition of ranges, used for properties with numeric values. Regarding the example, the spot list can be merged with the list of surf schools.

The described compositions can be performed automatically by exploiting semantic descriptions by applying the following rules:

Annotations referring to a single ontology, same concepts If the properties of two APIs refer to the same concepts in an ontology, the composition is straightforward.

Annotations referring to a single ontology, different concepts If the involved concepts are related to rdfs:subClassOf or rdfs:subPropertyOf, as defined by the RDF Schema [2], to indicate respectively the sub-class relationship, in which all instances of the class are also instances of the class indicated by the object, and the sub-property relationship, that is, a defined property as a specialization of another property, the composition can be performed by considering the parent classes.

Annotation referring to different ontologies If the involved concepts belongs to different ontologies, the composition becomes straightforward if the ontologies are *aligned* (e.g., relations of type owl:sameAs exist between the two ontologies).

The algorithms discussed in the previous sections have been implemented by extending the Swagger editor that can now support both the annotation of API descriptions and composition of API. According to the test-first principle, a set of API descriptions have been created. They are realistic since they derive from real ones identified in Programmable Web, include all relevant property types, and address possible composition patterns. The test phase is still ongoing, but the initial results are encouraging since about 70% of the tested patterns was successfully accomplished. The compositions that failed involved semantic descriptions that included hierarchical concepts, which will trigger a further refinement of the algorithm.

5 Conclusions and future work

The work presented in this paper is part of the EW-Shopp H2020 project that aims to provide real-time responsive services to integrate consumer and market data with weather and event data in the digital marketing domain. The semantic

annotation of such services is crucial to prepare the data to support analytics and decision making. It can be accomplished by linking properties and associated values of services to concepts in shared ontologies. Such knowledge can be extracted by techniques like Table Interpretation that has been introduced and exploited to populate OAS descriptions. The current activity deals with testing to perform an initial validation and tune up of the table annotation and annotation techniques against a set of selected artificial and real services. Future work will deal with extensive validation activities against the large set of real-world APIs developed within EW-Shopp to evaluate usability (the goal is to build effective tools for developers with little experience on semantic techniques), and effectiveness (the challenge is to be able to augment and compose generic APIs as well as generic data sources published in marketplaces) of the tools.

Listing 1.1. Example of API description following OAS with annotation of input parameter and properties.

```
prefix dbo: <http://dbpedia.org/ontology/>
prefix dbp:  <http://dbpedia.org/property/>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
[...]
paths:
  /spots:
    get:
      tags:
      - "Spot"
      description: "Returns the spots in the specified city"
      produces:
      - "application/json"
      parameters:
      - name: "city"
        description: "Name of the city"
        type: "string"
        classAnnotation: "dbo:City"
      responses:
        200:
          schema:
           $ref:  "#/definitions/Spot"
           [...]
definitions:
  Spot:
    type: "object"
    properties:
     name:
        type: "string"
        classAnnotation: "dbo:Place"
     address:
        type: "string"
        propertyAnnotation: "dbo:address"
        classAnnotation:  "rdfs:Literal"
     country:
        type: "string"
        propertyAnnotation: "dbp:country"
        classAnnotation: "dbo:country"
     [...]
```

References

1. Adida, B., Birbeck, M., McCarron, S., Pemberton, S.: Rdfa in xhtml: Syntax and processing. Recommendation, W3C 7 (2008)
2. Brickley, D., Guha, R.V., McBride, B.: Rdf schema 1.1. W3C recommendation 25, 2004–2014 (2014)
3. Chinnici, R., Moreau, J.J., Ryman, A., Weerawarana, S.: Web services description language (wsdl) version 2.0 part 1: Core language. W3C recommendation 26, 19 (2007)
4. Chowdhury, G.G.: Natural language processing. Annual Review of Information Science and Technology 37(1), 51–89 (2003)
5. Gomadam, K., Ranabahu, A., Sheth, A.: Sa-rest: semantic annotation of web resources. W3C Member Submission 5, 52 (2010)
6. Hadley, M.J.: Web application description language (wadl). Tech. rep., Mountain View, CA, USA (2006)
7. Kopeckỳ, J., Vitvar, T., Fensel, D., Gomadam, K.: hrests & microwsmo. STI International, Tech. Rep. (2009)
8. Lausen, H., Farrell, J.: Semantic annotations for wsdl and xml schema. W3C recommendation, W3C 69 (2007)
9. Li, P., Comerio, M., Maurino, A., De Paoli, F.: An approach to non-functional property evaluation of web services. In: Proc. IEEE International Conference on Web Services, ICWS 2009. pp. 1004–1005 (2009)
10. Lucky, M.N., Cremaschi, M., Lodigiani, B., Menolascina, A., De Paoli, F.: Enriching api descriptions by adding api profiles through semantic annotation. In: Proc. of the 14th ICSOC 2016. pp. 780–794. LNCS Springer (2016)
11. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., et al.: Owl-s: Semantic markup for web services. W3C member submission 22, 2007–04 (2004)
12. Ramnandan, S., Mittal, A., Knoblock, C.A., Szekely, P.: Assigning semantic labels to data sources. In: in Proc. ESWC 2015. pp. 403–417. Springer (2015)
13. Roman, D., Kopeckỳ, J., Vitvar, T., Domingue, J., Fensel, D.: Wsmo-lite and hrests: Lightweight semantic annotations for web services and restful apis. Web Semantics: Science, Services and Agents on the World Wide Web 31, 39–58 (2015)
14. Sheng, Q.Z., Qiao, X., Vasilakos, A.V., Szabo, C., Bourne, S., Xu, X.: Web services composition: A decades overview. Information Sciences 280, 218–238 (2014)
15. Tsouroplis, R., Petychakis, M., Alvertis, I., Biliri, E., Lampathaki, F., Askounis, D.: Community-based api builder to manage apis and their connections with cloud-based services. In: CAiSE Forum (2015)
16. Verborgh, R., Harth, A., Maleshkova, M., Stadtmüller, S., Steiner, T., Taheriyan, M., Van de Walle, R.: Survey of semantic description of rest apis. In: REST: Advanced Research Topics and Practical Applications, pp. 69–89. Springer (2014)
17. Verborgh, R., Mannnens, E., Van de Walle, R.: Bottom-up web apis with self-descriptive responses. In: Proceedings of the First Karlsruhe Service Summit Workshop-Advances in Service Research. p. 143. KIT Scientific Publishing (2015)
18. Zhang, Z.: Start small, build complete: Effective and efficient semantic table interpretation using tableminer. Under transparent review: The Semantic Web Journal (2014)