Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 137 (2018) 255–261

Procedia Computer Science

www.elsevier.com/locate/procedia

SEMANTiCS 2018 – 14th International Conference on Semantic Systems

Machine Readable Web APIs with Schema.org Action Annotations

Umutcan Şimşek*, Elias Kärle, Dieter Fensel

STI Innsbruck, Department of Computer Science, University of Innsbruck, Technikerstrasse 21a 6020, Innsbruck Austria

Abstract

The schema.org initiative led by the four major search engines curates a vocabulary for describing web content. The number of semantic annotations on the web are increasing, mostly due to the industrial incentives provided by those search engines. The annotations are not only consumed by search engines, but also by other automated agents like intelligent personal assistants (IPAs). However, only annotating data is not enough for automated agents to reach their full potential. Web APIs should also be annotated for automating service consumption, so the IPAs can complete tasks like booking a hotel room or buying a ticket for an event on the fly. Although there has been a vast amount of effort in the semantic web services field, the approaches did not gain too much adoption outside of academia, mainly due to lack of concrete incentives and steep learning curves. In this paper, we suggest a lightweight, bottom-up approach based on schema.org actions to annotate Web APIs. We analyse schema.org vocabulary in the scope of lightweight semantic web services literature and propose extensions where necessary. We demonstrate our work by annotating existing Web APIs of accommodation service providers. Additionally, we briefly demonstrate how these APIs can be used dynamically, for example, by a dialogue system.

Keywords: Web APIs; lightweight semantic web services; schema.org; schema.org actions

1. Introduction

The semantic annotation of web content is realizing the vision of making the web machine readable. More than a decade ago, researchers have identified that the challenge is not only bringing semantics to the data on the web, but also to web services, in order to enable automated agents to understand them and automate web service tasks. Initial efforts have been mostly focused on SOAP services. The semantic description of RESTful web services came later as they gained significant popularity within the last decade due to their lightweight approach and flexibility, for instance in terms of supported data formats.

* Corresponding author.
E-mail address: umutcan.simsek@sti2.at

However, outside of academia, the adoption of semantic web services have remained quite low. The main reason for the weak adoption is a so called ”chicken-egg problem”, that is, there is no interest in application development since there are no annotated web services and there is no annotation effort since there are no applications. Earlier approaches are semantically very strong and well designed, however, for the web service providers they look challenging [8]. We aim to overcome this chicken-egg problem by lowering the entry barrier to the semantically annotated Web APIs. For that, we utilize the already well adopted schema.org vocabulary and try to create a lightweight semantic web services vocabulary based on schema.org actions. This way, the machine readable Web APIs can be consumed by agents like goal-oriented dialogue systems (e.g. Intelligent Personal Assistants) to complete tasks like purchasing products or booking hotel rooms on the fly.

To realize our approach, we first analyse schema.org to see how it can be placed in the well established semantic web services literature. Then we propose some minor extensions for necessary points. Afterwards, we present our mapping and wrapping approach for semantic lifting of the Web APIs in JSON-LD format and grounding of JSON-LD requests annotated with schema.org to the accepted data format of individual Web APIs.

The remainder of this paper is structured as follows: Section 2 presents a literature review on the semantic web services field with a focus on the approaches for machine readable description of Web APIs. Section 3 gives an introduction to schema.org actions vocabulary and analyses it in terms of lightweight semantic web services. Section 4 explains our methodology for the mapping and implementation of a wrapper for existing Web APIs. Section 5 demonstrates the publication and consumption of such annotated Web APIs through our wrapper. Finally, Section 6 presents some concluding remarks and pointers to the future work.

2. Related Work

The initial efforts in the semantic web services field were targeting SOAP services. OWL-S [10] builds on top of OWL and uses Description Logic (DL) to describe web services, while SWSF [1] utilizes First Order Logic (FOL). WSMF [2] is a comprehensive approach that takes decoupled mediation to its core.

As for the RESTful web services, the initial efforts were adapted from SOAP services mainly to ensure the interoperability. For instance, the WSMO-Lite ontology was developed as a lightweight version of WSMO which is the modelling ontology of WSMF. It is more interoperable with W3C recommended technologies and is used as underlying semantic model for MicroWSMO , a language that extends hRESTS microformat for semantic description of RESTful services.[12]

Recently, several approaches specific to RESTful services have been proposed. Among those Hydra [9], RESTDesc [15], RESTDoc[5] and smartAPI [16] come to the fore. These lightweight API annotation approaches allow clients to be as generic (i.e. decoupled from the API) as possible and use the APIs with minimal apriori knowledge. This is the main difference between these approaches and API description languages like WADL[1], OpenAPI[2], RAML[3] and API Blueprint[4], since they merely give a structure to the documentation without any semantics and hypermedia-driven navigation possibilities. An active effort towards hypermedia-driven Web APIs is Hydra. Hydra is a vocabulary for bringing RESTful APIs and linked data together without any well-defined complex semantics to lower the entry barrier and ease the adoption. It allows Web API owners to describe a machine-oriented API documentation which contains all accessible resources and a client can navigate the API using links and supported operations. Hydra community working group also contributed to the inclusion of actions in schema.org.

smartAPI builds on top of OpenAPI specifications and provides means for semantically annotating Web APIs. smartAPI allows Web API developers to attach semantic metadata to their API descriptions. They provide specifications and set of tools for annotating and publishing API descriptions.

RESTDoc has a similar principle as Hydra with certain implications about the underlying semantics (i.e. RDF(S)) for discovery and composition tasks. The main drawback of this approach is that it is tied to a specialized microformat syntax. Additionally, it is not clear how the behavioural aspects can be represented. Another approach is RESTDesc,

[1]Web Application Description Language
[2]https://www.openapis.org
[3]https://raml.org
[4]https://apiblueprint.org

which uses N3-Notation syntax and semantics to represent preconditions and postconditions for operations defined on the resources of a Web API. This gives a relatively well-defined semantics and reasoning support for automated discovery and composition.

To the best of our knowledge, there has not been an effort so far to place schema.org actions into the semantic web services field. Incorporation of our approach with Hydra and smartAPI would also be an interesting research topic since the development goes in the similar direction. Our approach is completely based on schema.org and its extensions to describe resources and their specific instances (request and responses). The actions can be defined standalone as well as attached to a specific instance. This would allow us to specify operations not only on resource level but also on instance level (e.g. number of children may be required for one type of room and optional for another in a hotel booking API). This feature is aligned with the conversation based API design approach described in [4]. An interesting feature of schema.org actions vocabulary is that it provides a mechanism to attach high-level operations beyond HTTP CRUD operations. This provides an opportunity for (semi-)automatic generation of agents like dialogue systems [14]. The schema.org vocabulary is already a de-facto standard for annotation of data on the web, which is an advantage for the adoption of our approach. Additionally, we propose an extension for schema.org actions to facilitate description of authentication mechanisms.

3. Schema.org Actions as a Web API Annotation Vocabulary

In this section we will give a brief introduction to the actions in the schema.org vocabulary. Then we will investigate the vocabulary and its semantics in the scope of lightweight semantic web services.

3.1. Schema.org Actions

The schema.org actions have been included to the core vocabulary in 2014 to give a mechanism to annotate not only static entities, but also actions that can be taken on them[5]. The action vocabulary is built around the schema:Action[6], which is the most generic type of action. The action annotations can exist in two different forms: (a) as stand alone annotations with a value in the schema:object property (b) as potential actions defined on the instances of any type.

Schema.org does not provide any normative instructions for using the action vocabulary, only certain guidelines and suggestions[7]. In the next subsection we will discuss the schema.org actions vocabulary from the semantic web services point of view.

3.2. Lightweight Web Service Description with Schema.org Actions

Many Web APIs that operate on HTTP follows the REST principles [3] to some extent. That means, a Web API or a REST-like API is a collection of resources ideally linked via certain operations. As it is defined in the REST architectural elements, "a resource is an identifiable abstraction of a set of entities." The identification of resources are realized with URIs. Whenever a resource is requested, the server maps the request to a set of entities and responses to the client with a certain resource representation and its metadata.

Table 1 shows a mapping between REST data elements and schema.org types and properties. A Web API can be seen as a set of schema:Action annotations. The value of the schema:object property on a schema:Action instance represents the abstract concept of a resource. A resource is identified with the value of the schema:urlTemplate property. An action describes a high level operation on a resource (e.g. CommentAction, AddAction), which is in return connected to an HTTP method (e.g. POST, PUT). As for the parameterized operations, the input parameters can be described with <property name>-input properties and schema:PropertyValueSpecification instances. Similarly, a resource can promise to return certain property values with <property name>-output property definitions. Note that, the response to the resource request may have other property values as well, but the promised property values should definitely be included in the response. The concrete representation of a request, response and metadata about

[5]http://blog.schema.org/2014/04/announcing-schemaorg-actions.html

[6]Throughout the paper, the "schema" prefix will be used for http://schema.org/ namespace.

[7]See https://schema.org/docs/actions.html

REST Data Element	Relevant Schema.org Terms
Resource	schema:object
Resource Identifier	schema:urlTemplate
Resource Method	schema:Action, schema:PropertyValueSpecification and schema:httpMethod
Resource Representation and Metadata	schema:encodingType, schema:contentType

Table 1: Mapping of REST data elements to Schema.org

how they should be interpreted by the client can be described with schema:encodingType and schema:contentType properties. Although action annotations are useful for describing the public interface of an API, to truly enable the hypermedia-driven nature, the concrete entities to which the resource requests mapped (i.e. responses) should be also described with their potential actions. This is achieved by attaching such actions to a Web API response with schema:potentialAction property.

3.3. Service Semantics in Schema.org Actions

In this section we briefly analyze schema.org from a service semantics point of view. This analysis will be useful for the future, when the automation of web service tasks are considered. There is a distinction made between different type of service semantics in [13]. These semantics are important for automation of web service tasks like discovery/retrieval and composition.

3.3.1. Information Model and Functional Semantics

The information model defines the information transferred via the Web API, namely the inputs, outputs and fault messages. In our approach, we use a single domain ontology, namely schema.org, and/or its extensions for the information model. This reduces the heterogeneity and consequently the data representation mismatches. The property to attach fault messages to an action, schema:error, takes schema:Thing as range, which is the top concept in the vocabulary and open to extension. Schema.org has an RDFS based data model and RDFS entailment patterns regarding subclasses and subproperties are applicable[8]. The functional description specifies the functionality of a service, in other words, what a service can offer. We adopt a simple signature view [7], meaning the inputs and outputs of an operation is defined but no explicit mapping from inputs to outputs are given.

3.3.2. Behavioural Model and Semantics

The behavioural model defines the order of operations to consume the service functionality. With schema.org actions, the order of operations are implicitly dictated by the potential actions attached to the responses to a resource request.

3.3.3. Non-Functional Description and Semantics

The non-functional description specifies the policies for the consumption of the service as well as the meta-information such as the creator and version of a given implementation of a service. These properties are including but not limited to temporal availability, price, payment and security [11]. As for the Web APIs, this aspect is often neglected or limitedly represented. Furthermore, the existing non-functional properties are usually described in a human readable way only, sometimes even outside of the service, in an API repository. The pending schema:WebAPI class can be used for describing some non-functional aspects. Additionally, we propose an authentication extension. Schema.org offers a schema:instrument property which can be used to describe authentication tokens[9]. We define the authentication on the action, in other words at the resource method level. Based on the analysis we made on different API specification languages, we identified three different authentication methods: token-based, basic and custom form-based authentication.

The token-based authentication is represented by webapi:TokenAuthentication type. The value of the webapi:bearerToken property on this type is mapped to the Authorization header in "Bearer <token>" pattern. Similarly,

[8]Entailment patterns: Section 9.2.1 in https://www.w3.org/TR/rdf11-mt/ See for more details: https://actions.semantify.it/vocab
[9]The proposed authentication extension is defined and documented in https://actions.semantify.it/vocab/ and represented with 'webapi' prefix.

webapi:HTTPBasicAuthentication type is mapped to Authorization header in ”Basic <token>” pattern. The third method is for other custom authentication approaches. For these approaches, more information needs to be described such as, in which part of the request (header, body or url) the token is sent and what is the key and what is the value. This can be done through using name and value properties of the schema:PropertyValue type which is the supertype of all authentication types.

4. Methodology for Wrapping Web APIs

Our methodology for wrapping Web APIs consist of the following steps: API analysis, vocabulary selection, mapping and wrapper implementation.

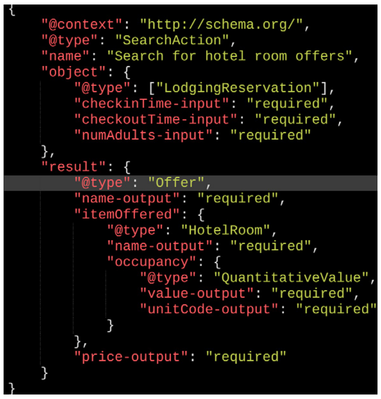
To find a mapping for an API, the first thing to do is the **API analysis**. We have to investigate the individual resources in terms of inputs, outputs, encoding types, error messages and operations. Then the service behaviour should be also analyzed to see what kind of responses are returned.

The second step is the **vocabulary selection** to describe web services. The relevant types and properties for the analyzed API in schema.org vocabulary are identified in this step.

The third step is the **mapping**. At this step, the identified types and properties are mapped to API constructs. There are two different kind of mappings: (a) mapping of resources and (b) mapping of instances. The details of the mapping shown in Table 2 is demonstrated with a use case in Section 5.

GET api/searchCatAvailability/ → schema:SearchAction	
schema:object → schema:LodgingReservation	
schema:result → schema:Offer	
Input Parameter name	**Mapped Schema.org property**
arrivalDate	checkinTime
departureDate	checkoutTime
numAdults	numAdults

Table 2: Partial mapping of an Easybooking Hotel Booking API resource to schema.org



```
{
    "@context": "http://schema.org/",
    "@type": "SearchAction",
    "name": "Search for hotel room offers",
    "object": {
        "@type": ["LodgingReservation"],
        "checkinTime-input": "required",
        "checkoutTime-input": "required",
        "numAdults-input": "required"
    },
    "result": {
        "@type": "Offer",
        "name-output": "required",
        "itemOffered": {
            "@type": "HotelRoom",
            "name-output": "required",
            "occupancy": {
                "@type": "QuantitativeValue",
                "value-output": "required",
                "unitCode-output": "required"
            }
        },
        "price-output": "required"
    }
}
```

Fig. 1: Partial JSON-LD representation of the Eventbrite resource in Table 2

The fourth and final step is the **wrapper implementation**. The wrapper is mostly a standalone software that implements the mappings defined in the previous step. The wrapper produces static JSON-LD annotations for resource descriptions and on the fly-annotations for responses returned from an API. The wrapper additionally creates another annotation that serves as a machine-readable documentation of the available actions for an API.

5. Use Case

As a proof-of-concept, we are presenting two use cases from the tourism domain. Tourism is a very convenient choice since it is a domain where the Web APIs are used extensively for tasks like booking rooms or buying event tickets.

The use cases focus on two aspects, namely the publication of actions as described above, and the consumption of those actions by a third-party software. First we show how an Internet Booking Engine can be described with schema.org actions and tourism related subset of schema.org [6]. Then, we show how a dialogue system can semi-automatically consume such semantically described API.

5.1. Easybooking Hotel Booking API Mapping and Action Publication

We created a mapping for one of the leading Internet Booking Engine providers in our region (Table 2). We first mapped ”searching a hotel room and its offers” operation to schema:SearchAction on */api/searchCatAvailability/* resource. Even though it is omitted in the Figure 1, the resource identifier would be the value of the schema:urlTemplate of the schema:target property of the schema:SearchAction. Then the input parameters required for the defined operation is mapped to a schema.org type and properties. The properties arrivalDate, departureDate and numAdults are mapped to schema:checkinTime, schema:checkoutTime and schema:numAdults of the schema:LodgingReservation type, respectively. What the operation returns will be the result of the action, which is, in this case a schema:Offer instance with a schema:HotelRoom. As shown in the Figure 1, the result of a request to this action contains at least the name and the allowed occupancy of the hotel room, as well as the price of the offer. Once a request is created based on the action description and sent to the resource, the response would be an action with an array of schema:Offer instances in the schema:result property. Each instance would have zero or more potential actions. In this case, the offers would have schema:BuyAction instances as potential actions. A client could follow the potential actions after each request to complete the workflow without having an apriori knowledge of the API[10].

5.2. Consumption of Actions with a Dialogue System

The actions in schema.org vocabulary allows us to define high-level operations on top of HTTP CRUD operations. For instance the semantics of reserving a room can be described with the schema:ReserveAction or of buying an offer with the schema:BuyAction. This feature has also implications towards goal-oriented dialogue systems. The goal-oriented dialogue systems typically access a back-end system based on a user’s intent to complete a task. The high-level resource-operation relationship can be converted to an intent (e.g. reserve.hotelroom intent based on schema:ReserveAction with a HotelRoom object). A Natural Language Understanding (NLU) module can classify user utterances to an extracted intent. The details of how the NLU trained for this intents is outside of the scope of this paper, but we demonstrate how a task-oriented dialogue system can consume Web APIs annotated with schema.org actions.

Listing 1 shows an example dialogue generated from Easybooking API[11]. First, Dialogflow[12] classifies the user utterance to an intent extracted from the API. Then, the values for required slots (e.g. check-in and check-out time) for the intent are elicited from the user. The slots are also extracted dynamically from the functional description of the API. The dialogue continues until the intent is fulfilled.

Listing 1: An example dialogue based for hotel booking over annotated Web APIs

```
1  User: I am searching for the Hotel STIInnsbruck.
2  Bot: One item has been found. Hotel STIInnsbruck. You can ask me something about it.
3  User: Tell me more.
4  Bot:  You can search and find hotel rooms and offers.
5  User: Let's do it.
6  Bot: Please tell me your check-in time.
7  User: 1.1.18
8  Bot: Please tell me your check-out time.
9  User: 2.1.18
10 Bot: I found 4 items. The first 2 are: 1. Einzelzimmer, 2. Doppelzimmer
11 User: 2
12 Bot: You can buy an offer of this room.
13 User: Let's do that.
14 Bot: Buy action completed. You can see the confirmation.
```

For the dialogue in Listing 1, User utterance (Line 1) is first matched with a schema:SearchAction. The response to this action itself has a schema:SearchAction attached to it, which allows the user to search for hotel rooms and

[10]The implemented wrapper can be found in https://actions.semantify.it
[11]Demo dialog system: https://bot.dialogflow.com/3aa58719-b665-4e7b-970a-564c1b9a64c5
[12]https://dialogflow.com/

offers, after eliciting certain filtering criteria extracted from the action description (Line 2-9). The returned offers of hotel rooms also have schema:BuyAction as potential action, so the flow is completed after the buying process (Line 10-14). Note that the dialogue system presents several options to the user based on the potential actions attached to instances (Line 4 and Line 12).

6. Conclusion and Future Work

In this paper, we showed how schema.org and schema.org actions can be used as a vocabulary to annotate Web APIs. We first analysed the schema.org vocabulary in terms of lightweight semantic web services. Afterwards, we explained a simple methodology for creating mappings and implementing a wrapper that does the lifting and grounding for individual APIs. Additionally, we also showed how APIs annotated with actions can be consumed by automated agents like dialogue systems.

Ideally, the APIs themselves should be semantically annotated to eliminate the need for a wrapper described in this paper which would not scale in long-term. JSON-LD format is very suitable for such APIs, since it is a bridge between RDF and Web APIs; on the one hand it is suitable for resource identification and linking, on the other it is fully compatible with JSON format which is well known by developers and has very good tool support. We think this is the main advantage of JSON-LD, as it is also pointed out in [9].

An obvious limitation of our approach is that schema.org is not expressive enough for every aspect of every domain. This is a trade off at the moment between expressiveness and simplicity. We are trying to solve such issues while not introducing too drastic extensions to schema.org. Another limitation of our approach is the task of mapping creation being very tedious. We will develop tools to recommend actions for resources to speed up the mapping process for the future work.

References

[1] Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., Su, J., Tebet, S., 2005. Semantic Web Services Framework (SWSF) Overview. URL: https://www.w3.org/Submission/SWSF/.

[2] Fensel, D., Bussler, C., 2002. The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications 1, 113–137.

[3] Fielding, R.T., 2000. Architectural Styles and the Design of Network-based Software Architectures (PhD Thesis). Ph.D. thesis. University of California.

[4] Haupt, F., Leymann, F., Pautasso, C., 2015. A conversation based approach for modeling rest apis, in: 12th Working IEEE / IFIP Conference on Software Architecture (WICSA 2015), Montreal, Canada. URL: http://wicsa2015.org/wicsa-sessions.html$#$WS5.

[5] John, D., Rajasree, M.S., 2013. RESTDoc: Describe, Discover and Compose RESTful Semantic Web Services using Annotated Documentations. International Journal of Web & Semantic Technology (IJWesT) 4. doi:10.5121/ijwest.2013.4103.

[6] Kärle, E., Simsek, U., Akbar, Z., Hepp, M., Fensel, D., 2017. Extending the schema. org vocabulary for more expressive accommodation annotations, in: Information and Communication Technologies in Tourism 2017. Springer, pp. 31–41.

[7] Keller, U., Lausen, H., 2006. Functional Description of Web Services. Technical Report. Digital Enterprise Research Intitute (DERI).

[8] Lanthaler, M., Gutl, C., 2011. A semantic description language for RESTful Data Services to combat Semaphobia, in: 5th IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2011), IEEE, Dajeon, Korea. pp. 47–53.

[9] Lanthaler, M., Gütl, C., 2013. Hydra: A vocabulary for hypermedia-driven web APIs. CEUR Workshop Proceedings 996.

[10] Martin, D., Burstein, M., McDermott, D., McIlraith, S., Paolucci, M., Sycara, K., McGuinness, D.L., Sirin, E., Srinivasan, N., 2007. Bringing Semantics to Web Services with OWL-S. World Wide Web 10, 243–277. doi:10.1007/s11280-007-0033-x.

[11] O 'Sullivan, J., Edmond, D., Ter Hofstede, A.H.M., 2005. Formal description of non-functional service properties. Technical Report. Business Process Management Group.

[12] Roman, D., Kopecký, J., Vitvar, T., Domingue, J., Fensel, D., 2015. 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.

[13] Sheth, A.P., 2003. Semantic Web Process Lifecycle: Role of Semantics in Annotation, Discovery, Composition and Orchestration. URL: http://corescholar.libraries.wright.edu/knoesis/33.

[14] Simsek, U., Fensel, D., 2018. Now We Are Talking! Flexible and Open Goal-Oriented Dialogue Systems for Accessing Touristic Services. e-Review of Tourism Research 9.

[15] Verborgh, R., Steiner, T., Van Deursen, D., Coppens, S., Valls, J.G., Van de Walle, R., 2012. Functional Descriptions As the Bridge Between Hypermedia APIs and the Semantic Web, in: Proceedings of the Third International Workshop on RESTful Design, ACM, New York, NY, USA. pp. 33–40. doi:10.1145/2307819.2307828.

[16] Zaveri, A., Dastgheib, S., Wu, C., Whetzel, T., Verborgh, R., Avillach, P., Korodi, G., Terryn, R., Jagodnik, K., Assis, P., Dumontier, M., 2017. smartapi: Towards a more intelligent network of web apis, in: Blomqvist, E., Maynard, D., Gangemi, A., Hoekstra, R., Hitzler, P., Hartig, O. (Eds.), The Semantic Web, Springer International Publishing, Cham. pp. 154–169.