From Open API to Semantic Specifications and Code Adapters

Conference Paper · May 2017

DOI: 10.1109/ICWS.2017.56

3 authors:

Simon Schwichtenberg

14 PUBLICATIONS **71** CITATIONS

SEE PROFILE

Christian Gerth

Hochschule Osnabrück

46 PUBLICATIONS **846** CITATIONS

SEE PROFILE

Gregor Engels

Universität Paderborn

441 PUBLICATIONS **7,244** CITATIONS

SEE PROFILE

From Open API to Semantic Specifications and Code Adapters

Simon Schwichtenberg
Paderborn University
s-lab – Software Quality Lab
Paderborn, Germany
Email: schwicht@mail.upb.de

Christian Gerth
Hochschule Osnabrück
University of Applied Sciences
Osnabrück, Germany
Email: c.gerth@hs-osnabrueck.de

Gregor Engels
Paderborn University
s-lab – Software Quality Lab
Paderborn, Germany
Email: engels@upb.de

***Abstract*—Today, modern IT-systems are often an interplay of third-party web services. Developers in their role as requesters integrate existing services of different providers into new IT-systems. Providers use frameworks like Open API to create syntactic service specifications from which requesters generate code to integrate services. Proper service discovery is crucial to identify usable services in the growing plethora of third-party services. Most advanced service discovery approaches rely on semantic specifications, e.g., OWL-S. While semantic specifications are crucial for a precise discovery, syntactical specification are needed for service invocation. To close the gap between semantic and syntactic specifications, service grounding establishes links between the semantic and syntactic specifications. However, for a large number of web services still no semantic specification or grounding exists.**

In this paper, we present an approach that semi-automates the semantic specification of web services for service providers and additionally helps service requesters to leverage semantic web services. Our approach enables a higher degree of automation than other approaches. This includes the creation of semantic specifications and service groundings for service providers as well as the integration of services for requesters by using our code generator. As proof-of-concept, we provide a case study, where we derive a sophisticated semantic OWL-S specification from a syntactic Open API specification.

***Keywords*-Open API, RESTful, Grounding, Semantic Web Services, Lifting, Lowering, Adapter Generation, Service Invocation**

I. Introduction

Modern IT-systems of today are often a loose interplay of different IT services. Such services can be assembled in manifold ways to shape a new IT-system. Service-oriented and microservice architectures are highly adopted in industry. In the sense of the Everything as a service (XaaS) philosophy, services may be software components or hardware devices, realized by different technologies. All these services are manufactured by different third-party service providers. These providers tend to use Software as a Service (SaaS) clouds to host their services. In order to interact with third-party services, the Application Program Interface (API) of these services needs to be known. Today, such services are typically realized as RESTful Web Services (Web APIs) [1] and are provided by different manufacturers. Thereby, the services are black-boxes to the requesters, which means that their source code is not publicly available. To make these services accessible, their APIs must be specified. For that purpose, the Open API framework becomes more and more popular with developers, since its generators help to keep the service implementation and documentation consistent.

In the context of this paper, we understand service requesters as developers that want to develop an IT-system. Instead of developing it from scratch, they wish to integrate existing services offered by service providers. The process to find appropriate services that provide the desired functionality is called service discovery. To that extent, a requester queries a service registry of available service offers. A service matchmaker is a computer program that matches a request with service offers from the registry. The service matchmaker finds relevant service offers that (partially) fulfill the request and recommends these offers to the requester. The requester inspects the results and decides for the service offer that fits the needs best and uses it.

Today, SaaS cloud vendors already host a multitude of third-party services. In the future, these cloud vendors will act more and more as brokers that bring requesters and providers together. For that purpose, a proper service discovery is crucial. Service discovery approaches that are category- or keyword-based are usually too coarse-grained and still need much manual effort on the part of the requester. More fine-grained approaches match operation signatures that are defined in a syntactic specification. Such approaches are prone to retrieve false positive or false negative results. This is because the names of operations, parameters, and types are ambiguous and service matchmakers cannot access their semantics.

Semantic service specifications, e.g., Web Ontology Language for Services (OWL-S) specifications, solve the problem of ambiguity by conceptualizing things and their relations in ontologies. To give an example, the meaning of `Plane` is ambiguous, because it might refer to an airplane or a two-dimensional surface. Instead of using an ambiguous string like `Plane`, seman-

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901)

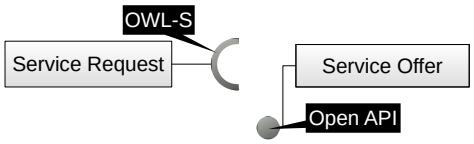

Figure 1. Mismatch of Semantic Request and Syntactic Offer

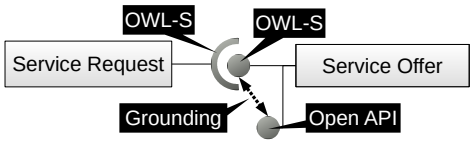

Figure 2. Deriving Semantic Specifications and Groundings

tic service models use unique Uniform Resource Identifiers (URI) like http://dbpedia.org/resource/Aircraft to define the meaning of things. Such URIs can also be used to define relations between things, like a relation between `Flight` and `Airport`: ⟨schema:Flight, schema:arrivalAirport, schema:Airport⟩.

Once a requester has discovered a service with the help of a service matchmaker, he wants to integrate this service into his IT-solution. While services are found by their semantic specification, the syntactic specification is still needed for the concrete invocation of the service. Service groundings connect the semantic and the syntactic specification. Commonly, these groundings are specified as Extensible Stylesheet Language Transformations (XSLT).

Although the discovery of semantically specified web services would lead to better search results and a higher reuse of existing services, there is still a large number of provided services for which only a purely syntactical specifications like Open API exists. For example, the Mashape Marketplace[1] is an API repository containing more than 1,800 public APIs which are described purely syntactically. Fig. 1 illustrates the mismatch between the specifications that are requested and those that are provided.

We propose an approach to derive semantic service specifications and groundings semi-automatically from syntactic specifications that are usually available anyways. Thereby, we obtain semantic specifications and enable an advanced service discovery even for services, which are currently only syntactically specified. In addition, our approach includes a code generator for requesters. This code generator eases the integration of semantic web services into an IT-solution by employing the adapter design pattern [2] to realize the service grounding between the semantic and the syntactic service specification. Fig. 2 illustrates how derived semantic specifications and groundings enable service discovery and invocation.

In the remainder, we introduce syntactic (Sect. II) and semantic specifications (Sect. III), discuss related work (Sect. IV), and present our solution in detail (Sect. V).

[1]https://market.mashape.com

II. Syntactic Specifications

Typically, web services are black-boxes, which means that their source code is not publicly available. To integrate an existing service, a requester relies on a specification of a service in order to integrate and invoke it. A syntactic specification describes exclusively the syntactic characteristics of a service. In context of this paper, we define the essential elements of a syntactic specification as follows:

(1) Operations, (2) Inputs of operations, i.e., the data an operation consumes, (3) Outputs of operation, i.e., the data an operation produces, (4) Types of in- and outputs, i.e., the data schema that prescribes all valid inputs and outputs.

One of the most popular languages to describe Web Services is the Web Service Description Language (WSDL). WSDL originally targeted the Remote Procedure Call (RPC) architectural style and the exchange of XML-based Simple Object Access Protocol (SOAP) messages. Since version 2.0, WSDL was extended to support RESTful Web Services. Analogously, the Web Application Description Language (WADL) [3] may be used. However, these specification languages have rarely been adopted in industry for the specification of RESTful Services [4]. Instead, RESTful web services are often described by custom-made HTML pages that do not have a uniform structure. The hRESTS microformat [5] has been proposed to annotate parts of an HTML documentation that describe certain syntactic elements of a web service.

Nowadays, the rising popularity of RESTful web services comes hand in hand with an increasing popularity of RESTful API specification languages and frameworks like RESTful API Modeling Language[2] (RAML), API Blueprint[3], OData[4], Open API[5] (formerly known as Swagger), etc.

In the remainder, we focus on Open API for illustration purposes. However, our approach can be adopted for other syntactic specification languages. Open API is platform-independent and comes with several tools which can generate program code, documentation, and test cases from Open API specifications. The code generator can generate client- and server-side code and are available for different programming languages, e.g., Java.

Open API provides several advantages for service providers. First, the server code and documentation generators help developers to keep the documentation and implementation consistent. Second, the client code generator eases the integration of Open APIs for requesters.

The language definition is available as a JSON schema[6].

[2]http://raml.org/

[3]https://apiblueprint.org/

[4]http://www.odata.org/

[5]https://www.openapis.org/

[6]https://github.com/OAI/OpenAPI-Specification/blob/master/schemas/v2.0/schema.json

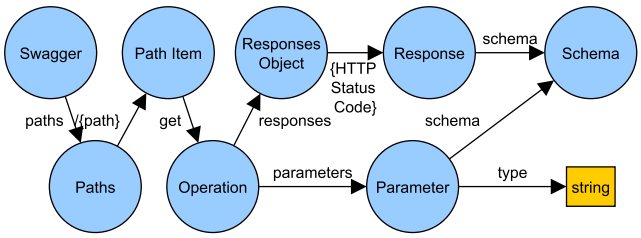

Figure 3. Open API Language (Excerpt)

```
"paths": {
  "/references/airports/nearest/{latitude},{longitude}": {
    "get": {
      "description": "Find the 5 closest airp...",
      "parameters": [
        {
          "name": "latitude",
          "type": "integer",
          "format": "int32",
          "description": "Latitude in decimal format..."
        },
        {
          "name": "longitude",
          "type": "integer",
          "format": "int32",
          "description": "Longitude in decimal format..."
        },
        {
          "name": "lang",
          "type": "string",
          "description": "Optional query parameter..."
        }
      ],
      "responses": {
        "200": {
          "description": "",
          "schema": {
            "$ref": "#/definitions/AirportResource"
```

Figure 4. Syntactic Specification for Lufthansa Open API

Fig. 3 is a visualization of this JSON schema. Operations can be described by `Path Items`, which describe the URI to be called, and an `Operation`, which describes an HTTP method, e.g., `get`. Inputs are described by `Parameters` and outputs by `Responses`. The `type` property must be used to define primitive types of `Parameters`. In addition, the `schema` property can be used to provide a JSON schema for complex types of `Parameters` and `Responses`.

To give an example, the airline Lufthansa runs a RESTful web service that provides current information on flight connections, flight status, etc[7]. Furthermore, there is an operation `Nearest Airports` that returns the five nearest airports for a given location. Fig. 4 is an excerpt of the Lufthansa API specification[8]. We have complemented the original version of the specification by JSON schemas for response messages by using an online tool[9].

A developer who wants to use the Lufthansa API can generate the client code from the Open API specification for the desired target platform, e.g. Java. For example, the client code comprises a Java class that exposes the API of

[7]http://developer.lufthansa.com

[8]https://github.com/LufthansaOpenAPI/Swagger/blob/master/LH_public_API_swagger_2_0.json

[9]http://jsonschema.net

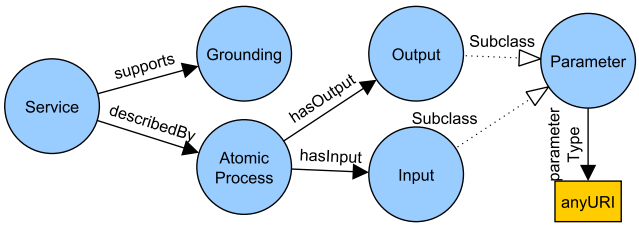

Figure 5. OWL-S Service Description Ontology (Simplified Excerpt)

the web service as Java API. That Java API has a method for each service operation. These Java methods ease the construction of HTTP request and the (de-)serialization of JSON responses to Java objects, etc.

On its own, service specifications based on Open API are purely syntactic. To run an accurate service discovery, the semantics of the services must be considered in order to reduce ambiguity and heterogeneity.

III. Semantic Web Services

Semantic web services have been introduced to make the semantics of services accessible to computers. Service matchmakers can utilize the semantic definitions, e.g., to improve the accuracy of service discovery.

An ontology is a formal specification of a conceptualization. The concepts of an ontology describe entities (`Classes`) of a certain domain and their relations (`Properties`). Each concept is defined by a unique URI. When the URI of a concept is referenced, the semantics of that concept are uniquely determined. Thereby, the semantics become accessible for computers. In the case that multiple different ontologies may be relevant, existing equivalent concepts from these different ontologies can also be related, i.e., ⟨dbpedia:Airport owl:equivalentClass schema:Airport⟩

Semantic service specifications define the semantics of concrete services. In the context of this paper, we focus only on the essential service elements, i.e., operations, inputs, outputs, and types. Semantic service specifications are defined in terms of a service description ontology that defines what a service is. Concrete services can be described by using this ontology. The types of inputs and outputs are defined in an additional domain ontology that defines the business objects of concrete services.

Several languages have been proposed to describe semantic web services, e.g., Web Ontology Language for Web Services (OWL-S) [6], WSML [7], SA-REST [8], WSMO-Lite [9], Hydra [10], etc.

Fig. 5 shows a simplified excerpt of the OWL-S service description ontology. Operations correspond to `AtomicProcesses`, which have `Inputs` and `Outputs`. The types of `Inputs` and `Outputs` are defined by a `parameterType` that refers to any URI of a domain

ontology concept. Sect. V-A provides an exemplary OWL-S specification for the Lufthansa service.

Service matchmakers can benefit from semantic specifications in two ways: First, requests and offers are uniformly described according to the service description ontology. As a consequence, their operations, inputs, and outputs can be compared more easily. Second, domain ontologies define the semantics and relations of business objects which helps to determine corresponding input / output types. When requester and providers use the same or equivalent concepts to describe their shared business objects, a matchmaker matches requests and offers more accurately and retrieve more relevant search results. In [11], an overview of existing semantic service matchmakers is presented. The performance of service matchmakers was repeatedly evaluated, e.g., in terms of the Semantic Service Selection (S3) contest [12].

A semantic service specification is an abstraction of a concrete realization of a service. Its purpose is to advertise the service to requesters and to improve the service discovery. However, when it comes to the integration of these potentially fitting services into an IT-system, a service requester has to understand how a particular service can be used, i.e., the requester must inspect the syntactic specification. Again, the requester needs to come from the abstract semantic service specification to the concrete syntactic interfaces. This gap can be closed by a service grounding, which links corresponding elements from the semantic and syntactic specifications.

The OWL-S has built-in support to ground semantic ontologies to syntactic WSDL specifications. WSDL-S [13] and SAWSDL (Semantic Annotations for WSDL and XML Schema) [14] extend the WSDL by attributes that can be used to annotate WSDL elements with concepts of domain and service description ontology. Hydra-enabled web services produce JSON responses that are annotated with links to domain ontologies [15].

As mentioned earlier, operations, inputs, and outputs are described in semantic and syntactic specifications. In OWL-S, the `Grounding` is used to link, e.g., `AtomicProcesses` to WSDL operations. The input and output types in syntactic specifications are described by a JSON or XSD schema. In contrast, the types of inputs and outputs of a semantic specification are usually concepts of a domain ontology, describing the service's business objects.

When requesters and providers agree on a common domain ontology like schema.org, it is easier during the service matchmaking to determine the shared business objects. The data schemas of web services and common domain ontologies are most likely heterogeneous because both might use different terminologies or logical structuring to describe the same domain. Often complex concept mappings are required to overcome ontological heterogeneity.

In OWL-S, the inputs and outputs of each operation are grounded to respective WSDL messages by providing two Extensible Stylesheet Language Transformation (XSLT) transformations: The lowering transformation transforms OWL-S input properties into WSDL inputs while the lifting transformation transforms OWL-S output properties into WSDL output message parts.

It is very costly to create semantic specifications and groundings manually. Several semi-automatic approaches have been proposed that support service providers to create such semantic specifications and groundings. These approaches are discussed in the next section.

IV. Related Work

Semantic web services have been studied intensively and much related work was developed in this area. In this section, we focus on related work that assists service providers to create semantic specifications or groundings.

JXML2OWL [16] is a tool that can be used to manually create mappings between an XML schema and an OWL ontology. These mappings can be exported as XSLT transformation scripts. The tool can also automatically generate XSLT transformations from these mappings.

Klímek et al. [17] propose a method that can be used to map an external domain ontology like schema.org to an existing conceptual model. XSLT-based lifting and lowering transformations can be generated from this conceptual model. The method supports 1:1 element correspondences between the conceptual model and the ontology. These correspondences have to be provided manually.

Sabou et al. [18] present a semi-automatic method to extract domain ontologies from textual web service descriptions. The extracted domain ontologies are used to classify web services.

The METEOR-S Web Service Annotation Framework (MWSAF) [19] is an approach to annotate WSDL specifications with semantics. XML schemas contained in a WSDL specification are matched with several domain ontologies. The best matching ontology is used to classify the web service.

ASSAM [20] includes a WSDL annotator that automatically suggest ontology concepts to annotate each WSDL element. Multiple machine learning classifiers are trained on already annotated service to determine relevant concepts for unannotated services. The annotations can be exported as OWL-S which also includes XSLT lifting and lowering transformations.

SWEET [4] is a tool to annotate hRESTS specifications according to a service description ontology. The tool integrates a semantic search engine to search ontologies describing the service's business objects. Each business object can be mapped to different ontologies which are not necessarily interlinked. Furthermore, it cannot be guaranteed that requesters and providers use the same or equivalent concepts to describe the same business objects.

In [21], we present an approach to normalize semantic request and offer specifications that use heterogeneous domain ontologies. An ontology matchmaker aligns these ontologies and produces a relational Query View Transformation (QVTr) [22] script which resolves the heterogeneity and eases service matchmaking.

Most approaches for the semantic annotation of syntactic specifications address RPC web service that are described in the XML-based languages WSDL and XSD. In contrast, the semantic specification of RESTful services are little studied. Usually, such services are not described in WSDL but rather in HTML, Open API and the like. Often, RESTful services use JSON instead of XML to exchange data. This complicates the usage of SAWSDL- and XSLT-based groundings. Approaches that extract domain ontologies from specifications barely consider that business objects of requesters and providers are preferably in a semantic relation that can be exploited in service discovery. To the best of our knowledge, there is no holistic approach that semi-automates the derivation of semantic specifications from syntactic specifications, their linking to existing ontologies, and the creation of service groundings. In the next section, we introduce our solution that aims to overcome these limitations.

V. Solution

In our previous work [23], we sketched an architecture for dynamic IT service markets, where a SaaS cloud vendor plays a central role as a broker between requesters and providers. In particular, this cloud vendor supports on the one hand providers to create semantic service specifications and on the other hand requesters to find suitable services and to integrate these into their IT-solutions. The approach that is presented in this paper takes this general architecture up and blends into it.

Fig. 6 illustrates the process how the cloud vendor assist service providers to derive semantic specifications and groundings from syntactic specifications. The steps 1-4 are described in more details in Sect. V-A. Furthermore, Fig. 6 also shows how the cloud vendor supports requesters to integrate services into their IT-solutions as if they were semantic web services. The steps 5-6 are described in more details in Sect. V-B. We illustrate our approach on the basis of the Lufthansa Open API.

A. Deriving Semantic Specifications and Groundings

Our approach assumes that a provider already has a syntactic specification for a service at hand, e.g., an Open API specification. The process of the SaaS cloud vendor to derive a semantic specification and groundings starts when the provider publishes his syntactic specification. Since syntactic and semantic specifications describe operations, inputs, outputs, and types, these elements of an Open API

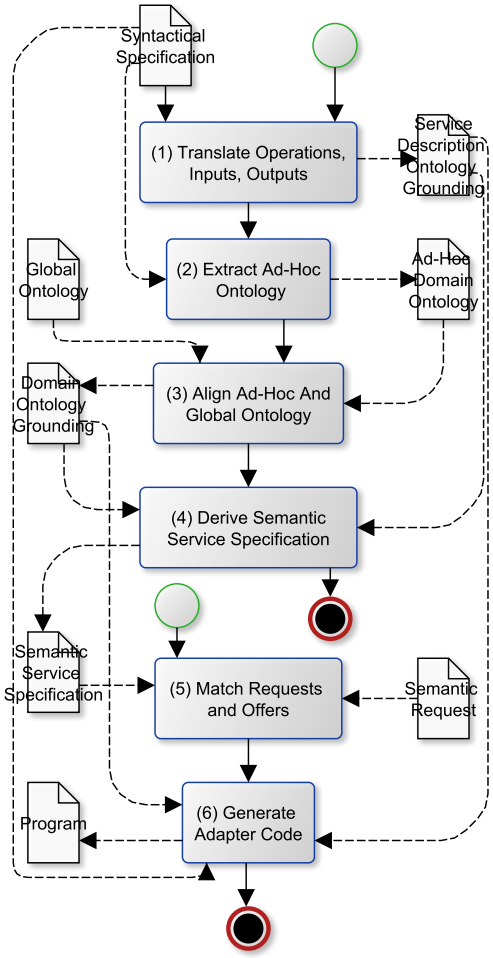

Figure 6. Process of the SaaS Cloud Vendor

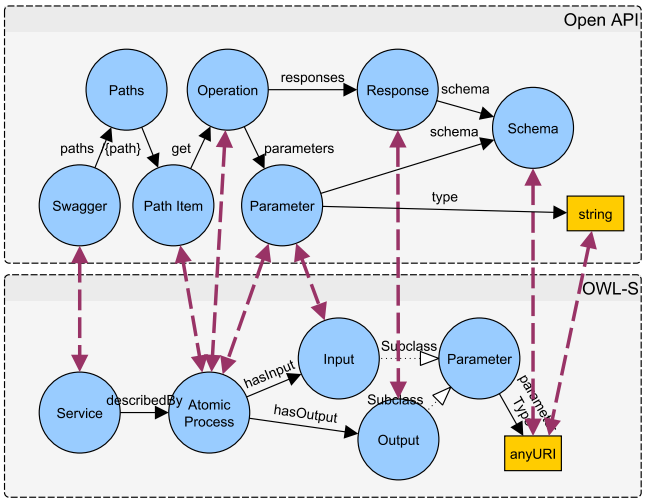

Figure 7. Translate Operations, Inputs, and Outputs to OWL-S

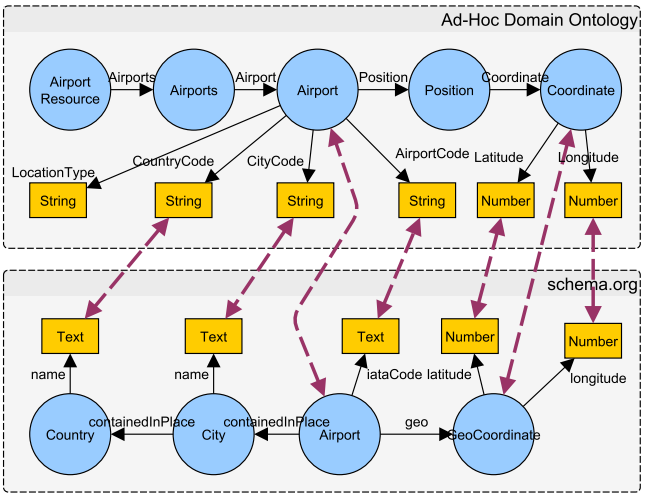

Figure 8. Align Ad-hoc and Global Domain Ontology

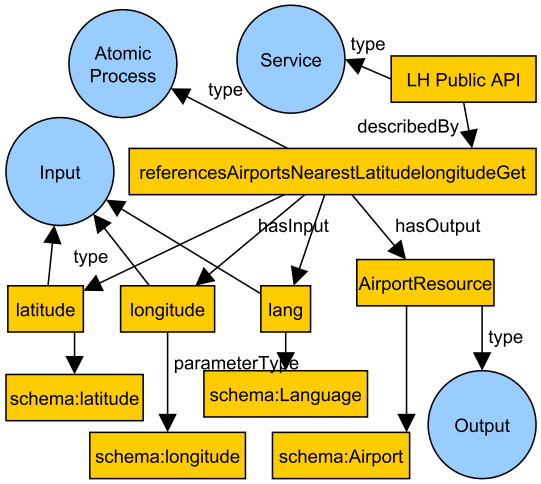

Figure 9. Derived OWL-S Specification for Lufthansa API

specification can be translated fully automatically in respective elements of an OWL-S specification (Step 1). Fig. 7 shows how we map operations, inputs, and outputs of Open API and OWL-S.

To map `AtomicProcesses`, `Inputs`, and `Outputs` to corresponding methods, input arguments, and return values of the client code, we use exactly the same identifiers as produced by the Open API client generator. Until now, we have syntactically converted an Open API into an OWL-S specification.

Extracting Domain Ontologies from JSON Schemas: The purpose of JSON schemas is to define the structure of data in order to ensure its validity. Often, structured data represents a cross-section of the domain ontology and different ontological concepts are hidden in a JSON schema. These single concepts have to be extracted from the JSON schema into a domain ontology so that it can be used in a semantic service specification. To this extent, we first create an ad-hoc domain ontology from the JSON schema that is contained in the syntactic specification (Step 2). For each property in the JSON Schema, a new concept is added to the ad-hoc ontology. Since these concepts are newly created, they are not interlinked with the concepts of other domain ontologies. The extracted ad-hoc domain ontologies could already be used together with existing service matchmakers. However, these ontologies are just different representations of data schemas and therefore no ontological semantics are gained. Consequently, semantic service matchers cannot utilize reasoning techniques due to the lack of explicit semantic relations across different ad-hoc domain ontologies.

In our approach, the SaaS cloud vendor provides a global domain ontology like schema.org, which requesters and providers use as a basis to describe their business objects within their semantic specifications. This simplifies to find corresponding input / output types of service requests and offers during service discovery.

Since ad-hoc ontologies and public domain ontologies like schema.org are most likely heterogeneous, it is tedious for the providers to align the ad-hoc ontology with the public domain ontology. Thus, we propose to use an ontology matcher [24] to help the provider to determine correspondences between the ad-hoc and global ontology (Step 3). The ontology matchmaker produces an ontology alignment, i.e., a set of corresponding concept pairs that represents the domain ontology grounding. Fig. 8 shows an alignment of the ad-hoc ontology and the global ontology schema.org, where the dashed arrows indicate particular concept mappings.

Ontology matchers usually uses heuristics like name similarity metrics to find correspondences. Consequently, the mapping from the global ontology to the ad-hoc may contain false positive or false negative correspondences. The provider can manually adjust this alignment by removing automatically suggested mappings or by adding new mappings. Furthermore, it might happen that some concepts of the ad-hoc ontology have no equivalent counterpart in the global ontology, which we leave unlinked.

The service description ontology and domain ontology groundings provide all information that is needed to derive a semantic service specification (Step 4). Fig. 9 shows the OWL-S specification for the Lufthansa API that is the final result of our approach. This semantic specification is published together with the grounding. The semantic service specifications we gained enable advanced service discovery with the help of service matchmakers that require, e.g., OWL-S specifications. In the next section, we describe how requesters can use these semantic specifications and groundings to integrate a service into their IT-solutions.

```
public Airport referencesAirportsNearestLatitudelongitudeGet(GeoCoordinates geoCoordinates, Language language){
  Float latitude = geoCoordinates.getLatitudeAsNumber().floatValue();
  Float longitude = geoCoordinates.getLongitudeAsNumber().floatValue();
  String lang = language.getNameAsText();
  AirportResource = openapi.referencesAirportsNearestLatitudelongitudeGet(latitude, longitude, lang);
  Airport airport = Factory.INSTANCE.createAirport();
  airport.setIataCodeAsText(airportResource.getAirportResource().getAirports().getAirport().getAirportCode().name());
  airport.setGeo(geoCoordinates);
  City city = Factory.INSTANCE.createCity();
  city.setName(airportResource.getAirportResource().getAirports().getAirport().getCityCode().name());
  ...
  return airport;
}
```

Figure 10. Generated Java Adapter Code

B. Adapter Code Generation

Service discovery starts with a requester that creates a semantic request. We assume that the requester uses concepts from the global ontology. The requester queries the SaaS cloud to find services specifications that match the request (Step 5). A service matchmaker finds corresponding operations, inputs, outputs, and types of requests and offers. The services are ranked by their relevance and shown to the requester. In general, a higher ranked offer has many correspondences with the request. The requester inspects these recommended services and selects the service(s) that best fits their needs. Up to this point, the described process conforms to common service discovery.

Now the requester needs to integrate this service into his IT-solution. The requester has selected a certain service on the basis of what was advertised in the semantic specification and wants to integrate this API like described in the semantic specification. However, the semantic specification serves only the purpose for service discovery. To integrate the service, the requester must retrace the correspondences between the semantic and syntactic specification.

Our approach comes with a client code generator that allows the requester to use the selected service as if it was a semantic web service. The requester can trigger the code generation at the SaaS cloud vendor and download the generated client code. This code implements the API of the advertised semantic web service and translates calls to this API to the actual API, which was described in the syntactic specification (Step 6). This adapter code actually realizes the lifting and lowering and is generated from the groundings.

Fig. 10 shows the adapter code that was generated from the derived OWL-S specification. Note that the signature of the operation is aligned to schema.org (line 1). The lowering is performed in lines 2-4. In line 5, the Java API that was generated with the Open API client code generator is called with the input data. The lifting are performed in lines 6-12.

After the requester has downloaded the generated code, he needs to tailor it according to his needs. He might need to manually adjust data conversions, formatting, or aggregations that could not be automatized by the code generator.

VI. Conclusion

Today, most web services are purely syntactically described, e.g., in terms of Open API specifications. However, advanced service discovery approaches require semantic service specifications.

In this paper, we present an approach that supports the semi-automatic creation of semantic service specifications out of purely syntactic service specifications. Additionally, our approach also supports service requesters to integrate semantic web services into their IT-systems.

In a case study, we derive a semantic OWL-S specification from a syntactic Open API specification. We extract domain ontologies from JSON schemas that are part of the syntactic specifications. Ontology matchers are used to relate the extracted ontology to a global ontology. During the extraction process, links between the syntactic and semantic specifications are established, which shape the service grounding.

When it comes to integrate a semantic web service, a requester is supported by our adapter code generator, which emits code that translates between the advertised (semantic) and the actual syntactic API.

Our approach aims to reduce the effort for providers to create semantic specifications and groundings and the effort of requesters to integrate existing services. The seamless integration of the semantic service specifications extraction and normalization, and the code generation enables a high degree of automation. Semantic web technologies are hidden from requesters and providers so that they can focus on technologies they are familiar with.

We have shown the feasibility and practicability of our approach on the basis of the real-world Lufthansa Open API in our case study. In future work, the application of our approach on other RESTful web services from diverse application domains, e.g., music, travel, medical, etc. is investigated. As a next step, the correctness and completeness of the produced domain ontology alignments and the resulting accuracy of service matching is in the focus of these investigations. We will further expand our scope from operations, inputs, outputs, and types to operation protocols, and pre- and post conditions of operations to increase the accuracy of service discovery.

References

[1] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[2] J. Vlissides, R. Helm, R. Johnson, and E. Gamma, "Design patterns: Elements of reusable object-oriented software," *Reading: Addison-Wesley*, vol. 49, no. 120, p. 11, 1995.

[3] M. Hadley, "Web application description language," https://www.w3.org/Submission/wadl/, 2009.

[4] M. Maleshkova, C. Pedrinaci, and J. Domingue, "Supporting the creation of semantic RESTful service descriptions," in *Proceedings of 8th International Semantic Web Conference (ISWC)*, 2009.

[5] J. Kopecký, K. Gomadam, and T. Vitvar, "hRESTS: An HTML microformat for describing RESTful web services," in *Proceedings of IEEE / WIC / ACM International Conference on Web Intelligence*, 2008, pp. 619–625. [Online]. Available: http://dx.doi.org/10.1109/WIIAT.2008.379

[6] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, "OWL-S: Semantic markup for web services," http://www.w3.org/Submission/OWL-S/, 2004.

[7] H. Lausen, J. de Bruijn, A. Polleres, and D. Fensel, "WSML - a language framework for semantic web services," in *W3C Workshop on Rule Languages for Interoperability, 27-28 April 2005, Washington, DC, USA*, 2005. [Online]. Available: http://www.w3.org/2004/12/rules-ws/paper/44

[8] A. P. Sheth, K. Gomadam, and J. Lathem, "SA-REST: semantically interoperable and easier-to-use services and mashups," *IEEE Internet Computing*, vol. 11, no. 6, pp. 91–94, 2007. [Online]. Available: http://dx.doi.org/10.1109/MIC.2007.133

[9] D. Roman, J. Kopecký, T. Vitvar, J. Domingue, and D. Fensel, "WSMO-Lite and hRESTS: Lightweight semantic annotations for web services and RESTful APIs," *J. Web Sem.*, vol. 31, pp. 39–58, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2014.11.006

[10] M. Lanthaler and C. Guetl, "Hydra: A vocabulary for hypermedia-driven web APIs," in *Proceedings of the WWW2013 Workshop on Linked Data on the Web*, 2013. [Online]. Available: http://ceur-ws.org/Vol-996/papers/ldow2013-paper-03.pdf

[11] H. Dong, F. K. Hussain, and E. Chang, "Semantic web service matchmakers: state of the art and challenges," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 7, pp. 961–988, 2013.

[12] M. Klusch, "Overview of the S3 contest: Performance evaluation of semantic service matchmakers," in *Semantic Web Services, Advancement through Evaluation*, 2012, pp. 17–34. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-28735-0_2

[13] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M.-T. Schmidt, A. Sheth, and K. Verma., "Web service semantics – WSDL-S," http://www.w3.org/Submission/2005/SUBM-WSDL-S-20051107/, 2005.

[14] J. Kopecký, T. Vitvar, C. Bournez, and J. Farrell, "SAWSDL: semantic annotations for WSDL and XML schema," *IEEE Internet Computing*, vol. 11, no. 6, pp. 60–67, 2007. [Online]. Available: http://dx.doi.org/10.1109/MIC.2007.134

[15] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, and N. Lindström, "JSON-LD 1.1 – a JSON-based serialization for Linked Data," http://json-ld.org/spec/latest/json-ld/, 2017.

[16] T. Rodrigues, P. Rosa, and J. Cardoso, "Moving from syntactic to semantic organizations using JXML2OWL," *Computers in Industry*, vol. 59, no. 8, pp. 808 – 819, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016636150800064X

[17] J. Klímek and M. Necaský, "Generating lowering and lifting schema mappings for semantic web services," in *Proceedings of the 25th IEEE International Conference on Advanced Information Networking and Applications Workshops, WAINA*, 2011, pp. 29–34. [Online]. Available: http://dx.doi.org/10.1109/WAINA.2011.13

[18] M. Sabou, C. Wroe, C. Goble, and H. Stuckenschmidt, "Learning domain ontologies for semantic web service descriptions," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 3, no. 4, pp. 340–365, 2005.

[19] N. Oldham, C. Thomas, A. P. Sheth, and K. Verma, "METEOR-S web service annotation framework with machine learning classification," in *Semantic Web Services and Web Process Composition, First International Workshop, SWSWPC*, 2004, pp. 137–146. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30581-1_12

[20] A. Heß, E. Johnston, and N. Kushmerick, "ASSAM: A tool for semi-automatically annotating semantic web services," in *Proceedings of the Third International Semantic Web Conference (ISWC)*, 2004, pp. 320–334. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30475-3_23

[21] S. Schwichtenberg, C. Gerth, Z. Huma, and G. Engels, "Normalizing heterogeneous service description models with generated QVT transformations," in *Modelling Foundations and Applications - 10th European Conference (ECMFA)*, 2014, pp. 180–195. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-09195-2_12

[22] *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Version 1.1*, http://www.omg.org/spec/QVT/1.1/PDF/, Object Management Group Std., January 2011.

[23] S. Schwichtenberg and G. Engels, "Automatized derivation of comprehensive specifications for black-box services," in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016 - Companion Volume*, 2016, pp. 815–818. [Online]. Available: http://doi.acm.org/10.1145/2889160.2889271

[24] J. Euzenat and P. Shvaiko, *Ontology Matching*. Springer Heidelberg, 2007, vol. 18.