


RESEARCH

Open Access



Transforming OpenAPI Specification 3.0 documents into RDF-based semantic web services

Wardani Muhamad^{1,2*} , Suhardi^{1*} and Yoanes Bandung¹

*Correspondence: wardani.muhamad@tasstelkomuniversity.ac.id; suhardi@itb.ac.id

¹ School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Jl. Ganesha 10, Bandung 40132, Jawa Barat, Indonesia
Full list of author information is available at the end of the article

Abstract

Web services are provided with documents that at the very least specify the endpoint, input parameters, and output or response of each operation to expose their capabilities. This should be considered through an understandable format for humans and/or machines. In the Representational State Transfer (REST) architectural style, the OpenAPI Specification (OAS) is used as a reference to create web service descriptions. However, it only supports syntactic interoperability, leading to the incapability of supporting the automated selection process. To overcome this, OAS documents must be enhanced by including semantics to each resource to provide “understandable” services. Therefore, this study aims to develop a system capable of transforming resources in OAS documents into RDF-based semantic web services. To begin, a relational database schema based on the OAS structure is created to store all objects in the OAS document. The published open-linked vocabulary was then used to create the ontology, which maps resources and their relationships on the RDF data model. To build RDF-based semantic web services, R2RML was used to generate the relational database model into triple RDF. The proposed system was also tested through prototyping and using a dataset of 106 OAS documents, which were downloaded from APIs.guru between 5–10 May 2021. The number of triple RDFs generated per document varied with resource rate. An OAS document generates 36 to 16,505 triple RDF in a dataset. The end product was a triple RDF knowledge base maintained by a graph management database. It is now possible to find service operations, input and output parameters, and service composition requirements utilizing the repository semantic web services using SPARQL. On the other hand, the use of relational databases to store OAS resources increased reuse efficiency by approximately 48%, owing to service developers designing interoperability between uniform parameter services, which were then used as input and output.

Keywords: Service composition, Semantic web services, RDF, OpenAPI Specification, Semantic ontology

Introduction

Simple Object Access Protocol (SOAP), Remote Procedure Call (RPC), and Representational State Transfer (REST) are some of the selected approaches or architectural styles to develop web services. However, SOAP and REST are presently the most widely

adopted options, with the main differences focusing on the availability of standardized interfaces. In SOAP-based services, developers should provide interfaces in Web Service Description Language (WSDL) format, although this does not apply to REST-based development. This is because REST offers flexibility and lighter access (without being required to adhere to strict protocols like SOAP-WSDL) to web services. The advantages exhibited by this program over SOAP makes it the most preferred business architectural style for developing web services. In the previous decade, REST-based services gained de facto recognition as a medium for exchanging data on the HTTP-based web and also enabling mechanical processing [1]. This provides several options for accessing the information provided by API, through the Uniform Resource Identifier (URI), query parameters (QP), HTTP headers, and the combination of URIs and QPs [2]. To provide easy access to resources while promoting business services [2], programmers should define the methods to expose the API (through the provision of a URI schema) and its characteristics (supported output formats), as well as provide web service description documentation. This description acts as an understandable interface and service contract to consumers, due to containing all the ideas regarding the web service business activities. Also, it is read and studied by service users to meet required needs [3], such as operations, input and output messages, as well as methods of making service calls [4]. Furthermore, the description of web services becomes the basic element to support various activities carried out by consumers, such as search, composition, and mediation [5].

OpenAPI (OAS) is the main reference towards creating the service descriptions in REST-style programs. This defines generally accepted interface standards and descriptions for all programming languages. The OAS documents also help humans and machines understand the capabilities of service, without accessing source code, additional documentation, or inspecting network traffic. When a web service is properly described using OAS, consumers are found to understand the capabilities of each operation on the API, and subsequently interact without comprehending the implemented logic. Therefore, an operation is technically an endpoint (URI for calling a web API) equipped with an HTTP method, where the provided OAS web service description format is capable of supporting interoperability. This was because the interoperability between web services generally had two levels, namely syntax and semantics. Syntactic interoperability is concerned with the fit between input and output operations, service binding types, data formats, and more. Meanwhile, semantic interoperability aims to produce seamless conditions through the use of machine-interpreted languages. Also, it focuses on solving semantic problems (resource meaning). OAS presently supports syntactic interoperability only, leading to the observation of limitations in understanding web service descriptions, which barely focused on semantics. In OAS documents, syntactical descriptions of the operations are generally supported by web services, to assist with server and client-side code generation [1]. However, the main weakness is the lack of interpretation of the provided resources. To solve these problems, the semantic program becomes a solution as complementary service, leading to the improvement of interoperability. This combines description (syntax) with web semantics, which focuses on publishing metadata within a shared knowledge framework. These semantics subsequently describes the functionality of web services using ontology terminology and annotations, to support automation and dynamic interactions between systems.

Moreover, ontology is generally used to describe a set of concepts and their relationships in a domain, while semantic annotation is a process of adding metadata to the description of web services. Therefore, the addition of semantic descriptions improves the capabilities of web services, by providing the defined interpretations [6] that are easier to read by machines [7]. These are conducted by determining the right vocabulary as a formal semantic provider, which contains a collection of classes representing concepts and properties. Additionally, API acts as a standard for transforming web data and interrelated vocabularies into a mechanism allowing the automatic integration of various services, according to their semantics [8].

Although a formal form for describing service semantics has not been found due to complexity [8], several approaches to create these descriptions have been carried out. This indicates the compilation of semantic description, by adding annotations through JSON-LD [1–4], SA-REST [5] or the Resource Description Framework (RDF) [9–11]. A knowledge graph was created by extracting data from set of web pages and data sources then subsequently converted into structured data, such as triple RDF [12]. Considering the various approaches to the development of semantic web services, the following research questions are raised in this study:

1. What is the appropriate relational database schema that can store data from OAS documents?
2. How to make resources in OAS documents meaningful?
3. What language is used to convert the data model in a relational database to triple RDF?

Meanwhile, the objective of the work is to design a semantic web service sourced from data stored in a relational database and then serve as a knowledge database on the service composition platform. The output is a triple RDF dataset obtained from the extraction of OAS documents published by service providers.

Related work

The service registry holds a central position in helping service-oriented software developers determine candidates for several composed programs. This is generally public and open to all users and programmers [13], due to primarily facilitating the discovery of web services as a basic requirement element. Through the registry, the presence (location) and assembly of services and media, including languages, tools, and machines are often demonstrated, for the required provision of more complex functionality to be met [14]. This subsequently determines the identity and description of each program, as well as other related information such as service group, availability, endpoint, and provider [15]. Although the registry is already storing complete information for each service, software developers still often determine difficulties in compiling complex composite programs, such as specification conformities, data and process compatibilities, as well as capability translations [16]. Therefore, the provision of a capable machine supporting the automation of service search is very important. The availability of complementary service descriptions is the main requirement to meet the automation process. This is because the selection recommendation for the provision of web services to clients

(service-oriented software developers) or developers refer to the program description [17]. Although this description uses a syntactic machine-interpreted language, it is not still equipped with the semantics allowing web services to be “understood”, and also automatically interacting with each other. To support automatic service composition, the semantic description of each resource is the main basis that should be met. This is based on the web semantic becoming a new paradigm that meets the creation of service descriptions. Also, it provides a set of standards and best practices for data sharing and semantics, which are automatically processed and used by various applications [18].

Based on the provision of appropriate resources, the concept of semantic technology led to the creation of the Semantic Web Services (SWS) model, which extended the web service capabilities. According to McIlraith et al. [3], SWS was introduced as an extension of web services through the addition of semantic descriptions. This was to provide a formal declarative definition of the program interface and its characteristics. Furthermore, SWS is a software component that provides dynamic search, composition, and use of web services to users [19]. This enriches the functional description of programs by adding the annotations defined by formal logic-based ontologies, for service-based agents and applications to understand semantics [20]. Semantic annotations are also the result of translating electronic resources through metadata, whose interpretation was formally determined in an ontology [7]. This indicated that the provision of a description or semantic annotation appropriately supported the search, composition, and execution of web services [16]. Without the provision of these specifications and required data, machines were unable to automatically translate and propose the required services [21]. In RESTful web services, the absence of a standard description led to a variety of methods and formats, which were used by programmers to expose annotations. However, these formats were difficult for clients due to accessing a set of web services from different providers, and also adapting to each description [2]. Each developer also has the freedom to compile a service description by providing a web page (hREST), using XML or the OAS rules. This is because the OAS is presently the standard adopted by most programmers, to describe the web API capabilities in JSON or YAML format. At the beginning of this study, the OpenAPI version 3.0 was used, with the OAS documents being an interface for the software developers that used web APIs for their business processes. This document contains operational details, which are used to ensure that clients understand the API endpoint, HTTP method, and other required parameters. Also, the OAS document naturally presents and improves the quality of syntactic service descriptions, leading to the possession of comprehensive semantics for easy automation processes.

Although no formal form has been found to describe semantics due to complexity [21], several previous studies have attempted and proposed the creation of service descriptions, by assigning interpretation to the OAS resources. According to Cremaschi and Paoli [4] and Michel et al. [1], a semantic description model was developed and sourced from the OpenAPI document. This was conducted through the addition of annotations to service descriptions, using the JSON-LD added by programmers before publication. The strategy was found to be appropriate for the service providers that developed complex web services through several different teams. Also, the addition of annotations helped each team to understand the meaning of the resources provided by

the web service. However, when the additional services were provided by a third party, the addition of JSON-LD-based annotations was not the appropriate solution to adopt. Based on Yu et al. [9], the development of the semantic description annotation used the RDF format, with the inclusion of several sections, namely (1) namespace construction, service and operation name, as well as endpoint invocation, (2) service categorization, (3) HTTP method declarations for service calls (GET, POST, PUT, DELETE, or PATCH), and (4) definition of input and output parameters, as well as their reference models. Therefore, registered web services are exposed to several capabilities, using the RDF-based semantic description annotations. As recommended by the World-Wide Consortium (W3C), the RDF and OWL-S schemes should be used as the models for representing data, and also the basis for RDF ontology descriptions [22]. This is due to RDF being used to integrate the web services originating from various sources and formats [23]. According to Guodong et al. [12], a knowledge graph was created by extracting the entities, attributes, and several program relationships of different structural web pages and data sources. These were subsequently converted into structured data, such as triple RDF. The study used RDF to define the semantics of web services, by applying the principle of linked data, where RDF and REST had similarities in identifying programmed resources through the Uniform Resource Identifier (URI). RDF is also a data model and language used to describe web resources [24], due to using a linked vocabulary that defines standard terminology. Based on Heath and Bizer [25], the principle of linked data was to solve the problem of information linkage, by proposing the methods of publishing parameters through a common machine-understandable format (RDF). This was conducted through shared vocabulary with clear semantic definitions and linkages between dataset resources. In this condition, any ambiguous problem in the service description was resolved. Furthermore, a collection of RDF statements formed a connected graph node, where they were graphically stored and queried using SPARQL. According to the service composition plan, the semantics of web services provides a database supporting the combination of program search and selection. Using annotations and a generally understandable semantic language, the definition of service functions and input/output parameters is found to support the program matching requested by the client [26]. Besides being complementary, SPARQL is used to express requests for service composition, and also identify appropriate operations in graphical format [21]. This indicates that a semantic matching mechanism is often implemented to determine the most appropriate service that meets the requirements of clients. Therefore, semantic matching is performed by matching service operations with appropriate input and output parameters [12]. The integration is subsequently used to help search services in the composition process. According to Sferruzza et al. [10] and Lucky et al. [27], OAS 3.0 was used as the basis for building semantic web services. This indicated that the provision of meaning to the OAS components was performed by adding a meta-model [10]. A different approach was also used through the addition of annotations to the OAS document, subsequently specifying vocabulary references to avoid ambiguity [27]. However, both solutions required the modification of the OAS docs, to add semantic descriptions. Furthermore, the creation of unmodified semantics was proposed by [11, 28] although the utilized vocabulary was incomplete and less comprehensive in providing meaning to each web resource.

Research methodology

As illustrated in Fig. 1, this research was conducted in four major stages. The stages of the research began with the application of the database normalization approach in order to create a database schema that corresponded to the components in the OAS document and concluded with triple RDF validation using ShExValidata [29].

The format for describing web service descriptions in OAS documents is JSON or YAML. Both formats share the same attributes, namely the ability to represent objects in attribute-value pair format and the array data type. Arrays are the most basic form of a relation or table in database science. The components of the OAS document, as a semi-structured document, can be extracted into tables that are related to each other in a relational database schema. Although creating a relational database schema can improve understanding and reuse of components in OAS documents, the data stored in the database is isolated (cannot be accessed using another database management system) and has no meaning. The semantic web must be used to give meaning to the data stored in the database. Vocabulary is used in the semantic web to determine semantic classes and the relationships between classes. Furthermore, vocabulary is used to categorize classes, characterize relationships, and define boundaries between classes that are connected.

After each class and relationship has been assigned a meaning, the RDF statement is formatted in a triple form of subject-predicate-object. Subjects and objects in the triple can be resources, which in this study are tables generated during the creation of a relational database schema. After all triples have been generated, the final step is to validate them. ShExValidata, which is available online, was chosen for triple validation in this study.

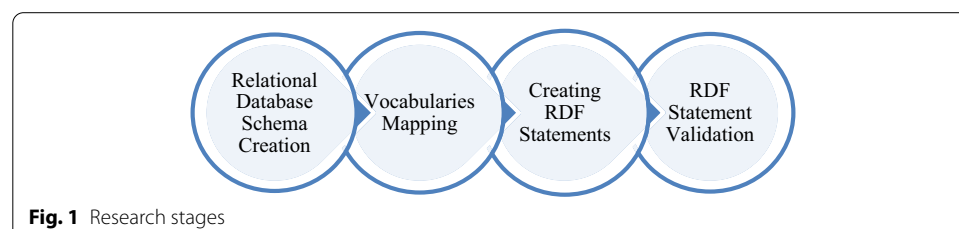
Results and discussion

System framework

To provide a complete and clear illustration for producing RDF-based semantics, a systematic framework was proposed as shown in Fig. 2. This contained three layers, namely data modelling and extraction, as well as RDF information management.

Data modelling

Based on the first layer, data modelling was carried out by creating the appropriate database schema, to store all objects in the OAS document. This was conducted through the implementation of the normalization techniques. The informal database was also designed by translating each OAS object into a table, to capture all OpenApis document fields without considering database constraints such as primary and foreign keys (PK and FK). Moreover, the relationship between the tables was



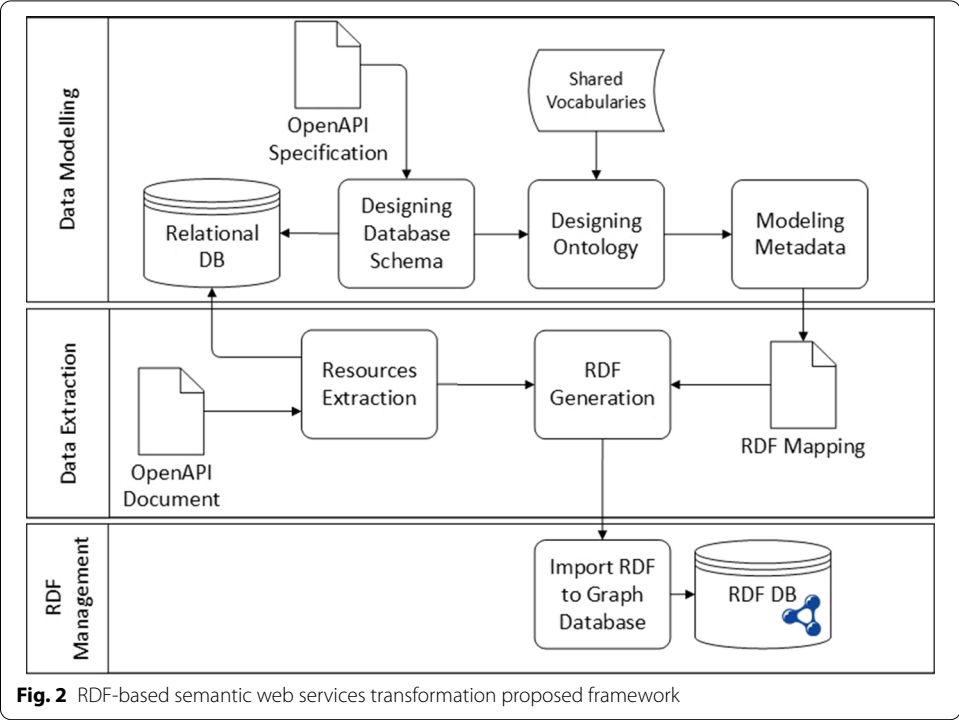


Fig. 2 RDF-based semantic web services transformation proposed framework

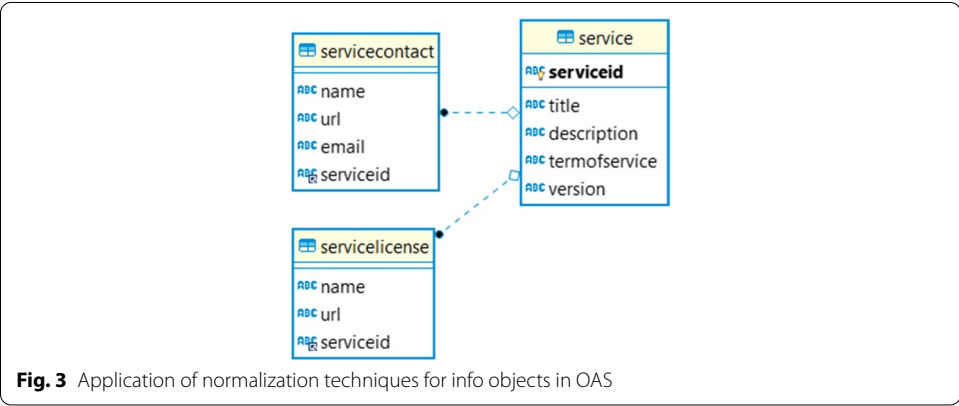


Fig. 3 Application of normalization techniques for info objects in OAS

normally determined. The final result was a relational database design, which had the characteristics of a well-structured table, based on avoiding redundant data, anomaly problems, and manipulation, as well as meeting the normal form (NF) rules [30]. The normalization process was carried out by evaluating the initial table structure (as a parent table) and eliminating repetitions (as a child table). Subsequently, the relationship between the parent and child tables was defined by determining the primary and foreign keys (PK and FK). Figure 3 is an example of applying the normalization technique for the OAS Info section (Fig. 4), to become a relational database schema that was connected from one table to another.

The implementation of the database normalization technique produced 26 tables, which transformed all the parts and objects within the OAS [31]. The results were

Field Name	Type
title	string
description	string
termsOfService	string
contact	Contact Object
license	License Object
version	string

Fig. 4 Structure of the info section on the OAS

Table 1 List of adopted vocabularies

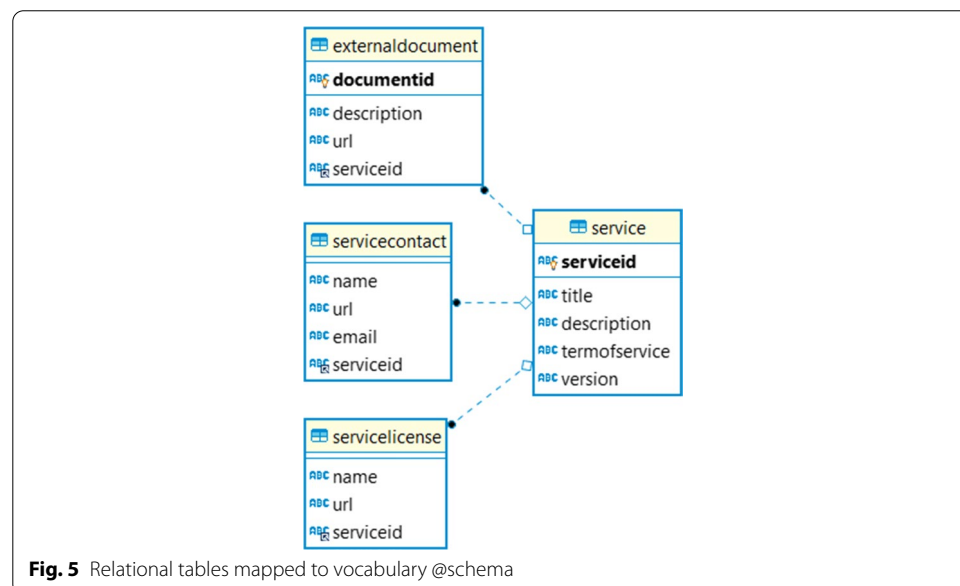
#	Prefix	URI
1	schema	http://schema.org
2	http	http://www.w3.org/2011/http#
3	jsonsc	http://www.w3.org/2019/wot/json-schema#
4	cnt	http://www.w3.org/2011/content#

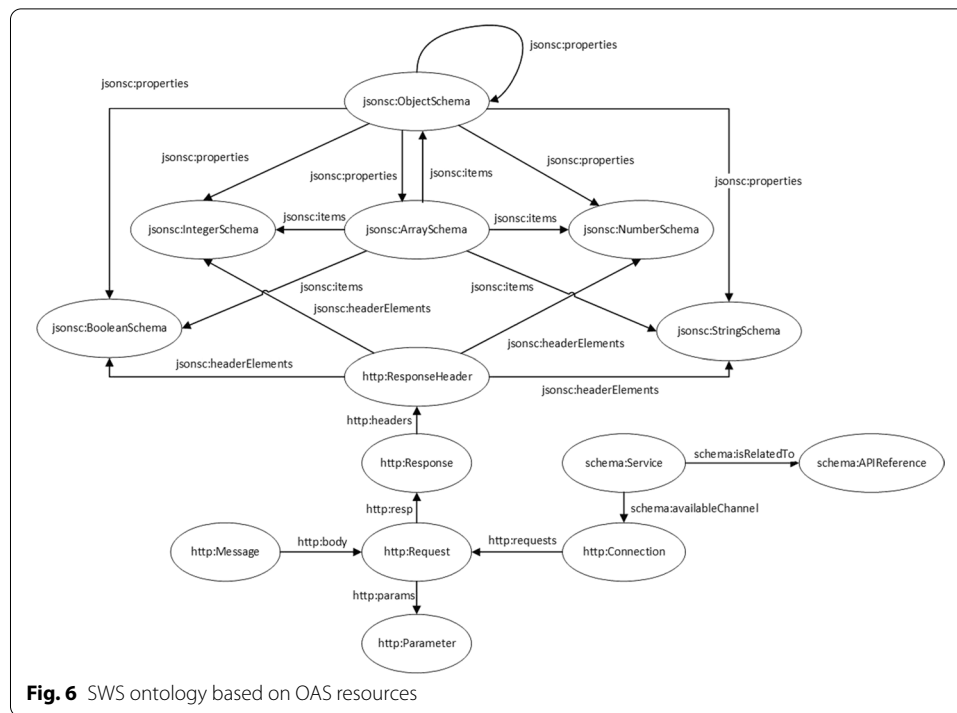
then grouped into strong and weak types, which had the potentials to become a parent and child table, with and without a PK, respectively. Furthermore, the tables within the database became a data model, whose quality was improved through important RDF concepts, which eliminated ambiguity, especially when sending information [32]. This was because RDF supported the interoperability of information exchange between applications, which was mechanically understood and had a data graph format to represent each statement. In a graphical illustration, the nodes and arrows represented entities and their relationships. This explicitly indicated that RDF formed a triple group, namely subject, predicate, and object, to explain a semantic statement. According to the principle of linked data [25], the subject should be a class with a URI, to explain its meaning. The predicate also contained the properties that described the relationship between classes and literal values. Meanwhile, the object was a class or a literal value. This indicated that each subject was connected to the object through the predicate. The selection reference for triple RDF classes and properties was subsequently found in the vocabulary, which described several service elements. Terminology also became a knowledge that expressed the meaning and connection between data, as recent trends related to the formal forms of connectivity provided reusable semantics to support automated composition methods [21]. To describe a resource, the use of common vocabulary facilitated a global understanding of the meaning, according to its relevance irrespective of the origin [33]. Table 1

presents a list of selected vocabulary for the development of the SWS. When compared to [11, 28], using shared vocabularies as the basis for creating ontology has the advantage of knowing the semantics of each class as well as the associated domains and ranges globally. When the resulting RDF dataset is made public, it will be easier to understand the meaning of each resource that has semantics added.

According to Table 1, the classes and properties contained in the vocabulary were mapped to each relational database illustration. This showed that a vocabulary was attached to a table based on the number of classes and properties provided to meet the data meaning needs. For example, the @schema vocabulary was attached to a group of interconnected tables (Fig. 5), to describe the information related to service owners.

Based on Fig. 5, two classes were observed for selection, namely Service and APIReference. The service class describes the programs provided by an organization, e.g., delivery and printing events, etc. This class was adopted to provide meaning to the service table, leading to the emphasis on the interpretation of a web program. Meanwhile, the properties of the Service Class were selected to emphasize the relationship between service, “servicecontact”, and “servicelicense” tables, which were described with their supporting data. Furthermore, the APIReference class is defined as a reference document used to describe an Application Programming Interface (API). This class was very suitable for adoption, to define the meaning of the “externaldocument” table. Subsequently, the relationship between service and externaldocument table was explained by using the “isRelatedTo” property. After all data models are mapped into classes and properties, an ontology capable of describing the positions and relationships between several groups was constructed. According to open standards and data structures, ontology creation aims to identify information connectivity and develop common semantics [34]. Moreover, Fig. 6 describes ontology as the basis for developing appropriate SWS with the data model generated through the selected vocabulary. The ontology is manually constructed by analyzing the suitability of the range, domain and predicate between classes





contained in the chosen vocabulary. This indicates that a labelled arrow represents the relationship between ontology classes. The direction of the arrow also determines the range (object) belonging to a class (acts as a subject) and has a predicate according to the label. For example, the `http:resp` predicate links the `http:Request` class as the subject, with the `http:Response` class being observed as the object.

As a formal form of concept specification and database schema (data model), ontology became the main basis for RDF creation. To translate the database model into RDF, R2RML was used. This is a language recommended by the W3C for customizing relational database mapping (as an RDF data model) into an RDF dataset. Based on this study, the RDF data model was structurally mapped with a vocabulary into a set of triple RDF, which represented resource information (in the semantic context, web services are components that explain the capabilities of web in OAS documents) as the interrelated graphs between one node and another. In R2RML, the input was a relational database model that matched the schema, while the output was a triple RDF according to a predetermined mapping. Moreover, the data model mapped using R2RML was a table, view, or a Structured Query Language (SQL). In creating R2RML mapping, the determination of subject-predicate-object was also determined from each data model represented by relational database tables. When a table was observed as an RDF class, the PK column became an identifier and a reusable resource. Meanwhile, other columns in the table completed the meaning of the resource. For example, three interconnected illustrations were found when observing the table schema (Fig. 3). The service table was found to be the parent and had a “serviced” column acting as a PK. However, the “servicecontact and servicelicense” tables were the child, related to the service table through the “serviced” column. The mapping of the data model subsequently stored in the service, servicecontact, and servicelicense tables using R2RML is described in [Appendix](#). Meanwhile, to

provide an overview of the data model mapping in these tables into a set of triple RDF, it is explained as follows,

Algorithm 1. Mapping for table service and servicecontact into triple RDF

```

01.  Define
02.      class Service = {brand, description, termOfService, version}
03.      class BlankNode = {name, url, license}
04.
05.  set table service as Service
06.  foreach (column) {
07.      if (column is PK) then
08.          set column value as subject in associated class
09.      else
10.          set column value as object and column name as predicate
11.  }
12.
13.  set table servicecontact as BlankNode with Service as domain
14.  foreach (column) {
15.      if (column is identifier) then
16.          set column value as subject in associated class
17.      elseif (column is FK)
18.          set column value as object referred to domain and column name as predicate
19.      else
20.          set column value as object and column name as predicate
21.  }

```

Using R2RML, the mapping of the data model stored in the triple RDF database was carried out by selecting a class in the vocabulary that matched the characteristics of the design. To use the selected class, rr should be added as the R2RML IRI vocabulary namespace. The rules for mapping a table into a triple RDF are described in Table 2.

B. Data extraction

The output generated by the data modeling layer became an artifact within the data extraction layer. Also, the relational databases and RDF metadata models translated into RDF mappings became key artifacts, to support goals at the extraction layer. This was because the data extraction layer had two main objectives, namely (1) extracting the OAS data into the records stored in a relational database, and (2) transforming the records in the relational database into a collection of triple RDE, using the predefined rules on the mapping process. To achieve the intended target, the sequential arrangement of activities is described in Fig. 7.

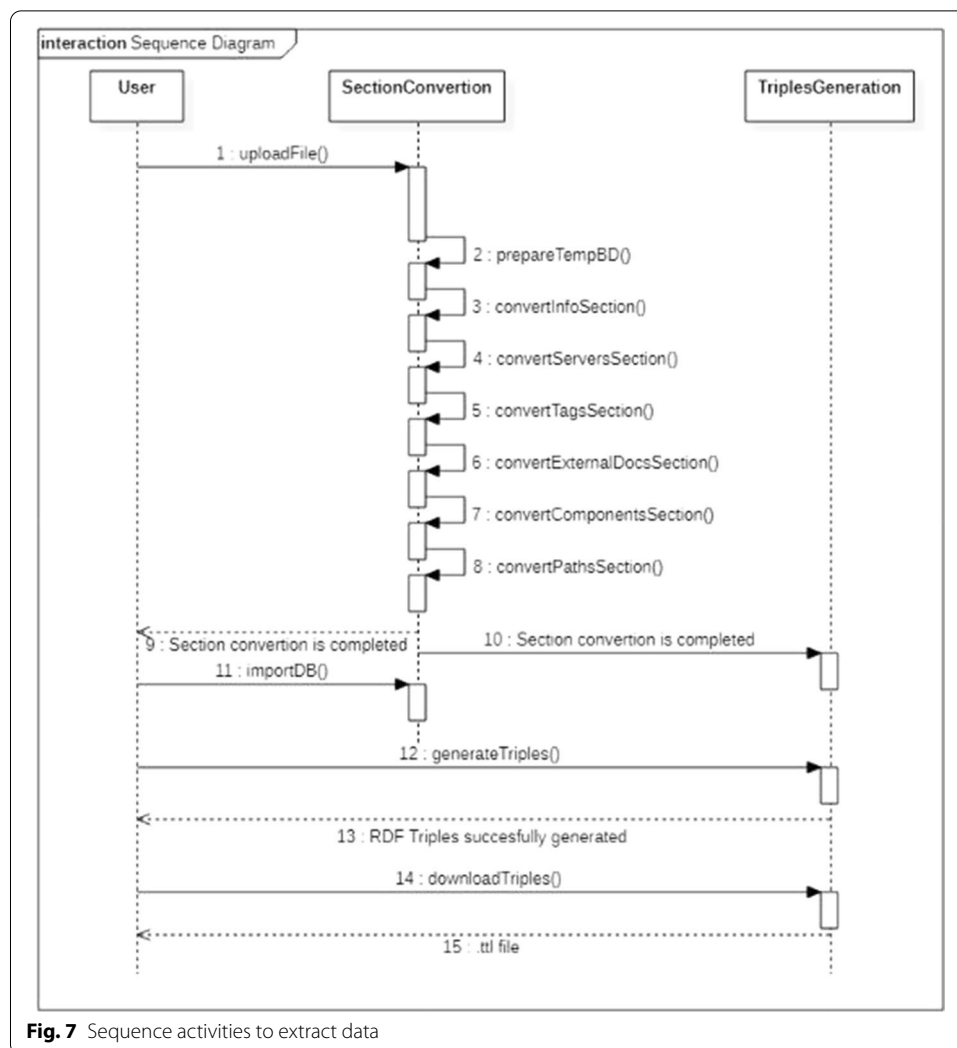
The extraction process began with the upload of the OAS document, which was the source of the information saved to the database. Each section and object in the document was also parsed and saved to appropriate tables. In this study, there were two types of databases with similar structure, namely temporary and production classes, which were used to manage the information obtained from the extraction process. Furthermore, the production and temporary databases stored the extracted data from all and present OAS documents, respectively. To improve the optimization of the production data reuse, the existence of the similar information should be initially evaluated before addition into the temporary system. When the data was found, the records in the production database was copied to the temporary system. However, a new record was added to the temporary database.

Table 2 General mapping rules

Table Type	Column Specification	RDF Form	Column Value Role	R2RML Mapping Template
Strong	Primary key	Subject	As identified resource in URI	<pre> <#_parentMapName_> rr:logicalTable [rr:sqlQuery "_SQL statement_"]; rr:subjectMap [rr:template "BaseURI/{_columnName_}"; rr:class _vocabulary class_ ;]; </pre>
Weak	Identifier	Subject	As a blank node	<pre> <#_childMapName_> rr:logicalTable [rr:sqlQuery "_SQL statement_"]; rr:subjectMap [rr:template "{_columnName_}"; rr:termType rr:BlankNode ;]; </pre>
Strong or weak	Non primary key or non-identifier	Predicate	As a string literal of object	<pre> rr:predicateObjectMap [rr:predicate _vocabulary properties_ ; rr:objectMap [rr:column "_columnName_" ;];]; </pre>
	Foreign key	Predicate	As a resource whose type refers to the parent table	<pre> rr:predicateObjectMap [rr:predicate _vocabulary properties_ ; rr:objectMap [rr:parentTriplesMap <#_parentMapName_> ; rr:joinCondition [rr:child "_columnName_" ; rr:parent "_parentColumnName_" ;]];]; </pre>

To add records to a database table, the consideration of the dependencies and relationships was very necessary. This indicated that the table with the fewest relationships and lowest dependencies obtained the first order within the process of adding records. These were subsequently found in the OAS structure, as an object in a section was reused in another. Therefore, this object should be able to be parsed and stored in the database, due to not causing dependency problems. Based on Fig. 7, the activity numbers 3–8 showed the order of information extraction from the OAS documents into the database. The relationship between the section and the table that stored the data extracted from the document is described in Table 3.

Based on this study, all sections and data were successfully parsed and stored in the database, with the user being provided with the option to import the information within the temporary system. Moreover, data extraction was continuously conducted with the activity of generating triple RDE, using the RDF mapping generated within the modeling layer. As an illustration of the RDF mapping, an example of the

**Table 3** Extraction mapping on OAS document and database schema

Activity	OAS section	Data storage table
convertInfoSection()	info	service, servicecontact, and servicelicense
convertServersSection()	servers	Server
convertTagsSection()	tags	tag
convertExternalDocsSection()	externalDocs	externaldocument
convertComponentsSection()	components	primitiveschema, objectschema, arrayschema, objectproperties, and arrayitem
convertPathSection()	paths	operation, parameter, header, content, response, requestbody, operationrequestbody, primitiveschema, objectschema, array-schema, objectproperties, and arrayitem

Table 4 Record in *service* table

Serviceid	Title	Description	Termofservice	Version
5f5d97f0-b09f-11eb-8c83-4fa8fcf43f30	Google Classroom API	Manages classes, rosters, and invitations in Google Classroom	https://developers.google.com/v1/terms/	v1

Table 5 Record in *servicecontact* table

Name	Url	Email	Serviceid
Google	https://google.com		5f5d97f0-b09f-11eb-8c83-4fa8fcf43f30

Table 6 Record in *servicelicense* table

Name	Url	Serviceid
Creative Commons Attribution 3.0	http://creativecommons.org/licenses/by/3.0/	5f5d97f0-b09f-11eb-8c83-4fa8fcf43f30

data stored in Tables 4, 5 and 6 was presented as a representation of the information within the service, servicecontact, and servicelicense tables.

Based on R2RML mapping described in Appendix, the service table was a resource included in the schema:Service class, which was represented by a value in the serviceid column. However, the other columns in the service table were the predicates reinforcing the interpretation of the schema:Service class. In the RDF generation process, a triple RDF was obtained in the turtle syntax, which matched the record in the service table, as shown below,

RDF Triple 1. RDF Generation from Table *service*

```
@prefix schema: <http://schema.org/>.
@prefix: <http://sws.itbsmartcampus.id/ont#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

:5f5d97f0-b09f-11eb-8c83-4fa8fcf43f30 a schema:Service ;
  schema:brand "Google Classroom API" ;
  schema:description "Manages classes, rosters, and invitations in Google Classroom." ;
  schema:termOfService <https://developers.google.com/terms/> ;
  http://schema:version "v1" .
```

In this study, servicecontact and servicelicense tables were weak illustrations with no PK. Therefore, the stored record was not identified as a resource, although described the service table as a blank node. This indicated that every column besides FK was a predicate connecting the RDF schema:Service class. RDF Triple 2 described the triple RDF as follows:

RDF Triple 2. RDF Generation from Table *servicecontact* and *servicelicense*

```

:5f5d97f0-b09f-11eb-8c83-4fa8fcf43f30 a schema:Service ;
    schema:license
    [
        schema:name "Creative Commons Attribution 3.0" ;
        schema:url "http://creativecommons.org/licenses/by/3.0/"
    ] ;
    schema:contactPoints
    [
        schema:name "Google" ;
        schema:url "https://google.com"
    ] .

```

C. RDF management

The produced set of triple RDF should be managed based on supporting graph processing as the basic form of RDF. This indicated the selection of various open-source graph management systems, including Apache Jena [35], Stardog [36], GraphDB [37], and Neo4j [38]. SPARQL was also a standard language used to perform queries, leading to the collection of the information stored within the RDF store. Based on service search, this language was used to determine the program candidates that matched the composition requirements.

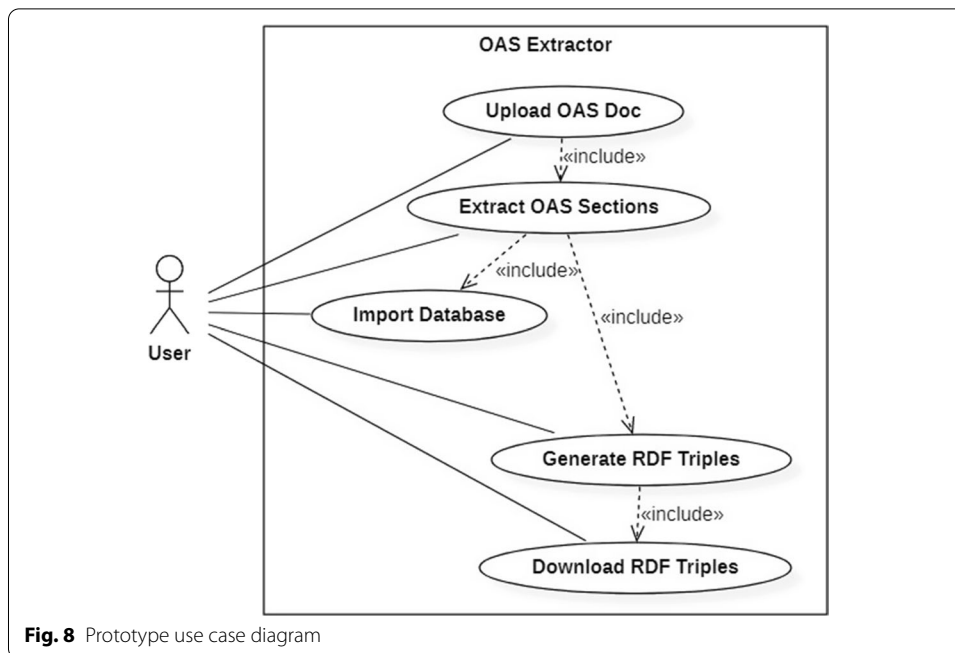
Experiment result

To support system testing, a software prototype was developed which had the ability to convert OAS documents into a triple RDF collection, by initially filling the records in the relational database. Moreover, datasets were generated from the published OAS documents with the development of prototypes. These were useful in supporting the search process in service composition, using SPARQL. Table 7 describes the prototype development environment to support testing.

The main programming language used to develop prototypes was Node.js, with express as its web server. Besides this, several modules integrated with Node.js were also used to support the achievement of the expected capabilities, such as node-jq [39]. This provided the main feature of extracting the OAS documents in JSON format. Also, the Node.js used an open-source engine based on the Java programming language, namely

Table 7 Prototype development environment

Hardware/Software	Specification
Processor	Intel(R) Core(TM) i5-8265U CPU @ 1.60 GHz 1.80 GHz
RAM	8.00 GB
Operating system	Windows 10 Home Single Language
Programming language	Node.js v10.15.3 Library: express v4.17.1 node-jq v1.12.0 r2rmlF
Database Management System (DBMS)	MariaDB v10.4.8



OpenAPI Specification 3.0 Extractor

This prototype is used as a proof of concept for the development of Semantic Web Service sourced from the OpenAPI Specification 3.0 document.

To start the process of extracting the OpenAPI Specification 3.0 document, please upload the .json file via the form below.

Browse...
No file selected.

Upload File
Cancel

Fig. 9 OAS document file upload interface

R2RML-F [40]. This engine satisfied all the specifications presented in R2RML, by creating data mappings to generate triple RDF. As a complement, MariaDB was selected as a database management system, to create a relational structure and also store records of the OAS document processing results. Therefore, the functionality designed in the prototype primarily supported the activities described at the data extraction layer of the system framework, as shown in Fig. 8.

In the uploaded OAS Doc functionality, users were required to upload the documents with JSON extension files. When the upload process was successful, the file was copied to the server. This indicated that a successful file upload was a prerequisite for being able to use the Extract OAS Sections functionality. In this functionality, the extracted data on the OAS document was appropriate with the sequence of processes, as shown in Fig. 7. Any data generated from this extraction process was stored in a table, whose mapping was explained in Table 3. After storage in a relational database, users downloaded the data for documentation or recovery needs, due to the unexpected incidence of damages to the production system. This indicated the abilities to generate a set of triple RDF,

zoom.json File uploaded successfully

#	OAS 3.0 Section	Exec. Time (ms)
1	Info section	Parse section
2	Servers section	Parse section
3	Tags section	Parse section
4	External Documents section	Parse section
5	Components Schemas section	Parse section
6	Components Request Bodies section	Parse section
7	Paths section	Parse section

Fig. 10 OAS document extraction interface

which was managed and transformed from the database information through a graphical system.

The use case illustrated in Fig. 8 serves as a reference for developing a prototype that enables proper and accurate interaction between users and the system, as illustrated in Figs. 9 and 10. The form displayed in Fig. 9 allows users to upload an OAS document file for extraction. The system restricts the file extensions that can be uploaded throughout the upload process, that is, .json. If it does not conform to the specified restrictions, an error notice is produced. After successfully uploading and copying the OAS document file to the server, the user can begin extracting each component of the OAS document. The user must follow a specific procedure while extracting the OAS document, as illustrated in Fig. 10. If the extraction procedure is successful, each section of the OAS document will be saved to the database as a separate record.

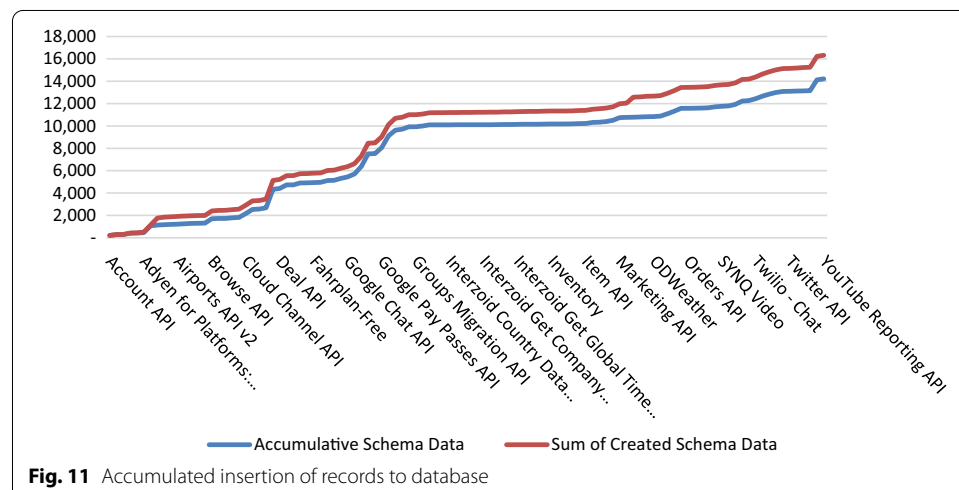
To test the prototype's ability to produce RDF-based semantic web services, a total of 106 OAS documents were selected and downloaded between 05–10 May 2021, to serve as an analytical dataset. The OAS documents were also obtained from APIs.guru [41], where 2283 files were registered by web service owners and other contributors. Based on Table 8, the distribution of OAS documents and the number of generated triple RDFs were described.

Discussion

Based on the experimental process, the OAS document transformation system was appropriately operated. This indicated that the sections within the OAS documents were fully translated into records in a relational database, which were subsequently transformed into triple RDF. According to the creation of a relational database, the benefit obtained was the avoidance of data duplication. Furthermore, the potential for data duplication was found in the schema definition (in relational databases stored in the primitiveschema, objectschema, and arrayschema tables) used in each operation within the OAS document section. The results showed that an operation had at least a

Table 8 OAS document distribution per service provider and number of generated triple RDF

Service provider	Number of services	Number of generated triples
ebay.com	12	11.366
adyen.com	9	9.963
transavia.com	1	437
mastercard.com	2	1.429
gov.bc.ca	1	1.382
deutschebahn.com	4	250
bikewise.org	1	44
regcheck.org.uk	1	59,287
googleapis.com	27	2.741
ticketmaster.com	2	71
exchangerate-api.com	1	1.021
nytimes.com	1	5.208
walletobjects.googleapis.com	1	4.409
getgo.com	2	1.191
instagram.com	1	1.602
interzoid.com	17	1.814
walmart.com	4	69
ip2location.com	1	36
ip2whois.com	1	76
iptwist.com	1	460
isbndb.com	1	341
oceandrivens.com	1	252
omdbapi.com	1	4.477
openchannel.io	1	2.017
openfintech.io	1	360
apache.org	1	434
synq.fm	1	16.505
trello.com	1	11.379
twilio.com	6	2.416
twitter.com	1	310
worldtimeapi.org	1	11.366

**Fig. 11** Accumulated insertion of records to database

response, which returned a status code and a message (in a specified data format) to the service user. This showed that the returned message had one data schema format defined in the OAS document. The data schema is also applied to the parameters and requests for an operation. Considering these characteristics, the reuse of data schemas from a set of OAS documents were predicted to provide efficiency in the addition of records to the database. From the dataset used as a test, an average efficiency of 14.59% was obtained, compared to not reusing similar data scheme. Figure 11 shows the addition of records into the database, for each iteration of OAS document processing.

Based on Fig. 11, two different lines indicated the addition of records in the database. This showed that the red line represented the addition of records, when the data schema reused was not applied to other OAS documents. Meanwhile, the blue line indicated the addition of records when the data schema stored in the identified database was found in the processed OAS document. This efficiency linearly affected the number of triple RDF produced. When the observation was focused on service providers, the efficiency reached 48.28%. This was because programmers had designed high interoperability between services. Through this interoperability, several services interacted with each other (forming a composite service) to meet a more complex business need. Table 9 provides an overview of the data schema reuse efficiency on several services.

Conclusion

Based on the study objective, the produced triple RDF met the requirements for the composition of web services, as a source of knowledge to support service discovery method. In addition, the results obtained resolved all research questions. To accomplish first research question, the SWS design process begins with the design of a relational database by applying the database normalization technique, then extracting the components in the OAS document into data in the relational database. 26 tables are constructed in the relational database to store all of the components contained in the OAS document. Selection of the appropriate shared dictionary to determine resource semantics, and creation of semantic statements in triple format (subject–predicate–object) based on semantic ontologies that relate classes from shared dictionaries becomes the next job after the relational database is generated. The results obtained are a solution to the second research question. To address the last research question, R2RML was selected as the standard language capable of transforming data from

Table 9 Efficient use of data schemes per service provider

Service provider	Number of services	Number of data schemes (without reuse mechanism)	Number of data schemes (with reuse mechanism)	Efficiency (%)
twilio.com	6	984	879	10.67
adyen.com	9	1.666	985	40.88
interzoid.com	17	145	75	48.28
googleapis.com	27	8.993	8.138	9.51
ebay.com	12	1.495	1.324	11.44

relational databases into triple RDF. To transform, a mapping between the relational database's table structure and the matching dictionary was created using the R2RML protocol. SWS is built using the RDF data schema with turtle syntax and is managed in a SWS repository, Apache Jena. Apache Jena is a Java-based open source that can be used to manage triple RDF and supports processing (querying) triple RDF using SPARQL. To support the proof-of-concept design of SWS, a software prototype was built with the main ability to extract OAS documents into relational database schemas and transform data in relational databases into triples. The software prototype only supports the extraction of OAS documents using the JSON format. The OAS document used to test the SWS design and software prototype was obtained from APIs.guru. Although this study generated a triple RDF dataset, it was not made publicly available. As a result, it cannot be used for comparable research. Further work will integrate the proposed framework with a service composition platform to enable seamless service discovery via a knowledge database. On the other hand, it is necessary to develop a more user-friendly user interface that is multi-platform compatible.

Appendix

R2RML mapping for table service, servicecontact and servicelicense

```

01. @prefix rr: <http://www.w3.org/ns/r2rml#> .
02. @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
03. @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
04. @prefix schema: <http://schema.org/> .
05. @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
06. @prefix : <http://sws.itbsmartcampus.id/ont#> .
07.
08. <#Service>
09. rr:logicalTable [ rr:sqlQuery "SELECT SERVICEID, TITLE, DESCRIPTION, TERMOFSERVICE,
    VERSION FROM SERVICE" ];
10. rr:subjectMap [
11.     rr:template "http://sws.itbsmartcampus.id/service/{SERVICEID}" ;
12.     rr:class schema:Service ;
13. ];
14. rr:predicateObjectMap [
15.     rr:predicate schema:brand ;
16.     rr:objectMap [
17.         rr:column "TITLE" ;
18.     ];
19. ];
20. rr:predicateObjectMap [
21.     rr:predicate schema:description ;
22.     rr:objectMap [
23.         rr:column "DESCRIPTION" ;
24.     ];
25. ];
26. rr:predicateObjectMap [
27.     rr:predicate schema:termOfService ;
28.     rr:objectMap [
29.         rr:column "TERMOFSERVICE" ;
30.         rr:termType rr:IRI ;
31.     ];
32. ];
33. rr:predicateObjectMap [
34.     rr:predicate schema:version ;
35.     rr:objectMap [
36.         rr:column "VERSION" ;
37.     ];
38. ];
39. rr:predicateObjectMap [
40.     rr:predicate schema:license ;
41.     rr:objectMap [
42.         rr:parentTriplesMap <#ServiceLicense> ;
43.         rr:joinCondition [
44.             rr:child "SERVICEID" ;
45.             rr:parent "SERVICEID" ;
46.         ]
47.     ];
48. ];
49. rr:predicateObjectMap [
50.     rr:predicate schema:contactPoints ;

```

```

51.      rr:objectMap [
52.          rr:parentTriplesMap <#ServiceContact> ;
53.          rr:joinCondition [
54.              rr:child "SERVICEID" ;
55.              rr:parent "SERVICEID" ;
56.          ]
57.      ];
58.  ].
59.
60.  <#ServiceLicense>
61.  rr:logicalTable [ rr:sqlQuery "SELECT NAME, URL, SERVICEID FROM SERVICELICENSE" ];
62.  rr:subjectMap [
63.      rr:template "{NAME}" ;
64.      rr:termType rr:BlankNode ;
65.  ];
66.  rr:predicateObjectMap [
67.      rr:predicate schema:name ;
68.      rr:objectMap [
69.          rr:column "NAME" ;
70.      ];
71.  ];
72.  rr:predicateObjectMap [
73.      rr:predicate schema:url ;
74.      rr:objectMap [
75.          rr:column "URL" ;
76.      ];
77.  ].
78.
79.  <#ServiceContact>
80.  rr:logicalTable [ rr:sqlQuery "SELECT NAME, URL, EMAIL, SERVICEID FROM
SERVICECONTACT" ];
81.  rr:subjectMap [
82.      rr:template "{EMAIL}" ;
83.      rr:termType rr:BlankNode ;
84.  ];
85.  rr:predicateObjectMap [
86.      rr:predicate schema:provider ;
87.      rr:objectMap [
88.          rr:column "NAME" ;
89.      ];
90.  ];
91.  rr:predicateObjectMap [
92.      rr:predicate schema:url ;
93.      rr:objectMap [
94.          rr:column "URL" ;
95.      ];
96.  ];
97.  rr:predicateObjectMap [
98.      rr:predicate schema:email ;
99.      rr:objectMap [
100.          rr:column "EMAIL" ;
101.      ];
102.  ].

```

Abbreviations

SOA: Service-Oriented Architecture; SWS: Semantic web services; OAS: OpenAPI Specification; REST: REpresentational State Transfer; RDF: Resource Description Framework; R2RML: RDB to RDF Mapping Language; SPARQL: SPARQL Protocol and RDF Query Language; SOAP: Simple Object Access Protocol; RPC: Remote Procedure Call; WSDL: Web Service Description Language.

Acknowledgements

The author is grateful to the Institut Teknologi Bandung and Telkom University, for supporting this study.

Author contributions

The author confirms the sole responsibility for this manuscript based on the following, study conception and design, data collection, analysis, and result interpretations, as well as manuscript preparation. All author read and approved the final manuscript.

Funding

Not applicable. This study received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Availability of data and materials

The original dataset used for this study is available on APIs.guru (<https://apis.guru/>).

Declarations**Ethics approval and consent to participate**

Not applicable.

Consent for publication

Not applicable.

Competing interests

The author reports no potential conflict of interest.

Author details

¹School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Jl. Ganesha 10, Bandung 40132, Jawa Barat, Indonesia. ²School of Applied Science, Telkom University, Jl. Telekomunikasi No.1, Bandung 40257, Jawa Barat, Indonesia.

Received: 23 November 2021 Accepted: 6 April 2022

Published online: 28 April 2022

References

- Michel F, Zucker C, Gargominy O, Gandon F. Integration of web APIs and linked data using SPARQL micro-services—application to biodiversity use cases. *Information*. 2018;9:310.
- Neumann A, Laranjeiro N, Bernardino J. An analysis of public REST web service APIs. *IEEE Trans Serv Comput*. 2021;14:957–70.
- McIlraith SA, Son TC, Zeng H. Semantic Web services. *IEEE Intell Syst*. 2001;16:46–53.
- Cremaschi M, De Paoli F. A practical approach to services composition through light semantic descriptions. In: Krikos K, Plebani P, de Paoli F, editors. *Service-oriented and cloud computing*. Berlin: Springer International Publishing; 2018. p. 130–45.
- Lathem J, Gomadam K, Sheth AP. SA-REST and (S)mashups: adding semantics to RESTful services. In: *International conference on semantic computing (ICSC 2007)*. Irvine: IEEE; 2007. p. 469–76.
- Klusck M. Semantic web service description. In: Schumacher M, Schuldt H, Helin H, editors. *CASCOM: intelligent service coordination in the semantic web*. Basel: Birkhäuser Basel; 2008. p. 31–57.
- Kurniawan K, Ekaputra FJ, Aryan PR. semantic service description and compositions: a systematic literature review. In: *2018 2nd international conference on informatics and computational sciences (ICICoS)*. Semarang: IEEE; 2018. p. 1–6.
- Serrano D, Stroulia E, Lau D, Ng T. Linked REST APIs: a middleware for semantic REST API integration. In: *2017 IEEE international conference on web services (ICWS)*. Honolulu: IEEE; 2017. p. 138–45.
- Yu HQ, Liu D, Dietze S, Domingue J. Developing RDF-based Web Services for supporting runtime matchmaking and invocation. In: *2011 7th international conference on next generation web services practices*. Salamanca: IEEE; 2011. p. 392–7.
- Sferruzza D, Rocheteau J, Attiogbé C, Lanoix A. Extending OpenAPI 3.0 to build web services from their specification. In: *Proceedings of the 14th international conference on web information systems and technologies*. Seville: SCITEPRESS - Science and Technology Publications; 2018. p. 412–9.
- Mainas N, Petrakis EGM, Sotiriadis S. Semantically enriched open API service descriptions in the cloud. In: *2017 8th IEEE international conference on software engineering and service science (ICSESS)*. Beijing: IEEE; 2017. p. 66–9.
- Guodong L, Zhang Q, Ding Y, Zhe W. Research on service discovery methods based on knowledge graph. *IEEE Access*. 2020;8:138934–43.
- Rathod D. REGISTRY FOR RESTful WEB SERVICE: RESTRegistry. *Int J Res-GRANTHAALAYAH*. 2017;5:128–35.
- Papazoglou MP. *Web services: principles and technology*. Boston: Pearson/Prentice Hall; 2007.
- Verma R, Srivastava A. A dynamic web service registry framework for mobile environments. *Peer-to-Peer Netw Appl*. 2018;11:409–30.
- Pedrinaci C, Domingue J, Sheth AP. Semantic web services. In: Domingue J, Fensel D, Hendler JA, editors. *Handbook of semantic web technologies*. Berlin: Springer; 2011. p. 977–1035.

17. Roman D, Kopecký J, Vitvar T, Domingue J, Fensel D. WSMO-Lite and hRESTS: lightweight semantic annotations for Web services and RESTful APIs. *J Web Semant.* 2015;31:39–58.
18. DuCharme B. Learning SPARQL: querying and updating with SPARQL 1.1. 2nd ed. Sebastopol: O'Reilly Media; 2013.
19. Hesami Rostami N, Kheirkhah E, Jalali M. Web services composition methods and techniques: a review. *IJCSEIT.* 2013;3:15–29.
20. Klusch M, Kapahnke P, Schulte S, Lecue F, Bernstein A. Semantic web service search: a brief survey. *Künstl Intell.* 2016;30:139–47.
21. Serrano D, Stroulia E. Semantics-based API discovery, matching and composition with linked metadata. *SOC.* 2020;14:283–96.
22. Jacksi K, Dimililer N, Subhi R. State of the art exploration systems for linked data: a review. *Int J Adv Comput Sci Appl.* 2016;7:155–64.
23. Hougue YPE, Sagbo KAR, Yetongnon K. RDF-based web information integration system: a travel system use case. In: 2018 14th international conference on signal-image technology & internet-based systems (SITIS). Las Palmas de Gran Canaria: IEEE; 2018. p. 500–7.
24. Klyne G, Carroll JJ, McBride B, editors. Resource Description framework (RDF): concepts and abstract syntax. W3C recommendation. <https://www.w3.org/TR/rdf-concepts/>. Accessed 14 Sept 2020.
25. Heath T, Bizer C. Linked data: evolving the web into a global data space. Synthesis lectures on the semantic web: theory and technology. 2011;1:1–136.
26. Chen N, Cardozo N, Clarke S. Goal-driven service composition in mobile and pervasive computing. *IEEE Trans Serv Comput.* 2018;11:49–62.
27. Lucky MN, Cremaschi M, Lodigiani B, Menolascina A, De Paoli F. Enriching API descriptions by adding API profiles through semantic annotation. In: Sheng QZ, Stroulia E, Tata S, Bhiri S, editors. Service-oriented computing. Berlin: Springer International Publishing; 2016. p. 780–94.
28. Mainas N, Petrakis EGM. SOAS 3.0: semantically enriched OpenAPI 3.0 descriptions and ontology for REST services. San Diego, California; 2020.
29. Validata: RDF validator using shape expressions. <https://www.w3.org/2015/03/ShExValidata/>. Accessed 15 Aug 2020.
30. Ponniah P. Database design and development: an essential guide for IT professionals. Wiley-Interscience: IEEE Press; 2003.
31. Muhamad W, Suhardi, Bandung Y. Designing semantic web service based on OAS 3.0 through relational database. In: 2020 international conference on information technology systems and innovation (ICITSI). Bandung - Padang: IEEE; 2020. p. 306–11.
32. Segaran T, Taylor J, Evans C. Programming the Semantic web. 1st ed. Beijing: O'Reilly; 2009.
33. d'Aquin M. Linked data for open and distance learning. Milton Keynes: The Open University, UK; 2012.
34. Pattuelli MC, Provo A, Thorsen H. Ontology building for linked open data: a pragmatic perspective. *J Libr Metadata.* 2015;15:265–94.
35. Apache Jena. <https://jena.apache.org/>. Accessed 1 Sept 2020.
36. Stardog. <https://www.stardog.com/>. Accessed 1 Sept 2020.
37. GraphDB. <https://graphdb.ontotext.com/>. Accessed 1 Sept 2020.
38. neo4j. <https://neo4j.com/>. Accessed 1 Sept 2020.
39. Ackermans M. node-jq. <https://www.npmjs.com/package/node-jq>.
40. Debruyne C, O'Sullivan D. R2RML-F: towards sharing and executing domain logic in R2RML mappings. Montreal, Canada; 2016.
41. APIs.guru. <https://apis.guru/>. Accessed 5 May 2021.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen[®] journal and benefit from:

- Convenient online submission
- Rigorous peer review
- Open access: articles freely available online
- High visibility within the field
- Retaining the copyright to your article

Submit your next manuscript at ► [springeropen.com](https://www.springeropen.com)