Using RESTful API and SHACL to Invoke Executable Semantics in the Context of Core Software Ontology

Xianming Zhang
Aviation Industry Development and Research Center of China
Beijing, China
e-mail: forzxm@163.com

***Abstract*—It is known that executable semantics can be constructed for tasks in the context of some ontologies for computational domain (here, it is Core Software Ontology), but it cannot support that the constructed executable semantics be invoked to launch an actual computing process for returned values. The goal of this paper is to put forward a framework in which Representational State Transfer Application Programming Interface (RESTful API), Shapes Constraint Language (SHACL) and Core Software Ontology can work together to invoke constructed executable semantics and finally get the result. Firstly, a link between RESTful API and Core Software Ontology is necessary and conceptually established. With such a link, we can see that the former will energize and refine the latter. Secondly, this paper investigates how to take advantage of SHACL to invoke RESTful API for returned value. Thirdly, with the help of SHACL, we can see how RESTful API energizes Core Software Ontology, namely by invoking the executable semantics in the context of Core Software Ontology to get the result. With an example in this paper, we can use this framework to get the returned value and prove the feasibility of this framework.**

Keywords-Executable Semantics; Core Software Ontology; DUL Ontology; SHACL; RESTful API.

I. INTRODUCTION

DOLCE Ontology is a well-known foundation ontology that provides a few design patterns providing a substantial foundation for building multiple core ontologies and domain ontologies [1][11]. One of these ontologies is the Core Software Ontology (CSO). It must be acknowledged that CSO diligently achieves the goal of describing computational domain, particularly portraying aspects of computing object (software, data and their realizations) and activity (execution of software) semantically, all of which implement executable semantics. With SPARQL QUERY, it is easy to catch sight of semantic aspects of a computing configuration, including its I/O, execution plan, execution situation and so on. By combining the SPARQL CONSTRUCT with the execution plan, an execution situation embodying computing objects and activities to portray computing configuration can be constructed. However, the execution situation still cannot be used to invoke the included computing objects (software) and the initial hope of building software is to launch a computing/execution and return a value, so it is clear that the current state of CSO fails to achieve that.

Today, more and more RESTful APIs are put into use as computing resources; their emergence means that software can be regarded as computing resources published on the Web. Both users and running codes can access them and get the result via their open URLs. It is known that a typical ontology is based on the Resource Description Framework (RDF) and the RDF provides a format basis in which URLs are encapsulated to represent multiple entities, either types or instances. While navigating ontology, users can access these URLs that are often pointers to html files, texts and multimedia files, the contents of which can be either directly downloaded or viewed on the Web browsers. But RESTful APIs often focus on computing, which means that their URLs should not be simply accessed and need some values as input and then the subsequent returned values should be interpreted semantically and used for further computing tasks. It is simple to directly fit RESTful APIs as URLs into the context of CSO for content management, but that cannot enable executable semantics, including these RESTful APIs, to be invoked to launch computing in the context of CSO.

Shapes Constraint Language (SHACL) provides SHACL JavaScript Extensions (SHACL-JS) engine that can access and invoke JavaScript code on the Web by its URL, with which SHACL Specification based on SPARQL (SHACL-SPARQL) can be used to infer new triples containing values coming from invoking JavaScript code. From this perspective, SHACL can help to invoke executable semantic in the context of CSO if JavaScript file URLs are fitted into it. However, JavaScript cannot support important intermediate data, often stored in databases or data files, for returning the final result and too complex algorithms such as Matrix or Calculus. As result, it is less necessary and applicable to have JavaScript files fit into the context of CSO than RESTful APIs.

This paper presents a framework in which the problem above can be solved.

- a) Fitting RESTful API into the context of CSO enables the URL of RESTful API to become part of the context of CSO as computing resource, which testifies the rationality that with the help of RESTful API, the applicability of CSO makes a great progress. It is not easy to access RESTful APIs as computing resources only by their URLs because the fault of CSO is that the first paper on CSO is before the emergence of RESTful API.
- b) Breaking the hurdle between the SHACL-JS engine and RESTful API to enable the former to invoke the latter to get the result, which can be done by investigating the running mechanism of the SHACL-JS engine and taking advantage of it to

make a devised scenario in which the engine invokes a specific RESTful API as JavaScript code.

- c) With a, b and SHACL-SPARQL Construct, RESTful API can fit into the context of CSO very well and the constructed executable semantics can be invoked for the desired purpose.

This paper is structured as follows. Section 2 presents the related work, mainly why to choose SHACL. Section 3 introduces a framework to achieve the purpose of this paper. Section 4 presents a use case of computing lift coefficient of airfoil, where readers can get a sense of the motivation. Section 5 describes how to apply the framework to the use case and get the desired result to verify the framework. Section 6 is the conclusion.

II. RELATED WORK

In the past years, a large number of RDF-based applications have been developed for various domains. In order to take advantage of the semantic feature of RDF, several query and rule languages, such as SPARQL [10], Jena rule [14] and SWRL [13], have been developed and adopted widely with a number of inbuilt functions [15][16] for users to execute various computations during either querying or reasoning.

Unfortunately, due to computing complexity in the real world, such as matrix calculation, linear operation and those requiring external data source, the execution above is insufficient to accomplish such computing tasks.

In order to solve such problems, Zhang [12] turns to SPARQL Inferencing Notation (SPIN) [17]. In SPIN, SPARQL queries can be stored together with RDF data models in RDF graphs to define executable semantics of classes and their members. SPIN provides a special framework (SPINx) that allows a user-defined function to link an external JavaScript file to RDF data by RDF property; this user-defined function can be used for executing computing by invoking this linked JavaScript file [18]. The work in [12] investigates the mechanism of SPINx framework and devises a method to link RESTful API with RDF data, and invoke RESTful API while either querying or reasoning. It must be pointed out that this paper opens a new window in Semantic Web technology.

SHACL is strongly influenced by SPIN and can be regarded as its successor [19]. SHACL includes basically all features of SPIN, and more. Most importantly, SHACL is an official W3C Recommendation, which makes it far more likely that other vendors will support it [19]. As result of this, this paper adopts SHACL rather than SPIN.

III. INTRODUCTION TO THE FRAMEWORK

This paper puts forward a framework working as basis for invoking executable semantics, in which the following are cooperating with each other to achieve the goal: Core Software Ontology (CSO), Shapes Constraint Language (SHACL) and RESTful API. We address how SHACL-JS invokes a RESTful API.

A. *Introduction to Core Software Ontology*

In this paper, a lightweight, easy-to-apply foundational ontology known as DOLCE+DnS Ultralite (DUL) [1] [5] is used as basis for CSO. Due to space limit of this paper, only some aspects concerned are discussed here. For more details, readers can refer to [1]-[4].

1) *Software and Data*

The programs that manipulate the data are usually referred to as CSO: Software, a special kind of DUL: InformationObject. CSO: Data also can be considered as a special kind of DUL: InformationObject. The difference from CSO: Software is that CSO: Data does not DUL: express a DUL: Plan.

CSO: ComputationalObject as a special kind of DUL: InformationRealization can be the appearance of an algorithm in memory or disks. Just like the fact that DUL: InformationRealization DUL: realizes DUL: InformationObject in DUL, CSO: ComputationalObject DUL: realizes CSO: Data as well as CSO: Software.

CSO: ComputationalActivity as a special kind of DUL: Activity is the correspondence of the execution of a CSO: ComputationalObject. This is the form of software which manifests itself in a sequence of activities in the computing domain.

2) *Task, Input and Output*

We use CSO: ComputationalTask, a special kind of DUL: Task, to represent invocations, and the actual executions of CSO: ComputationalTask can be CSO: ComputationalActivity. A set of CSO: ComputationalTask are grouped and linked via the DUL: follows and DUL: precedes associations in a DUL: Plan.

We also need to model the CSO: Input and CSO: Output for CSO: ComputationalTask. The CSO: Input and CSO: Output are required to represent the input and output for computing task, and they are special kinds of DUL: Role, which are both DUL: isRoleOf CSO: Data and DUL: definedIn a DUL: Plan. The relationships between CSO: Input (CSO: Output) and CSO: ComputationalTask are modeled by CSO: inputFor (CSO: outputFor).

CSO: executes is also introduced to formalize that a CSO: Software is used to complete a CSO: ComputationalTask. That means that CSO:Software begins to CSO:executes a CSO: ComputationalTask in the DUL:Plan and then a DUL:Situation regarded as a computing configuration holding DUL:satisfies association with the DUL:Plan comes into being, where a DUL: CSO:ComputationalObject that DUL:realizes this CSO:Software holds a DUL:hasParticipant association with a CSO: ComputationalActivity that also has DUL:executesTask association with the CSO: ComputationalTask (see D1).

CSO:executes(x, y) =def CSO:Software(x) $\wedge$
CSO:ComputationalTask(y) $\wedge$
$\exists$co,ca,p(CSO:ComputationalObject(co) $\wedge$
CSO:ComputationalActivity(ca) $\wedge$ DUL:Plan(p) $\wedge$
DUL:realizes(co,x) $\wedge$ DUL:express(x, p) $\wedge$ DUL:

defines(p, y) ∧ DUL: executesTask (ca, y) ∧ DUL: hasParticipant (ca, co)) (D1)

B. *Simple Introduction to RESTful API*

A RESTful API is an application program interface (API) that uses existing HTTP methodologies defined by the RFC 2616 protocol. They use GET to retrieve a resource, PUT to change the state of or update a resource, which can be an object, file or block, POST to create that resource, and DELETE to remove it [9].

RESTful APIs for a website are codes that allow software programs to communicate with each other. The RESTful API spells out the proper way for a developer to write a program requesting services from an operating system or other applications.

With RESTful APIs, networked components are resources the user requests access to - a black box whose implementation details are unclear. All calls are stateless; nothing can be retained by the RESTful service between executions [9].

C. *Introduction to SHACL, SHACL-SPARQL and SHACL-JS*

SHACL is a W3C recommendation [6] [7]. A SHACL processor takes a shapes graph and a data graph as input. The shapes graph defines so-called shapes, which are a collection of constraints. A shape also tells the engine for which nodes in the data graph it applies to (using sh: targetNode).

SHACL-SPARQL is one extension mechanism for SHACL to express constraints in SPARQL, allowing shape definitions to point at executable SPARQL queries to perform the validations [8].

SHACL-JS engine is an advanced extension mechanism for SHACL, allowing users to express SHACL constraints and the advanced features of custom targets, functions and rules with the help of JavaScript. The principle of calling JavaScript function is to download JavaScript file contents via HTTP to the engine, and then the engine resolves the contents to execute the code specified by the function name. Figure 1 below shows how SHACL-JS invokes a RESTful API.

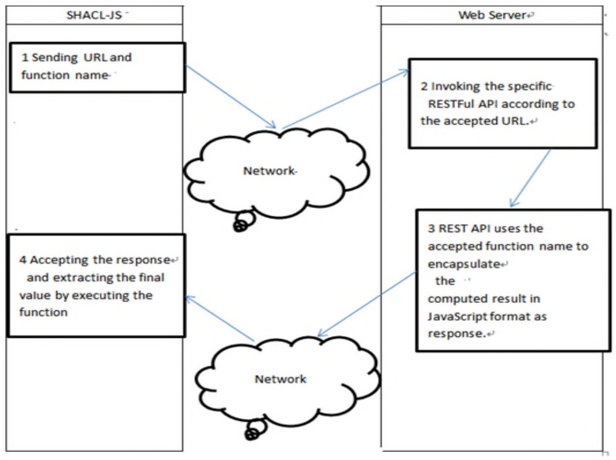

SHACL-JS
Web Server
1 Sending URL and function name
2 Invoking the specific RESTful API according to the accepted URL.
Network
3 REST API uses the accepted function name to encapsulate the computed result in JavaScript format as response.
4 Accepting the response and extracting the final value by executing the function
Network

Figure 1. How to use SHAL-JS to invoke a RESTful API.

According to Figure 1, a computer with SHACL-JS is called a client and a computer running RESTful API on it is called Web Server.

1) A user operating the client submits data in RDF that contain URL of a RESTful API and a function name into SHACL-JS. SHACL-JS sends the URL to the Web Server after parsing. In this case, the URL is http://ip/lift-coefficient?attack-angle=13 and the function name is called func.

2) The Web Server runs the received URL and then gets 1.51, the returned value.

3) The Web Server encapsulates the value as the string “function func () {return 1.55 ;}” in JavaScript code format and then sends it as feedbacks to the client.

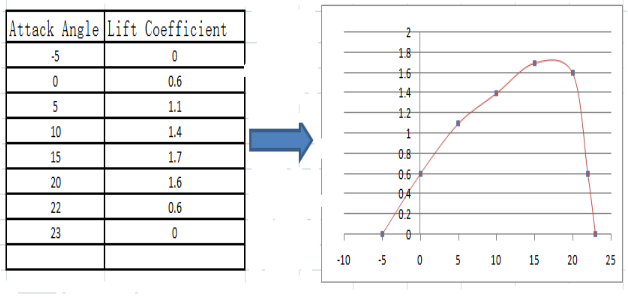
4) The client simply runs the string and returns 1.51 to the user.

IV. Introduction to a Use Case of Computing Lift Coefficient of Airfoil

The dedicated function of an airfoil on an airplane is to provide lift during flight and it is necessary for computing varying lift with continuously changing attack-angle. The lift-coefficient formula is as follows.

Lift-coefficient=f(attack-angle) (1)

In (1), there is no explicit formula (or calculation script) to accurately calculate the lift coefficient from the attack angle. Typically, the actual lift-coefficient is a list of data through a limited number of experiments that record the data under different attack angles, as shown in Figure 2.



Attack Angle	Lift Coefficient
-5	0
0	0.6
5	1.1
10	1.4
15	1.7
20	1.6
22	0.6
23	0

Figure 2. The left part is a list of attack angles and corresponding lift coefficients; The right part is the curve graph reflecting the left part.

Here, these experiments are conducted independently, which means these data are locally stored in a Web server not freely accessible to others. A dedicated RESTful API can be developed and deployed on the Web server. The RESTful API implements numerical approximation, such as least square and interpolating to allow users to query for any given attack-angle. In this paper, the URL for this RESTful API is below:

http://ip/lift-coefficient?attack-angle={value}

In order to work out the lift coefficient while the attack angle is 13, the access URL is:

http://ip/lift-coefficient?attack-angle=13

V. APPLYING THE FRAMEWORK TO THE USE CASE

In this section, the framework is applied to the use case and we can freely get the returned value of the lift coefficient for a given attack angle. In this paper, a temporary ontology is created in the context of CSO for this use case, known as S-C ontology.

A. Modeling the Basic Entities and Properties of S-C Ontology for Use Case in the Context of CSO

A few of entities and properties can be extracted from this use case and be aligned to predefined concepts in CSO.

1. The s-c: attack-angle and s-c: lift-coefficient can be regarded as an instance of CSO: Data because their goals are to be manipulated by the software as input and output, respectively.
2. The s-c: computation1 can be regarded as an instance of CSO: Software, which is responsible for computing the lift-coefficient with an attack-angle.
3. The s-c: computation1-computational-object can be regarded as an instance of CSO: ComputationalObject, which DUL: realizes s-c: computation1 and rdfs:seeAlso "http://server:8080/lift-coefficient?Attack-angle={value}".
4. The s-c: activity1 can be regarded as an instance of CSO: ComputationalActivity, which identifies the execution of a certain CSO: ComputationalObject/CSO: Software.
5. The s-c: run is rdfs:subProperty of DUL: hasParticipant, which associates a CSO: ComputationalActivity with a CSO: ComputationalObject and means giving rise to an actual execution of a software.
6. The s-c:in is rdfs: subProperty of DUL: hasParticipant, which associates a CSO: ComputationalActivity with a CSO: Data as input for computing.
7. The s-c:out is rdfs:subProperty of DUL: hasParticipant, which associates a CSO: ComputationalActivity with a CSO: Data as output for computing.

B. Modeling Plan and Situation in S-C ontology

The s-c:computation-plan is an instance of DUL: Plan, the design of computing configuration, which includes the following entities: s-c:input-attack-angle, s-c:output-lift-coefficient and s-c:task1.The s-c:input-attack-angle can be regarded as an instance of CSO:Input and DUL:isRoleOf s-c:attack-angle outside the plan. The s-c:output-lift-angle can be regarded as an instance of CSO:Output and DUL:isRoleOf s-c:lift-coefficient outside the plan. The s-c:task1 can be regarded as an instance of CSO:ComputationalTask, which has CSO:inputFor association with s-c:input-attack-angle, CSO:outputFor association with s-c:output-lift-angle and CSO:executes association with s-c:computation1.

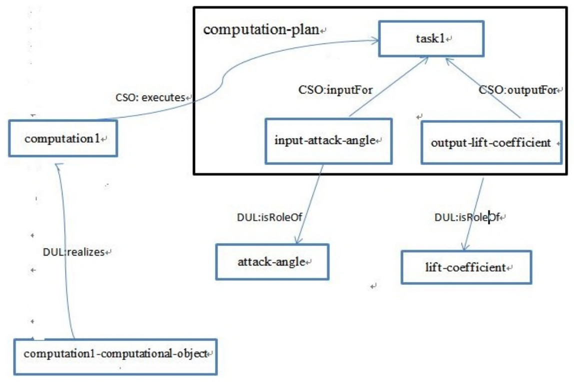


Figure 3. The structure of s-c: computation-plan and its association with software and data.

Although s-c:computation-plan is the design of computing configuration and the design does not come into use until the situation known as s-c:computation-situation (an instance of DUL:Situation) holding DUL:satisfies association with it, includes s-c:attack-angle,s-c:lift-coefficient,s-c:activity1, s-ccomputation1-computational-object.In s-c:computation-situation, s-c:attack-angle s-c:in s-c:activity1,s-c:lift-coefficient s-c:out s-c:activity1 and s-c:computation1-computational-object s-c:run s-activity1. According to D1 (Section 2), s-c: computation-situation will be constructed by s-c: computation-plan and the associated CSO: Software and CSO: Data outside the plan.

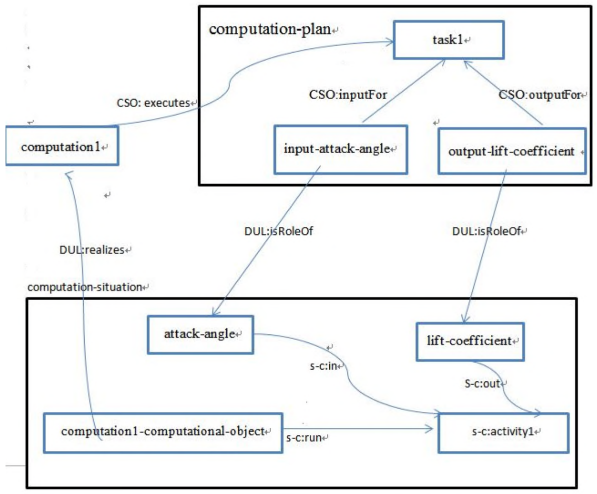


Figure 4. The structure of s-c: computation-situation and association with s-c: computation-plan.

C. Inferencing Situation from Plan

The s-c: computation-plan is the design of computing configuration and, according to CSO, the actual computing configuration (executable semantics) which return values should be s-c: computation-situation, the special kind of DUL: Situation. In this paper, the CONSTRUCT clause of SPARQL Update is used to construct s-c: computation-situation from s-c: computation-plan with associated CSO: Data and CSO: Software. The SPARQL statement is below.

```
      construct {
        ?software_computational_object s-c: run s-c: activity1.
        ?inData s-c: in s-c: activity1.
        ?inData DUL: hasDataValue ?value.
        ?outData s-c: out s-c: activity1.
        ?software_computational_object rdfs: seeAlso ?url.
       }
      where {
        GRAPH       <http://semantic-computing/#computation-
plan>{
             ?inRole cso: input-for s-c: task1.
             ?outRole cso: output-for s-c: task1.
       }
        GRAPH <http://semantic-computing/#software-plan>{
          ?software cso: executes s-c: task1.
          ?inData   DUL: hasRole   ?inRole.
         ?outData DUL: hasRole    ?outRole.}
         GRAPH <http://semantic-computing/#software-data>{
      ?software_computational_object DUL:realizes ?software.
         ?software_computational_object rdfs: seeAlso  ?url.
          ?inData DUL: hasDataValue ?value. }
 }
```

There are several graphs in this statement: <http://semantic-computing/#computation-plan> contains triples of s-c: computation-plan; <http://semantic-computing/#software-plan> contains triples representing associations between CSO: Data/CSO: Software and entities in s-c:computation-plan; <http://semantic-computing/#software-data> contains triples representing associations among CSO:Data, CSO:Software. The constructed result is below. It is noted that the s-c: computation1-computational-object (an instance of CSO: ComputationalObject) links a RESTful API via rdfs: seeAlso.

```
      s-c: computation1-computational-object s-c: run s-c:
activity1.
      s-c: attack-angle s-c: in s-c: activity1.
      s-c: attack-angle DUL: hasDataValue 13.0.
      s-c: lift-coefficient s-c: out s-c: activity1.
s-c: computation1-computational-object rdfs: seeAlso
<http://ip/lift-coefficient?attack-angle=>.
```

D. Using SHACL to Invoke Executable Semantics

The s-c: computation-situation forms executable semantics and now its goal is to create a triple of "s-c: lift-coefficient DUL: hasDataValue ?value".

By using the triples of s-c: computation-situation to create a model named Shape-Function, triples of which meet the standard of SHACL-JS, we form a function that will be further used to invoke RESTful API. The SPARQL Construct statement to use is below.

```
      prefix sh: <http://www.w3.org/ns/shacl#>
      construct {
      s-c: dynamicFunc sh: jsFunctionName "dynamicFunc".
      s-c: dynamicFunc sh: jsLibrary <http://jsLibrary/temp>.
      s-c: dynamicFunc sh: returnType xsd: double.
      s-c: dynamicFunc sh: parameter <http://parameter/temp>.
      s-c: dynamicFunc rdf: type sh: JSFunction.
      <http://parameter/temp> sh: datatype   xsd: double.
      <http://parameter/temp> sh: path s-c: number.
      <http://jsLibrary/temp> sh: jsLibraryURL ?dynamicFuncURL.
      }
      where{
        ?software_computational_object s-c: run s-c: activity1.
        ?inData s-c: in s-c: activity1.
        ?inData DUL: hasDataValue ?value.
        ?software_computational_object rdfs: seeAlso ?url.
        BIND(('http://IP/RESTTemplate/access?url='+STR(?url)+
      '? value='+STR (?value)) as ?dynamicFuncURL).
}
```

The constructed model is called Shape-Function; it can also be regarded as a shape according to SHACL, see below.

```
      s-c: dynamicFunc sh: jsFunctionName "dynamicFunc".
      s-c: dynamicFunc sh: jsLibrary <http://jsLibrary/temp>.
      s-c: dynamicFunc sh: returnType xsd:double.
      s-c: dynamicFunc sh: parameter <http://parameter/temp>.
      s-c: dynamicFunc rdf: type sh: JSFunction.
      <http://parameter/temp> sh: datatype xsd: double.
      <http://parameter/temp> sh: path s-c: number.
      <http://jsLibrary/temp>
sh:jsLibraryURL
'http://ip/RESTTemplate/access?url=http://ip/lift-
coefficient?attack-angle=13'.
```

It is noted that there are two RESTful APIs that should be discussed. They are 'http://ip/RESTTemplate/access?url=" and http://ip/lift-coefficient?attack-angle=. The function of the former is to invoke the latter and encapsulate the return value in JavaScript format with "dynamicFunc" as function name. We show below the concise code.

In addition to the Shape-Function complied with SHACL-JS, another model named Shape–Construct complied with SHACL-SPARQL is needed to work with the Model-Function to achieve the goal. The Shape –Construct is shown below.

```
      @prefix s-c :< http://semantic-computing/#>.
      s-c:rule1
        a rdfs: Class, sh:NodeShape ;
        sh:targetNode s-c:activity1 ;
        rdfs:label "to run sc:activity1" ;
        sh: rule[
       a sh:SPARQLRule ;
      sh: construct """
      CONSTRUCT {
      ?outdata DUL: hasDataValue ?value.}
      WHERE {
      ?outdata s-c: out ?this.
      BIND       (<http://semantic-computing/#dynamicFunc>()
AS ?value).
                  }
                              """ ;
   ] .
```

Figure 5 below shows the process and the result.

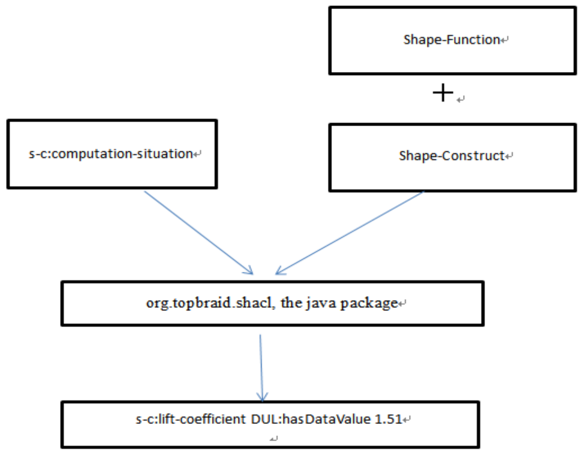

Shape-Function
+
s-c:computation-situation
Shape-Construct
org.topbraid.shacl, the java package
s-c:lift-coefficient DUL:hasDataValue 1.51

Figure 5. The process and method of creating the new triple.

In Figure 5, after interference a new triple is generated, the content of which is "s-c: lift-coefficient DUL: hasDataValue 1.51".

VI. Conclusion

With the abundance in Information Technology (IT) infrastructure today, the number of RESTful APIs is growing and applications of ontologies for computational domain should be constructed by fully taking advantage of this situation. This paper discusses that usefulness and feasibility of using SHACL and RESTful API to invoke executable semantics. It can be said that the paper's achievement is useful in the development of ontology-based knowledge system.

In our opinion, the study of this paper can make the Semantic Web models, here Core Software Ontology, have powerful computing capacity. Of course, the coordinating asynchronous requests, latency, availability and security must be taken into account. These problems should be solved effectively (at least in part) as the technologies for RESTful API, exemplified by SPRING BOOT, have made much effort to solve them from birth.

References

[1] A. Gangemi et al., "Sweetening Ontologies with DOLCE", EKAW 2002, pp. 166-181, 2002.

[2] A. Gangemi and P. Mika, "Understanding the Semantic Web through Descriptions and Situations", ODBASE 2003, pp. 689-706, 2003.

[3] A. Gangemi, "Task taxonomies for knowledge content. Metokis Deliverable D07", 2004.

[4] D. Oberle et al.,"Towards Ontologies for Formalizing Modularization and Communication in Large Software Systems" , Journal of Applied Ontology, vol. 1, no. 2, pp. 163–202, 2006.

[5] https://www.w3.org/2005/Incubator/ssn/wiki/DUL_ssn [retrieved: November, 2020]

[6] https://www.w3.org/TR/shacl/.[retrieved: November, 2020]

[7] J. Corman et al.," Semantics and validation of recursive SHAC", ISWC 2018, pp. 318– 336, 2018.

[8] https://w3c.github.io/data-shapes/shacl-js/#introduction [retrieved: September, 2020]

[9] https://searchapparchitecture.techtarget.com/definition/RESTful-API. [retrieved: November, 2020]

[10] https://www.w3.org/TR/sparql11-query [retrieved: November, 2020]

[11] S. Borgo and C. Masolo," chapter Foundational choices in DOLCE", Handbook on Ontologies, 2009.

[12] X. Zhang, "An approach to enabling RDF data in querying to invoke REST API for complex calculating",International Conference on Dublin Core and Metadata Applications 2018, pp. 34-42, 2018.

[13] http://www.w3.org/Submission/SWRL. [retrieved: September, 2020]

[14] J. Carroll et al., "The Jena Semantic Web Platform: Architecture and Design", HP Laboratories Technical Report HPL-2003-146,2003.

[15] https://github.com/dotnetrdf/dotnetrdf/wiki/DeveloperGuide-SPARQL-XPath-Functions. [retrieved: November, 2020]

[16] http://jena.apache.org/documentation/query/library-function.html. [retrieved: November, 2020]

[17] https://www.topquadrant.com/technology/sparql-rules-spin. [retrieved: November, 2020]

[18] http://spinrdf.org/spinx.html. [retrieved: September, 2020]

[19] https://www.spinrdf.org/spin-shacl.html. [retrieved: September, 2020]