2017 IEEE 1st International Conference on Cognitive Computing

An Automation Method of SLA Contract of Web APIs and Its Platform Based on Blockchain Concept

Hiroki Nakashima*
Graduate Program of Software Engineering
Graduate School of Science and Engineering
Nanzan University
Nagoya, Japan
nakashimaorz@gmail.com

Mikio Aoyama
Graduate Program of Software Engineering
Graduate School of Science and Engineering
Nanzan University
Nagoya, Japan
mikio.aoyama@nifty.com

***Abstract*— As the number of Web APIs is rapidly increasing, it is an urgent issue to discover qualified Web APIs and provide value-added services by orchestrating them. However, most of the interface descriptions of Web APIs are informal, and the Web API SLA contracts, which are a key to quality of services orchestration, require manual operations at the consumers. Meanwhile, applying the blockchain, the distributed ledger technology, to various domains beyond Fintech is attracting attention because of its fault-tolerant and anti-tampering. However, it isn't applied to Web API SLA contracts, yet. In this article, the authors propose a formal specification description of Web APIs together with its associated SLA specifications based on RDF, and an SLA contract method based on the common SLA contract platform built on the blockchain. We implemented the prototype of the SLA contract platform, and applied it to the examples for demonstrating its feasibility. Those experiences prove the feasibility of the proposed Web API SLA contract method and its supporting platform.**

Keywords-Blockchin; WebAPI; SLA, RDF; Contract, Ethereum

I. Introduction

As the rapid spread of Web APIs, it is an urgent issue to select qualified Web APIs, and orchestrate them to provide value-added services [17]. However, most of the interface descriptions of Web APIs are informal, and the Web API SLA contracts, which are a key to quality of services orchestration, require manual operations at the consumers (Fig. 1). The manual contract is the bottleneck to the agility, efficiency and quality of the service orchestration. The contract, depending on the provider's contract platform, might not be trusted.

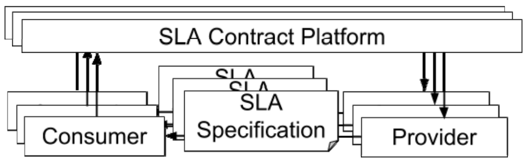

Fig. 1 SLA Contract Method and Its Actors

* Currently with Freee K. K.

II. Research Questions

There are following two major obstacles in orchestrating Web APIs, which are the research questions of this article.

(1) Informal specification descriptions of Web APIs and their associated SLA, and
(2) The difference of SLA contract specifications and their supporting platforms by the service providers.

III. Related Works

A. Web Service Level Agreement (WSLA)

WSLA is a specification language of SLA for Web services requiring the obligations of each of the service providers and the consumers for contracting the SLA [14]. It assumes WSDL (Web Services Description Language) [4] as the interface description, and SOAP messaging over HTTP.

B. Web API Description

Similar but different Web API description languages have been proposed. They include API Blueprint [2], RAML [1], and OpenAPI [16]. Development support tools are provided for each language. However, most of the description languages lack some formality and may cause problems of agility, efficiency and quality in orchestrating them [9].

C. RDF Document Verification based on the Shape

Shape is a constraint definition for RDF graph [19]. Resource Shape is one of the earliest definition languages of a shape. The shape enables to verify properties of an RDF graph against the constraints defined by the shape. Various software tools have been developed to verify the RDF graphs with RDF query language SPARQL (SPARQL Protocol and RDF Query Language) [12]. The verification tools provide the results of the verification and the details of the violation parts of the RDF graph.

D. SHACL (Shapes Constraint Language)

SHACL is a definition language of the shape under development [13]. The constraints on an RDF document defined by SHACL can verify the document with SPARQL queries. The specification of SHACL also includes the definition of the output format of the verification result.

E. Blockchain

Blockchain is the underlying technology of Bitcoin [15]. It is a distributed ledger based on the distributed database and hash chain working on P2P network [3, 11]. It is expected to apply to the contract conclusion and payment from its fault-tolerant and anti-tampering [6]. However, the SLA contract method has not been established, yet.

F. Smart Contract

Smart contract is a mechanism to digitize contracts and record and execute them on the blockchain [5, 6, 7]. A smart contract is defined as a program and is executed autonomously according to its conditions. Smart contracts enable the automation of contract conclusion that depends on a trusted third party conventionally.

G. Ethereum

Ethereum is a decentralized application platform based on blockchain [10]. A smart contract called *contract* executes on the Virtual Machine called EVM (Ethereum Virtual Machine). The operation is recorded in the blockchain with the account information of the caller. The accounts are guaranteed by a signature of public key cryptography. With the contract, it is possible to handle payments between users of Ether, the virtual currency on the Ethereum.

IV. Approach

The proposed *SLA contract* is defined as a building agreement based on an SLA specification description, and concluding the contract.

A. RDF-Based Specification Description of Web APIs and Associated SLA, and Verification with Shape

We propose a method of specifying Web API and associated SLA based on RDF.

Since RDF is extensible, it is possible to describe the specifications of the Web API and associated SLAs at the equivalent level of conventional description methods. The RDF graph can be verified against its shape. Therefore, satisfaction of the SLA document against shape can guarantee that the SLA contract satisfies the constraints.

B. SLA Contract Based on Blockchain

We propose a method of the SLA contract as an application of Smart Contract based on blockchain.

In the conventional SLA contract method, a contract depends on the contract platform of the service provider. On the other hand, the proposed method together with the common SLA contract platform operates on the P2P environment independent of any specific platform of the parties. A secure contract platform can be realized by the fault-tolerance and anti-tampering of the underlying blockchain. In addition, multiple providers can provide Web APIs via the common SLA contract platform, and allow consumers to use them via the common interface specification description.

The proposed SLA contract can be automated as a smart contract on the blockchain (Fig. 2).

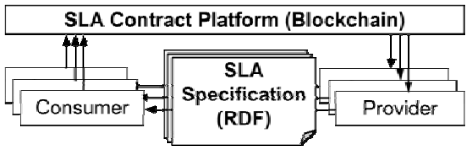

Fig. 2 The proposed method and its actors

V. SLA Specification Description of Web APIs Based on RDF

We propose an RDF-based specification description of both Web APIs and associated SLA. We call the specification description as the SLA specification description of the Web API. The RDF document of the SLA specification description is called SLA documents.

A. Model of SLA Specification Description

We defined the SLA specification description based on WSLA. For WSLA, the interface of Web APIs is defined by WSDL. WSLA extends the WSDL for SLA description.

It is known that more than 90% of Web APIs is based on REST [17]. However, WSDL description is based on SOAP over HTTP, and does not support REST APIs. We defined a specification description of Web APIs based on API Blueprint [2] since it is one of a few commonly used descriptions of Web APIs.

The specification description of API Blueprint employs MSON (Markdown Syntax for Object Notation) as its underlying description language. We also employ the description of JSON and a part of JSON Schema for the specification description.

We designed the SLA specification description of the Web API with the following three specification descriptions. The meta-model of the Web API SLA specification is shown in Fig. 3.

(1) SLAMS: Specification Description of SLA
(2) APIMS: Specification Description of Web API
(3) TSON: Specification Description of JSON and a part of JSON Schema

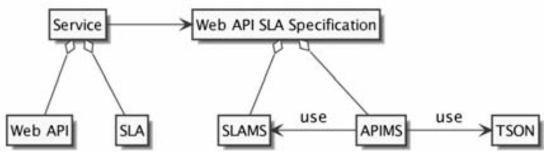

Fig. 3 Meta-model of Web API SLA Specification

B. SLAMS (SLA Metadata Specification)

We propose SLAMS as a specification description of SLA in RDF.

1) Required Expression of SLA Specification Description

In order to contract the SLA with the specification description, the SLA specification description requires to express the following expression.

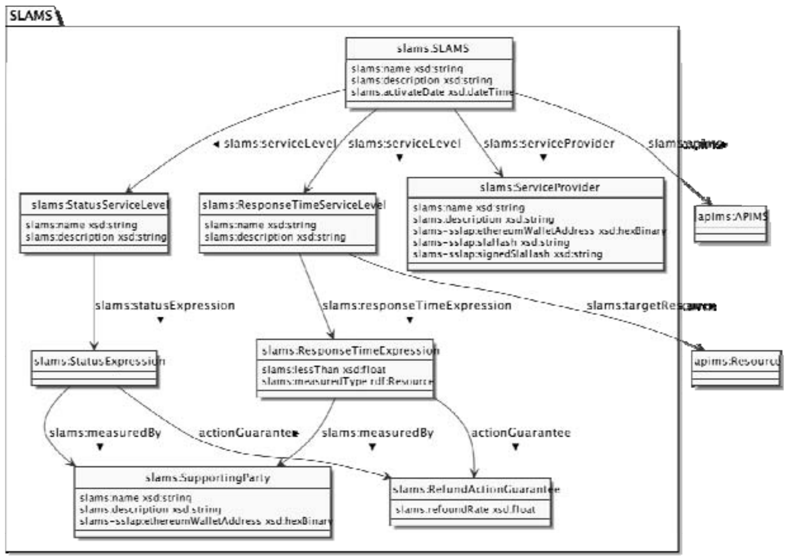
Fig. 4 Data Model of SLAMS

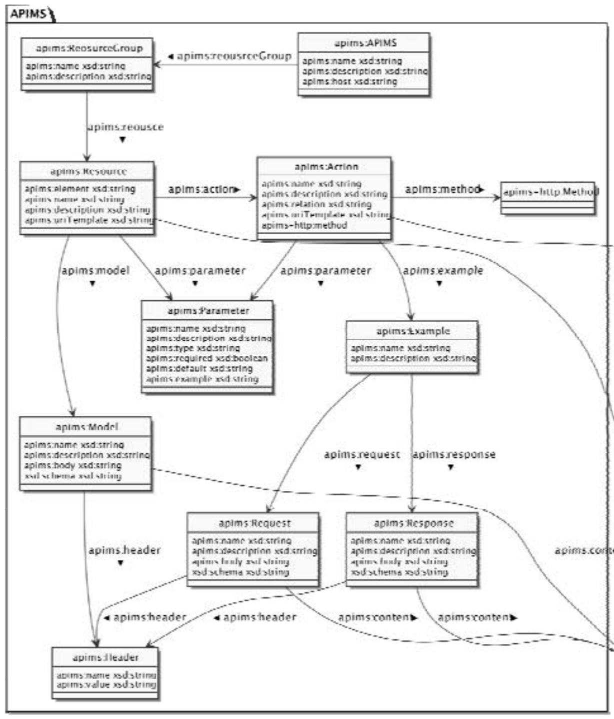
Fig. 5 Data Model of APIMS

(1) Description of service level and its obligations
(2) Information on the parties involving in the contract
(3) Link for extending Web APIs

2) Data Model of SLAMS

We show the data model of SLAMS in Fig. 4.

Definition of SLAMS is based on WSLA. A specification of SLA is defined by a resource that root resource is slams:SLAMS (1). The slams:SLAMS resource has links to the APIMS resources, service provider resources, service level resources and Web API resources. The service level has links to the monitor resource (slams:SupportingParty), and a resource to evaluate the level of achievement of the contracted service level. The level of achievement of service level is evaluated with the service level evaluation expression defined by the service level evaluation expression resource.

C. APIMS (API Metadata Specification)

We propose APIMS as a specification description of Web APIs in RDF.

1) Required Expression of Web API Specification Description

The proposed specification description of Web APIs requires to represent the conventional specification descriptions of Web APIs.

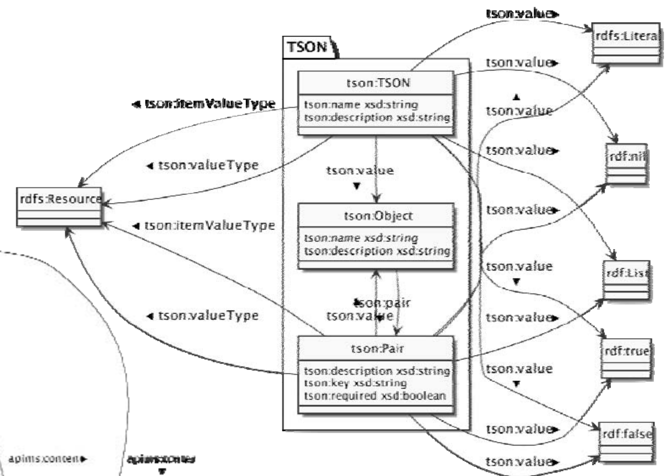
Fig. 6 Data Model of TSON

2) Data Model of APIMS

We show the data model of SLAMS in Fig. 4.

The definition of APIMS is based on API Blueprint. The description of APIMS is defined with an RDF whose root resource is apims:APIMS (2). The apims:APIMS resource has a hierarchic structure corresponding to that of API Blueprint. The apims:APIMS has links to tson:TSON (3) resources.

D. TSON (Triple Syntax for Object Notation)

We propose TSON as a specification description of JSON and a part of JSON Schema in RDF.

1) Required Expression of TSON

(1) slams: means the name space of SLAMS.

(2) apims: means the name space of APIMS.

(3) tson: means the name space of TSON.

TSON represents an orchestration between resources by reusing Web resources. The following two characteristics are required to TSON.

(1) Compatibility with MSON
(2) Reusing the RDF regular resource

2) Data Model of TSON

We show the data model of TSON in Fig. 6. TSON is described as a RDF graph based on MSON and ECMA-404 [8]. The specification of TSON is defined with resources whose root resource is tson:TSON.

E. Shape of Specification Description of Web APIs and SLA

We defined the shape of the three specification descriptions with using SHACL [13] since SHACL is the most powerful shape description language under standardization at W3C. The shape enables to verify the SLA document. We show the part of one of a part of shape of SLAMS in Fig. 7.

```
@prefix slams: <http://slams.example.org/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.example.org/> .
...

slams:Shape a sh:Shape ;
   sh:targetClass slams:SLAMS ;

...

  sh:property [
    sh:predicate slams:activateDate ;
    sh:minCount 1 ;
    sh:maxCount 1 ;
    sh:datatype xsd:dateTime ;
  ] ;
...
```

Fig. 7 A Part of Shape of SLAMS

F. Valiation of SLA Specification Description with Shape

We show a part of a SLAMS graph in Fig. 8. When the shape applies to this RDF graph, the value type of slams:activateDate constraints xsd:dateTime, but this value

```
@prefix xsd:
<http://www.w3.org/2001/XMLSchema#> .
@prefix ex: <http://example.org/> .
@prefix slams: <http://slams.example.org/> .
@prefix slams-sslap:
<http://slams.example.org/sslap/> .

   ex:slams a slams:SLAMS ;
   slams:activateDate
      "2017-01-01T00:00:00Z"^^xsd:string ;
   slams:serviceProvider ex:org0 ;
   slams:apims ex:apims ;
   slams:serviceLevel ex:sl0 .

ex:org0 a slams:ServiceProvider ;
   slams:name "Weather Company inc." ;
...
```

Fig. 8 A Part of SLAMS graph

type is xsd:string. There is a violation for value type constraint.

We show the validation result graph in Fig. 9. This graph includes the following information of identifying constraint violation point.

(1) sh:focusNode: subject that violation causing
(2) sh:resultPath: property that violation causing
(3) sh:value: object that violation causing
(4) sh:resourceConstraintComponent: violated constraint
(5) sh:sourceShape: shape that violation causing

If there is no violation, this resource does not generate. If SLA document has no violation when the three shape applied, SLA document is valid for SLA contract.

```
@prefix sh:   <http://www.w3.org/ns/shacl#> .
@prefix ex: <http://example.org/> .
@prefix slams: <http://slams.example.org/> .

[  a sh:ValidationReport ;
   sh:conforms false ;
   sh:result [
      a sh:ValidationResult ;
      sh:resultSeverity sh:Violation ;
      sh:focusNode ex:slams ;
      sh:resultPath slams:activateDate ;
      sh:value
         "2017-01-01T00:00:00Z"^^xsd:string ;
      sh:resultMessage
         "slams:activateDate expects a literal of
datatype xsd:dataTime." ;
      sh:sourceConstraintComponent
         sh:DatatypeConstraintComponent ;
      sh:sourceShape  slams:Shape .
] ,
    ...
```

Fig. 9 A Part of Validation Result Graph

VI. SLA Contract Method Based on the Blockchain

A. SLA Contract Based on Common SLA Contract Platform

We propose an SLA contract method based on common SLA contract platform. The consumer can contract the Web APIs with the common interface of the common contract platform. The consumer can also contract all the Web APIs registered to the platform, and reuse them with the messages supported by the platform.

1) Actors

Fig. 10 shows the relationships between the actors and messages exchanged on the in the SLA contract platform.

In the proposed method, we defined the three actors, including service consumer, service provider, and SLA contract platform. SLA contract can conclude between them. However, to make the proposed contract method work, the following two problems need to be solved.

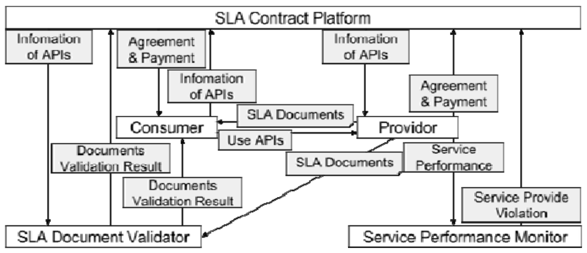

Fig. 10 Messaging Structure between Actors

(1) Concluding a contract with the invalid SLA document
(2) The service provider does not meet the service level and obligations

To solve the issues, we introduce the following two additional actors:

a) SLA document validator, and
b) Service performance monitor.

2) SLA Document Validtor

The SLA document validator validates a SAL document registered to the SLA contract platform, and responds to the platform with the validation results.

A consumer can get the SLA document and associated validation results, and confirm the validity of the SLA document. Therefore, SLA document validator enables to avoid a contract with invalid SLA documents.

3) Service Performance Monitor

The service performance monitor monitors the service the performance of service provisioning based on the SLA document. If the monitor detects any violation, it sends a notice to the SLA platform.

4) Common SLA Contract Platform

To realize the SLA contract, the SLA contract platform provides the following seven services.

(1) Registering the specification description of Web APIs
(2) Registering the validation result of SLA documents
(3) Providing the specification description of Web APIs, validation result of SLA documents, and execution information of any obligations.
(4) Paying the usage fee from consumer to provider
(5) Providing the information on contracted consumers
(6) Registering the refund request corresponding to the service violation
(7) Accepting the refund request

B. SSLAP (Smart SLAP Platform)

We propose the SSLAP (Smart SLA Platform) as a common SLA contract platform based on smart contract.

1) Resource identification by URI

The above mentioned services (1) and (2) of SSLAP accepts the URI of SLA document. Then, the service (3) can get the SLA document from the service provider.

A smart contract has a limitation of the size of data, and refer only the URI of documents to identify the document on the Web.

2) Contract Conclusion with Inner Currency

The services (4) and (7) of SSLAP need the payment function. These functions use inner currency of the smart contract. Sending the inner currency is programmable as a smart contract, and assures the conclusion of SLA contract.

3) Assurance of Document Publisher Based on the Signature

There is a risk of spoofing of the publisher, that is, a publisher registered the SLA document which actually other publisher published. Therefore, identifying the SLA document with a URI is not enough to assure the identity of the publisher. To avoid the problem, SSLAP use the signature of the publisher.

Fig. 11 shows the messaging structure for the identify validation of publisher of an SLA document.

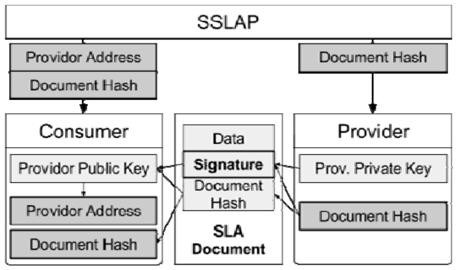

Fig. 11 Identity Validation of Publisher of an SLA Document

A provider gets a hash of an SLA document, and register it to SSLAP with service (1). SSLAP register the publisher address and hash of the SLA document to a storage on a blockchain. The provider generates the signature with the hash and its own private key, and attach it to the published document.

A consumer gets the address of a provider, a hash of the document, and the URI of the SLA document, and gets the SLA document based on the URI. The consumer generates the provider public key from the signature and hash of the SLA document, and validates the identity of the document publisher.

4) Assurance of Web API Request Publidsher Based on the Signature

The provider must identify the consumer when providing services. Fig. 12 shows the messaging structure for validating the identity of the Web API request.

A consumer gets a hash of the SLA contract for paying on the SSLAP with service (4). SSLAP registers the

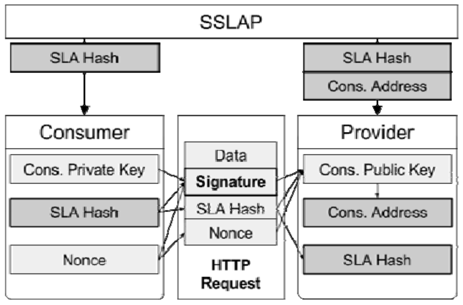

Fig. 12 Validation of Web API Request

addresses of contract parties and hash to a storage on a blockchain. The consumer generates the signature with the hash, its own private key and nonce, and attach it to the Web API request.

When a provider gets the Web API request, it gets the address of the consumer and a hash of the contract. The provider generates the provider public key from the signature and hash of the contract and nonce, and validates the request publisher.

5) Model of SSLAP Contract

Fig. 13 shows the model of SSLAP contract.

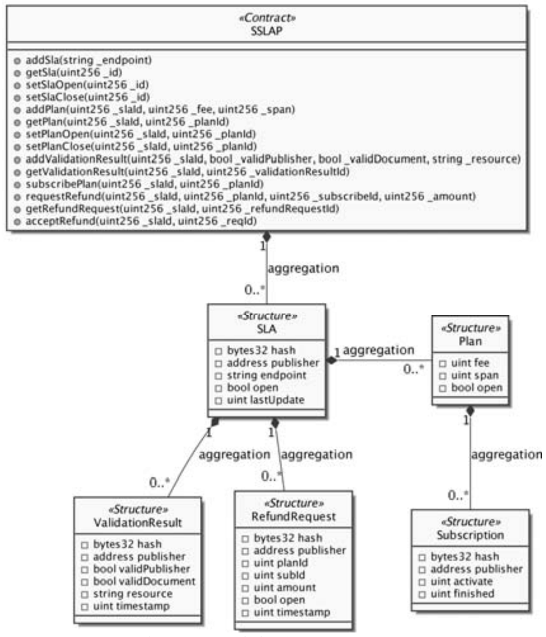

Fig. 13 Model of SSLAP Contract

The SSLAP works with the following five entities for realizing the services of the common SLA contract platform.

(1) SLA: Information on SLA document
(2) Plan: Subscribe plan of Web API
(3) Validation Result: Information of validation
(4) Subscription: Information of SLA contracting
(5) Refund Request: Information of refund

The Information of SLA, Plan, and Refund Request have "open" property indicating the availability. The property of SLA and Plan are updated when the functions such as setSlaOpen are called. The property of Refund request is updated when the consumer receives the refund payment.

The expiration time of SLA contract is set by the values of "activate" property and "finished" property. Those values are decided by the time of contract conclusion and duration of activation, i.e. the value of span property.

All the contract functions have execution permission. When the function called, its caller must be validated.

The subscribePlan function and acceptRefund function involve a payment. When the function called, its amount must be validated.

VII. Implementation of SSLAP Prototype and Its Evaluation

A. SSLAP Contract

We implemented the SSLAP contract on Ethereum with the contract definition language, Solidity, of 300 LOC.

1) Applying to The Example Case

We deployed the implemented contract to EVM emulator on JavaScript, and applied it six use cases (Fig. 14).

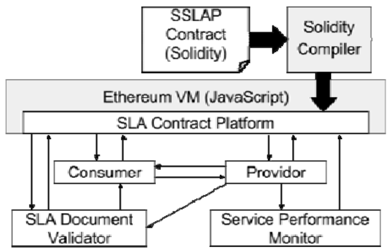

Fig. 14 The SSLAP Contract on Ethereum

We show the part of one use case scenario SLA contract execution that includes the SLA contract conclusion and the Web API request validation.

The Plan 0 of the SLA 0 was set the fee, 1,000,000,000 wei (4). The account that has following parameter was used for the evaluation.

```
(1)   Private Key:
      "0x1be3fd99e9819aca25bec9383602e49fe6dfa9d6e75b1d1
      16555663a7138539f"
(2)   Public Key:
      "0x31800bf1ebed99f11977d81f6ba71cdcc0670664c7ebe7a
      03ea33de0982e15e0ebfad233d83e4a154c63157fdbb2f991f
      0a0e39ff3f444f626eef762869a5e8a"
(3)   Address:
      "0xb2c5a2fe26458387822481bc718a4f92f6e91ad4"
```

Fig.15 shows the execution result of the subscribe request for Plan 0 of SLA 0.

```
> subscribe(0,0) with 1000000000 wei
Result: "0x6a696d59deaf6ad98489800e74be94b5
faab00f26ef2765e8dc835a98e65bc9f000(…)
Transaction cost: 130964 gas.
Execution cost: 109436 gas.
Decoded:
bytes32 hash: 0x6a696d59deaf6ad98489800e7
4be94b5faab00f26ef2765e8dc835a98e65bc9f
uint256 id: 0
```

Fig. 15 The Subscribe Request for Plan 0 of SLA 0.

The message generated from nonce 0 and the hash of an SLA is given below:

"0x6a696d59deaf6ad98489800e74be94b5faab00f26ef2765e8dc835a98e65bc9f,0"

The signature generated from the message is given below:

```
"0x3099eb8b5ff4a74fb9e3968ff64abd75f197d57fa526d3
59517db0d2296d8b451b9e15cb908028e4b2be99363e4c4f7
bfda77e0f42b668d041c21c834e7beac21c"
```

4 wei is the minimal unit of Ether, the Ethereum inner currency.

The public key generated for the signature and message is given below:

"0x31800bf1ebed99f11977d81f6ba71cdcc0670664c7ebe7a03ea33de0982e15e0ebfad233d83e4a154c63157fdbb2f991f0a0e39ff3f444f626eef762869a5e8a"

The address generated from the public key is given below:

"0xb2c5a2fe26458387822481bc718a4f92f6e91ad4"

From the above results, the identity of the Web API request publisher was verified.

2) Evaluation

Applying the proposed method to the example use cases proved the concept of SLA contracting based on the common SLA contract platform.

The restriction of the invocation of unpermitted consumer, and operations on the unpermitted document can be implemented with "modifier" functions.

A function shown Fig. 16 is invoked for recording an SLA document onto the blockchain after validating the permission of the consumer and document.

```
Function setSlaOpen(uint _id)
  slaExsist(_id)
  onlyOwner(_id) {
  slas[_id].open = true;
}
```

Fig. 16 setSlaOpen Function

Fig. 17 shows the function of permission validation. The variable "msg" is build-in. The message "msg.sender" has the address of callee. This function compares the address of SLA register and the function caller, and raises an exception if they are not identical (Fig. 17).

If the contract function with a payment called with not enough fee, an exception will be raised.

The function shown in Fig. 18 has a payable modifier. It can attach an arbitrary amount of ether for function call. If the attached amount of ether is greater than or equal to the required fee of a subscribe plan, sla.publisher.send(plan.fee) becomes true, and the function executes. If not, an exception will be raised.

The validation of SLA document and Web API request were implemented with Ethereum basic functions, ecsing function and ecrecover function.

B. APIMS Document Genetator

The APIMS document generator generates an APIMS document from an API Bluerpint document. It is used by service providers.

It is implemented in Node.js of 742 LOC based on the API Blueprin parser (Fig. 19).

```
Modifier onlyOwner(uint _id) {
  if (slas[_id].publisher != msg.sender)
    throw;
  _;
}
```

Fig. 17 The Permission Validate Function

```
function subscribe(uint _slaId, uint _planId)
  payable slaExsist(_slaId)
planExsist(_slaId, _planId)
  returns (bytes32 hash, uint id) {

  SLA sla = slas[_slaId];
  Plan plan = sla.plans[_planId];
  if (sla.publisher.send(plan.fee) == false)
    throw;

  id = plan.subscriptions.length++;
  hash = sha256(_slaId, _planId, now);

  Subscription subscription =
plan.subscriptions[id];
  subscription.hash = hash;
  subscription.publisher = msg.sender;

  uint t = now;
  subscription.activate = t;
  subscription.finished = t +
sla.plans[_planId].span;
}
```

Fig. 18 Subscribe Function

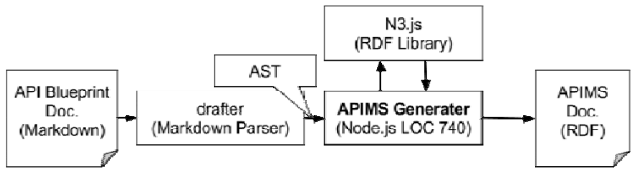
N3.js
(RDF Library)
AST
API Blueprint Doc. (Markdown)
drafter (Markdown Parser)
APIMS Generater (Node.js LOC 740)
APIMS Doc. (RDF)

Fig. 19 APIMS Document Generator

1) Applying to the Example Use Cases

We generate the APIMS documents of 3,200 LOC from the 20 examples in API Blueprint of 2,300 LOC [1].

2) Evaluation

We compared the number of entities in the API Blueprint documents with the number of resources in the APIMS documents, and verified the no missing entities in the generated APIMS. This experiment assures the compatibility from API Blueprint to APIMS.

Fig. 20 shows the function generating the ResourceGroup resource.

VIII. Discussion

A. Extensibility of Web API Specification

The contract based on the proposed common SLA contract platform is driven by the consumers, and the change history of the SLA documents should be traceable. Furthermore, an extensibility of the SLA documents is needed for service orchestration.

However, conventional specification descriptions of Web APIs are based on Markdown [2] and JSON [16] [18], and hard to extend. The proposed specification description is based on RDF, and can be extendable by its very nature.

```
var getResourceGroupNode =
  function(writer, resourceGroup) {
  // resourceGroup Data

  var triples =[ {
      predicate: 'rdf:type',
      object: 'apims:ResourceGroup'
  } ];
  var name = resourceGroup.name;
  if(name != "") {
    triples.push({
      predicate: 'apims:name',
      object: '"' + name + '"'
    });
  }
  ...
  var resources = resourceGroup.resources;
  resources.forEach(
    function(resource, index, resources) {
      triples.push({
        predicate: "apims:resource",
        object: getResourceNode(
          writer, resource
        )
      }
    );
  });

  return writer.blank(triples);
};
```

Fig. 20 Function Generating the ResourceGroup

B. *Verifiability of Web APIs*

Specification descriptions of the most Web APIs are in natural language, and can't be verified. Thus, they are error prone. However, the proposed Web API specification description can be verified by shape, and can assure the quality of the specifications. Furthermore, it is possible to mine the Web APIs with SPARQL queries based on the shapes [1] [13]. This is particularly useful if it is required to orchestrate a large number of Web APIs.

C. *Security of SLA Contract Platform*

The conventional SLA contract is concluded with the information of a payment service, and its behavior and security are closed under the hood of service providers. With the proposed method, the contract is concluded with the fact of inner currency payment with using open data.

D. *Provider Evaluation Based on Performance*

The open data of Web APIs may differ by providers. The open data of the SLA contract based on SSLAP are stored on blockchain and traceable. The quality and trust of the providers can be evaluated with the open data.

E. *On Demand Usage of Web APIs*

When a consumer uses a new Web API, the deployment and contracting of the service are required. The contract based on the proposed common SLA platform automates that process, and the consumer can use it on demand.

IX. FUTURE WORKS

We are considering the following future works.

(1) Minimize the cost of contract execution
(2) Implementing the verification method of Web API and SLA documents with SHACL
(3) Extending the description of obligations and metrics.

X. CONCLUSION

The service orchestration of Web APIs is increasing, but needs painful and error prone tasks [9]. This article proposed the specification description of Web APIs and associated SLA based on RDF, and SLA contract platform based on the blockchain. We applied them to example use cases, and demonstrated the feasibility. The proposed system enables to conclude SLA contracts based on a secure contract platform on demand and provide services to consumers on demand.

XI. REFERENCES

[1] apiaryio,apiaryio/api-blueprint/examples/, https://github.com/apiaryio/api-blueprint/tree/master/examples, (2017-02-02 Accessed).
[2] API Blueprint, https://apiblueprint.org, (2017-02-02 Accessed).
[3] M. Atzori, et al., Blockchain-Based Architectures for the Internet of Things: A Survey, University College of London, May 2016, 8 pages.
[4] K. Ballinger, et al., Basic Profile Version 1.0, http://www.ws-i.org/profiles/BasicProfile-1.0-2004-04-16.html, 2004.
[5] V. Buterin, A Next Generation Smart Contract and Decentralized Application Platform, Ethereum White Paper, 2014, https:// www.weusecoins.com/assets/pdf/library/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf (2017-2-20 Accessed).
[6] K. Christides, et al., Blockchains and Smart Contracts for the Internet of Things, IEEE Access, Vol. 4, 2016, pp. 2292-2303.
[7] K. Delmolino, et al., Step by Step towards Creating a Safe Smart Contract, Financial Cryptography and Data Security (Proc. of FC 2016), LNCS Vol. 9604, Springer, Feb. 2016, pp. 79-94.
[8] ECMA, Standard ECMA-404 The JSON Data Interchange Format, https://www.ecma-international.org/publications/standards/Ecma-404.htm, (2017-02-02 Accessed).
[9] T. Espinha, et al., Web API Growing Pains: Stories from Client Developers and Their Code, Proc. of CSMR-WCRE 2014, IEEE Computer Society, Feb. 2014, pp. 84-93.
[10] Ethereum Foundation, Ethereum Project, https://www.ethereum.org, (2017-02-02 Accessed).
[11] A. Hari, et al., The Internet Blockchain: A Distributed, Tamper-Resistant Transaction Framework for the Internet, Proc. of ACM HotNets 2016, ACM, Nov. 2016, pp. 204-210.
[12] S. Harris, et al., SPARQL 1.1 Query Language, Mar 2013, https://www.w3.org/TR/sparql11-query/.
[13] H. Knublauch, et al., Shapes Constraint Language (SHACL), Aug 2016, https://www.w3.org/TR/shacl/.
[14] H. Ludwig, et al., Web Service Level Agreement (WSLA) Language Specification, IBM, 2003.
[15] S. Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, https://bitcoin.org/bitcoin.pdf (2017-02-02 Accessed).
[16] OpenAPI Specification, http://swagger.io/specification/, (2017-02-02 Accessed).
[17] ProgrammableWeb, https://www.programmableweb.com/, (2017-02-02 Accessed).
[18] RAML, http://raml.org, (2017-02-02 Accessed).
[19] A. G. Ryman, Resource Shape 2.0, 2014, http://www.w3.org/Submission/2014/SUBM-shapes-20140211/.
[20] M. Vukovic, et al., Riding and Thriving on the API Hype Cycle, CACM, Vol. 59, No. 3, Mar. 2016, pp. 35-37.