A RESTful Approach to Service Level Agreements for Cloud Environments

Florian Blümel
Dortmunder Initiative zur rechnerintegrierten Fertigung (RIF) e.V.
Dortmund, Germany
florian.bluemel@rt.rif-ev.de

Thijs Metsch
Platform Computing GmbH
Ratingen, Germany
tmetsch@platform.com

Alexander Papaspyrou
Institut für Roboterforschung
Technische Universität Dortmund
Dortmund, Germany
alexander.papaspyrou@tu-dortmund.de

Abstract—Cloud Computing is becoming more and more a commodity service to all kinds of businesses. This leads to a stronger need for dependable service guarantees on the resources or applications offered to the customer by the provider via Service Level Agreements. Most offerings on the market, however, rely on non-electronic modes rather than machine-manageable means. In this paper, we propose a protocol that closes this gap for REST-based services and leverages the benefits of HTTP. We build upon the widely established WS-Agreement document format and show how our protocol integrates with the currently popular Open Cloud Computing Interface.

Index Terms—RESTful Web Services; Service Level Agreements; Open Cloud Computing Interface

I. INTRODUCTION

Cloud Computing has become a new driver for the global IT market. With its vision of delivering services to the customer over simple, commoditized interfaces and its paradigm of paying per usage for the provided resources, Cloud Computing poses a main actor in the ongoing change of the IT landscape.

Still, not only the underlying technology is evolving: much more, the new way of providing a service to the customer via mainly self-service mechanisms requires the modernization of current contracting best practices. Especially with respect to service guarantees, most interaction is done in a very traditional way, that is providing the customer with the so-called "Terms of Service (ToS)" usually through a written document which is implicitly accepted with the usage of the service.

This rather gross, albeit very wide-spread approach to QoS has obviously disadvantages: the management of the ToS is disconnected from the service itself. Moreover, no electronic representation for the individual contract is available. Also, it is impossible to induce flexibility into the QoS terms on a per-customer basis without complex out-of-band communication (typically via special contracts between the legal representatives of both the consumer and the provider).

Electronic Service Level Agreements (SLAs) [1] provide a solution to this problem for many years. These can be used to make an machine-readable agreement on the ToS while at the providing enough flexibility for both parties to change them to their needs. In this paper, we propose a protocol and format for REST-based services to create, modify, agree upon, and manage electronic SLAs by leveraging the features of HTTP-based interaction [2].

The rest of the paper is organized as follows: in Section II, we formulate scenarios for the usage of electronic SLAs; in Section III, we describe the overall architecture and RESTful resources used for interaction; in Section IV we discuss the used protocol; in Section V, we outline an integration path for the Open Cloud Computing Interface; in Section VI, we place our work into the greater research context and compare similar projects to ours; and in Section VII, we conclude our work.

II. USAGE SCENARIO

In today’s service marketplaces, service offerings are very limited: customers usually do not have much choice in what is offered by a service provider. This leads to a very simple ”take it or leave it” situation. However, this is not only problematic for the customer (having little influence on what is offered and even less transparency in terms of QoS violations), but also for the provider, not being able to make near-line customized offerings based on the current supply and demand.

In fact, modern service marketplaces offer little support for other than non-negotiable, non-machine readable SLAs. This is not only uncomfortable for both contractor sides, but also may pose a major judicial problem if data for cloud comes into play: data privacy law in many countries requires certain protection for stored information being exposed to such services, to the extent of determined physical locations requirements on the data centers.

As such, it is necessary to offer Predictability, Dependability, and Automation to allow for a user environment that fulfills the compliance requirements in modern businesses. The

1Quality of Service.

2E.g. the Amazon WebServices, Microsoft Azure, Longjump PaaS, and others.
requirements directly deriving from this on a protocol level are:

1) Advertise a service offering (along with the conditions of a template SLA),
2) Negotiate on an instance of this offering, allowing customization within certain bounds, and
3) Be notified when the actual service performance deviates from the terms of the agreed offering.

The aforementioned scenario covers practically all current IaaS offerings on the market. We will show later an example of how to conduct service integration the currently popular the Open Cloud Computing Interface.

III. Architecture

We describe our architecture along the model of REST, starting with a definition of the resources we use and the renderings we support. Then, we detail the protocol, walking through the different steps until coming to an agreement.

As a basis for our system, we use the widely-accepted Web Services Agreement (WS-Agreement) recommendation as published by the Open Grid Forum in Version 1 [3]. More specifically, we rely on WS-Agreement for the overall structure of the SLA document itself, and provide the original document format as one rendering of our resource, see Section III-A; on the other hand, we loosely refer to the the templating mechanisms of WS-Agreement as described in Section IV.

A. Resources

The resources comprising the interface can be grouped by use cases:

- Offering and accepting agreements
- Querying existing agreements and their state
- Publishing templates as a basis for agreements
- Interactive negotiation

Agreements and templates cover a common range of general information; this functionality is presented separately as a common base.

Figure 1 illustrates the resources described in this section, and the relations between them. A hierarchical structuring with sub-resources is depicted as a set of aggregations and compositions. The associations in the figure show the realization of the HATEOAS\(^3\) principle:

- Aggregations and compositions imply forward links from the whole to its parts and backward links from the parts to the whole.
- An undirected general association models mutual links without hierarchical implications.
- A link without a back link is pictured as a directed general association.

\(^3\)Hypertext As The Engine Of Application State.

1) General information: Any service level agreement has to declare the involved parties and other frame conditions, e.g. an expiration time. Apart from these, the central points to an agreement can be summarised as conditions and terms. This includes the specification of agreed upon services and guarantees for the quality of these services.

Agreements and templates have to be addressable by some identifier. This identification has to be immutable and unique between the parties. It is made available by an agreement’s sub-resource Identifier.

Agreements and templates can be named. This aims to provide a way for memorable, human readable naming; such names are optional and not necessarily unique. They could be used for categorizing or searching. The name is provided by the sub-resource Name.

The resource Context models the context information to an agreement, i.e. the involved parties and expiration time. For agreements created from templates, a reference to the template can be included.

The identification of parties could include authentification data or digital signatures. In practice, concrete technologies for authentication are often dictated by the negotiated web services and is thus out of scope for this document.

The terms defined in an agreement or template are made available by the resource Terms. In addition to the descriptions of services and guarantees for their quality, this also includes associated business values.

2) Agreement: In the considered negotiation scenarios without renegotiation, the general informations are fixed as soon as the agreement is submitted as an offer. The resource Agreement provides these in the described sub-resources.

In addition, the state of the agreement—e.g. “Pending” or “Terminated”—and compliance to the terms has to be communicated.

The resource AgreementState models the state of an agreement. Possible states are Rejected, Observed, Complete and Terminated; with asynchronous communication, additional states Pending, PendingAndTerminating and ObservedAndTerminating are introduced.

![Fig. 2. Agreement states and the transitions between them.](image-url)
The state of individual services is provided by the sub-resources ServiceState. The basic states NotReady, Ready and Completed could be extended with application specific substates.

For each negotiated guarantee, a GuaranteeState resource provides information on its fulfillment. Possible values are Fulfilled, Violated and NotDetermined. The exact interpretation of these states has to be defined in context, e.g. whether fulfillment is checked on each request for the guarantee’s state or whether the guarantee and thus the provided state correspond to some time interval.

A client can subscribe to notifications about state changes for an agreement. Subscriptions are managed through the SubscriptionCollection resource; the collection contains Subscription resources. Subscribers are notified about changes for the AgreementState resource and any ServiceState and GuaranteeState resources.

3) Templates: The resource TemplateCollection models an enumeration of templates for agreements, that would be accepted by the provider (with reservations). Individual templates are provided as Template sub-resources. A template includes the general information described, in particular the sub-resources Context and Terms. The sub-resources Identifier and Name are used for the template itself and are not to be interpreted as defaults for derived agreements.

With the terms in an agreement, acceptable values for parameters to an agreement are specified. In addition, constraints and value for parameters can be defined, implying customization points for agreements. The constraints are provided by the ConstraintCollection; the elements of this enumeration are Constraint resources.

4) Enumeration and offering: The resource AgreementFactory models a collection of accepted offers and agreements. It contains not only agreed-upon and running, but also rejected, terminated or completed agreements. Inserting into the collection is interpreted as offering a new agreement; that is, this resource provides the negotiation of agreements.

In applications where agreements are legally binding contracts, agreements may have to be held available after completion or termination. Whether agreements can be removed from the collection and under what conditions is thus to defined in application context.

5) Interactive negotiation: As a basis for more complex negotiation scenarios, an AgreementDraft resource models a modifiable draft based on a template. Apart from the general information, parameters for the agreement’s terms can be edited; for this, modifiable parameters are implied by constraints and value ranges in the template. With any change, validity of the new supplied values and conformity to the constraints can be verified; violating changes are rejected. This enables an interactive negotiation of agreements.

The modifiable parameters are provided by OfferItem resources through the OfferItemCollection. Drafts are managed with the AgreementDraftCollection resource.
B. Representation

The resources described can be represented in XML as WS-Agreement documents. Sub resources can be mapped to XML sub documents (e.g. the Context resource) or to textual representations used therein (e.g. AgreementState). Overall, this mechanism is governed by MIME [4], which we use for our model.

Specifications of services are embedded as ServiceDescriptionTerms. To describe interface semantics, domain specific languages like APP⁴ [5] or OpenSearch [6] can be used. More general specification languages for web-based service interfaces are WSDL⁵ and WADL⁶.

Resource Uniform Resource Identifiers (URIs) can be used as ServiceReferences; this addressing can be used without additional knowledge about the service description employed.

Using the content type negotiation mechanisms specified by HTTP, the described REST interface can be used to provide human readable views on agreements. With this, an overview over running agreements or guarantee fulfillment could be requested as HTML documents and viewed in any web browser, with embedded links enabling navigation.

HTML representations are provided for viewing only and are not accepted for processing. These representations aim for ease of human readability and are not suited for automatic interpretation. Sophisticated browser-based user interfaces could be implemented using JavaScript and Ajax but would use other representations for communication.

IV. PROTOCOL

For the protocol description, we walk through the main steps of an SLA lifecycle, including template choosing, negotiation, offer making, and notification.

A. Choosing a template

A list of available templates can be obtained with a GET request for the resource TemplateCollection. Depending on the usage scenario, that list could be represented as an XML document containing complete templates or as a text/uri-list pointing to individual resources. Representations of these Template resources and their components – Context, Terms and Constraints – can likewise be requested with GET.

Templates can not be altered using this interface, and no templates can be added or deleted, meaning GET is the only method allowed on these resources.

```
GET /templates/123 HTTP/1.1
Host: server.example.com
Accept: application/x-wsag+xml

Response:
HTTP/1.1 200 OK
Content-Type: application/x-wsag+xml
```

Listing 1. Requesting a template in WS-Agreement XML representation.

B. Interactive Negotiation

To initiate a negotiation, a new AgreementDraft resource is created via POST on the AgreementDraftCollection. The identifier to a server-side template is transferred as plain text in the request body. The constraints given in that template define the negotiable terms and items. At creation, the new agreement draft is prepared, i.e. a unique identifier is chosen. The response 201 Created then points to the newly created resource via Location.

```
Request:
POST /drafts/ HTTP/1.1
Host: server.example.com
Content-Type: text/plain

Location: /drafts/123

Response:
HTTP/1.1 201 Created
```

Listing 2. Creating a draft for interactive negotiation.

The list of negotiable items is supplied by the OfferItemCollection. This resource responds to GET with a text/uri-list addressing OfferItem resources. The current values of these items can be requested with GET and new values can be PUT. New values are validated and possibly, if they violate constraints, rejected.

```
Request:
GET /drafts/123/items/Availability HTTP/1.1
Host: server.example.com
Content-Type: text/plain

Response:
HTTP/1.1 200 OK
```

Listing 3. Setting a new value of 99.9% for the negotiable item “Availability”.

The current state of an agreement draft can be obtained at any time by requesting a representation of the AgreementDraft resource. If a draft is no longer needed, removal from the AgreementDraftCollection can be requested with DELETE.

C. Making an offer

An offer can be submitted by POSTing it to the AgreementFactory resource. By supplying the header field X-Allow-Deferral, the client can allow a deferred decision. The default is to require an immediate decision, which makes supporting the asynchronous scenario optional for both parties.

If the offer is not immediately rejected, a new Agreement resource is created; however this does not necessarily

```
<?xml version="1.0" encoding="UTF-8"?>
<Template
  xmlns="http://www.ggf.org/namespaces/ws-agreement"
  TemplateId="123">
  <Name>Template for agreements</Name>
  <Context>
    ...
  </Context>
</Template>
```

Listing 1. Requesting a template in WS-Agreement XML representation.

⁴Atom Publishing Protocol.
⁵Web Service Description Language.
⁶Web Application Description Language.
mean that an agreement has been reached. To indicate a deferred decision, the server responds with 202 Accepted; the agreement state is Pending. If an immediate decision is possible and the agreement is made upon creation, the response is 201 Created. In this case the agreement is binding and in state Observed. Lastly, instant rejection is expressed with 403 Forbidden and no resource is created.

After an Agreement resource has been created, a client can subscribe to notifications via the subresource Subscriptions. Since events could have been missed at this point, the subscription can also be passed with the offer. To this end, the subscribing, client-side resource is addressed through a Link header field with relation “Subscription”. This subscription is added while creating the server-side Agreement resource and is notified of the agreement’s initial state (Pending or Observed).

Request:
POST /agreements HTTP/1.1
Host: server.example.com
Content-Type: application/x-wsag+xml
X-Allow-Deferral: true
Link: <http://client.example.com/ \notifications/agreement>; rel=Subscription

<?xml version="1.0" encoding="UTF-8" ?>
<Agreement AgreementId="123" xmlns="..." >
 <Name>Test</Name>
 <Context>...</Context>
 <Terms>...</Terms>
</Agreement>

Response:
HTTP/1.1 201 Created
Location: /agreements/123

Listing 4. Submitting an offer, immediate acceptance.

D. Notifications

To subscribe to notifications for state changes, a new subscription is POSTed to the SubscriptionCollection. A client-side resource will be the target for notification and has to be specified with the request. The created Subscription is referenced in the Location field of the 201 Created response. This subscription can later be terminated by requesting DELETE.

Notifications are published by the server as a POST on the specified client-side resource. A representation of the new resource state is transmitted with the notification; the changed resource is indicated as a Link field with relation “Resource”. The subscription causing the notification is specified as a Link to “Subscription”.

Request:
POST /notifications/agreement HTTP/1.1
Host: client.example.com
Link: <http://server.example.com/ag/state >; rel=Resource
Link: <http://server.example.com/ag/subscriptions/I >; rel=Subscription

Completed

Listing 5. Notification about an agreement expiring.

This simple resource based realization of notifications follows the pattern for RESTful Webhooks [7].

E. Enumerating Agreements

In addition to accepting new agreement offers, the resource AgreementFactory supplies an overview of made agreements and pending offers. A text/uri-list representation can be obtained with GET. The listed Agreement resources and their components can be retrieved as XML documents, but human readable representations, e.g. HTML documents, can also be provided.

Agreements can be deleted by requesting DELETE. However this may or may not be admissible for a specific usage scenario; agreements may have to be preserved whenever they correspond to legally binding contracts. Deleting agreements is generally only acceptable when the agreement is Complete, Rejected or Terminated. For all other agreement states, the deletion request is rejected with a 409 Conflict response. In particular, running agreements must be explicitly terminated prior to deletion.

Since the agreement and its components are binding and immutable, the methods PUT and POST are not allowed; furthermore, no component will allow DELETE.

F. Terminating agreements

Terminating an agreement corresponds to requesting a transition of its AgreementState resource to the state Terminated, i.e. a PUT request. If the request is accepted, the response is 200 OK or 204 No Content. Rejection is expressed with 403 Forbidden; a domain specific reason can be supplied as message body. The decision can also be deferred, resulting in an immediate transition to ObservedAndTerminating or PendingAndTerminating, a 202 Accepted response, and a later transition to Terminated or back to the original state.

Termination requests for Rejected or Complete agreements are rejected with 409 Conflict; likewise, any transitions to other states than Terminated are rejected with 403 Forbidden.

Request:
PUT /agreements/123/state HTTP/1.1
Host: server.example.com
Content-Type: text/plain

Terminated

Response:
HTTP/1.1 202 Accepted
Content-Type: text/plain

ObservedAndTerminating

Listing 6. Terminating an agreement. The request is accepted, but a decision can not be made immediately. The response is thus 202 Accepted and the new resource state is ObservedAndTerminating.
In order to show the real-world applicability and benefits of our RESTful SLA approach, we show the integration points and effort of the proposed protocol into the popular Open Cloud Computing Interface [8]–[10].

The OCCI family of specifications was developed as an open, interoperable protocol for Cloud Computing applications. The main focus was on IaaS management, but due to the modularity of the interface, other responsibilities in the Cloud stack can be covered as well.

OCCI provides a simple model for infrastructure provisioning by offering three simple resource types, namely Compute, Storage, and Network on the basis of a more generic type system, see [11]. Besides these, link resources are offered to interconnect them. Through the rendering over HTTP with a RESTful model, OCCI provides a number of additional features like a simple discoverable type system, Actions and the handling of Collections.

The type of a REST resource within the OCCI domain is exposed using a Category. The Category description not only indicates the type of a resource within the OCCI type system but also exposes the attributes, actions, and relations a resource might have. Categories can related to each other and therefore a hierarchy can be setup.

Each resource has a single kind (the type definition) and can have multiple mixins. Mixins are a way of adding capabilities, tagging resources/features as well as a way of templating. The Category can be rendered using the Category field as described by Johnston, and Links between resources are exposed through the Link as recommended by RFC 5988 [12].

All Categories are exposed too the client using a Query Interface. Which makes is possible for clients to discover the features a Service Provider offers. When Categories define a template those are also queryable.

With this query interface in place the client can always query the Service Provider and determine if the Categories needed to setup an Agreement are available. Next to these Category definitions the client can request templates which could represent certain Levels of an Agreement (Platinum, Gold, Silver, ...).

After filling out the templates an negotiation process can be started. If an agreement can successfully be agreed upon the Client has the final word and ensures that the state of the resource instance agreement is set to ‘agreed’

Since it is necessary to provide a way of locating Agreement resources from the services agreed upon, we use the Links as described by the OCCI model to embed the agreement URI in an non-intrusive way: by pointing to an agreement, a relation like sla can be used to reference the resource. This particularly means that a certain resource falls under an certain Agreement. REST Resource can be added to an agreement during their creation or later in their life-cycle. By removing the link between a resource and an Agreement it is indicated that the resource no longer falls under an Agreement.

It it noted here that the Agreement resources and e.g. compute resource instance do not necessarily need to belong to the same user. A company could have an agreement with an Service Provider while individual employees of the company have compute instances which link to the company wide Agreement instance.

In particular, by specifying a way to locate agreements related to known services, RESTful SLAs can be used as a purely informational interface and without providing negotiation. As such, the integration looks like the following:

```
Request:
GET /compute/123 HTTP/1.1
User-Agent: curl/7.21.0 [...] Host: localhost:8888
Accept: */*
Cookie: [...]  

Response:
HTTP/1.1 200 OK
Content-Type: text/plain
Server: pyocci OCCI/1.1
Category: compute;
scheme=http://schemas.ogf.org/occi/...
Link: </agreements/456> rel=sla
X-OCCI-Attribute: ...
```

Listing 7. A resource featuring a Link to an SLA.

Protocol-wise, after settling an agreement, the resource creation on the infrastructure level exposes an additional Link used for referring to the settled agreement resource. Within the agreement, the attributes (as exposed by the X-OCCI-Attribute) header field are exposed as service terms along with guarantee constraints for fulfillment and violation. This way, monitoring and auditing systems may check the current value of any attribute against the settled agreement resource (which, of course, may be hosted by a trusted third party).

Overall, this approach allows for an integration into other RESTful protocols with a very small footprint. In addition to that, REST resources comprising the business logic of a given service and the entities managed hereby are kept strictly separate from the agreement itself, allowing for a non-domain specific implementation of the SLA part while still being able to integrate properly for a concrete service offering.

VI. RELATED WORK

REST as an architectural style (as described by Fielding [13]) and the usage of electronic Service Level Agreements [14], [15] are both well-covered research fields, with many projects working on the topic.

Phosphorus [16] makes use of SLAs to manage co-allocation of resources from the compute and network domain for traditional HPC job scenarios which require dedicated QoS parameters. BREIN [17] aims to support outsourcing for an increased competitiveness of SMEs by enhancing classical IaaS scenarios with a standards-based environment for eBusiness, including SLAs, to allow for provision and
sale of services. Bazaar\(^7\) aims towards transparent SLA management for eInfrastructure in the European EGEE efforts. SLA@SOI [18] addresses the entire SLA framework in Service Oriented Infrastructure (SOIs) and delivers architecture and framework for such environments.

Regarding SLA languages and protocols, only WS-Agreement is currently under active development and uses a broad approach to SLAs. Other languages like NextGrid SLA [19], SLAng [20], or WSLA [21] either ceased support, or are proprietary and not well-established.

As such, most projects and efforts (including the ones listed above) base on WS-Agreement, however focus on working the business requirements rather than introducing new ideas to the protocol level. Only little effort has been put into the RESTful world so far besides the work from Küber, Kataros and Wang [22], who propose a direct transformation of WS-Agreement to a REST-based interface. This paper describes how the entities in the WS-Agreement specification can be mapped to a RESTful interface. Similar effort - and therefore also naming conventions - have been used within this paper. Certainly the work presented within this paper goes further than just mapping WS-Agreement and looks into the Negotiation process as well as how the proposal can be used for Cloud Computing Environment.

**VII. CONCLUSION**

In this paper, we presented a RESTful approach to Service Level Agreements for modern Cloud Computing environments. For the usage scenario at hand, we proposed an architecture for SLA creation, modification, agreement, and basic management of SLAs using a purely REST-based system. To this end, we described our template-based resource model and introduced a protocol that is able to leverage the features of HTTP, requiring only minimally invasive changes to already available REST interfaces. In order to prove the benefits of our approach, we showed the service integration steps for the popular Open Cloud Computing Interface. A draft specification, describing Agreements and Negotiation of Agreements for the Open Cloud Computing Interface, have been submitted to the working group.

Future work will address a more standards-based notification mechanism (e.g. moving to RSS feeds for notification) and additional MIME renderings for the agreement resource, such as WSLA. Moreover, we will extend the current protocol towards negotiation and renegotiation capabilities in order to address change requests after SLA settlement in an appropriate, trustful way.

**ACKNOWLEDGEMENT**

This work is partly funded by the German Federal Ministry of Education and Research under project grant #01IG09009.

---

\(^7\)https://www.ce-egee.org/weblog/bazaar.


