Automated Reverse-Engineering of a Cloud API

Stéphanie Challita, PhD
Associate Professor @University of Rennes 1, ESIR & IRISA/Inria, DiverSE team
Brief bio

- **Degree of Systems and Networks Engineering, 2010 - 2015**
- **Research Master’s degree in Computer Science, 2014 - 2015**
- **PhD in Computer Science, 2015 - 2018**
  - Spirals team (Lille, CRIStAL, Inria)
- **Postdoctoral Researcher, 2019 - 2020**
  - Kairos team (Sophia Antipolis, I3S, Inria)
- **Associate Professor, since September 2020**
  - DiverSE team (Rennes, IRISA, Inria)
Research project

Towards the automatic construction of reliable and co-evolving software systems

Scientific challenges

- **Inference of Domain-Specific Modeling Languages (DSML) from APIs**
  - Precision & Genericity
  - Learning

- **Verification**
  - Generation of instances of inferred DSMLs
  - Generation of oracles

- **Co-evolution** of APIs and languages
  - Identify the impact of API changes on DSML
  - Co-evolve the impacted components
Research team

- Modeling & Language Engineering
- Advanced testing
- DevOps for distributed and heterogeneous systems
- Variability Engineering

https://www.diverse-team.fr/
Cloud computing

Created by Sam Johnston, downloaded from https://en.wikipedia.org/wiki/Cloud_computing
Multi-cloud computing

Multi-clouds ensures:
- Increase performance
- Reduce costs
- Ensure resiliency during outages
Multi-cloud computing

Multi-clouds ensures:
- Increase performance
- Reduce costs
- Ensure resiliency during outages
- No interoperability
- Vendor lock-in

Different deployment models
Different management interfaces
Different service models
Different semantics
Lack of formalization
Approaches for multi-clouds - Actors

Cloud provider

Cloud developer

Cloud architect

Offer

Use

Use
Approaches for multi-clouds

Diagram showing the connections between multi-cloud libraries, cloud brokers, APIs, and cloud providers. The diagram includes AWS SDK, DigitalOcean SDK, GCP SDK, OCCI SDK, AWS API, DigitalOcean API, GCP API, OCCI API, and CIMI, with arrows indicating the flow between these components.
Approaches for multi-clouds

Cloud Meta-metamodel

Cloud Metamodel

M3

Cloud Model

M2

Cloud architect

defines

represented by

M1

M0

Documentation

Code generation

Model

Static analysis

Transformation
Approaches for multi-clouds
Approaches for multi-clouds

Issue:

Fuzziness of the concepts of the cloud modeling languages
Research question

**RQ:** Is it possible to automatically extract precise models from cloud APIs and to synchronize them with the cloud evolution?

- How to **provide an accurate description** for a cloud API?
- How to **correct** the existing drawbacks in a cloud API documentation?
- How to **analyze** a cloud API documentation?

**Research topics:** API mining, reverse-engineering, NLP
Global vision

Model-Driven Approach for the Cloud

AWS  GCP  OCCI
An agreement with the developer on exactly how the system will operate

Cloud documents are written in natural language → human errors and/or semantic confusions
Vision

- Inferring models from cloud APIs
  - Work of API mining, reverse-engineering
    - HTML Model
  - Model refinement (NLP techniques, graphical output...)

Cloud API → Generic parser → Model Generator → Text Analysis Engine → Model Validator → Cloud DSML
Google Cloud Platform (GCP) use case
List of GCP documentation drawbacks

- Informal heterogeneous documentation
- Imprecise types
- Implicit attribute metadata
- Hidden links
- Redundancy
- Lack of visual support
Imprecise types

1. selfLink: string (Output Only) Server-defined URL for the resource.

Available at
https://cloud.google.com/compute/docs/reference/latest/targetHttpsProxies

2. email: string

The email address of the service account.
Note: This field is used in responses only. Any value specified here in a request is ignored.

Available at
https://cloud.google.com/iam/reference/rest/v1/projects.serviceAccounts

3. instanceClass: string

Instance class that is used to run this version. Valid values are:
- AutomaticScaling: F1, F2, F4, F4_1G
- ManualScaling or BasicScaling: B1, B2, B4, B8, B4_1G
Defaults to F1 for AutomaticScaling and B1 for ManualScaling or BasicScaling.

Available at
https://cloud.google.com/appengine/docs/admin-api/reference/rest/v1beta5/apps.services.versions

4. locations[]: string

The list of Google Compute Engine locations in which the cluster's nodes should be located.

Available at
**Implicit attribute metadata**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>unsigned</td>
<td>[Output Only] The unique identifier for the resource. This identifier is defined by the server.</td>
</tr>
<tr>
<td>location</td>
<td>string</td>
<td>The geographic location where the dataset should reside. Possible values include EU and US. The default value is US.</td>
</tr>
</tbody>
</table>

Available at
https://cloud.google.com/compute/docs/reference/latest/networks

Available at
https://cloud.google.com/bigquery/docs/reference/rest/v2/datasets
GCP snapshot

- GCP engineers could update/correct GCP documentation
- Continuously following up with GCP documentation is costly
- Snapshot of GCP API
GCP crawler & GCP model

- **GCP Crawler** to extract all GCP resources, their attributes and actions
- **GCP Model** for a better description of the GCP resources
GCP crawler & GCP model

**Algorithm:** Crawler and model generator

**Input:** Documentation’s URL: `url`

**Output:** Model conforms to OCCI: `model`

```java
1: model ← OCCIFactory.createModel()
2: document ← connect(url)
3: for resource in document do
4:   resourceHTML ← connect(resource.url)
5:   resourceOCCI ← OCCIFactory.createResource()
6:   resourceOCCI.properties ← extract(resourceHTML)
7:   for attributeHTML in resourceHTML do
8:     attributeOCCI ← OCCIFactory.createAttribute()
9:     attributeOCCI.properties ← extract(attributeHTML)
10:    resourceOCCI.attributes.add(attributeOCCI)
11:   model.append(resourceOCCI)
12: return model
```
Algorithm: Crawler and model generator

**Input:** Documentation’s URL: `url`

**Output:** Model conforms to OCCI: `model`

```
1: model ← OCCIFactory.createModel()
2: document ← connect(url)
3: for resource in document do
4:   resourceHTML ← connect(resource.url)
5:   resourceOCCI ← OCCIFactory.createResource()
6:   resourceOCCI.properties ← extract(resourceHTML)
7: for attributeHTML in resourceHTML do
8:   attributeOCCI ← OCCIFactory.createAttribute()
9:   attributeOCCI.properties ← extract(attributeHTML)
10: resourceOCCI.attributes.add(attributeOCCI)
11: model.append(resourceOCCI)
12: return model
```
GCP crawler & GCP model

**Algorithm:** Crawler and model generator

**Input:** Documentation’s URL: `url`

**Output:** Model conforms to OCCI: `model`

1. `model ← OCCI.Factory.createModel ()`
2. `document ← connect (url)`
3. **for resource in document do**
   4. `resourceHTML ← connect (resource.url)`
   5. `resourceOCCI ← OCCI.Factory.createResource ()`
   6. `resourceOCCI.properties ← extract (resourceHTML)`
   7. **for attributeHTML in resourceHTML do**
      8. `attributeOCCI ← OCCI.Factory.createAttribute ()`
      9. `attributeOCCI.properties ← extract (attributeHTML)`
     10. `resourceOCCI.attributes.add (attributeOCCI)`
    11. `model.append (resourceOCCI)`
4. `return model`
GCP crawler & GCP model

Algorithm: Crawler and model generator
Input: Documentation’s URL: url
Output: Model conforms to OCCI: model
1: model ← OCCIFactory.createModel()
2: document ← connect(url)
3: for resource in document do
   4: resourceHTML ← connect(resource.url)
   5:  resourceOCCI ← OCCIFactory.createResource()
   6:  resourceOCCI.properties ← extract(resourceHTML)
   7:  for attributeHTML in resourceHTML do
      8:     attributeOCCI ← OCCIFactory.createAttribute()
      9:     attributeOCCI.properties ← extract(attributeHTML)
     10:    resourceOCCI.attributes.add(attributeOCCI)
     11:    model.append(resourceOCCI)
4: return model
GCP crawler & GCP model

Algorithm: Crawler and model generator

Input: Documentation’s URL: url

Output: Model conforms to OCCI: model

```java
1: model ← OCCIFactory.createModel()
2: document ← connect(url)
3: for resource in document do
   4:   resourceHTML ← connect(resource.url)
   5:   resourceOCCI ← OCCIFactory.createResource()
   6:   resourceOCCI.properties ← extract(resourceHTML)
   7: for attributeHTML in resourceHTML do
      8:       attributeOCCI ← OCCIFactory.createAttribute()
      9:       attributeOCCI.properties ← extract(attributeHTML)
     10:      resourceOCCI.attributes.add(attributeOCCI)
     11:     model.append(resourceOCCI)
8: return model
```

<p>
Google BigQuery is a data warehouse [...] queries
</p>

```html
<h3>BigQuery</h3>

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>descr</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>string</td>
<td>The geographic [...] The default value is US.</td>
</tr>
<tr>
<td>lastModifiedTime</td>
<td>long</td>
<td>[Output-only] The date [...] the epoch.</td>
</tr>
</tbody>
</table>
```

30/09/2020

Stéphanie Challita @ENS seminar
Algorithm: Crawler and model generator

Input: Documentation’s URL: url

Output: Model conforms to OCCI: model

1: model ← OCCIFactory.createModel()
2: document ← connect(url)
3: for resource in document do
   4:    resourceHTML ← connect(resource.url)
5:   resourceOCCI ← OCCIFactory.createResource()
6:   resourceOCCI.properties ← extract(resourceHTML)
7: for attributeHTML in resourceHTML do
   8:      attributeOCCI ← OCCIFactory.createAttribute()
9:      attributeOCCI.properties ← extract(attributeHTML)
10:     resourceOCCI.attributes.add(attributeOCCI)
11:    model.append(resourceOCCI)
12: return model

<h3>BigQuery</h3>

Google BigQuery is a data warehouse [...] queries

<table>
<thead>
<tr>
<th>name</th>
<th>location</th>
<th>type</th>
<th>string</th>
<th>descr</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Output-only] The date [...] the epoch.
Algorithm: Crawler and model generator

Input: Documentation’s URL: `url`

Output: Model conforms to OCCI: `model`

1: `model ← OCCIFactory.createModel()`  
2: `document ← connect(url)`  
3: for resource in document do  
4: `resourceHTML ← connect(resource.url)`  
5: `resourceOCCI ← OCCIFactory.createResource()`  
6: `resourceOCCI.properties ← extract(resourceHTML)`  
7: for attributeHTML in resourceHTML do  
8: `attributeOCCI ← OCCIFactory.createAttribute()`  
9: `attributeOCCI.properties ← extract(attributeHTML)`  
10: `resourceOCCI.attributes.add(attributeOCCI)`  
11: `model.append(resourceOCCI)`  
12: return `model`

---

<h3>BigQuery</h3>

Google BigQuery is a data warehouse [...] queries

```
<tr>
<td>location</td>
<td>string</td>
<td descr="The geographic [...] The default value is US."
</tr>
<tr>
<td>lastModifiedTime</td>
<td descr="[Output-only] The date [...] the epoch."</td>
</tr>
```

---

30/09/2020 Stéphanie Challita @ENS seminar 29/41
GCP crawler & GCP model

Algorithm: Crawler and model generator
Input: Documentation’s URL: url
Output: Model conforms to OCCI: model

1: model ← OCCIFactory.createModel()
2: document ← connect(url)
3: for resource in document do
4:   resourceHTML ← connect(resource.url)
5:   resourceOCCI ← OCCIFactory.createResource()
6:   resourceOCCI.properties ← extract(resourceHTML)
7: for attributeHTML in resourceHTML do
8:   attributeOCCI ← OCCIFactory.createAttribute()
9:   attributeOCCI.properties ← extract(attributeHTML)
10: resourceOCCI.attributes.add(attributeOCCI)
11: model.append(resourceOCCI)
12: return model

<h3>BigQuery</h3>
<p>Google BigQuery is a data warehouse [...] queries</p>
<table>
  <tr>
    <!-- name -->
    <td>location</td>
    <!-- type -->
    <td>string</td>
    <!-- descr -->
    <td>The geographic [...] The default value is US.</td>
  </tr>
  <tr>
    <!-- name -->
    <td>lastModifiedTime</td>
    <!-- type -->
    <td>long</td>
    <!-- descr -->
    <td>[Output-only] The date [...] the epoch.</td>
  </tr>
</table>
Algorithm: Crawler and model generator

Input: Documentation’s URL: url

Output: Model conforms to OCCI: model

1: model ← OCCIFactory.createModel()
2: document ← connect(url)
3: for resource in document do
4:   resourceHTML ← connect(resource.url)
5:   resourceOCCI ← OCCIFactory.createResource()
6:   resourceOCCI.properties ← extract(resourceHTML)
7: for attributeHTML in resourceHTML do
8:   attributeOCCI ← OCCIFactory.createAttribute()
9:   attributeOCCI.properties ← extract(attributeHTML)
10: resourceOCCI.attributes.add(attributeOCCI)
11: model.append(resourceOCCI)
12: return model
GCP crawler & GCP model

- **GCP Crawler** to extract all GCP resources, their attributes and actions
- **GCP Model** for a better description of the GCP resources
GCP crawler & GCP model

GCP HTML pages → Snapshot → GCP documentation → GCP Crawler

- Implicit Attribute Metadata Detection
- Link Identification
- Redundancy Removal
- Model Visualization
- Type Refinement

Model Transformations
Implicit attribute metadata detection

- To explicitly store information into additional attributes defined in the ATTRIBUTE concept of our GCP MODEL

- We use Natural Language Processing (NLP) techniques
  - Word Tagging/Part-of-Speech (PoS)

- We declare pre-defined tags for some GCP specific attribute properties:
  - mutable = true if [Input-Only]
  - mutable = false if [Output-only]/read only
  - required = true if [Required]
  - required = false if [Optional]
  - default = X if The default value is X
Implicit attribute metadata detection

Algorithm: Implicit attribute metadata detection.
Input: AttributeOCCI: OCCI attribute extracted from HTML page.
Output: AttributeOCCI: OCCI attribute with explicit metadata.
1: rules ← {“The default value is”, “always”, “…” }
2: desc ← AttributeOCCI.description
3: for rule in rules do
4:    if desc.contains (rule) then
5:        value ← rule.getValue (desc)
6:        rule.setValue (AttributeOCCI, value)
7: return AttributeOCCI

Google BigQuery is a data warehouse [...] queries

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>descr</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>string</td>
<td>The geographic [...] The default value is US.</td>
</tr>
<tr>
<td>lastModifiedTime</td>
<td>typeTime</td>
<td>[Output-only] The date [...] the epoch.</td>
</tr>
</tbody>
</table>

Implicit attribute metadata detection

Algorithm: Implicit attribute metadata detection.
Input: AttributeOCCI: OCCI attribute extracted from HTML page.
Output: AttributeOCCI: OCCI attribute with explicit metadata.

1: rules ← {“The default value is”, “always”, “...”}
2: desc ← AttributeOCCI.description
3: for rule in rules do
4:     if desc.contains(rule) then
5:         value ← rule.getValue(desc)
6:         rule.setValue(AttributeOCCI, value)
7: return AttributeOCCI
Implicit attribute metadata detection

**Algorithm:** Implicit attribute metadata detection.

**Input:** AttributeOCCI: OCCI attribute extracted from HTML page.

**Output:** AttributeOCCI: OCCI attribute with explicit metadata.

1. rules ← \{ “The default value is”, “always”, “...” \}
2. desc ← AttributeOCCI.description
3. for rule in rules do
4. if desc.contains (rule) then
5. value ← rule.getValue (desc)
6. rule.setValue (AttributeOCCI, value)
7. return AttributeOCCI

```html
<h3>BigQuery</h3>
<p>
Google BigQuery is a data warehouse [...] queries
</p>
<table>
<tr>
<!-- name -->
<td>location</td>
<!-- type -->
<td>string</td>
<!-- descr -->
<td>
The geographic [...] The default value is US.
</td>
</tr>
<tr>
<!-- name -->
<td>lastModifiedTime</td>
<!-- type -->
<td>long</td>
<!-- descr -->
<td>[Output-only] The date [...] the epoch.
</td>
</tr>
</table>
```
Implicit attribute metadata detection

**Algorithm:** Implicit attribute metadata detection.

**Input:** AttributeOCCI: OCCI attribute extracted from HTML page.

**Output:** AttributeOCCI: OCCI attribute with explicit metadata.

1. `rules ← {“The default value is”, “always”, “…” }`
2. `desc ← AttributeOCCI.description`
3. `for rule in rules do`
4.   `if desc.contains (rule) then`
5.      `value ← rule.getValue (desc)`
6.      `rule.setValue (AttributeOCCI, value)`
7. `return AttributeOCCI`

```html
<h3>BigQuery</h3>
<p>
Google BigQuery is a data warehouse [...] queries
</p>
<table>
<tr>
  <!-- name -->
  <td>location</td>
  <!-- type -->
  <td>string</td>
  <!-- descr -->
  <td>The geographic [...] The default value is US.</td>
</tr>
<tr>
  <!-- name -->
  <td>lastModifiedTime</td>
  <!-- type -->
  <td>long</td>
  <!-- descr -->
  <td>[Output-only] The date [...] the epoch.</td>
</tr>
</table>
```
Perspectives

- More generic crawler to support any API
  - Do not conform to OCCI anymore, other than HTML pages

- Implement GCP studio, a model-based framework that relies on this approach to design and deploy GCP applications

- Automated approach that would automatically handle the evolution of GCP
  - Incrementally detect streaming modifications, by calculating and modifying only the differences between the initially processed version and the newly modified one
Research internship for M2 students

Research needs you!
Thank you!


https://github.com/occiware/GCP-Model

stephanie.challita@inria.fr

https://stephaniechallita.github.io/