05: Service-oriented and Resource-based Architectures

Distributed Software Architectures

Stefan Huber <shuber.lba@fh-salzburg.ac.at>

May 9, 2019

Section 1

Service-oriented Architectures

Service-oriented Architectures

What is a service?

- A key concept to object-base architectures is encapsulation.
- A service is a self-contained entity: it can live by itself.
- A service can make use of other services, but in essence operates independently. The other services may actually be run by different organizations.

A SOA (Service-Oriented Architectures) separates an application into services.

Example: Web shop selling goods

- Application logic: Selecting ordered item, registering, checking delivery channels (e.g., e-mail), checking payment.
- Payment could be a separate service, maybe run by a different organization.
- Delivery channel handling could be a separate service.

Example: Traveling website

- Multiple services of various airlines, hotels, car rental are used.

Misconceptions and fuzz

SOAs are like snowflakes – no two are alike.

– David Linthicum

SOA has become a well-known and somewhat divisive acronym. If one asks two people to define SOA one is likely to receive two very different, possibly conflicting, answers. – msdn.microsoft.com

Wrong myths about SOA:[1]

- SOA would require web services.
- SOA would be new and revolutionary.

[SOA is defined as] a loosely-coupled architecture designed to meet the business needs of the organization. – msdn.microsoft.com

[1] https://web.archive.org/web/20160206132542/https://msdn.microsoft.com/en-us/library/bb833022.aspx#_Introduction_to_SOA

The Open Group SOA Working Group

The Open Group SOA Working Group provides the following definitions.[2]

Definition

Service-Oriented Architecture (SOA) is an architectural style that supports service-orientation.

Definition

Service-orientation is a way of thinking in terms of services and service-based development and the outcomes of services.

Definition

A service

- is a logical representation of a repeatable *business activity* that has a specified outcome (e.g., check customer credit, provide weather data, consolidate drilling reports),
- is self-contained,
- *may be* composed of other services, and
- is a black box to consumers of the service.

[2] https://web.archive.org/web/20160819141303/http://opengroup.org/soa/source-book/soa/soa.htm

Interfaces

Each service defines its own interface.

- Traveling example: Each airline may define its own, different, interface.
- This makes service composition often non-trivial.
- A typical use-case of the Adapter[3] design pattern.

[3] https://en.wikipedia.org/wiki/Adapter_pattern

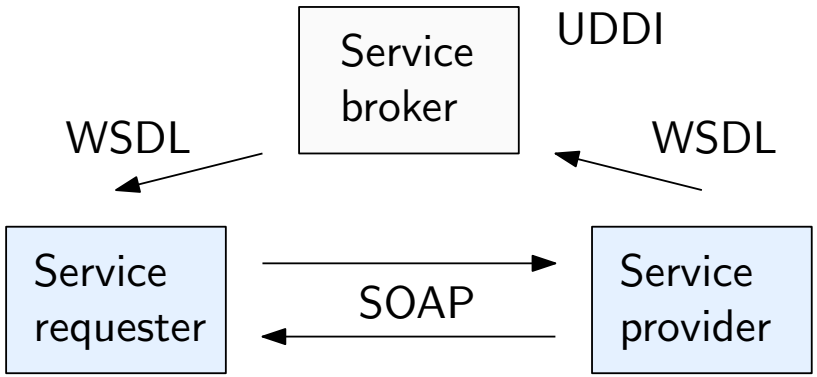
W3C Web Services

The W3C defines a web service as follows:[4]

Definition

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network.

- It has an interface described in a machine-processable format (specifically WSDL).
- Other systems interact with the Web service [...] using SOAP-messages [...].



UDDI (Universal Description, Discovery and Integration):

- A world-wide web service registry in the internet.
- In contrast, WS-Discovery is a multicast discovery protocol for a local network.

[4] https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice

Zeep

The python package `zeep` provides a simple SOAP client to access web services:

```
pip3 install zeep
python3 -m zeep "http://www.dneonline.com/calculator.asmx?WSDL"
```

Example call to invoke the web service method `Add`:

```
import zeep

if __name__ == '__main__':
    client = zeep.Client('http://www.dneonline.com/calculator.asmx?WSDL')
    result = client.service.Add(10, 13)
    print(result)
```

A more complex example of a SOAP API is provided by Amazon.[5]

[5] https://docs.aws.amazon.com/AWSECommerceService/latest/DG/WSDLLocation.html, http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl

Section 2

Resource-based Architectures

Resource-based Architectures

Issues with services:

- Service composition easily turned into a integration nightmare.
- E.g., SOAP client and server are quite tightly coupled by a contract (WSDL).

REST provides a different approach:

- Model: A distributed system interpreted as a collection resources.
- Resources can be added, removed, retrieved, modified.
- "Browsing" resources rather than invoking RPCs.
- Stateless communication, self-contained messages, uniform interface.
- A client can start using a REST API with zero knowledge about the API, in contrast to SOAP.

REST is an architectural style, not a protocol.[6]

[6] It may not make sense, but one could theoretically implement a REST architecture over SOAP.

RESTful architecture

REST means REpresentational State Transfer

- Resources have states. A state has a representation (e.g., XML or JSON).
- REST is about the transfer of representations of states (of resources).

Basic principles:

- Client-server communication
- Stateless
  The messages sent to and from the service are fully self-described. There is no session state; the server forgets about the client after command execution.
- Cacheable
  There might be a layer between client and server that caches responses. Retrieving a state twice gives the same result (idempotence).
- Uniform interface
  There is a single uniform interface to all components. There is a single naming scheme (URIs) for resources. HATEOAS[7]: Only the initial URI but no further knowledge is necessary for access; further information is dynamically given through hypermedia.

[7] Hypermedia As The Engine Of Application State. `http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven`

REST via HTTP in practice

REST via HTTP in practice:

- Resource identifiers are URIs like `http://www.example.com/book/341234`
- The media type is the format of the representation, e.g., XML or JSON.
- Uniform interface using HTTP operations:
  - PUT: Create a new resource (or overwrite the exiting one)
  - DELETE: Delete a resource (if it exists)
  - GET: Retrieve the state of a resource
  - POST: Modify a resource[8]
- The HTTP operations GET, PUT and DELETE are by definition idempotent.
- The HTTP operation GET is cacheable.

```
curl "https://api.github.com"
{
  "current_user_url": "https://api.github.com/user",
[...]

curl "https://api.github.com/users/torvalds"
{
  "login": "torvalds",
  "id": 1024025,
[...]
```

[8] Often, POST is applied to a resource that is a collection of elements in order to add a new element.

RESTful webservice in Python

Flask is a web development framework.

- `pip3 install Flask`
- It is based on Werkzeug and Jinja2.
- It provides a routing mechanism to handle URLs of some pattern.[9]

```
1  import flask
2
3  app = flask.Flask(__name__)
4
5  @app.route("/")
6  def helloworld():
7      return "Hello world"
8
9  @app.route("/users/<int:uid>")
10 def users(uid):
11     return "User with uid {}.".format(uid)
```

For RESTful services, however, there is an extension to Flask:

- `pip3 install Flask-RESTful`
- It adds features that remove manual work for implementing RESTful APIs.

[9] The function `app.route()` is a decorator: It wraps around functions.

RESTful webservices in Python

```python
import flask
from flask_restful import Resource, Api

class EntryPoint(Resource):
    def get(self):
        # HATEOAS: Add URL to tell where to go
        return {"message": "Hello world!",
                "items_url": api.url_for(ItemList, _external=True)}

class ItemList(Resource):
    def get(self):
        return {"message": "get all"}
```

- Resources modeled by classes that override `get()`, `put()`, `delete()`, `post()`.
- The returned data structure is automatically converted to JSON.

```python
app = flask.Flask(__name__)
api = Api(app)
api.add_resource(EntryPoint, "/")
api.add_resource(ItemList, "/items")
api.add_resource(Item, "/items/<int:itemid>")
```

- Routing patterns are used when adding the resources to the API.