RO-Verifier:

A Remote-Operations Invocations Framework

Tools And Strategies For Generalized OpenAPI/Swagger Based Verification Of Web-Services

Mohsen BANAN

Email: http://mohsen.1.banan.byname.net

PLPC-180057

November 1, 2019
Version 0.5

Available on-line at:
http://www.by-star.net/PLPC/180057

Contents

List of Figures

Part I

RO-Verifier: Generalized Web-Services Validation Tools

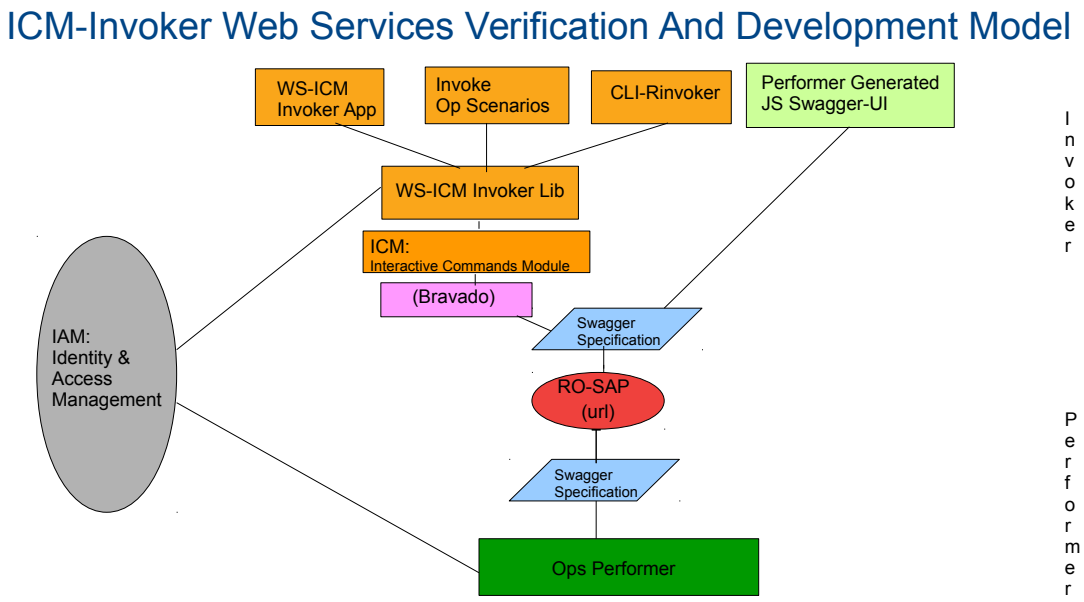

Figure 1: Swagger Based ICM Web Services Invoker Model

1 Overview Of The Web-Services Validation Tools

The Web-Services Validation Tools assume that a correct Swagger Specification is available. These tools do not focus on validating the service specification – there are a variety of tools available for validation of the swagger specifications, that is a different topic. These tools focus on validating the service based on its assumed correct specification.

These tools are invocation tools. They operate purely as the client side and other than verification and use of the service their scope does not extent to the server side (performers).

These tools are purely python based and they are purely Open Source (FOSS, Libre-Halaal). The python packages are available at PyPi and the complete source code is available at github.

You can think of these tools as a layer on top of
Bravado – `https://bravado.readthedocs.io/en/stable/`, `https://github.com/Yelp/bravado`.

Bravado ingests the swagger service specification in json or yaml and maps it to python on the fly. Bravado is a complete replacement to swagger codegen – the traditional code generation phase is eliminated. The Web-Services Validation Tools then create a higher level of convenience for invoking the operations specified in the service specification.

The Web-Services Validation Tools fall into two broad categories of:

1. Command Line Service Invocation Tools – described in Section 1.1.

2. Python Scenarios Service Invocation Tools – described in Section 1.2.

1.1 Command-Line Remote Invocations – rinvoker.py

Based on a given a swagger specification, rinvoker.py maps the json/yaml specification to python (using Bravado) and then python functions corresponding to remote-operation invocations are exposed as command-line using the ICM package (Interactive Command Modules).

All of this happens on the fly. Given a service-specification and a base service url, all operations become available for invocation at command line.

Pointers to examples and additional details are provided below.

1.2 Python Operations Scenarios – opScnSvc.py

In many situations the command-line interface may not be adequate for operations invocations as the parameters syntax may be complex, and as the results syntax may complex and as operation invocations sequences may be chained or interdependent. For such situation, a high level python interface called Operation-Scenarios (opScn) is provided.

You can then customize remote operation invocations as concrete scenarios (opScn) specifications.

Pointers to examples and additional details are provided below.

2 Installation Of The Web-Services Validation Tools

If you don't have Python 2.7 already installed, execute the following steps.

- Install Python 2.7
- Install pip and upgrade to the latest
- Optionally, if you are familiar with virtualenv, create a virtualenv.

With Python and pip in place, you can now install the unisos.mmwsIcm package.

- pip install unisos.mmwsIcm

All needed dependencies will be installed by just doing that.

2.1 Obtaining (OAuth) Authorization Tokens And Access Credentials

In the OAuth model, to invoke operations, the invoker needs to present tokens that the performer expects.

"Web Services Validation Tools", includes facilities that provide for obtaining oauth2 tokens.

Typically, the swagger specification includes a section such as:

```
"securityDefinitions": {
  "Oauth": {
    "type": "oauth2",
    "description": "Get OAuth access token \n",
    "tokenUrl": "https://xxx/v1/oauth2/accesstoken",
    "flow": "application",
    "scopes": {
      "resource-access": "Get Resource Access"
```

```
      }
    }
  }
```

By presentating its credentials to the token service at the “tokenUrl” RO-SAP, the invoker receives a token that it can then use in future invocations.

Obtaining of an oauth token for a given “serviceName” requires the following:

- Token Service Invocations Syntax (details of how the obtainToken operation is invoked)
- tokenUrl (RO-SAP) – typically included in the ”securityDefinitions” section of the swagger file
- accessKey – typically a user-name – Communicated out of band
- secretKey – typically a password -name – Communicated out of band

Use of the oauth2 tokens involves the following steps:

2.1.1 Install The Needed Modules (symCrypt, cryptKeyring)

```
pip install unisos.symCrypt
pip install unisos.cryptKeyring
```

The utility that facilitates invoker’s credentials storage and that facilitates obtaining of tokens is: getTokenWithCryptKeyring-svc.py.

2.1.2 Prepare The KeyRing

The invoker uses the keyring to store accessKey and secretKey for serviceName.

We encrypt the password (secretKey) in the keyring. For this, the interfaces to the keyring needs to be initialized (prepared). This is a one time activity. This preparartion involves:

```
getTokenWithCryptKeyring-svc.py --rsrc="keyring" -i prepare
```

2.1.3 Store The Invoker Credentials In The Keyring

Storing the invoker credentials in the keyring involves:

```
getTokenWithCryptKeyring-svc.py --rsrc="keyring/serviceName/accessKey"
                            --passwdPolicy="prompt" -i cryptPasswdSet
```

2.1.4 Obtain The Token

The following parameters:

- Token Service Invocations Syntax (details of how the obtainToken operation is invoked)
- tokenUrl (RO-SAP) – typically included in the ”securityDefinitions” section of the swagger file

need to be customized for getTokenWithCryptKeyring-svc.py.

Obatining the token for the serviceName and accessKey then involves:

```
getTokenWithCryptKeyring-svc.py --rsrc="keyring/serviceName/accessKey"
                            -i getTokenForKeyringUser > current.token
```

The token is then stored in a file called current.token.

2.2 Run The Web-Services Validation Tools Against The Canonical Petstore

Upon installation of the python packages, the relevant executable python scripts are placed in the bin directory of where Python was installed. On Windows locate the default bin directory by running: "where rinvoker.py". On Linux locate the default bin directory by running: "which -a rinvoker.py".

Now, verify that you can run the example commands against the petstore.

In https://pypi.org/project/unisos.mmwsIcm/, go through the instruction in the sections titled:
"Binaries And Command-Line Examples"
"Remote Invoker (rinvoker-svc.py) Examples"
"Operation Scenario (opScn-svc.py) Examples"

Just run the mentioned commands and verify that you are seeing the mentioned expected outputs.

If these all work right, then you know that the Web-Services Validation Framework has been properly installed and is operational.

2.3 Source Code And Packages Repositories

Complete sources are at:
https://github.com/bisos-pip/mmwsIcm

The PYPI page is at:
https://pypi.org/project/unisos.mmwsIcm

2.4 Applying This Services Validation Framework To Your Own Swagger Specifications

When you have a formal swagger service specification in place, use of these service validation tools can be very convenient and productive.

Simply follow the instructions that were provided for the canonical petstore and replace petstore's service specification with your own.

Follow the documentation to build your own scenarios for service specification and use the provided framework to combine multiple scenarios to form regression tests.

3 Use Of Command Line Remote Invocation (rinvoker) Validation Tools

You can invoke a swagger specification's operations directlly from the command line using the rinvoker.py command.

Typically for each swagger specification you create a customized version based on rinvoker as a seed.

In the following sections we describe common features and parameters and arguments of rinvoker and its derivatives and provide rinvokerPetstore.py as an example.

3.1 rinvoker Seed Features – Commands – Parameters – Arguments

rinvoker is an ICM (Interactive Command Module) and its command syntax is based on the ICM model.

The specific commands, parameters and arguments that are implemented on rinvoker are enumerated below.

3.1.1 rinvoker.py Seed Features – Commands

rinvoker.py commands are enumerated below.

- Cmnd: -i svcOpsList

```
svcOpsList command digests the Service Specification (swagger-file)
specified on command line as --svcSpec= parameter and produces a
complete list of ALL remotely invokable commands with their corresponding
--resource, --opName and url or body arguments.

Applicable options, parameters and arguments are:

  * Parameter (Mandatory) : --svcSpec=
  * Parameter (Optional)  : --perfSap=  --headers=
```

- Cmnd: -i rinvoker

```
rinvoker command invokes the "opName" operation at "resource" with
specified arguments.

Applicable options, parameters and arguments are:

  * Parameter (Mandatory) : --svcSpec=  --resource=  --opName=
  * Parameter (Optional)  : --perfSap=  --headers=

  * Arguments             : name=value  bodyStr=jsonStr
```

3.1.2 rinvoker.py Seed Features – Parameters

rinvoker.py parameters are enumerated below.

- Parameter: –svcSpec= (url, or swagger-file)
  The swagger file as a url or as a json/yaml file is specified with the –svcSpec= parameter.
- Parameter: –perfSap= (url)
  The Performer Service Access Point Address (perfSap) is specified as a URL with the –perfSap= parameter.
- Parameter: –header= (file)
  Additional headers (e.g., a token) can be included with the –svcSpec= parameter.
- Parameter: –resource= (string, corresponding to SvcSpec)
  The resource to be invoked should be specified with the –resource= parameter
- Parameter: –opName= (string, corresponding to SvcSpec)
  The operation name to be invoked should be specified with the –opName= parameter

3.1.3 rinvoker.py Seed Features – Arguments

rinvoker.py arguments are enumerated below.

- Argument: name=value (string=string corresponding to SvcSpec's URL Params)
- Argument: bodyStr=jsonStr (bodyStr=string corresponding to SvcSpec's Body)

3.2 rinvokerPetstore.py Example

rinvoker allows you to list all possible invocations based on a service specification (swagger file). For example:

```
rinvoker.py --svcSpec="http://petstore.swagger.io/v2/swagger.json" -i svcOpsList
```

rinvoker allows you to fully specify an invocation on command line. For example:

```
rinvoker.py --svcSpec="http://petstore.swagger.io/v2/swagger.json"
     --resource="user" --opName="createUser" -i rinvoke
     bodyStr="{...}"
```

4 Scenarios Invocation Validation Tools

You can specify one or more invocations as a ”scenario”.

Scenarios are python scripts that specify operations and their arguments and expectations of results.

Scenarios allow for the results of operations to be used as arguments of future operations.

4.1 Model Of Invoke – Specification, Verification And Reporting – Scenarios

Invoke Scenarios Are pure python specification of sequence of invocations.

Invoke-Expect Scenarios Are pure python specification of sequence of invocations subject to preparations and post-invoke verification and reporting.

OpInvoke class allows for complete invoke specification and complete results to be fully captured.

4.1.1 Scenario Specification For Sequences Of Invocations

In pure python you can specify invocation of each operation, for example:

```
thisRo = ro.Ro_Op(
    svcSpec=petstoreSvcSpec,
    perfSap=petstoreSvcPerfSap,
    resource="pet",
    opName="getPetById",
    roParams=ro.Ro_Params(
        headerParams=None,
        urlParams={ "petId": 1},
        bodyParams=None,
        ),
    roResults=None,
    )
rosList.opAppend(thisRo)
```

Validation And Reporting Of Invokations

Building on the previously mentioned Operation Specification, in pure python you can the specify Operation Expectations, for example:

```
thisExpectation = ro.Ro_OpExpectation(
    roOp=thisRo,
    preInvokeCallables=[sleep1Sec],
```

```
    postInvokeCallables=[ verify_petstoreSvcCommonRo, ],
    expectedResults=None,
    )
roExpectationsList.opExpectationAppend(thisExpectation)
```

preInvokeCallables(ro.Ro_OpExpectation) can include a function that initializes the DB or sleepFor1Sec.

postInvokeCallables(ro.Ro_OpExpectation) can include a function that verifies the result was as expected and then reports success or failure.

4.2 opScn-Seed (Remote Operation Scenarios) – Commands – Paramters – Arguments

Commands drived from the opScn seed are ICMs (Interactive Command Module) and their command syntax are based on the ICM model.

4.2.1 opScn Seed Features – Commands

opScn-seed provides the following commands and parameters:

- Cmnd: -i roListInv

  ```
  roListInv command serially invokes the list of ro.Ro_Op() opersations
  specified in the loaded scenario files.

  roListInv displays the invocation and its results. But does not do any verifications.

  Applicable options, parameters and arguments are:

    * Parameter (Mandatory) : --load=
  ```

- Cmnd: -i roListExpectations

  ```
  roListExpectations command serially invokes the list of ro.Ro_OpExpectation()
  specified in the loaded scenario files.

  roListExpectations displays the invocation and its results and
  additionally runs the list of preInvokeCallables and
  postInvokeCallables.

  postInvokeCallables can include functions that verify the results of the invocation
  were as expected.

  Applicable options, parameters and arguments are:

    * Parameter (Mandatory) : --load=
  ```

4.2.2 OpScn Outputs And Reportings

The output format is:

```
 * ->:: Invoke Request
 * <-:: Invoke Response
 * ==:: Invoke Validation (SUCCESS or FAILURE)
```

Additional information for each is include with "**" tags.

This output format can then be used in outline or org-mode.

5 Complete Invoker-Applications Development

RO-Verifier framework can also be used to develop complete invoker-application in python.

5.1 Invoker-Apps Development Model

Invoker-Apps development model is an extension of opScenarios facilities.

5.1.1 Invoker-Apps Can Easily Build On unisos.mmwsIcm Capabilities

The following modules:

- Bravado does invoker code-generation on the fly.
- unisos.mmwsIcm.opInvoke – Abstracts invoke-specifications
- unisos.mmwsIcm.wsInvoker – Allows for invokation and verification of opInvoke

provide a consistent framework for Invoker-Apps development.

Part II

Uses Of RO-Verifier For Public Web Services

6 Public Uses Of RO-Verifier

The RO-Verifier has been used to validate and verify a number of Web Services based on their swagger files.

Generally speaking, uses of RO-Verifier fall into two categories:

Specific Service Verifications: Where we take a web service and its swagger file and custmize a set of OpScenarios for verificatin of that web service.

Generalized Service Verifications: Where we apply a set of exisiting common OpScenarios and apply them to a repository of web services.

6.1 Specific Web Service Verifcation Scenarios

6.1.1 PetStore Verifcation Scenarios

Verifcation scenarios for the PetStore web service are maintained at:
https://github.com/bxexamples/roVerifier-petstore

6.2 Generalized Web Service Verifcations

A set of scripts that pass all swagger files availble at:
https://github.com/APIs-guru/openapi-directory through the rinvoker.py are available at:
https://github.com/bxexamples

References