March 29, 2017

API Test framework using Meta-Descriptor

Murugesan Arumugam  
_Hewlett Packard Enterprise_

Hiro Rameshlal Lalwani  
_Hewlett Packard Enterprise_

Jayasankar Nallasamy  
_Hewlett Packard Enterprise_

Follow this and additional works at: [http://www.tdcommons.org/dpubs_series](http://www.tdcommons.org/dpubs_series)

Recommended Citation

Arumugam, Murugesan; Lalwani, Hiro Rameshlal; and Nallasamy, Jayasankar, "API Test framework using Meta-Descriptor", Technical Disclosure Commons, (March 29, 2017)  

This work is licensed under a Creative Commons Attribution 4.0 License.
This Article is brought to you for free and open access by Technical Disclosure Commons. It has been accepted for inclusion in Defensive Publications Series by an authorized administrator of Technical Disclosure Commons.
API Test framework using Meta-Descriptor

Abstract - Current trend demands the IT infrastructure to move towards cloud centric service adapting open standards like open source software, open data-center hardware, open networking standards etc. There has been a steady increase in Open API software but it lacks a common validation frameworks to meet the growing demand. So there arises the need for a generic API validation framework that can apparently help developers to validate public as well as private APIs. Testing a software sub-system, which houses a large number of such APIs each of which accepts a number of parameters always ends up as a highly monotonous work. When the parameters can intake any value from the vast range available, the permutations and combinations of the tests will be huge. When the APIs are inter-dependent, it further muscles up the complexity by offering more than one way to verify and validate an outcome. Our approach can be used to test the APIs on distributed systems as well, apart from the conventional standalone API testing.

This disclosure is related to Test methodologies and automation frameworks.

Application Programming Interface (API) enables application developers and service providers seemingly access data regardless of underlying software. Conventional API Test frameworks currently in the market were built to validate combinatorial as well as input parameter variation. While testing the parameter permutations and combinations is greatly addressed, they lack in identifying the inter-dependency between the APIs and preserve the state of the system to derive working logic among the APIs. Essentially the intention of designing the APIs is to form a business logic using various sets of APIs and enable a service in the system. Conversely, traditional API testing fails to discover dependent APIs and frame inter-dependency matrix since the sequence in which the APIs are called would influence the outcome. When the APIs are inter-dependent, it further muscles up the complexity by offering more than one way to verify and validate an outcome. Additionally, other environment variables within the system under test too play a major role, as the behavior of the APIs may change accordingly. All these complexities explained until now are in perspective of a standalone system. It exponentially shoots-up when the system environment becomes distributed in nature. Therefore, we felt the need of a generic API validation Methodology that can apparently help developers to validate public as well as private APIs.

In our solution, we propose an API Virtualization Provider (AVP) that dynamically generates the test data based on the inputs provided by three components i.e. i) an API Meta Descriptor (AMD) ii) Rules Engine and iii) a Combinatorial Test Classifier (CTC).
API Meta Descriptor defines “How an API behaves”? It acts as a template in which the functional attributes of each and every API to be tested are defined in all the aspects, such as the parameters accepted, range of values those parameters accept and the restrictions, possible user-space programs or utilities that can invoke the API, command line to be executed for invoking the API, type of return action/value, dependent APIs in sequential order (identified by a sequence ID), etc. Every API will be assigned a unique ID. Once all the APIs are defined, they are ‘zoned’ based on the objects they act upon. Zoning is a process where the dependent APIs are identified across the zones, grouped and assigned a priority based on the sequence ID which determines the order of API calls depending on the test strategies. In this way, a Relationship matrix is formed which helps us in solving the inter-dependency problem between the APIs. For instance, the sequence can be followed as such within a zone for positive testing and altered for negative testing. Zoning also allows us to pick any random way to verify and validate the outcome of an API, as there is always more than one way to do it when it comes to inter-dependent APIs. Provisions for defining the entire structure of any API is generalized in the API Meta Descriptor so that it grows dynamically based on the attributes of the API. Provisions for data related to distributed systems are provided to deal with that perspective of our problem statement. When no explicit verification requirements are mentioned, the system takes a random way to verify and validate on any of the system within the network.

Rules Engine supplies the necessary rules that are to be applied along with the behavior of the APIs in constructing the test dataset. These rules are mainly to dictate the limitations and restrictions certain objects hold. API Virtualization Provider follows these rules while populating the tests. For instance with reference to Fig 1, an object name can hold any alphabet value of size up to 255 bytes, with ‘@’ character being the restriction. Set of tests in which the parameter ‘object_name’ consisting of random string with “All characters except ‘@’” and the parameter ‘object_size’ with value ranging “1<=255bytes” are populated for positive tests and “Characters with ‘@’” for object_name and object_size being “>255bytes” are populated for negative tests in any permutations and combinations. And with the same rules cited in Fig 1, Create APIs (zone1) will use -1, 0, 1, 127, 128 and 129 as values for “number of objects” parameter while populating tests for boundary value testing.

Combinatorial Test Classifier offers modularity and coverage in constructing the test dataset. It grants flexibility in combining conventional test design strategies with a mix of user-preferred techniques. For instance you can choose positive/negative tests or apply ECP/BVA techniques (classes will be defined in the Rules Engine), or the combination of both. Since we deal with APIs, test design from API perspective should be covered as well. For instance, a set of APIs can be chosen for thorough testing with all the permutations and combinations irrespective of any precedence. All these logic will be built in the API Virtualization Provider.

Test dataset constructed by API Virtualization Provider acts as the test scripts, which contains test-cases in the form of one-liners with each test being given a unique ID for traceability purpose. Each test is then parsed by a parser for command line building, which is aided by pre-defined
connector scripts. Once the command-line is built, the API is called for execution (using any user-space program). On execution, the control will be transferred to the APIs and returned back to the caller (program) once it’s over. After execution, return value/action has to be validated, since we cannot wholly rely on the return code to arrive on the success/failure of the purported API call. This sort of action/data verification and validation can be done from any node-to-node basis in a distributed environment (system related complexities), which was already handled by the API Virtualization Provider in this case. For instance, when a system call to write a file F1 on node N1 is made, it can be verified by accessing the same file from another node N2 in a distributed environment.

Advantages

- Handling of distributed systems complexity and inter-dependent APIs has been addressed while no other existing solutions does.
- Helps the end user to easily add the API definition using Meta-descriptor format.
- Selective regression testing becomes very handy, where only few APIs can be thoroughly tested based on the requirements/code-changes.
- Test data-set population is entirely taken care by the framework which allows the user to focus on fine tuning the meta-descriptor attributes for better test efficiency.

Fig 1:

Disclosed by Murugesan Arumugam, Hiro Rameshlal Lalwani and Jayasankar Nallasamy, Hewlett Packard Enterprise