ETSI GR CIM 011 V1.1.1 (2021-04)

GROUP REPORT

Context Information Management (CIM); NGSI-LD Testing Framework: Test Purposes Description Language (TPDL)

Reference

DGR/CIM-0011v111

Keywords

API, IoT, testing

Contents

Foreword

This Group Report (GR) has been produced by ETSI Industry Specification Group (ISG) cross-cutting Context Information Management (CIM).

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Executive summary

The present document presents an analysis of candidate languages to be used for development of test suites by ETSI ISG CIM team of experts who developed the present document. A list of criteria, covering the whole chain from requirements extraction, test purpose drafting, abstract test suites and execution production has been developed. The language candidates, identified from expert's knowledge, desk research and interviews with ETSI CTI members have been analysed. They are shortly described in the present document and compared in respect with the criteria list. From that analysis, two languages/framework emerged: TTCN-3 and Robot framework. These candidates have been further analysed to evaluate their aptitude to cover the whole chain of tests developments and execution. Despite having TTCN-3 as an ETSI promoted language, the Robot Framework appears for several reasons to be a better suited candidate.

Introduction

The ISG CIM group has defined an API for exchange of information contextualized in time, space and relation to other information using a property graph model with the intent that the associated protocol (called NGSI-LD) becomes the "glue" between all kinds of applications and databases associated with services for Smart Cities, Smart Agriculture, Smart Manufacturing, etc.

To be successful, the NGSI-LD API specification needs to be well understood and well implemented. The community of users will not be solely highly professional engineers employed by big companies but will include many small teams and SMEs and even hobbyists. Therefore, it is essential that the developers have access to not only the standard but also a test specification and a testing environment to check that their work is (and remains) conformant to the ETSI NGSI-LD specification.

The developers will usually write integration tests to validate the behaviour of their NGSI-LD implementation, but it is important to assert compliance to the specification based on a test suite agreed by the group creating the API specification, i.e. ETSI ISG CIM. Therefore, it is very important to create a set of ETSI-approved test cases.

What is more, the existence of such a test suite will likely help to increase the adoption of the NGSI-LD specification by giving developers a ready to use and complete set of sample requests.

The present document (report/study) aims at analysing the options which exist for the development of such a test environment to be adopted by the developer's community. It develops a list of criteria against which, test purpose description languages are analysed so to provide a recommendation for the forthcoming activities of the ETSI ISG CIM team of experts who developed the present document.

1 Scope

The present document is a choice of Test Purposes Description Language (TPDL), with the intention to capture all of the information required by the Test Template and should be parseable using software.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document, but they assist the user with regard to a particular subject area.

[i.1] ETSI GS CIM 009 (V1.3.1): "Context Information Management (CIM); NGSI-LD API".

NOTE: Available at https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.03.01_60/gs_CIM009v010301p.pdf.

[i.2] Asha Channenahalli Ramachandraiah, API Testing using Robot Framework, March 13 2019.

NOTE: Available at https://devonblog.com/test-automation/api-testing-using-robot-framework/.

[i.3] Adam Hepner, Parameterizing Your Robot Framework Tests, March 03 2020.

NOTE: Available at https://wonderproxy.com/blog/parameterizing-robot-framework/.

[i.4] OpenAPI Specification: fka Swagger RESTful API Documentation Specification,Version 2.0.

NOTE: Available at https://swagger.io/specification/v2/.

[i.5] ITU-T (Z.160 multipart series): "Series Z: Languages And General Software Aspects For Telecommunication Systems".

3 Definition of terms, symbols and abbreviations

3.1 Terms

Void.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

API	Application Programming Interface
BDD	Behavior Driven Development
CD	Coding and Decoding
CH	Component Handling
CI	Continuous Integration
CL	Community and License
DO	Definition and Organisation
DSL	Domain Specific Language
DX	Developer eXperience
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IP	Internet Protocol
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
MIT	Massachusetts Institute of Technology
MQTT	Message Queuing Telemetry Transport
OAS	OpenAPI™ Specification
OS	Operating System
PA	Platform Adaptor
PRO	PROfessional
SA	System Adaptor
SOAP	Simple Object Access Protocol
SUT	System Under Test
TCI	TTCN-3 Control Interface
TDL	Test Description Language
TEE	Test Execution Environment
TL	Test Logging
TM	Test Management
TOP	TDL Open source Project
TP	Test Purpose
TPDL	Test Purpose Description Language
TRI	TTCN-3 Runtime Interface
TTCN	Testing and Control Notation
UI	User Interface
URL	Uniform Resource Locator
VS	Visual Studio

4 Criteria for Test Purpose Description Language (TPDL) selection

4.1 Purpose

The purpose of this clause is to define the list of criteria to be used during the evaluation of the Test Purposes Description Languages.

For clarity and ease of processing, it has been divided into categories grouping related criteria:

- Definition and Organization of tests (DO): how tests can be written, grouped and organized in a flexible way?
- Developer eXperience (DX): what support is provided to developers working with the TPDL?
- NGSI-LD specific points of attention (NGSI-LD): how it can handle some specific implementation points of the NGSI-LD API?

- Test Execution Environment (TEE): what are the characteristics of the Test Execution Environment provided by the TPDL?
- Community and License (CL): under which conditions is the TPDL available and how well it is supported?

4.2 Criteria list

Definition and organization of tests

1) Is the programming language similar enough to "normal English" that it can be partially or completely used to define the Abstract Tests in NGSI-LD Test Purposes Descriptions specifications?
2) How can the final list of test cases written in the TPDL be extended (under conditions compatible with ETSI IPR rules) by contributions from a developer community?
3) How "easy" it is to write HTTP API tests (is there a DSL? are there existing functions to make assertions on the result?)
4) Is it possible to group/organize tests in a suite (shall/may, subscription/entity, etc.)?
5) Is it possible to reuse fixture (parameterized) data between tests?
6) Is it possible to reuse functions/components between tests?

Developer experience

1) What is the IDE support to develop tests with the TPDL (syntax highlighting, code completion, run tests directly from the IDE, etc.)?
2) Can the programming language be executed in a software environment which is familiar to most web developers, e.g. Eclipse™, or does it run as a stand-alone?

NGSI-LD specific points of attention

1) Is it able to check that after an entity creation/update and a matching subscription the notifications are correctly delivered to the IP/port specified in the subscription?
2) Is it able to check that notifications are received over MQTT (ETSI GS CIM 009 [i.1])?
3) How can it be integrated with authentication (and authorization) concerns (even though it is out of scope of the present document (report/study))?

Test Execution Environment

1) Is the Test Execution Environment offering tools familiar to the developer community such as display of results in a browser window and writing full results in log files, etc.
2) Is it possible to run the NGSI-LD software and the Test Execution Environment on the same host?
3) Can the tooling environment be integrated with continuous integration tools (e.g. Jenkins™)?
4) Is it possible, in the execution environment, to run individual test cases as well as specific sets or the entire set of test cases?
5) Is the execution environment able to generate error messages in case the test case cannot be executed due to any cause, including malformed definition of the test case?
6) Can it supply detailed step-by-step logs and overall summary of pass/fail/skip?
7) After a complete test suite generates errors/fails, can individual failed tests be re-tried?
8) Is it possible to have the full details of requests and responses (payloads, headers)?
9) Is it possible to define if the suite continues or aborts (or forks?) after a failed test?

Community and license

1) What is the strength of the community behind the TPDL (number of committers, frequency of releases, roadmap, etc.)?

2) Is it available as Open Source or free runtime license? If so, under which license?

5 Candidates analysis

5.1 Analysis

5.1.0 Foreword

Here is the complete list of tools considered for the TPDL. Some already appear to not be the right tool for the needs of the NGSI-LD Test Suite but they have been considered for completeness of the study.

Candidate	Homepage	Rationale
TDL & TTCN-3	http://tdl.etsi.org http://www.ttcn-3.org/	Test purpose description language and abstract test language developed and used within ETSI
Robot Framework	https://robotframework.org/	Widely used tool for conformance and interoperability testing
Karate	https://intuit.github.io/karate/	Offers natively a DSL that allows testing RESTful APIs
OpenAPI™ based tooling	https://www.openapis.org/	The NGSI-LD exposes an OpenAPI™ representation that could be a base for testing
SoapUI	https://www.soapui.org/	Commonly used tool to interact with SOAP Web services, it has since evolved to interact with RESTful APIs
Dredd	https://dredd.org/en/latest/	Recent project that aims at validating an API against an OpenAPI™ description
Postman	https://www.postman.com	Widely used tool to interact with RESTful APIs
Selenium/Cucumber	https://www.selenium.dev/	Commonly used tool to test web-based application, it has since evolved to allow testing RESTful APIs
Javascript™ based tooling (Mocha, Chai, etc.)		Environment already used by some Open Source implementations of NGSI-LD

5.1.1 TTCN-3

1) Is the programming language similar enough to "normal English" that it can be partially or completely used to define the Abstract Tests in NGSI-LD Test Purposes Descriptions specifications?

TTCN-3 addresses not only the language for specifying tests but also the interfaces that control and adapt a test to any given environment. It provides a syntax for the definition of tests independent of any application domain. The abstract nature of a TTCN-3 test system makes it possible to adapt a test system to any test environment. This separation significantly reduces the effort required for maintenance allows experts to concentrate on what to test and not on how. The standardization of TTCN-3 means that users are not forced to rely on the use of one proprietary tool.

The language part is actually handled by TDL, which is close to normal English. TDL allows to generate skeletons for the Abstract Test Suite. Those skeletons have then to be manually enhanced and protocol bindings have to developed.

2) How can the final list of test cases written in the TPDL be extended (under conditions compatible with ETSI IPR rules) by contributions from a developer community?

TTCN-3 is a standardized testing technology developed and maintained by ETSI and specifically designed for testing and certification. The ETSI TTCN-3 standards have also been adopted by the International Telecommunication Union (ITU-T) in the Z.160 series [i.5]. The test cases can be maintained in a Git repository, allowing contributions from a developer community.

3) How "easy" it is to write HTTP API tests (is there a DSL? are there existing functions to make assertions on the result?)

The standardized interfaces make it possible for a suite of TTCN-3 tests to be adapted to execute in a range of different environments, platforms or operating systems including HTTP API tests. Writing HTTP API tests requires the development of a System Adaptor (SA) for HTTP protocol which may be complex along with the corresponding Coding and Decoding (CD) for adapting message-based communication to and from the SUT, and SUT action operations and encoding and decoding data associated with message-based communication within the TEE according to the following TTCN3 system architecture.

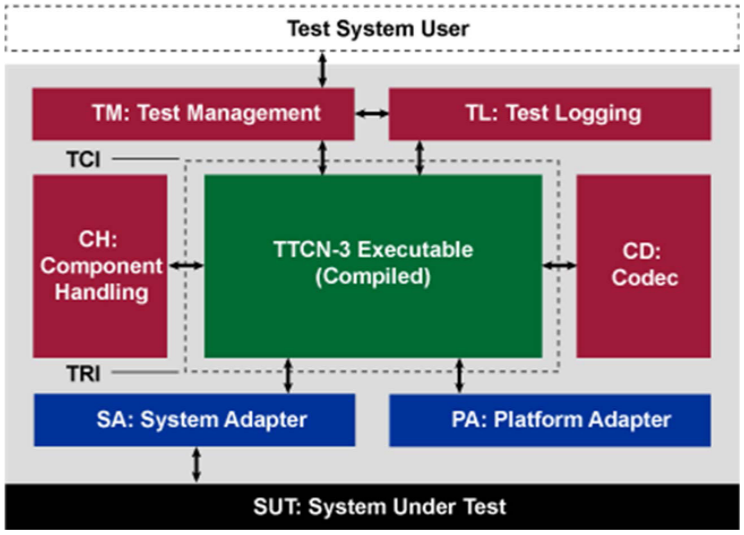


Figure 5.1.1-1: TTCN-3 elements

- TTCN-3 Control Interface (TCI):
  - Test Management (TM): Overall management of the test system, test system user interface, execution of tests, provider of parameters.
  - Test Logging (TL): Handling of all log events of the test system.
  - Coding and Decoding (CD): Encoding and decoding data associated with message based or procedure-based communication within the TEE.
  - Component Handling (CH): Distributing and communication between parallel test components.
- TTCN-3 Runtime Interface (TRI):
  - System Adaptor (SA): Adapts message based or procedure-based communication to and from the SUT, and SUT action operations.
  - Platform Adaptor (PA): Implements TTCN-3 external functions and provides a TTCN-3 test system with a single notion of time.

4) Is it possible to group/organize tests in a suite (shall/may, subscription/entity, etc.)?

The language is organized into:

- Module: The top-level container in a test suite is a module. It is usually a file.
- Component: A component is an execution entity. A test case or a function is executed on a component.
- Port: Components communicate with each other or with the SUT through ports that are mapped to each other.
- Test case: A test case is a sequence of sends and receives. When a message is sent to the SUT (System Under Test) several possible replies can be received.
- Alternative: Since a test case is a sequence of stimuli followed by a set of possible responses, the notation includes alternatives. It is a compact way to list all the possible alternatives in a scenario.
- Template: The template construct aims at defining the parameters values when sent or verifying the parameter values when received. The template allows complex verification in a single statement so that the test case stays legible.
- Verdict: The verdict is the result of a test case execution. It has 5 possible values: none, pass, inconclusive, fail, error.

Hierarchy of groups allows to define any needed structure.

5) Is it possible to reuse fixture data between tests?

Yes. It is possible to define global variables that will be accessible from all tests and keywords. The variables can also be defined on suite, test case or keyword level.

6) Is it possible to reuse functions/components between tests?

It is possible to reuse TTCN3 Modules, Components, Ports and Templates between tests.

7) Does it support data-driven approach?

Data-driven tests can be created in TTCN3 using its Rich type system and Data and signature templates with built-in matching mechanisms.

Developer experience

1) What is the IDE support to develop tests with the TPDL/Abstract Test Suite (syntax highlighting, code completion, run tests directly from the IDE, etc.)?

Requirements:

- A TTCN-3 test suite.
- A TTCN-3 tool, i.e. a TTCN-3 compiler (or interpreter) plus execution environment.
- Optionally implementations of test execution control, logging, and codecs.
- A SUT Adaptor implementing the means of communication required by SUT interfaces.
- A Platform Adaptor implementing a timing model and external functions (if there any defined in the test suite).

This support is typically provided by TTworkbench (a commercial product) and Titan (Open Source but with limited features and no support for all the usual OS, e.g. there are no binaries for macOS®).

There is also a specific IDE, packaged as an Eclipse plugin, for writing the TP based on TDL/TOP.

2) Can the programming language be executed in a software environment which is familiar to a majority of web developers, e.g. Eclipse™, or does it run as a stand-alone?

Yes. For instance, Eclipse IDE for Titan and TTworkbench, it depends on the chosen tool.

NGSI-LD specific points of attention

1) Is it able to check that after an entity creation/update and a matching subscription the notifications are correctly delivered to the IP/port specified in the subscription?

Yes. The corresponding test could be written in TTCN3.

2) Is it able to check that notifications are received over MQTT (ETSI GS CIM 009 [i.1])?

An MQTT adapter need to be implemented in order to receive notification over an MQTT broker.

3) How can it be integrated with authentication (and authorization) concerns (even though it is out of scope of the present document)?

Authentication and authorization tests can be written in TTCN3 and tested if needed.

4) Possibility/difficulty to add a new protocol binding?

TTCN3 has an abstraction over the protocol binding.

Test Execution Environment

1) Is the Test Execution Environment offering tools familiar to the developer community such as display of results in a browser window and writing full results in log files, etc.

TTCN3 is a test specification language and not a programming language, however existing implementation (open source or commercial) offer some tools to display the result in graphical or log formats. But the current state for each tool is not known and would require more investigation.

2) Is it possible to run the NGSI-LD software and the Test Execution Environment on the same host?

Yes. However, they do not support all the usual environments for developers (e.g. there are no Titan binaries for macOS®).

3) Can the tooling environment be integrated with continuous integration tools (e.g. Jenkins™)?

Most tools offer command line for test execution and could be easily integrated with continuous integration tools like Jenkins™.

4) Is it possible, in the execution environment, to run individual test cases as well as specific sets or the entire set of test cases?

It is possible to select all test suites, a specific test suite or a specific test case.

5) Is the execution environment able to generate error messages in case the test case cannot be executed due to any cause, including malformed definition of the test case?

Yes. Tests details and assertions result are displayed.

6) Can it supply detailed step-by-step logs and overall summary of pass/fail/skip?

Yes.

7) After a complete test suite generates errors/fails, can individual failed tests be re-tried?

They can be retried one by one.

8) Is it possible to have the full details of requests and responses (payloads, headers)?

Yes. A log file can be created for each request that contains the request full details.

9) Is it possible to define if the suite continues or aborts (or forks?) after a failed test?

Yes. During a test campaign, it is possible to define conditions to stop, continue or retry.

Community and license

1) What is the strength of the community behind the TPDL (number of committers, frequency of releases, roadmap, etc.)?

The unique open source implementation Eclipse Titan provided limited features, weak graphical interface, and not well documented. It is nowadays providing new release regularly every 3 months (roughly)

TTCN-3 is specified by ETSI and used in different technical committees inside ETSI (ITS, 3GPP, oneM2M, etc.).

2) Is it available as Open Source or free runtime license? If so, under which license?

TTCN-3 is fully available as Open Source.

For the runtime environment:

- Eclipse Titan (Open Source): A TTCN-3 compilation and execution environment with an Eclipse-based IDE. Titan consists of a core part, executing in a UNIX®/Linux®-like environment and a set of Eclipse plug-ins.

NOTE 1: UNIX® is a registered trademark of The Open Group.

NOTE 2: Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

- TT Workbench (Commercial): A full-featured integrated test development and execution environment (IDE) for any kind of test automation project. This powerful, user-friendly tool allows to test software products and services regardless of technology, operating system, or implementation domain.

- There are other tools but the team of experts who developed the present document had not enough resources to evaluate them: http://www.ttcn-3.org/index.php/tools.

5.1.2 Robot framework

Definition and organization of tests

1) Is the programming language similar enough to "normal English" that it can be partially or completely used to define the Abstract Tests in NGSI-LD Test Purposes Descriptions specifications?

Yes, Robot Framework allows testers to easily write automated test scripts using a Human-readable, keyword-driven syntax; also supporting Gherkin.

2) How can the final list of test cases written in the TPDL be extended (under conditions compatible with ETSI IPR rules) by contributions from a developer community?

The test cases can be maintained in a Git repository, allowing contributions from a developer community.

3) How "easy" it is to write HTTP API tests (is there a DSL? are there existing functions to make assertions on the result?

Robot provides some libraries that provide a DSL for testing HTTP APIs:

- RequestsLibrary: https://github.com/MarketSquare/robotframework-requests.

- RESTinstance: https://github.com/asyrjasalo/RESTinstance/.

- HTTPRequestLibrary: https://github.com/Hi-Fi/robotframework-httprequestlibrary.

- HTTPLibrary: https://github.com/peritus/robotframework-httplibrary (API documentation: http://peritus.github.io/robotframework-httplibrary/HttpLibrary.html): last commit is 7 years ago.

RequestLibrary is the most active of the four, with 33 contributors, and an activity peak for one year. It seems HttpLibrary.HTTP is often used in conjunction with RequestsLibrary.

RESTinstance integrates support for JSONPath, which allows for powerful matching of JSON payloads (e.g. String $[0].myProp.value "value of my prop").

HTTRequestLibrary has no real activity for about 2 years (only dependencies upgrade) and its associated DSL is far less fluent than the one provided by RequestLibrary (e.g."Should Be Equal As Strings ${resp.status_code} 200" to check a response code status vs "Status Should Be 200 ${resp}" with RequestLibrary).

4) Is it possible to group/organize tests in a suite?

Yes, Robot Framework test cases are created using test case tables in test case files. Such a file automatically creates a test suite from all the test cases it contains. Test case files can be organized into directories, and these directories create higher-level test suites. These directories can then be placed into other directories creating an even higher-level suite.

5) Is it possible to reuse fixture data between tests?

Yes. It is possible to define global variables that will be accessible from all tests and keywords. The variables can also be defined on suite, test case or keyword level. It can be also set using the setup and teardown (on test case or test suite level).

6) Is it possible to reuse functions/components between tests?

Yes. Keywords and other Python™ (language) implemented functions can be reused between tests.

7) Does it support data-driven approach?

Data-driven tests can be created in Robot Framework using the test template functionality. Test cases with template contain only the arguments for the template keyword. Instead of repeating the same keyword multiple times per test and/or with all tests in a file, it is possible to use it only per test or just once per file.

Developer experience

1) What is the IDE support for the TPDL?

It has its own editor (RIDE) but there are plugins/extension for most common editors as well (Visual Studio Code, Eclipse™, IntelliJ™, etc.). Recommended: Intellisense extension for Visual Studio Code:

- IntelliJ™: https://plugins.jetbrains.com/plugin/7415-robot-framework-support/versions.
- Eclipse™: https://nokia.github.io/RED/.
- VS Code: https://marketplace.visualstudio.com/items?itemName=TomiTurtiainen.rf-intellisense (no support for any of the above HTTP libraries).

2) Can the programming language be executed in a software environment which is familiar to a majority of web developers, e.g. Eclipse™, or does it run as a stand-alone?

Since it is built in Python™ language, it can be used and further extended using Python™ or Java™. It can also easily be launched on the command line or via some shell scripting.

NGSI-LD specific points of attention

1) Is it able to check that after an entity creation/update and a matching subscription the notifications are correctly delivered to the ip/port specified in the subscription?

It should be possible with the usage of available libraries (PythonRemoteServer, SSHLibrary, ...). Other Python™ functions can be implemented if needed.

2) Is it able to check that notifications are received over MQTT (ETSI GS CIM 009 [i.1])?

Robot has an MQTT library: https://github.com/randomsync/robotframework-mqttlibrary. It allows to subscribe to a topic and validate a message is received (at least). However, it needs an MQTT broker to be launched as part of the test suite.

3) (Keep in mind authentication and more generally security, etc. even if out of scope of the present document).

Robot allows to easily bind an authentication process as part of each or some requests, for instance to get an access token in the case of an OpenID Connect authentication.

Test Execution Environment

1) Is the Test Execution Environment offering tools familiar to the developer community such as display of results in a browser window and writing full results in log files, etc.

Robot Framework generates an HTML report. It has statistics based on tags and executed test suites, as well as a list of all executed test cases. It allows the possibility to navigate to more detailed information by adding links to the log files. The overall test execution status is easily seen by the report's background color: green if all critical tests pass, and bright red otherwise. For people that like dashboards, results can even be presented in a Grafana dashboard (https://cognitiveqe.com/robot-framework-test-results-in-grafana-postgresql/).

2) Is it possible to run the NGSI-LD software and the Test Execution Environment on the same host?

Yes. The Test Execution Environment requirements are simple (Python™) and it can also be executed in Docker.

3) Is the Test Execution Environment available as open source or free runtime license?

Yes, Robot Framework is open source.

4) Can the tooling environment be integrated with continuous integration tools (e.g. Jenkins™)?

Yes, their execution can be configured in CI tools such as Gitlab and Jenkins™. In Jenkins™, there is a plugin to collect and publish Robot Framework execution results.

5) The possibility, in the execution environment, to run individual test cases as well as specific sets or the entire set of test cases.

Robot Framework test cases are created in files and directories, and they are executed by giving the path to the file or directory in question to the runner script. A set of tests can be selected by referring a specific folder or file from a lower level. It is also possible to give paths to several test case files or directories at once, separated with spaces. Tests can also be associated with tags, that can be included or excluded on run instructions.

6) The execution environment should generate error messages in case the test case cannot be executed due to any cause, including malformed definition of the test case

Robot Framework provides logs on console and on the report. Different log levels can be set: for instance, the log level can be set to debug or trace to get more information on the console.

7) Can it supply detailed step-by-step logs and overall summary of pass/fail/skip?

Yes, Robot Framework contains a statistics table with the total number of pass/fail tests, as well as the result for each test. Each individual test can be expanded to see in detail each step executed and its logs.

8) After a complete test suite generates errors/fails, can individual failed tests be re-tried?

Yes. It is possible to select tests by previous status (--rerunfailed or --rerunfailedsuites) and re-try the failed tests. It is also possible to select them by names (--test or --suite options). Results from both runs can be merged in a single report at the end with "rebot".

9) Is it possible to have the full details of requests and responses (payloads, headers)?

Yes. It is possible to do it by using the keyword "output", from the rest library, to output the request (URL, method, headers, body, etc) and the response (status, body, seconds).

10) Is it possible to define if the suite continues or aborts (or forks?) after a failed test?

Continuing the suite is the default behavior. However, it is possible to stop the suite after a failed test case by passing the -exitonfailure option parameter when launching the suite.

Community and license

1) Is the TPDL available as Open Source?

Yes, Robot Framework is a generic open-source automation framework released under Apache License 2.0. Most of libraries and tools are also published as Open Source software.

2) What is the strength of the community behind the TPDL (number of committers, frequency of releases, roadmap, etc.)?

Robot Framework is actively supported, with many industry-leading companies using it in their software development.

At the date (19.06.2020), their GitHub project has 4.8 K stars and more than 100 contributors, major releases at around twice a year (they usually have pre-releases from alpha and beta versions in between) and a roadmap for 3 more versions (v3.2.2, v3.3 and v4.0) with 124 issues open. It is used by more than 2 300 other projects. The last version is 3.2.1, it has been released on May 2020 the 4th. At the time of this writing, there are 129 issues 35 pull requests opened, but also 3 128 issues and 315 pull requests closed.

Other resources

- API Testing using Robot Framework [i.2], see https://devonblog.com/test-automation/api-testing-using-robot-framework/.
- Parameterizing Your Robot Framework Tests [i.3], see https://wonderproxy.com/blog/parameterizing-robot-framework/.
- Robot test suite developed by the ETSI GS MEC group: https://forge.etsi.org/rep/mec/gs032p3-robot-test-suite.

5.1.3 Karate

Karate is an "open source tool to combine API test-automation, mocks, performance-testing and even UI automation into a single, unified framework".

It uses the BDD syntax popularized by Cucumber. It exposes a DSL that ease testing an HTTP API and has first-class support for JSON. The BDD language is a normal English programming language for tests.

Definition and organization of tests

1) Is the programming language similar enough to "normal English" that it can be partially or completely used to define the Abstract Tests in NGSI-LD Test Purposes Descriptions specifications?

The programming language is quite readable and expressed in English.

For instance, here is a simple feature:

```
Feature: karate 'hello world' example
Scenario: create and retrieve a cat
Given url 'http://myhost.com/v1/cats'
And request { name: 'Billie' }
When method post
Then status 201
And match response == { id: '#notnull', name: 'Billie' }
```

So, it can at least partially be used to define the Abstract Tests in NGSI-LD Test Purposes Descriptions specifications.

2) How can the final list of test cases written in the TPDL be extended (under conditions compatible with ETSI IPR rules) by contributions from a developer community?

The test cases can be maintained in a Git repository, allowing contributions from a developer community. The test cases are plain text files which can easily be extended or improved.

3) How "easy" it is to write HTTP API tests (is there a DSL? are there existing functions to make assertions on the result?, etc.)

Karate is natively designed to test HTTP APIs. As such, it defines a domain specific language that offers the vocabulary needed to easily test an HTTP API test.

4) Is it possible to group/organize tests in a suite?

Karate allows to put tags (https://intuit.github.io/karate/#tags) on features.

5) Is it possible to reuse fixture data between tests?

Fixture data can be reused between different tests.

6) Is it possible to reuse functions/components between tests?

Functions can be defined in Javascript™ or Java™ and then called from features. Features can also be reused by other features (e.g. a feature to create an entity that will then be used for query testing).

7) Does it support data-driven approach?

Karate supports a data driven approach to tests (https://intuit.github.io/karate/#data-driven-tests), called a scenario outline. It can also use a dynamic scenario outline where the test data is resolved at run-time.

Developer experience

1) What is the IDE support for the TPDL?

Karate benefits from the Gherkin support already available in many IDEs. Among them, it has a specific support in IntelliJ™, Eclipse™ and VS Code (https://github.com/intuit/karate/wiki/IDE-Support).

2) Can the programming language be executed in a software environment which is familiar to a majority of web developers, e.g. Eclipse™, or does it run as a stand-alone?

The programming language can be executed inside an IDE, but it can also easily be executed on the command line or via shell scripting. As it is written in Java™, it only requires to have a JRE installed on the host.

NGSI-LD specific points of attention

1) Is it able to check that after an entity creation/update and a matching subscription the notifications are correctly delivered to the ip/port specified in the subscription?

Karate does not natively offer a support to create and start a listening HTTP server as part of a test and check for the data it receives.

However, it is possible to leverage the Async support (https://intuit.github.io/karate/#async) in order to wait for and check received notifications.

2) Is it able to check that notifications are received over MQTT (ETSI GS CIM 009 [i.1])?

Same as the point above.

3) (Keep in mind authentication and more generally security, etc. even if out of scope of the present document).

Karate allows to easily bind an authentication process as part of each or some requests, for instance to get an access token in the case of an OpenID Connect authentication.

Test Execution Environment

1) Is the Test Execution Environment offering tools familiar to the developer community such as display of results in a browser window and writing full results in log files, etc.

Yes, Karate benefits from the underlying Cucumber tool to offer tests reports in different formats (HTML, JSON, etc.). It can also be easily extended with custom formatters. Some screenshots are available in https://intuit.github.io/karate/#test-reports.

2) Is it possible to run the NGSI-LD software and the Test Execution Environment on the same host?

Yes, both can run on the same host.

3) Is the Test Execution Environment available as open source or free runtime license?

Yes, the Test Execution Environment is available as Open Source.

4) Can the tooling environment be integrated with continuous integration tools (e.g. Jenkins™)?

Yes, the environment can be invoked via a script (it requires having a JRE installed on the host), thus it can be integrated in a CI tool.

5) The possibility, in the execution environment, to run individual test cases as well as specific sets or the entire set of test cases.

Karate allows to choose to run one or more features (via a name or by specifying a path), or a set of features selected (or excluded) via tags.

6) The execution environment should generate error messages in case the test case cannot be executed due to any cause, including malformed definition of the test case.

Yes, the execution environment displays meaningful error messages in case the test case cannot be executed.

7) Can it supply detailed step-by-step logs and overall summary of pass/fail/skip?

Yes, the generated reports include a detail of all the steps, with the statuses. They also display the overall summary of the last execution.

8) After a complete test suite generates errors/fails, can individual failed tests be re-tried?

It is possible to rerun individually the failed tests when the Test Execution Environment is launched from an IDE like Eclipse™ or VS Code. It does not seem to be possible when tests are run from the command line.

9) Is it possible to have the full details of requests and responses (payloads and headers)?

Yes, the generated reports include the full details of requests and responses (request and response bodies and headers).

10) Is it possible to define if the suite continues or aborts (or forks?) after a failed test?

Karate continues the test suite if a test fails, it does not allow to abort the test suite in such case (see https://stackoverflow.com/questions/54699292/how-to-configure-karate-to-stop-execution-when-any-scenario-fails).

Community and license

1) Is the TPDL available as Open Source?

Yes, Karate is available as Open Source under the MIT license.

2) What is the strength of the community behind the TPDL (number of committers, frequency of releases, roadmap, etc.)?

At the time of this writing (March 2021), the GitHub project has 3,4 K stars and 37 contributors, new releases every few months but no clearly defined roadmaps for in-progress and future versions. It is used by more than 1 100 other projects. The last version is 0.9.5, it has been released on February 2020 the 16th. At the time of this writing, there are 47 issues and 0 pull requests opened, but also 1 009 issues and 165 pull requests closed.

The contributors page also shows that the activity is mainly driven by one person.

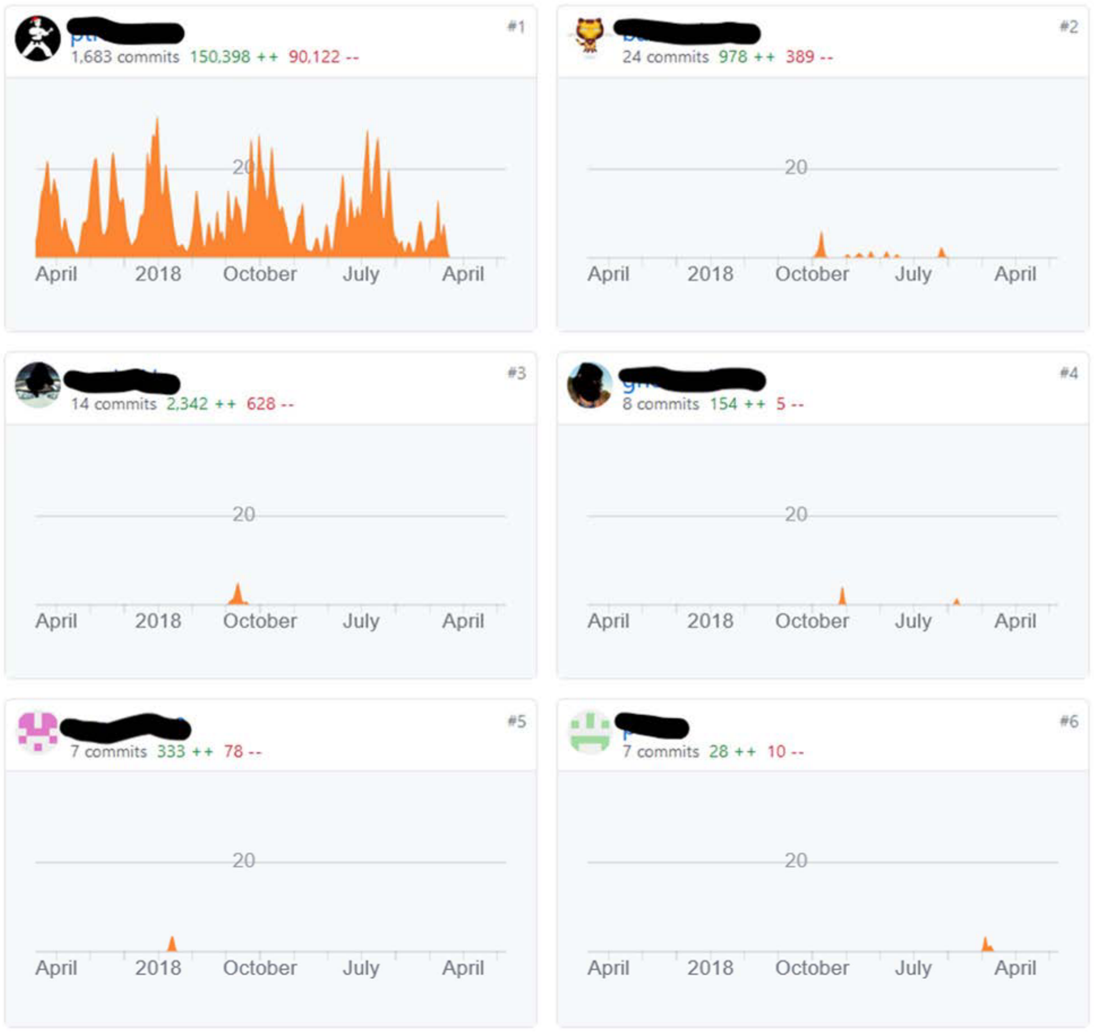


Figure 5.1.3-1: Karate GitHub users

5.1.4 OpenAPI™

OpenAPI™ is mainly a specification to define the contract of a RESTful HTTP API.

However, it also provides some tooling related to testing (https://openapi.tools/#testing). They are mainly aimed at bootstrapping some test data based on an OpenAPI™ specification file. But there also exists some tools that are able to generate test cases based on this same specification file.

As the NGSI-LD API already provides an OpenAPI™ specification file (versioned in https://forge.etsi.org/rep/NGSI-LD/NGSI-LD and exposed on https://forge.etsi.org/swagger/ui/?url=https://forge.etsi.org/gitlab/NGSI-LD/NGSI-LD/raw/develop/spec/updated/full_api.jsonht), it is interesting to experiment the existing generation tools, at least to bootstrap the structure of the tests cases and alleviate this tedious task.

This was the same process that was envisioned by the STF 576 (Methodology for RESTful APIs specs and testing, https://portal.etsi.org/STF/STFs/STF-HomePages/STF576), where an OpenAPI™ specification file is meant to be used as a basis for the generation of a skeleton Abstract Test Suite that could be then derived and enriched to form the final Abstract Test Suite. The proprietary capability extension provided by the OAS could also be used to add NGSI-LD specific metadata on each API (e.g. whether it is a mandatory endpoint, or to transmit group/subgroup information).

5.1.5 SoapUI

Definition and organization of tests

1) Is the programming language similar enough to "normal English" that it can be partially or completely used to define the Abstract Tests in NGSI-LD Test Purposes Descriptions specifications?

It is not possible to change the text that define the assertions. Tests are added by using assertions that are predefined and completing with the value desired.

2) How can the final list of test cases written in the TPDL be extended (under conditions compatible with ETSI IPR rules) by contributions from a developer community?

Install SoapUI, import Project, make the changes and then save the project (.xml format).

EXAMPLE: https://raw.github.com/SmartBear/soapui-sample-projects/master/GoogleMaps/Google-Maps-soapui-project.xml.

3) How "easy" it is to write HTTP API tests (is there a DSL? are there existing functions to make assertions on the result?)

There are predefined assertions.

4) Is it possible to group/organize tests in a suite (shall/may, subscription/entity, etc.)?

Yes. SoapUI structures functional tests into three levels; TestSuites, TestCases and TestSteps. But it is not possible to define TestSuites in different hierarchy levels (e.g: one higher level TestSuite that includes lower level TestSuites).

5) Is it possible to reuse fixture data between tests?

Groovy scripts can be written to setup a required state and teardown at the end.

6) Is it possible to reuse functions/components between tests?

It is possible to reuse other test cases, for example, using the Run TestCase TestStep.

7) Does it support data-driven approach?

Data Source test steps are only available in SoapUI Pro.

Developer experience

1) What is the IDE support to develop tests with the TPDL (syntax highlighting, code completion, run tests directly from the IDE, etc.)?

SoapUI has its own IDE which allows to create the tests, select the assertions and run the tests.

2) Can the programming language be executed in a software environment which is familiar to a majority of web developers, e.g. Eclipse™, or does it run as a stand-alone?

SoapUI needs to be installed and has Java™ as a requirement.

NGSI-LD specific points of attention

1) Is it able to check that after an entity creation/update and a matching subscription the notifications are correctly delivered to the ip/port specified in the subscription?

As long as it is possible to be verified by a Groovy script that would run in the test Tear Down.

2) Is it able to check that notifications are received over MQTT (ETSI GS CIM 009 [i.1])?

As long as it is possible to be verified by a Groovy script that would run in the test Tear Down.

3) How can it be integrated with authentication (and authorization) concerns (even though it is out of scope of the present document)?

Authentication tests can be called from other tests, which makes it easy to modify authentication if the requirements change.

Test Execution Environment

1) Is the Test Execution Environment offering tools familiar to the developer community such as display of results in a browser window and writing full results in log files, etc.

Exporting to PDF or HTML is only available in the PRO version.

2) Is it possible to run the NGSI-LD software and the Test Execution Environment on the same host?

It should be. SoapUI can be installed in either Linux, macOS® or Windows®.

3) Can the tooling environment be integrated with continuous integration tools (e.g. Jenkins™)?

It is possible by installing SoapUI in the Jenkins™ machine and calling the $SOAPUI_HOME/bin/testrunner.sh.

4) Is it possible, in the execution environment, to run individual test cases as well as specific sets or the entire set of test cases?

It is possible to select all test suites, a specific test suite or a specific test case.

5) Is the execution environment able to generate error messages in case the test case cannot be executed due to any cause, including malformed definition of the test case?

The output of the tests and the results of assertions can be seen in the terminal and in a log file.

6) Can it supply detailed step-by-step logs and overall summary of pass/fail/skip?

It is only possible in the PRO version.

7) After a complete test suite generates errors/fails, can individual failed tests be re-tried?

They can be retried one by one.

8) Is it possible to have the full details of requests and responses (payloads, headers)?

Yes. A log file can be created for each request that contains the request full details.

9) Is it possible to define if the suite continues or aborts (or forks?) after a failed test?

No. It is only possible to abort the current test after an error occurs.

Community and license

1) What is the strength of the community behind the TPDL (number of committers, frequency of releases, roadmap, etc.)?

The SoapUI Open Source project had the latest release (5.5.0) on 11 November 2019. There is a branch for a newer version (5.6.0) with recent activity but there is no information about any planned release date. According to the release history, there is a new release once a year. The Github repository has 60 contributors.

2) Is it available as Open Source or free runtime license? If so, under which license?

There is an Open Source version with the basic features. For information, the pricing for the PRO version is available on the SoapUI website: https://www.soapui.org/tools/soapui-pro/pricing/.

5.1.6 Dredd

Definition and organization of tests

1) Is the programming language similar enough to "normal English" that it can be partially or completely used to define the Abstract Tests in NGSI-LD Test Purposes Descriptions specifications?

No, focus is on validating the API based on the Swagger™/OpenAPI™ definition. It is not standard English, but different development languages can be used.

2) How can the final list of test cases written in the TPDL be extended (under conditions compatible with ETSI IPR rules) by contributions from a developer community?

Focus of the OpenAPI 2 [i.4], see https://swagger.io/specification/v2/, is on the definition and validation of an API and cannot really be used natively to write extendable test cases.

3) How "easy" it is to write HTTP API tests (is there a DSL? are there existing functions to make assertions on the result?...)?

The tool uses the OpenAPI™ specification to evaluate the implementation versus the specification. No custom tests. The tool is a specification evaluation tool.

4) Is it possible to group/organize tests in a suite?

Dredd can be integrated in a test suite like grunt, rack or meteor. It can for be integrated in grunt using the following setup: https://www.npmjs.com/package/grunt-dredd.

5) Is it possible to reuse fixture data between tests?

Yes.

6) Is it possible to reuse functions/components between tests?

The tool is based on the OpenAPI™ specification, but it can use hooks as setup or teardown. These can be reused.

7) Does it support data-driven approach?

No.

Developer experience

1) What is the IDE support for the TPDL?

Any IDE with support for Javascript™.

2) Can the programming language be executed in a software environment which is familiar to a majority of web developers, e.g. Eclipse™, or does it run as a stand-alone?

Yes, OpenAPI 2 (formerly known as Swagger™) and the different programming languages can be executed in environments similar to web developers.

NGSI-LD specific points of attention

1) Is it able to check that after an entity creation/update and a matching subscription the notifications are correctly delivered to the IP/port specified in the subscription?

No.

2) Is it able to check that notifications are received over MQTT (ETSI GS CIM 009 [i.1])?

No.

Test Execution Environment

1) Is the Test Execution Environment offering tools familiar to the developer community such as display of results in a browser window and writing full results in log files, etc.

Yes, Hooks can be used to write output to log files.

2) Is it possible to run the NGSI-LD software and the Test Execution Environment on the same host?

Yes, installed via node.js.

3) Is the Test Execution Environment available as open source or free runtime license?

Dredd runs under MIT license https://github.com/apiaryio/dredd.

4) Can the tooling environment be integrated with continuous integration tools (e.g. Jenkins™)?

Yes, examples are given for CircleCI and Travis CI.

5) The possibility, in the execution environment, to run individual test cases as well as specific sets or the entire set of test cases

Yes.

6) The execution environment should generate error messages in case the test case cannot be executed due to any cause, including malformed definition of the test case

Yes.

7) Can it supply detailed step-by-step logs and overall summary of pass/fail/skip?

With custom code in hooks

8) After a complete test suite generates errors/fails, can individual failed tests be re-tried?

No.

9) Is it possible to have the full details of requests and responses (payloads and headers)?

Yes.

10) Is it possible to define if the suite continues or aborts (or forks?) after a failed test?

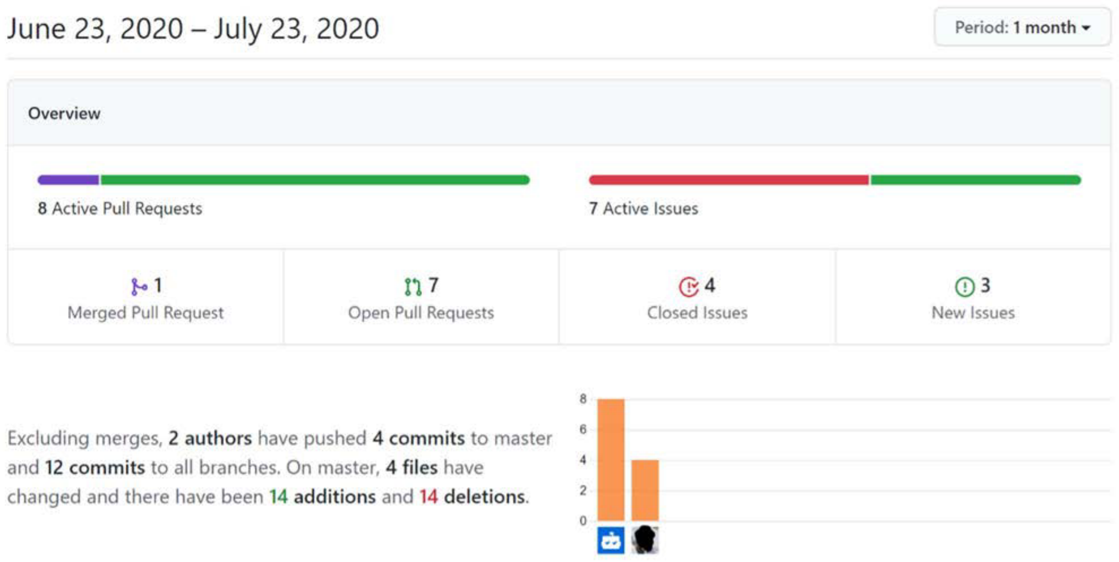
Yes.

Community and license

1) Is the TPDL available as Open Source?

Yes: https://github.com/apiaryio/dredd

2) What is the strength of the community behind the TPDL (number of committers, frequency of releases, roadmap, etc.)?



Excluding merges, **2 authors** have pushed **4 commits** to master and **12 commits** to all branches. On master, **4 files** have changed and there have been **14 additions** and **14 deletions**.

Figure 5.1.6-1: Strength of the community behind Dredd

5.1.7 Postman

Postman is a toolset for the development and the testing of APIs. It contains a set for automated testing, but it is created in the closed and proprietary Postman environment and cannot be integrated in other environments.

5.1.8 Selenium

Selenium is a toolset for web browser automation that uses the multiple techniques to remotely control browser instances and emulate a user's interaction with the browser. It allows users to simulate common activities performed by end-users such as entering text into fields, selecting drop-down values and checking boxes, clicking links in documents, mouse movement, and arbitrary JavaScript™ execution.

Selenium is used for front-end testing of websites therefore not adapted for NGSI-LD conformance testing.

5.1.9 Cucumber

Cucumber is a software tool that supports Behavior-Driven Development (BDD). It allows expected software behaviors to be specified in a logical language that customers can understand and promote BDD practices across an entire development team, including business analysts and managers. Gherkin is the language that Cucumber uses to define test cases. It is designed to be non-technical and human readable, and collectively describes use cases relating to a software system.

Cucumber is used for BDD testing to bridge the gap between business and technical people by collaborating on executable specification, therefore it not adapted for NGSI-LD conformance testing.

What is more, it is actually considered via Karate which provides a DSL for API testing above the Cucumber/Gherkin couple.

5.2 Synthesis and proposal

Tool	Synthesis
TTCN-3 & TDL	It covers the expected criteria. However, it is a complex environment and has a higher learning curve compared to other tools. There are also some uncertainties about the development environment.
Robot Framework	Robot covers the totality of the requirements and offers nice features to describe the tests purposes as "normal English". What is more, it is supported by a very large community of contributors and users.
Karate	Karate covers the large majority of the requirements. It has however two major weaknesses: it does not natively offer support for asynchronous checking, nor MQTT binding and the development effort behind Karate is mainly driven by one single person (even if it is extending, Karate is still young).
OpenAPI™ based tooling	OpenAPI™ does not provide the expected tooling and environment. However, it has to be considered as a complementary tool to be used in the global process of the test suite management.
SoapUI	SoapUI is very limited in the features it provides, even when considering the PRO version. It is also a closed environment, with few to no possibility to extend and adapt.
Dredd	Dredd is interesting in the way it binds to an OpenAPI™ specification file. But it misses a lot of the expected requirements and it is still a very young product.
Postman	Closed and proprietary environment not adapted to what is looked for.
Selenium	Not adapted to API conformance testing.
Cucumber/Gherkin	Not adapted, see Karate for API testing DSL.

Therefore, two tools meet most or all of the expected requirements:

- Robot Framework.
- TDL & TTCN-3.

However, some key points were identified in favor of Robot Framework:

- A larger community in general, and especially a larger pool of contributors.
- The existence of library to integrate protocols other than HTTP (e.g. Robot already offers a library to bind to MQTT brokers).
- The native flexibility in defining custom keywords can be of great help when translating from Test Purposes (and Test Plan) to Test Cases.
- The development and runtime environments are easy to learn and familiar to many developers (Python™).
- The execution environment is simple to set up.
- Some Specialist Task Forces (STF) inside ETSI are already using this tool.

About TDL & TTCN-3, it has been identified as a less suited candidate for the following reasons:

- Smaller community than Robot.
- It implies a lot more to handle and learn, resources are less accessible. That of course implies major risks in costs for NGSI-LD Test Purposes Descriptions & NSGSI-LD Test Suite specifications.
- There are major risks related to IDEs:
    - Titan is the only known Open Source IDE but it can only be installed on some platforms (e.g. not on macOS® which is nowadays very used among the developer community). Currently, it evolves quite regularly, but its state of implementation of the specification is not exactly know and it can be considered as a sort of single point of failure.
    - Another specific IDE is needed for TOP: https://labs.etsi.org/rep/top/ide.
- TOP only generates a skeleton, so that implies manual work afterward to complete the executable test cases.

- TTCN-3 is the Abstract Test Suite above the protocol binding (HTTP, MQTT, etc.), so the Executable Test Suite with the protocol binding has to be implemented (there exists some partial code for HTTP, nothing for MQTT for instance).

In conclusion, the Robot Framework is chosen.

Annex A: Change History

Date	Version	Information about changes
September 2020	0.1.1	Milestone A
October 2020	0.2.0	Update TDL & TTCN-3 analysis, add recommendations
October 2020	0.3.0	Change document from GS to GR
October 2020	1.0.1	Align version with expectation from the ToR
October 2020	1.0.1	Final review and analysis of TPDL, recommendations from the team of experts who developed the present document
March 2021	1.1.0	Final version
March 2021	1.1.1	Technical Officer review for EditHelp Publication pre-processing

History

Document history		
V1.1.1	April 2021	Publication