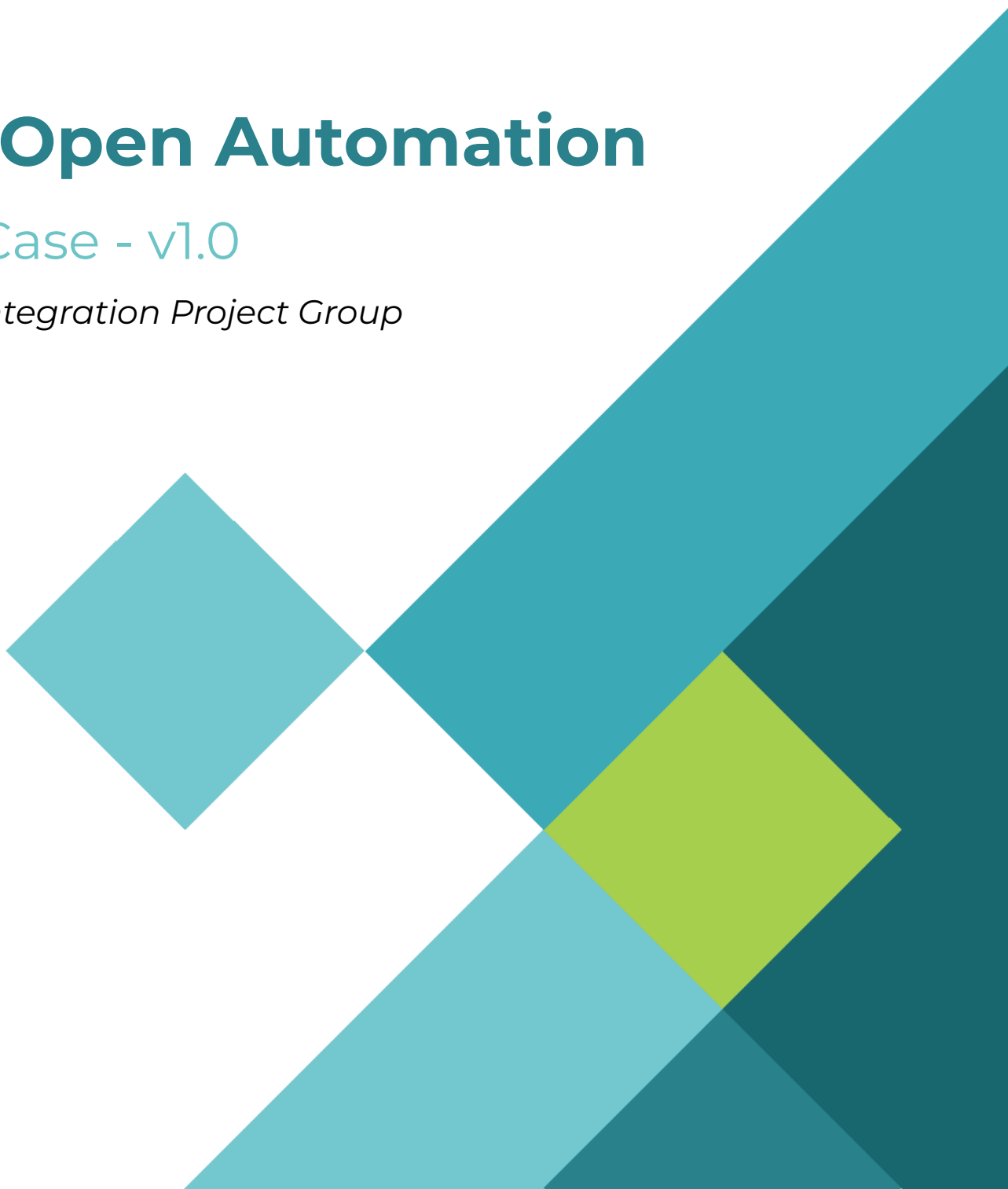
TELECOM INFRA PROJECT

TIP Open Automation

Use Case - v1.0

Test & Integration Project Group



Authors

Christophe Chevallier

Connectivity Technologies & Ecosystems Manager, Meta

christophec@meta.com

Alistair Scott

5G Solutions Planning, Keysight Technologies

alistair.scott@keysight.com

Siming Yuan

Architect, Dell Open Telecom Ecosystems Lab, Dell Telecom System Business

siming.yuan@dell.com

Jonathan Borrill

Head of Global Market Technology, Anritsu Corporation

jonathan.borrill@anritsu.com

*Releasing of the requirement document is for further study within the T&I Project Group

Contributors

Jamie Li

Software System Senior Principal Engineer, Telecom Systems Business, Dell Technologies

jamie_li@dell.com

Eng Wei Koo

Head of Standards and Solution Strategy, Keysight Technologies

engwei.koo@keysight.com

Carsten Rossenhoevel

Managing Director, EANTC

cross@eantc.de

*Releasing of the requirement document is for further study within the T&I Project Group

Exhibit B

TIP DOCUMENT LICENSE FOR USE WITH FINAL DOCUMENTS

TIP Document License

By using and/or copying this document, or the TIP document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to copy, display and distribute the contents of this document, or the TIP document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted under the copyrights of TIP and its Contributors, provided that you include the following on ALL copies of the document, or portions thereof, that you use:

• A link or URL to the original TIP document.

• The pre-existing copyright notice of the original author, or if it doesn't exist, a notice (hypertext is preferred, but a textual representation is permitted) of the form: "Copyright © 2023, TIP and its Contributors. All rights Reserved "

• When space permits, inclusion of the full text of this License should be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of TIP documents is granted pursuant to this License. except as follows: To facilitate implementation of software or specifications that may be the subject of this document, anyone may prepare and distribute derivative works and portions of this document in such implementations, in supporting materials accompanying the implementations, PROVIDED that all such materials include the copyright notice above and this License. HOWEVER, the publication of derivative works of this document for any other purpose is expressly prohibited.

For the avoidance of doubt, Software and Specifications, as those terms are defined in TIP's Organizational Documents (which may be accessed at https://telecominfraproject.com/organizational-documents/), and components thereof incorporated into the Document are licensed in accordance with the applicable Organizational Document(s).

*Releasing of the requirement document is for further study within the T&I Project Group

Disclaimers

THIS DOCUMENT IS PROVIDED "AS IS," AND TIP MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

TIP WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

*Releasing of the requirement document is for further study within the T&I Project Group

Change Tracking

Revision	Author(s)	Comment
1.0	Christophe Chevallier	Initial Draft
1.1	Siming Yuan	Review/Comments
1.2	Alistair Scott	CH6 Arch Diagram update
1.3	Jonathan Borrill	Chapter 7

Table of Contents

Table of Contents

Table of Contents

Authors 3

Contributors 4

TIP Document License 5

Disclaimers 6

Exhibit A 7

Draft Document Notice 7

TIP CONFIDENTIAL 7

Change Tracking 8

Table of Contents 9

1. Objectives 12

2. Benefits to the Industry 12

3. Out of Scope 13

4. Guiding Principles 13

4.1. Openness 13

4.2. Scalability/Extensibility 14

4.3. Maintainability 14

5. Target Audience 15

6. High-Level Architecture 17

6.1. Architecture Level Breakdown 19

*Releasing of the requirement document is for further study within the T&I Project Group

6.1.1. Test Inputs 19
6.1.1.1. Test Case Library 19
6.1.1.2. Test Scripts 19
6.1.1.3. Inputs Requirements 19
6.1.1.4. Business/Test Logic 20
6.1.1.5. Test Environment/Execution Dependencies 20
6.1.2. Common Framework 20
6.1.2.1. Automation Framework 20
6.1.2.2. Interfaces 21
6.1.2.3. Test Management 21
6.1.3. Test Executive/Sequencer 21
6.1.3.1. Test Results 22
6.1.3.2. Telemetry 22
6.1.3.3. Extension and Plugins 22
6.1.4. Libraries 23
6.1.4.1. Instrument Drivers 23
6.1.4.2. Test Harness Drivers 23
6.1.4.3. DuT/SuT Drivers 23
6.1.4.4. Telemetry Collectors 23
6.1.4.5. Connection Implementation 24
6.1.4.6. Simulator/Emulator Driver 24
6.1.5. Test Output 24
6.1.6. Test Automation Control 24
6.1.6.1. CI/CD Pipeline 24

7. Architecture for O-RU Test Automation 25

7.1 Test Automation Platform 25

7.1.1. User Repository 27

8. Requirements 29

9. References 31

[1] RU Product Test Plan 2.0 31

[2] 3GPP Conformance Document [to be included] 31

[3] O-RAN Conformance Document 31

[4] TIP T&I Open Test Automation Requirements 31

10. Terminology 32

*Releasing of the requirement document is for further study within the T&I Project Group

1 Introduction

*Releasing of the requirement document is for further study within the T&I Project Group

1. Objectives

The Open Automation project within the Test and Integration (T&I) Project Group is designed to:

- Recommend implementation of test automation to facilitate the exchange of test automation artifacts/scripts across different labs
- Ensure that the artifacts/scripts are compatible with different end-to-end test automation implementations, (i.e., test automation platform)
- Ensure that the artifacts/scripts are compatible with different test and measurement equipment (TME)
- Ensure that the different test automation platforms provide consistent and shareable test results

In this document, the term “test artifacts” refers to scripts, drivers and test cases that are used during testing.

The requirements defined as part of this project will focus on the test automation platform, assuming that the initial application will lead to the implementation of a test automation system that allows users to test Open RAN Radio Units (O-RU), following the test plan developed by the OpenRAN RU subgroup [1]. It shall be noted that this test plan references other industry test specifications or requirements, in particular as developed by 3GPP [2] and the O-RAN ALLIANCE [3].

The requirements are defined in such a way that an implementation of the platform fulfilling the requirements shall be able to evolve to accommodate other test plans, i.e., testing different devices, or systems, under test (DuT or SuT).

2. Benefits to the Industry

Test automation, so far, has been done in complete isolation by each lab, to gain efficiency in their own test execution. Such isolated implementation of test automation leads to duplication of efforts between labs, and typically the impossibility to share basic artifacts between labs. The impossibility of sharing is typically due to having

*Releasing of the requirement document is for further study within the T&I Project Group

completely different architectures, different programming languages, etcetera which would make the adaptation of a script to a new architecture a more complex endeavor than writing the script from scratch. Effectively, the impossibility to share makes the effort needed to implement test automation so high, that many testers continue to do manual testing due to the high barrier of entry for a complete test automation system.

The current Open Test Automation effort is aimed at reducing the barrier of entry for test automation by defining common test architecture that would allow re-use of test scripts, with minimal adaptation to interface with different TMEs of similar capabilities.

Re-using scripts with minimal change would allow for distribution of the effort of automating test plans for each individual lab. This also enables one lab to reproduce the testing done by another lab with minimum effort or uncertainty, enabling labs to co-operate together to resolve any issues. The stated goal will also require defined requirements on the test automation architecture, including the software architecture, in order to control different TME performing similar functions but having different models/SKUs from the same or different manufacturers. Similar to interaction with TME, the test automation platform would require flexibility to integrate into different lab environments, including but not limited to different IP networks, different user management, etc.

3.Out of Scope

The implementation of an Open Test Automation platform is not in the initial scope of the effort. The implementation phase may be considered at a later stage to validate the mandatory requirements for a given use case.

4. Guiding Principles

4.1. Openness

The test automation platform should not prescribe the use of any particular tool but allow for the integration of any of the tools currently used by the lab where the test automation will be deployed.

The adherence to the Open Test Automation requirement would NOT eliminate the

*Releasing of the requirement document is for further study within the T&I Project Group

need to have a system integrator build the overall test automation platform but would simplify the effort for building such a platform.

4.2. Scalability/Extensibility

The test automation platform should allow control of different types or models of test and measurement equipment (TME).

The test automation platform should accommodate any test campaign design to test products or systems that fulfill TIP requirements, or other industry standards with similar scope.

The initial intent for the open test automation platform is to focus on OpenRAN RU testing. The platform scalability would allow additional test cases to be implemented, as the OpenRAN RU test plan is updated. The platform scalability should also allow other devices or systems to be tested with the addition of extra TMEs, scripts, drivers, etcetera to the platform.

4.3. Maintainability

The test automation artifacts (e.g., test scripts, drivers, etc.) contributed are not centrally maintained. Each contributor of artifacts is responsible for documenting its usage and state its level of integration.

*Releasing of the requirement document is for further study within the T&I Project Group

5. Target Audience

The requirements defined may be used by System Integrators, Test Labs, Mobile Network Operators, or more generally test engineers that are developing test automation solutions or running test scripts designed to test that product or systems that are fulfilling Telecom Infra Project (TIP) requirements. Furthermore, manufacturers of systems under test are encouraged to take these requirements into account during product development, with the goal to develop an integrated CI/CD testing pipeline from vendors through system integrators and test labs, to carrier labs.

*Releasing of the requirement document is for further study within the T&I Project Group

2

Architecture

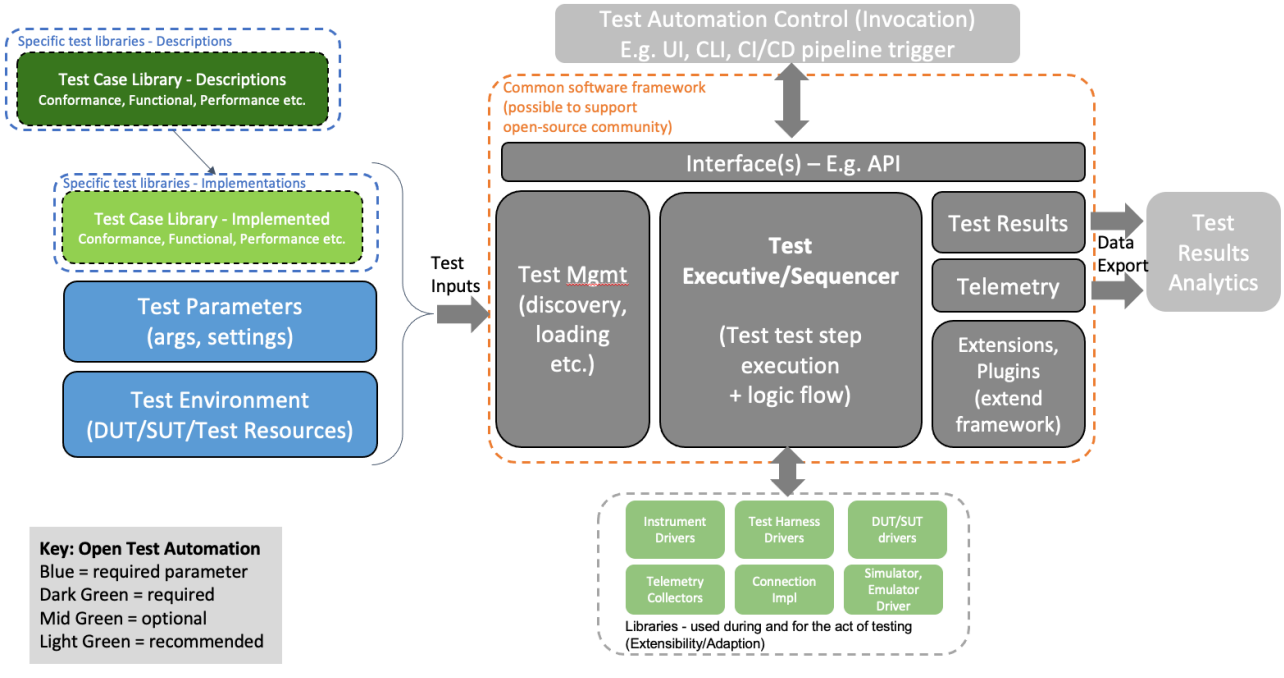
Only High-level architecture of the solution is presented in this section. Low-level architecture would typically be included in the technical requirements document.

*Releasing of the requirement document is for further study within the T&I Project Group

6. High-Level Architecture

The overall architecture defined below focuses mainly on interaction between the different functions to facilitate the implementation of an open test automation. These main functions can be divided into:

- Common software framework (dark gray). This represents the typically customized deployment of test automation in the different labs
    - *Note 1 - The implementation of the functional blocks within the framework is not prescribed in this document*
    - *Note 2 - The blocks are functional areas that may be implemented by one or more software components, depending on each the lab's automation framework*
- Inputs (left side) include both description of the test environment, test cases and parameters that need to feed into the common framework
- Libraries (bottom) that the common framework need to rely on to interact with the test environment
- Test output (right side) are defined to interface with external systems, e.g., repository
- Test automation control (top) defines how test execution can be triggered from external sources



-

Figure 1

*Releasing of the requirement document is for further study within the T&I Project Group

The architecture above has a limited (controlled) scope, focusing strictly on the core needs of testing and test execution. This is defined as the foundation, or basic necessities required for the definition of test cases, defining test inputs (e.g., environment and parameters), to test execution, to obtaining results. Any further integration (e.g., smart test selection, result analysis, database, KPIs, etc.) is considered extended capabilities that are lab/user dependent and come in the form of extensions and plugins.

Similarly, this architecture does not specify the requirements for CI/CD. It assumes that users, and labs will have their own forms of CI/CD, and that the CI/CD system provides the mechanism of integrating and testing continuously. It shall not dictate how the testing is performed or executed (as described by this architecture diagram). CI/CD systems are considered northbound to TIP Open Automation architecture and shall invoke testing cycles by calling the standard interface supported by this architecture.

To support portability and openness of test automation, and re-use across different platforms, automation frameworks and infrastructures:

- MUST support passing testing parameters (arguments, settings) to the tests, varying and/or tweaking the underlying tests as needed
- MUST support passing test environment details (DUT/SUT/test resources information such as IPs, creds) to the tests being executed
- Test case libraries MUST be data driven and decoupled from the actual system under test: all required hardware/software information that describes the system under test are to be provided (as test environment parameters) as part of test execution (as input arguments, such as files in YAML syntax, TOSCA, etc).
- MUST provide test case descriptions and references, usage details. and/or all supporting documents needed to effectively use/run the test cases
- SHOULD share (open source) the testing approach (test plan), i.e., overall testing goal, prerequisite, passed/failed criteria, and test steps are to be implemented
- SHOULD share (open source) the testing logic/implementation (e.g., open-source python test cases, or Robot-Framework scripts)
- SHOULD share (open source) the library implementation (e.g., Robot keyword libraries, simulator drivers, etc.)

*Releasing of the requirement document is for further study within the T&I Project Group

The specific format for data inputs and outputs that MUST be used (e.g., test environments, test parameters, and test results) shall be defined in the related platform definition document.

6.1. Architecture Level Breakdown

The solution allows several types of test artifacts to interwork together. This section describes each of the blocks and interfaces from Figure 1 above.

6.1.1. Test Inputs

6.1.1.1. Test Case Library

The set of instructions (words, steps) defined by a subject matter expert on how the test is to be conducted and how the outcome of the test will be evaluated, i.e., pass/fail criteria

6.1.1.2. Test Scripts

The set of code (scripts, software code) that implements any of the test cases from the library(ies)

6.1.1.3. Inputs Requirements

The set of arguments, parameters, environment variables, files, etc. that are provided to the test script as inputs and used to drive the test scripts to perform the required testing. Variances in these inputs allows the test scripts to adapt and run in the various labs and platforms it needs to be portable to.
These inputs should follow a common standard, e.g., TOSCA, YAML, where:

- they are human readable and easily modifiable
- machine parsable with cross-platform library support
- support for various data payload types, e.g., string, integer, list, dictionaries
- support for raw data (e.g., string text, binary)

6.1.1.4. Business/Test Logic

The layer where test plans and test library (e.g., the step-by-step approach of how to accomplish the coverage of an intended test) are implemented. Depending on the test framework of choice (e.g., ROBOT, pyTest, etc.), the business logic is the implementation layer of the actual test requirements, leveraging said frameworks, in a machine executable format.

6.1.1.5. Test Environment/Execution Dependencies

Description of the testing environment (aka. testbeds, networking etc.) shall be fed into the test cases as input. Examples (not limited to):

- hostnames
- IP addresses
- credentials
- static configuration (for the given lab)
- helper devices
    - router, switches that help with the testing but are not system under test
    - power distribution units (PDUs), programmable switches, etc.
    - traffic generators, simulators, emulators analyzers

6.1.2. Common Framework

6.1.2.1. Automation Framework

Foundation/low level infrastructure that dictates and guidelines the format and syntax of which the test automation test suites, test cases and steps (in general, code that gets executed) are written/implemented in. This is defined as the black box that consumes the test case implementations (in its executable format), the various test inputs, and executes them (e.g., carries out the logic necessary).
Examples of automation frameworks: OpenTAP, ROBOT Framework, Cisco pyATS, FOSS pyTest, etc.

Note that automation frameworks' impact and scope may vary; not all frameworks are the same. Some operate at a lower level (e.g., execution logic only), whereas others operate at a higher level, and govern also how reusable libraries shall be written, how the input files shall be defined, etc. The automation framework of choice largely impacts the automation engineering implementation efforts: as long as it is modular enough to

*Releasing of the requirement document is for further study within the T&I Project Group

satisfy the requirements outlined in this document, it will remain portable across labs and users.

6.1.2.2. Interfaces

Interface in architecture describes the boundary layer between the test automation system and any northbound caller. Examples of this would be the Application Programming Interface ("API", e.g., REST, gRPC) and CLI (e.g., shell), where the caller can use to perform actions such as:

- inspect the available test cases, scripts, libraries, the details, metadata, groupings, etc.
- launch new runs (execution)
- view/probe current runs/executions
- fetch recent run results (note that the functionality of databasing results for long term archiving, telemetry and metrics analysis is not a function of our system - as previously established, are not in scope of this architecture

6.1.2.3. Test Management

Test management describes the set of features and functions necessary to define, track, and organize test cases and test scripts, mapping them to their equivalent test plans. Test management may also need to provide the necessary metadata-storage mechanism where information linking (e.g., test case id) between this framework and other external tools (such as qTest, Jira etc.) are required.
Common test management system feature highlights:

- name, description, documentation, identifiers
- grouping
- relationships (pre-requisites, dependencies, commonalities)
- etc.

6.1.3. Test Executive/Sequencer

The test executive performs the task of interpreting the test logic (script & code), evaluating it using the provided inputs, and collecting logs and results that they generate. In the cases where tests have ordering dependencies, are grouped together, etc., and the end user asks the execution to be run in a certain order or sequence (through the interface described above), the test executive is responsible for carrying out these

requirements whilst meeting test dependencies described in the test management system.

6.1.3.1. Test Results

Any test configurations (of solutions under test and test tools), output (logs, files) and test verdicts (such as pass, fail, error) are to be captured in a structured format (e.g., YAML, JSON, XML) that will be archived for long term consumption. The rationale is to collect a full set of information which will enable reproducible test results.
Test results need to be indicative as opposed to binary 1/0 (e.g., basic process exit code). Often, tests could be failing but expected, passing but unexpected, errored (logic fault), aborted (user intervention), skipped (requirement not met), etc., and such cases should be captured in the result system.

6.1.3.2. Telemetry

In scope of this architecture, telemetry describes the set of data and metrics generated by the framework itself (e.g., test case start, end, number of test cases executed, lines of code, coverage). Such telemetry should be made available to an external consumer where necessary, through a commonly defined interface such as Prometheus, OpenTelemetry, etc.

6.1.3.3. Extension and Plugins

The framework should be implemented in such a way where non-core functionalities can be integrated via extensions and plugins. This requires a set of foundations (e.g., callback interfaces, hook points etc.) to be predefined, enabling third parties to integrate with.
Some examples of plugins and extensions:

- result upload to third party repositories (Jira, qTest, Rally)
- result database support (save to database)
- notification system (e.g., MS Teams, Slack channels)
- business analytical functions
- etc.

6.1.4. Libraries

Libraries are the collection of non-volatile, reusable implementations, software and programs that drive southbound devices (e.g., testbed members). Libraries are expected to be used by both the framework and the test scripts to perform the necessary testing steps, shimming and/or abstracting the underlying device functionality into a more common, manageable, API form.

6.1.4.1. Instrument Drivers

Libraries that implement the APIs for driving instruments such as analyzers, radio instrumentation, RF matrix switches, etc.

6.1.4.2. Test Harness Drivers

Libraries that implement the APIs for driving other test infrastructures, such as Jenkins, etc.

6.1.4.3. DuT/SuT Drivers

Libraries that implement the APIs for driving the system and or device under test. Typically, this abstracts away a low-level remote interface (e.g., CLI, gRPC, REST) into a higher-level, feature/function-based API set.

6.1.4.4. Telemetry Collectors

Implementing the necessary logic and actions required to initiate and collect telemetry (e.g., metrics, logs, traces) from the system under test (e.g., K8s telemetry, radio statistics, CU/DU operational telemetry, hardware temperature). Not to be confused with framework telemetry. This library should abstract away the complexity of the system under test, its various interfaces, and finally harmonize and/or aggregate the collected data streams into a more common, manageable format such as OpenTelemetry.

6.1.4.5. Connection Implementation

Low level interface that implements a connection protocol into programmable APIs. (e.g., SSH implementation, CLI driver)

6.1.4.6. Simulator/Emulator Driver

Library that drives helper devices such as traffic generators, simulators, emulators, traffic capture devices, etc. Typically comes from the actual vendor but may require an additional adaptation layer to be able to accommodate the framework requirements.

6.1.5. Test Output

Defined as everything the framework can collect during the span of testing, including but not limited to:

- test events (start, end, timing, sequence)
- test results (logs and result generated by the test code itself)
- telemetry collected front the system under test
  - logs
  - metrics
  - traces

Test outputs should be aggregated and archived into a portable format.

6.1.6. Test Automation Control

6.1.6.1. CI/CD Pipeline

Top level infrastructure responsible for continuously running (executing) test automation on a set of systems under test, collecting and analyzing the results. Whereas the test framework is responsible of the act of testing, the CI/CD pipeline decides when/where/how the test framework should be invoked and is responsible as the broker between variable inputs (such as build systems, partner software ingest) to “when testing should be performed on which testbed”.

7. Architecture for O-RU Test Automation

The details of the test automation platform implementation are not prescribed in this document. This is left for implementation.

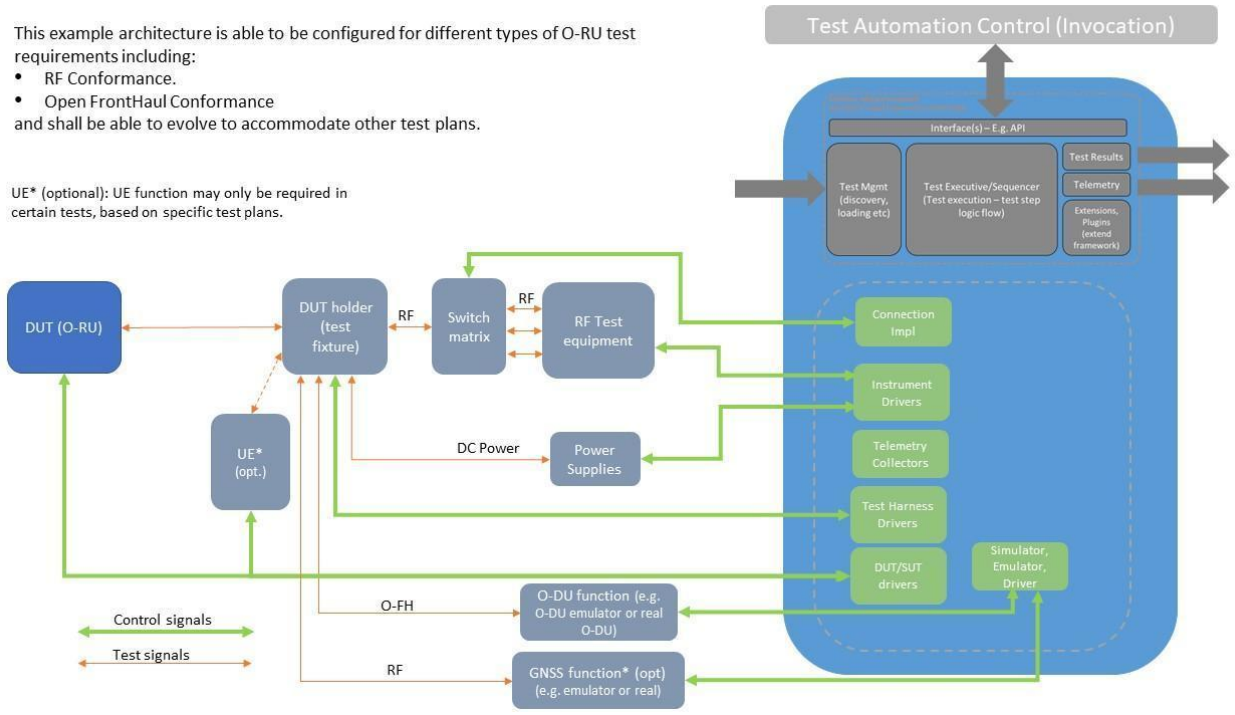


Figure 2

7.1 Test Automation Platform

The architecture for an O-RU test automation platform is defined in the above diagram, in line with the high-level architecture and 'guiding principles', and shows where required interfaces (e.g., software API's) can be specified. This will enable a solution provider (e.g., T&M vendor, system integrator) to provide O-RU test automation that can meet the objective for portability of scripts and results, by following the described API's.

The study work has defined the architecture elements that are relevant and necessary to meet this objective and describe the related parameters for each required API.

The architecture consists of the following hardware elements:

Device Under Test (DUT), which is the O-RU to be tested.

DUT Test Fixture provides the physical interfaces to connect appropriate signals in/out of the DUT. This is normally converting standardized physical interfaces on commercial test/control equipment into the proprietary interfaces of the DUT vendor.

- Switch matrix and RF test equipment, used to make measurements on the DUT. This may be physically implemented as separate switches and measurement equipment (e.g., discrete instruments) or maybe as an integrated solution (e.g., modular chassis with combined functionalities). The use of separate components is not mandated here, only the overall function to couple the signal from DUT fixture to the measuring equipment. The coupling may be done through RF cables or may be done 'over the air' (OTA), depending on the specific hardware implementation of the DUT, and the DUT Test Fixture is implemented in the corresponding way (e.g., cable ports for cable connection, or antenna ports for OTA connection).
- Power Supplies, providing the DUT input power to support O-RU hardware test cases, such as Power Outage Recovery.
- O-DU function, which provides the O-FH interface towards the DUT (O-RU). The method of implementation for DU function, and scope of the capabilities, is left open and should be corresponding to capability specific in the related test requirements.
- GNSS function, this is an optional element which provides a GNSS signal over RF. This may be required by the O-RU, depending on internal O-RU implementation, to enable correct operation of the O-RU (e.g., for timing sync).

The architecture consists of the following software (e.g., libraries) elements:

- Test automation platform, which takes inputs in terms of 'test scripts' and executes the tests based upon these test scripts, and then provides an output in terms of 'test results.' The test automation platform controls the different hardware, software, and DUT elements of the test system using API's specific to each element.
- DUT/SUT drivers, which provide the O-RU specific control and configuration commands, are used to control the DUT and provide the required configuration commands to ensure the DUT is in the correct state for measurements to be taken. The DUT driver may also return self-test and diagnostic/measurement data from within the O-RU.
- Test Harness drivers, these provide control of the DUT holder to ensure all required input and output signals are correctly routed and conditioned so the O-RU may operate correctly with different fronthaul configurations, e.g., M-plane Architecture Model or S-plane LLS mode
- Connection Implementation, this controls the Switch Matrix function to ensure the RF test equipment is correctly connected to the O-RU and able to make valid measurements. This driver will normally manage aspects such as calibration loss for the switch matrix, and any associated impedance/matching requirements.

*Releasing of the requirement document is for further study within the T&I Project Group

- Instrument drivers, e.g., O-DU emulator supporting the Fronthaul Ethernet link (e.g., 25G, 10G) as defined in O-RAN WG4 IOT profiles
- Simulator/Emulator Driver
- Telemetry Collectors

The following APIs are required for an O-RU test automation platform, in alignment with section 6 above:

- Test Management Libraries (test sequences)
- Test Management Parameters (input arguments, parameters)
- Test Management Environment (DUT/SUT settings)
- Test Results
- Test automation control and configuration API
- Equipment libraries, for DUT control, Measurement equipment control and configuration, emulator control and configuration

The defined architecture above also includes the User Equipment (UE) as an optional component in the test configuration. Normally a UE is not required for O-RU 'standalone' test, as the connection with a UE would require Open RAN DU, CU and Core Network signaling elements that enable registration procedures to take place. For the initial set of O-RU test cases (RF Conformance, O-FH Conformance, O-RU production) then a UE is not usually required. If the test plan scope is (optionally) extended to O-RU performance characteristics, including 'End to End' aspects, then UE may be supported if the O-DU interface emulation function can provide support for the required signaling.

7.1.1. User Repository

The user repository is the system where the test artifacts are stored and can also be retrieved.

Test artifacts include the following elements:

- Test Case descriptions for Test Case Libraries (e.g., reference, heading, chapter, sub-chapter, etc.).
- Test conditions (settings allowed within the test case description, other settings such as environmental conditions).
- DUT reference information (type, model, firmware, configuration, etc.).
- Test automation platform reference information (type, models, firmware, etc.).
- Test results (including references, formats, watermarks, verdicts, graphics, data, etc.).

*Releasing of the requirement document is for further study within the T&I Project Group

3

High-Level Requirements

Only the high-level requirement, or categories, are described here. Low-level requirements are defined in either the Technical Requirement document, or the Detailed Technical Requirement table

*Releasing of the requirement document is for further study within the T&I Project Group

8. Requirements

The complete set of requirements are listed in Open Test Automation Requirements [4], and are grouped as follows:

- Architecture
    - Defines the requirements relating to the overall test automation platform architecture. These requirements are completed with the architecture defined in Section 2
- Open Solution/Portability
    - Defines the requirements which make the architecture open, considering how the solution can be deployed, how TME can be controlled and integrated in the solution and specify how new artifacts can be added to the solution
- Scalability
    - Define how new artifacts can be added to the solution
- Reporting
    - Defines how the solution interface with test management tools, either as defined by TIP, or as used by the solution owner
- Maintainability
    - Defines how both the solution and the artifacts are version controlled and documented

A solution, as well as its components, implemented following the defined requirement may be considered a TIP validated solution, be badged, if it meets all the mandatory requirements. It should be noted that the requirements are defined as either:

- Mandatory: such requirements are worded as “shall”
- Optional, but recommended: such requirements are worded as “should”
- Optional, left to implementation decision: such requirements are worded as “may”

*Releasing of the requirement document is for further study within the T&I Project Group

4

References

*Releasing of the requirement document is for further study within the T&I Project Group

9. References

[1] RU Product Test Plan 2.0

[2] 3GPP Conformance Document

[3] O-RAN Conformance Document

[4] TIP T&I Open Test Automation Requirements

*Releasing of the requirement document is for further study within the T&I Project Group

10. Terminology

For consistency, the following terminology shall be used:

- Test Suite/Test Campaign: The execution of a given test plan on a given device or system under test to validate such device or system. For the validation of a DuT/SuT, the test campaign may require multiple execution of test run(s) until all mandatory test cases have passed.
- Test Plan: A written document that specifies how a device or system shall be tested to verify that such device or system fulfills the requirement that such device or system was designed to comply with. A test plan typically contains a collection of test cases, each test case defining the purpose, the expected outcome, or pass/fail criteria, and at least a summary of the steps that need to be taken to execute such a test case.
- Test Case: A single test that has a clearly defined objective, steps, and verdict. A test case would typically validate one, or part of a, or multiple, product or system requirement.
- Test Script: a computer code designed to automatically execute all the steps of a test case, including necessary configuration of either/or the DuT, SuT, or TME, set the test verdict (passed, or failed), and collect the necessary artifacts to be included in the test report. When a test case includes a prerequisite, the test scripts should validate such prerequisite, or be executed in sequence with additional scripts that validate such prerequisite(s).
- Test Verdict or outcome: Expected results of a given test case, typically pass or fail. The test verdict will typically be limited to “pass”, “fail”, or a [numerical value(s)] supplemented by evidence of the test verdict. Such evidence may include logs, observation of equipment graphical interfaces, or representation of these. Pass/Fail/Undetermined test run status based on test results and behavior. Only used if test was successfully executed, otherwise Test Fault should be indicated
- Test Report: an aggregation summary of test verdicts
- Test Session/Test Run: An individual execution of a test Plan. A given test run may include all, or some, of the test cases, depending on the test verdict of previous test sessions. More generally, a test run is a collection of test cases.

In the context of Open Test Automation, the purpose of the platform should be to:

*Releasing of the requirement document is for further study within the T&I Project Group

- As a minimum, execute a test run independently and collect evidence so that the test engineer will be able to determine the test verdict for each of the test cases
- Ideally, execute a test run independently, collect evidence, and determine the test verdict so that the test engineer will be able to determine the test verdict for each of the test cases

Releasing of the requirement document is for further study within the T&I Project Group