



**5TH GENERATION END-TO-END NETWORK, EXPERIMENTATION,
SYSTEM INTEGRATION, AND SHOWCASING**

[H2020 - Grant Agreement No. 815178]

Deliverable D3.16

Experiment Lifecycle Manager (Release B)

Editor B. García (UMA)

Contributors UMA (Universidad de Malaga), FhG (Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.)

Version 1.0

Date March 31, 2021

Distribution PUBLIC (PU)



List of Authors

UMA	University of Malaga
B. García, MM Gallardo, P. Merino	
FhG	Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.
F. Eichhorn	
NCSRD	National Center For Scientific Research “DEMOKRITOS”
H. Koumaras	

Disclaimer

The information, documentation and figures available in this deliverable are written by the 5GENESIS Consortium partners under EC co-financing (project H2020-ICT-815178) and do not necessarily reflect the view of the European Commission.

The information in this document is provided “as is”, and no guarantee or warranty is given that the information is fit for any particular purpose. The reader uses the information at his/her sole risk and liability.

Copyright

Copyright © 2021 the 5GENESIS Consortium. All rights reserved.

The 5GENESIS Consortium consists of:

NATIONAL CENTER FOR SCIENTIFIC RESEARCH “DEMOKRITOS”	Greece
AIRBUS DS SLC	France
ATHONET SRL	Italy
ATOS SPAIN SA	Spain
AVANTI HYLAS 2 CYPRUS LIMITED	Cyprus
AYUNTAMIENTO DE MALAGA	Spain
COSMOTE KINITES TILEPIKOINONIES AE	Greece
EURECOM	France
FOGUS INNOVATIONS & SERVICES P.C.	Greece
FON TECHNOLOGY SL	Spain
FRAUNHOFER GESELLSCHAFT ZUR FOERDERUNG DER ANGEWANDTEN FORSCHUNG E.V.	Germany
IHP GMBH – INNOVATIONS FOR HIGH PERFORMANCE MICROELECTRONICS/LEIBNIZ-INSTITUT FUER INNOVATIVE MIKROELEKTRONIK	Germany
INFOLYSIS P.C.	Greece
INSTITUTO DE TELECOMUNICACOES	Portugal
INTEL DEUTSCHLAND GMBH	Germany
KARLSTADS UNIVERSITET	Sweden
L.M. ERICSSON LIMITED	Ireland
MARAN (UK) LIMITED	UK
MUNICIPALITY OF EGALEO	Greece
NEMERGENT SOLUTIONS S.L.	Spain
ONEACCESS	France
PRIMETEL PLC	Cyprus
RUNEL NGMT LTD	Israel
SIMULA RESEARCH LABORATORY AS	Norway
SPACE HELLAS (CYPRUS) LTD	Cyprus

TELEFONICA INVESTIGACION Y DESARROLLO SA	Spain
UNIVERSIDAD DE MALAGA	Spain
UNIVERSITAT POLITECNICA DE VALENCIA	Spain
UNIVERSITY OF SURREY	UK

This document may not be copied, reproduced or modified in whole or in part for any purpose without written permission from the 5GENESIS Consortium. In addition to such written permission to copy, reproduce or modify this document in whole or part, an acknowledgement of the authors of the document and all applicable portions of the copyright notice must be clearly referenced.

Version History

Rev. N	Description	Author	Date
1.0	Release of D3.16	Bruno García (UMA)	31/03/2021

Index of Figures

Figure 1 The ELCM component in the Coordination Layer of the 5GENESIS reference architecture .	17
Figure 2 General architecture of the ELCM.....	20
Figure 3 Resource availability wait loop	22
Figure 4 ELCM Administration Interface	25
Figure 5 Log viewer	26
Figure 6 RTT experiment dashboard generated by the ELCM	29
Figure 7 Main classes on the Experiment life-cycle implementation.....	30
Figure 8 PlatformConfiguration and auxiliary classes	33
Figure 9 TAP test steps for MONROE experiments.....	35
Figure 10 Throughput Grafana dashboard.....	38
Figure 11 Example TestPlan and external parameters for ELCM.....	46
Figure 12 SSH Instrument configuration	47
Figure 13 Run SSH Command step settings	47
Figure 14 Retrieve Background SSH Command step settings	48
Figure 15 SCP Transfer step settings	48
Figure 16 InfluxDb result listener settings	50
Figure 17 DateTime overrides	50
Figure 18 Python debugger	51
Figure 19 ELCM model checking results.....	52

Index of Listings

Listing 1 Example of an UE definition file	27
Listing 2 Example of a test case definition file.....	28
Listing 3 NEST payload format.....	37
Listing 4 Experiment descriptor format.....	42
Listing 5 LogInfo format.....	42
Listing 6 East/West results payload	44

Index of Tables

Table 1 Document dependencies.....	16
Table 2 Available parameters for Grafana panel definition.	38
Table 3 Experiment related endpoints exposed through the Open APIs	40
Table 4 Facility related endpoints exposed through the Open APIs	41
Table 5 East/West interface endpoints.....	43
Table 6 Slice Manager endpoints used by the ELCM	44

LIST OF ACRONYMS

Acronym	Meaning
3GPP	Third Generation Partnership Project
5G PPP	5G Infrastructure Public Private Partnership
API	Application programming interface
CPU	Central Processing Unit
CQI	Channel Quality Indicator
C-RAN	Cloud-RAN
CSI	Channel State Information
DUT	Device Under Test
E2E	End To End
EaaS	Experimentation as a Service
EARFCN	Evolved-UTRA Absolute Radio Frequency Number
eMBB	Enhanced Mobile Broadband-5G Generic Service
eNB	eNodeB, evolved NodeB, LTE eq. of base station
ELCM	Experiment Lifecycle Manager
EU	European Union
EPC	Evolved Packet Core
ETL	Extract, Transform, and Load
ETSI	European Telecommunications Standards Institute
EUTRAN	Evolved Universal Terrestrial Access network
FDD	Frequency Division Duplexing
GPS	Global Positioning System
ICCID	Integrated Circuit Card Identifier
ICMP	Internet Control Message protocol
IMEI	International Mobile Station Equipment Identity
IMSI	International Mobile Subscriber Identity
IP	Internet Protocol
IOT	Internet of Things
KPI	Key Performance Indicator
LAC	Location Area Code
LTE	Long-Term Evolution
LTE-A	Long-Term Evolution - Advanced
MAC	Medium Access Control
MANO	NFV MANagement and Organisation
MCC	Mobile Country Code
MCS	Mission Critical Services
MCSI	Modulation and Coding Scheme Index
MEC	Mobile Edge Computing
MIMO	Multiple Input Multiple Output
MME	Mobility Management Entity
mMTC	Massive Machine Type Communications-5G Generic Service
MNC	Mobile Network Code

MOCN	Multiple Operator Core Network
MONROE	Measuring Mobile Broadband Networks in Europe.
NEST	Network Slice Type
NFV	Network Function Virtualisation
NGMN	Next generation mobile networks
NMS	Network Managment System
OFDM	Orthogonal Frequency Division Multiplexing
PoC	Proof of concept
PCRF	Policy and Charging Rules Function
PDCP	Packet Data Convergence Protocol (PDCP)
PDSCH	Physical Downlink Shared Channel
PoP	Point of Presence
POSIX	Portable Operating System Interface
P-GW	Packet Data Node Gateway
PLMN	Public Land Mobile Network
PMI	Precoding Matrix Indicator
PNF	Physical Network Functions
PRB	Physical Resource Block
RAN	Radio Access Network
REST	Representational State Transfer
RSCP	Received Signal Code Power
RSRP	Reference Signal Received Power
RSRQ	Reference Signal Received Quality
RSSI	Received Signal Strength Indicator
RTT	Round trip time
SCPI	Standard Commands for Programmable Instruments
SIM	Subscriber Identity Module
SIMO	Single input, multiple output
TAP	Test Automation Platform
UDP	User datagram Protocol
UE	User Equipment
uRLLC	Ultra-Reliable, Low-Latency Communications
YAML	YAML Ain't Markup Language (human readable data serialization language)

Executive Summary

The Experiment Life Cycle Manager, or ELCM, is part of the coordination layer of the 5GENESIS architecture responsible for the scheduling and execution of experiments. It handles the life cycle of an experiment from start to end. Keeping the experiment in an internal queue until all the resources required are available. It employs independent executors to run the experiment, communicates with the lower layers and provides information about the experiment execution status and metadata to the upper layers.

The general codebase of the ELCM will be common to all of the 5GENESIS platforms. However, it can be customized to meet the needs of each platform. Platforms can modify the contents of the Platform Registry or extend the ELCM by developing additional plugins.

The ELCM has been developed from the ground up using Python [1], and uses different interfaces for communicating with specific elements of the platforms. Additionally, the ELCM exposes an internal web administration interface developed in Flask [2].

Table of Contents

LIST OF ACRONYMS	10
1. INTRODUCTION	15
1.1. Purpose of the document	15
1.2. Document dependencies	15
1.3. Structure of the document	16
1.4. Target audience	16
2. RELEASE A SUMMARY AND RELEASE B INTRODUCTION	17
2.1. Release A Summary	17
2.2. Release B Introduction.....	18
3. ELCM DESIGN.....	20
3.1. Scheduler.....	21
3.1.1. Feasibility and resource availability	22
3.2. Composer.....	23
3.2.1. Variable expansion.....	23
3.3. Execution Engine	24
3.4. Other components.....	25
3.4.1. Administration interface	25
3.4.2. Platform Registry	26
3.4.2.1. Test case and UE description	27
3.4.2.2. Resources.....	28
3.4.2.3. Scenarios.....	28
3.4.3. Experiment Registry.....	28
3.4.4. Grafana dashboard generator	28
4. ELCM IMPLEMENTATION	30
4.1. Experiment life-cycle implementation.....	30
4.1.1. The ExperimentRun class.....	30
4.1.2. The ExecutorBase and Child classes	31
4.1.3. Tasks	32
4.2. Composer.....	33
4.2.1. The composition process	33
4.3. Experiment execution workflow	34

4.3.1. Standard and custom experiments	34
4.3.2. MONROE experiments	35
4.3.3. Distributed experiments	35
4.4. Network Services deployment	36
4.5. Grafana dashboard generation	37
4.5.1. Grafana dashboard auto-generation	39
5. ELCM INTERFACES	40
5.1. Northbound interfaces	40
5.1.1. Open APIs	40
5.1.2. 5Genesis Portal	42
5.2. East/West interface	42
5.3. Southbound interfaces	44
5.3.1. Katana Slice Manager	44
5.3.2. Network management system (NMS) and platform infrastructure.....	45
5.3.2.1. TAP (OpenTAP) as execution environment.....	45
(a) Integration of TAP test plans	45
(b) SSH TAP Plugin	46
5.3.2.2. Generic script execution and additional integration	48
5.3.3. Analytics module and Results Registry	49
5.3.3.1. InfluxDb Helper class and CsvToInflux	49
5.3.3.2. InfluxDb Result Listener	49
6. TESTING AND VALIDATION.....	51
7. CONCLUSIONS	53
8. REFERENCES.....	54

1. INTRODUCTION

1.1. Purpose of the document

This deliverable describes the progress of designing and implementing the Experiment Life Cycle Manager (ELCM) for Release B, as well as the changes performed since the publication of the Release A of this component. The ELCM is the entity that performs the management, orchestration and execution of Experiments in the 5GENESIS Platforms, and has been developed from the ground up as part of the 5Genesis Open Experimentation Framework.

In this document the reader can also find details about the different interfaces of this entity, namely the northbound interface (exposed by the ELCM), the southbound interfaces (used for controlling the different elements of the 5GENESIS facilities) and the East/West interface (used for communicating two different ELCM instances during a distributed experiment execution).

Additionally, some information regarding the basic testing of the developed functionality (as an initial step before the in-depth testing performed as part of Work Package 5) is also included.

1.2. Document dependencies

The ELCM design and implementation is based on specifications and requirements described in the Architecture related deliverables, which are detailed in Table 1.

It should be noted that this deliverable does not follow the same convention as most of the other Work Package 3 deliverables, where only the new components and functionality added in Release B are detailed, while referencing the original Release A deliverables for the rest of the content.

This deliverable can be considered self-contained and supersedes D3.15 in order to improve readability, given that many of the design and implementation concepts were introduced during Release A and it would be difficult to understand the changes on Release B without these concepts.

However, readers who have previously read D3.15 can find, on sections where content overlaps, a section called '**Differences from Release A**', which shows in a concise way the changes performed during the development of Release B.

Table 1 summarizes the relevance of previous deliverables produced by the 5Genesis project towards this document.

Id	Document title	Relevance
D2.2 [3]	5GENESIS Overall Facility Design and Specifications	The 5GENESIS facility architecture is defined in this document. The list of functional components to be deployed in each testbed is defined.
D2.3 [4]	Initial planning of tests and experimentation	This document describes the different components of the coordination layer and defines the sequence of interactions between the components of the facility during an experiment execution.
D2.4 [5]	Final report on facility design and experimentation planning	This document describes the final architecture of the coordination layer, as well as the requirements and work flow followed during the execution of a distributed experiment.
D3.15 [8]	Experiment and Lifecycle Manager	This deliverable describes the design and implementation of the ELCM during Release A

Table 1 Document dependencies

1.3. Structure of the document

The document is structured as follows:

- Section 1, *Introduction* (the present section)
- Section 2, *Release A summary and Release B introduction*, summarizes the state and available features of the ELCM during Release A, and specifies the improvements available as part of Release B
- Section 3, *ELCM Design*, describes in detail the design principles of the ELCM
- Section 4, *ELCM Implementation*, describes the implementation of the ELCM Release B
- Section 5, *ELCM Interfaces*, presents the interfaces and helpers that can be used by the ELCM during the execution of an experiment.
- Section 6, *Testing and Validation*, provides a small summary of the initial tests performed in order to confirm the correct operation of the ELCM.

1.4. Target audience

This document provides details about the functionality supported by the ELCM, as well as high-level information regarding its design and implementation. However, specific details that may only be useful while contributing to the codebase of the ELCM are not included.

Therefore, the target audience of this document are the 5GENESIS Platform administrators, who are required to know about the general implementation aspects of the ELCM in order to effectively configure and manage this element.

2. RELEASE A SUMMARY AND RELEASE B INTRODUCTION

2.1. Release A Summary

The Release A of the ELCM contained part of the functionality included in the Release B, and has been used as the base for the development of the second release.

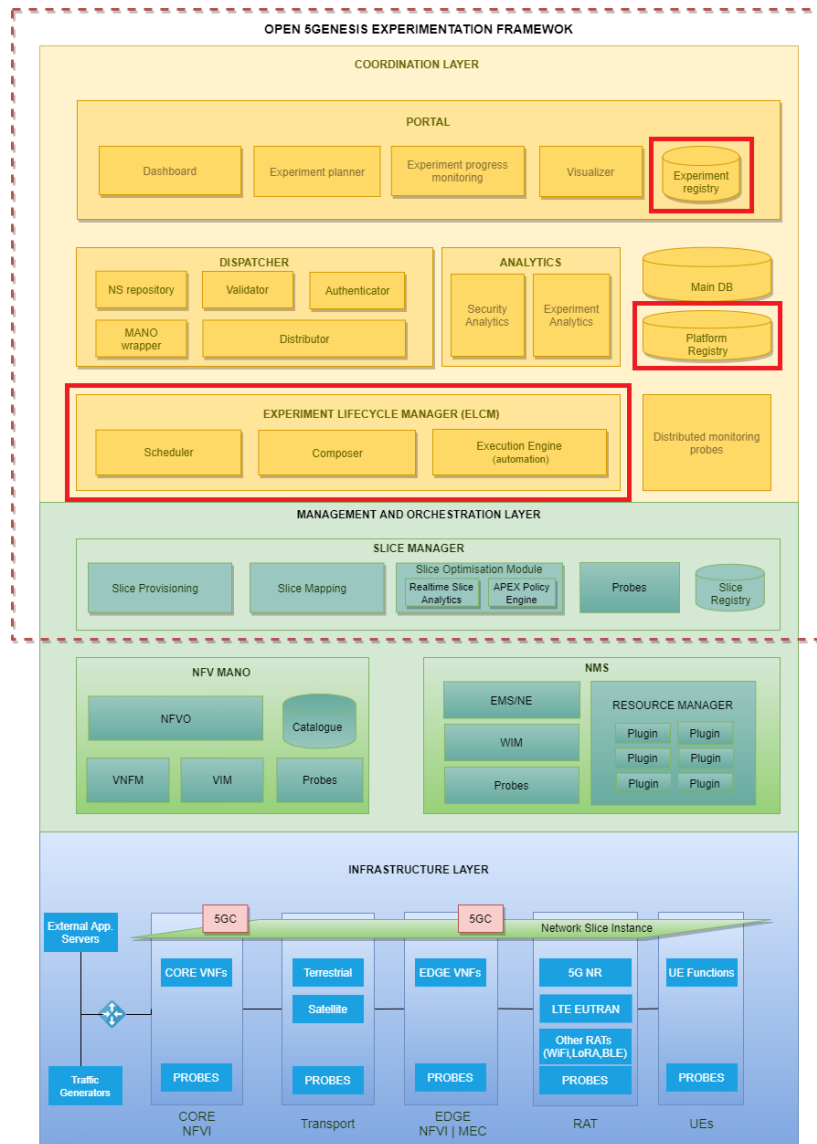


Figure 1 The ELCM component in the Coordination Layer of the 5GENESIS reference architecture

This initial release has been successfully integrated on all of the 5Genesis platforms, being used during the experimentation phase reported on Deliverable D6.2 [10], which demonstrate that the architecture designed for the Release A is suitable for the requirements of the 5Genesis project.

The Release A development cycle included:

- The design and implementation of the ELCM's general architecture, including the separation on the Scheduler, Composer and Execution Engine components, as well as the Platform-Specific Configuration composition process.
- The implementation of the initial logic for executing experiments, separated in three different stages (Pre-Run, Run and Post-Run), with the execution of a specific test-case defined by a set of Tasks.
- The implementation of several Task types, for use by platform administrators during the definition of the available facility test cases.
- The creation of the administration interface

The Release A of this component became open source software in May 2020, receiving its final revision on March of the same year, and is available for reference as a separate branch in the ELCM Github repository ¹the implementation details of this version can be seen on Deliverable D3.15 [8], which the present document extends by detailing the changes and additions introduced during the development of the Release B.

2.2. Release B Introduction

The development of the Release B began on October 2019, in parallel with the support and development of the final features of the Release A. The source code of the Release B is available as a separate branch in Github ², which, at the time of writing, is also the default branch of the repository.

The following list includes some of the changes and additions implemented as part of the Release B development:

- Scheduling of experiments based on the resources available in the platform, which includes:
 - o Blocking the execution of an experiment until the required equipment has been released by other experiments that are making use of.
 - o Blocking the execution until the required computational resources are available in the Management and Orchestration layer
 - o Automatically cancelling the execution of experiments that are deemed unfeasible due to their requirements.
 - o Allowing experimenters to request the execution of an experiment exclusively in the platform, without any other experiments running at the same time, regardless of the requirements.
- Support for experiments with a set of user customized parameters.
- Support for the execution of experiments using a MONROE node.
- Implementation of an East/West interface for communication between two ELCM instances and support for the execution of distributed experiments.
- An update on the composition logic, in order to support the changes introduced in the second version of the Experiment Descriptor.
- The integration with the Dispatcher component, which provides the required privacy and security management.
- Additional support for the functionality exposed by the Open APIs.

¹ https://github.com/5genesis/ELCM/tree/release_A

² https://github.com/5genesis/ELCM/tree/release_B

- The implementation of new Task types that give additional possibilities to platform administrators while describing the implementation of the platform's test cases.

In addition, many of the features present in the Release A received further refinement, extending and improving the functionality or fixing detected issues.

3. ELCM DESIGN

The ELCM is divided in 3 main components, as well as several auxiliary elements. These main components are:

- The **Scheduler**, which is responsible for managing the execution of the experiments on a higher level: An experiment execution comprises three stages (Pre-Run, Run and Post-Run), and the Scheduler keeps track of the execution of each of these stages for multiple experiments in parallel.
- The **Execution Engine** includes the logic for managing the execution of each experiment stage, by generating an independent Executor. The progress in each Executor is further divided in different Tasks, which are dependent on the test case and the equipment involved in the experiment.
- The **Composer** is the entity responsible for creating the Platform Specific Configuration of the received experiments. The configuration generated includes the Tasks to be run by the Executors and will depend on the contents of the Facility Registry and the contents of the Experiment Descriptor.

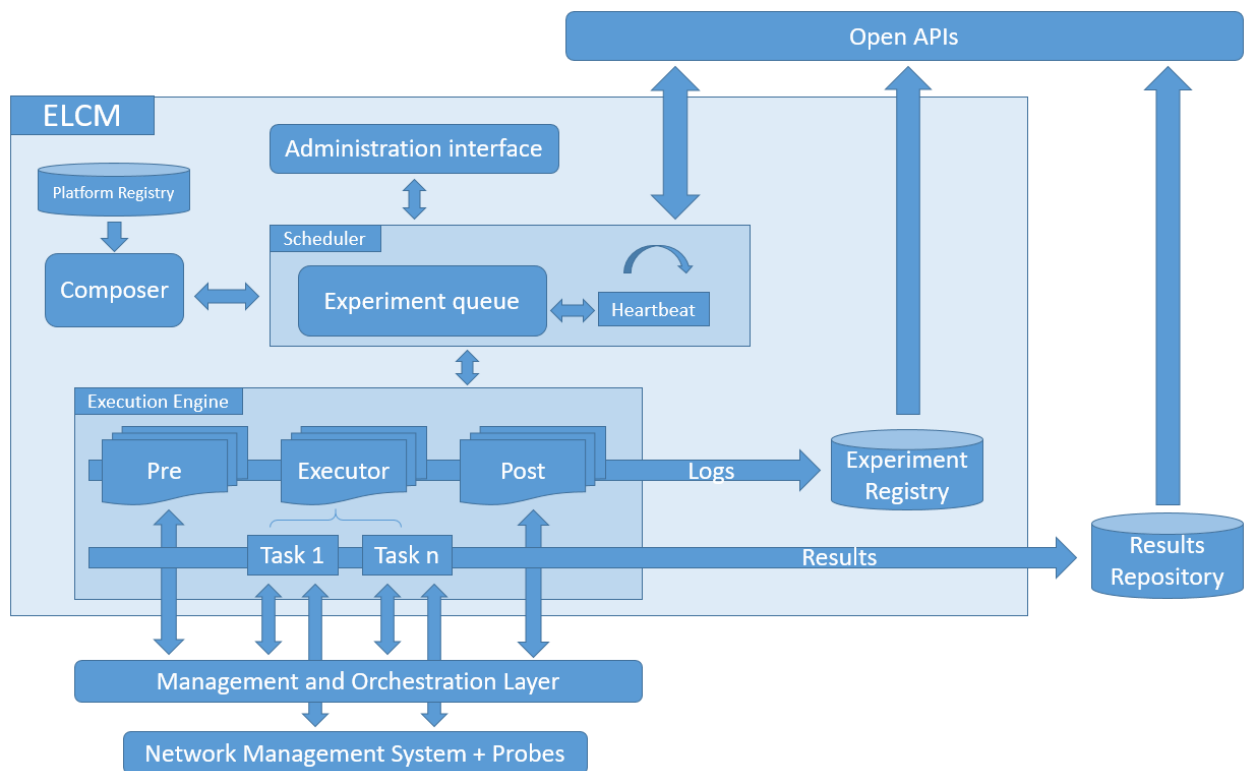


Figure 2 General architecture of the ELCM

Additionally, the following elements are included within the ELCM instance:

- The **Platform Registry** is the set of configuration files that define the capabilities and expected behaviour of the platform when specific test cases and equipment are tested. These configuration files are organized in different folders that reside alongside the ELCM.

- The **Experiment Registry**, which stores the logs generated by the different experiment executions, as well as certain metadata, is also part of the ELCM.

The work-flow of the ELCM when an experiment execution is requested is as follows:

- The Scheduler creates a new Experiment Run instance. These objects contain all the information about a particular execution.
- The Scheduler requests the creation of a Platform Specific Configuration to the Composer, using the Experiment Descriptor received on the request.
- The Composer uses the information contained in the Platform Registry along with the specific requests contained in the Experiment Descriptor to generate this configuration (including the Tasks to execute).
- The Scheduler queues the experiment execution, starting from the Pre-Run stage. The execution is then handled by the Pre-Run Executor, which runs on a separate thread and waits until all resources are available (among other actions).
- When the Pre-Run executor finishes (which means that all the required resources are available), the Scheduler moves the experiment to the Run stage. Again, the execution of the specific experiment Tasks is handled by a different thread, allowing the concurrent execution of different experiments.
- The Scheduler moves the execution to the Post-Run stage once the Run stage finishes, and additional Tasks run on the new Executor.
- When finished, the Scheduler removes the Experiment Run from the queue.

Differences from Release A:

- The ELCM exposes the northbound interface to the Open APIs, instead of communicating directly with the Experimenter Portal.
- The Platform and Experiment Registries are now officially part of the ELCM (instead of being temporarily included). Although they were separate entities in the 5Genesis architecture, these components are tightly coupled with the ELCM. To improve performance and ease of operation it was decided to deploy them together.
- The communication options with the lower layers have been clarified in Figure 2: Though it was already the case for Release A, it is now explicit that the experiment's Tasks can communicate directly with the different probes and components of the platform's infrastructure.

3.1. Scheduler

The Scheduler is the component that manages the execution of the experiments at the Stage level. The stages defined are:

- **Pre-Run:** This stage includes the experiment registration and configuration, the feasibility check and the wait loop until for the required resources, and the instantiation of the required network services.
- **Run:** The Run stage is different for each experiment type, and consist on the execution of the actions required for running the experiment's test cases in the platform.
- **Post-Run:** During this stage, the resources used by an experiment are released, including the decommissioning of network services.

All the experiments are kept on an internal Execution queue, where they will transition from one stage to the next. When an experiment enters one of the stages the execution is handled by an independent Executor, which is able to run in parallel with any other Executors belonging to other experiments. These Executors will run their specific tasks one after another, until they reach completion or an error is detected.

The ELCM is able to respond to events in an asynchronous manner as they are received via the REST API or the administration interface, while separate Executors can run in parallel. However, the transitions between different stages of an experiment execution are coordinated by a separated background thread known as the Heartbeat. This thread will periodically trigger a check on the status of every Executor from active experiments (i.e. on any of the Run stages), triggering the transition to the next stage when an Executor has finished, or marking them as errored or user cancelled when necessary.

The use of the Heartbeat thread allows us to reach a good balance between the performance gains of the parallel execution of multiple experiments, and the predictability of handling the stage transitions in a sequential order, which makes, for example, the resource handling logic more robust.

Differences from Release A:

- The instantiation and decommission of network services are now part of the Pre and Post-Run stages respectively, instead of the first and last Tasks of the Run stage.
- Resource management and the feasibility check did not exist in Release A, meaning that multiple experiments could make use of the same equipment in parallel, possibly causing execution errors or inconsistent results.

3.1.1. Feasibility and resource availability

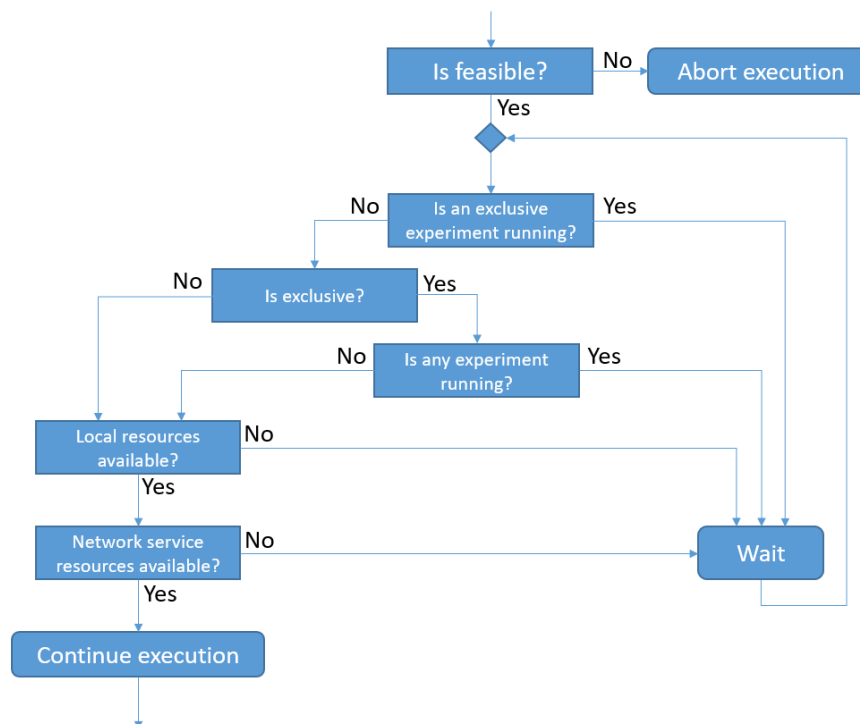


Figure 3 Resource availability wait loop

As part of the Pre-Run stage, every experiment execution is tested for feasibility, and is forced to wait until the required resources are available. By following the flow presented in Figure 3 the ELCM ensures that every experiment can safely run in the platform, without interfering with others. Additionally, the ELCM handles the execution of experiments on a first come-first served basis, meaning that for cases where two experiments are waiting for the same resource, and both can be run, the ELCM will give priority to the one that has been waiting for longer.

The feasibility check verifies that the experiment does not require more resources than those available in the Management and Orchestration layer, as well as confirms that the required local resources have been defined in the platform. The ELCM requests information about the maximum number of computation cores, memory and storage that can be used for the deployment of network services from the Slice Manager, as well as the requirements for each of the network services included in the Experiment Descriptor. Then compares the maximum values with the total amount required for the network services deployment, aborting the execution of the experiment if this total exceeds the maximum amount of resources available in the facility.

If the experiment is feasible, the ELCM checks if an exclusive experiment is being executed in the platform, or if the experiment is exclusive itself and other experiments are running. In both circumstances, the experiment must wait for a certain time before checking the testbed availability again. For every failed test the experiment must wait for 10 seconds before the conditions are checked again. This value was selected because it presents a good balance between loading the system with many unnecessary checks and making the experiment wait for additional time when it is not required.

When the experiment passes the previous test, the availability of the required resources is checked. First, by testing that all the physical resources of the facility are available for use and, if needed, that enough computational resources in the MANO layer are free. In this case, the execution of the experiment is permitted and the next steps can be performed.

3.2. Composer

The Composer is able to create the Platform Specific Configuration of the experiments, by using the information available in the Facility Registry in conjunction with the Experiment Descriptor received along with the execution request. By using this information, the Composer can generate the list of Tasks that are to be executed during the experiment run, as well as any other configuration value needed to support the execution.

3.2.1. Variable expansion

Certain values cannot be known while the platform administrator defines the Tasks that take part in a test case's execution, for example, the Execution ID (that depends on the number of experiments executed previous to the current one in the platform) or the location of an experiment execution's temporary folder, which is created at run-time. Other values may depend on the output provided by external applications or, in the case of distributed experiments, be defined by the remote platform. For this reason, the ELCM offers a set of 'placeholders'. The placeholders follow a predefined format (a known identifier enclosed in

curly or square brackets, preceded by '@'), to ease their detection. The ELCM will substitute them with the appropriate run-time values, when the respective Task starts running.

The following values are recognized while enclosed by curly brackets ('@{ ... }'):

- ExecutionId: Unique ID of the experiment execution.
- SliceId: ID of the network slice deployed during the Pre-Run stage.
- TempFolder: Path to the temporary folder created exclusively for use of the current experiment executor.
- Application: The contents of the 'Application' field in the Experiment Descriptor.
- JSONParameters: The complete 'Parameters' dictionary of the Experiment Descriptor, encoded in JSON format.
- ReservationTime and ReservationTimeSeconds: The value of the 'ReservationTime' field of the Experiment Descriptor.
- TapFolder and TapResults: Paths to the folders where the OpenTAP instance and result output reside, as set in the configuration of the ELCM.

It is also possible to expand the values contained in the 'Parameters' dictionary independently, using the following expressions:

- @[Params.key]: The value of 'key' in the dictionary, or '<<UNDEFINED>>' if not found
- @[Params.key:default]: The value of 'key' in the dictionary, or 'default' if not found

Additionally, platform administrators have access to Tasks that are able to generate new (key, value) pairs at runtime, either with a fixed value, a value extracted from a file or taken from a previous task. Once a value has been published using either of these methods, it becomes available for variable expansion using the following expressions:

- @[key] or @[Publish.key]: Expands to the value of 'key' if defined, '<<UNDEFINED>>' otherwise
- @[key:default] or @[Publish.key:default]: Expands to the value of 'key' if defined, or 'default'.

Differences from Release A:

- The composition logic was updated in order to support the second version of the Experiment Descriptor.
- Many variables were not available during Release A, and the 'Parameters' dictionary did not exist.

3.3. Execution Engine

The Execution Engine is responsible for performing the specific actions required for the execution of a specific execution Stage. These Stages are, in turn, composed of different Tasks. For every Stage, a different Executor will handle the execution of the defined Tasks.

The Executors will run all the tasks one after another, in the order defined by the Composer, however, multiple Executors can run in parallel. Due to this, the ELCM is able to run any number of experiments at the same time, provided that there are enough resources in the platform for all of them.

Each of the different Tasks may perform any action on other components of the platform. For example, it's possible to define a Task that calls a shell script for enabling iPerf on a remote machine or another that executes a TAP[1][12] (Test Automation Platform) TestPlan that activates a probe running on a mobile phone.

Differences from Release A:

- There are no important differences on the implementation of the Execution Engine.

3.4. Other components

3.4.1. Administration interface

The Administration Interface is a web application developed in Flask[2] that gives a unified interface to platform administrators. Here they can review the execution status of an active experiment run and the usage of physical resources in the platform. From this interface, it's also possible to cancel the execution of an experiment and review the generated execution logs.

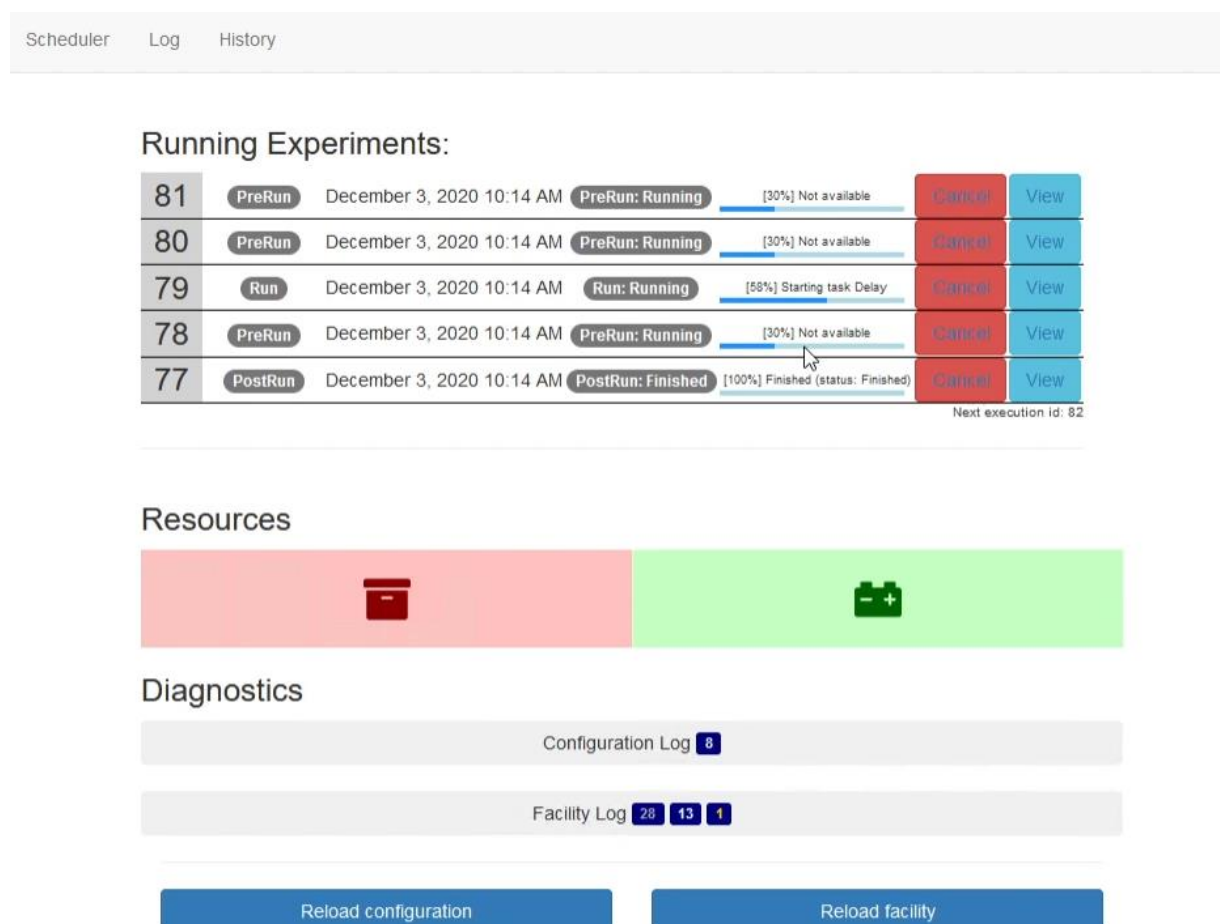


Figure 4 ELCM Administration Interface

Scheduler Log History

Status: **Finished** Created: June 10, 2019 11:11 AM (21 days ago)

Pre-Run

Started: June 10, 2019 11:11 AM (21 days ago, waited a few seconds)

Finished: June 10, 2019 11:11 AM (21 days ago, ran for a few seconds)

Pre-Run Log 7 10

Debug 7 Info 10 Warning Error Critical

```

2019-06-10 11:11:22,202 - INFO - Started
2019-06-10 11:11:22,207 - INFO - [Starting Task Check Availability]
2019-06-10 11:11:22,209 - INFO - Requesting availability
2019-06-10 11:11:22,210 - INFO - Resources available
2019-06-10 11:11:22,210 - INFO - [Task Check Availability finished]
2019-06-10 11:11:22,213 - INFO - [Starting Task Add Experiment Entry]
2019-06-10 11:11:22,214 - INFO - Sending entry information
2019-06-10 11:11:25,215 - INFO - Information sent
2019-06-10 11:11:25,216 - INFO - [Task Add Experiment Entry finished]
2019-06-10 11:11:25,222 - INFO - Finished (status: Finished)

```

Run

Started: June 10, 2019 11:11 AM (21 days ago, waited a few seconds)

Finished: June 10, 2019 11:11 AM (21 days ago, ran for a few seconds)

Run Log 11 34

Post-Run

Started: June 10, 2019 11:11 AM (21 days ago, waited a few seconds)

Finished: June 10, 2019 11:11 AM (21 days ago, ran for a few seconds)

Post-Run Log 9 14

Figure 5 Log viewer

Differences from Release A:

- The interface includes the visualization of used resources in the platform.
- Diagnostic logs are now visible in the interface, making it easier for platform administrators to detect errors in the configuration or in the definition of the facility test cases.

3.4.2. Platform Registry

The Platform Registry defines the exposed functionality and behaviour of a 5Genesis facility, and comprises a set of configuration files in YAML format, distributed across four different folders that reside along with the ELCM files. These folders are:

- TestCases: Contains information about the available test cases that can be run in the facility.
- UEs: Contains specific actions that are required for using and releasing specific equipment of the facility during the execution of test cases.
- Resources: Contains the description of certain facility equipment, in particular those that can be used only by a single experiment at a time.
- Scenarios: Contains additional configuration values that can be set during the deployment of a network slice.

Differences from Release A:

- During Release A, a single file (facility.yml) defined both test cases and UEs. Resources and scenarios were not available.

3.4.2.1. Test case and UE description

The contents of the 'UEs' and 'TestCases' sub-folders describe the behaviour of the 5Genesis Platform when an Experiment execution request is received. The ELCM will load the contents of every YAML file contained in these folders on start-up and whenever the 'Reload facility' button on the web dashboard is pressed. The dashboard will also display a validation log ('Facility log') which can be used in order to detect errors on a test case or UE configuration.

The files in both folders share a similar format, both containing a main dictionary key that defines the name of the test case or UE, which contains the set of actions to perform when the UE is in use or the test case is being executed. For test cases, the configuration files contain additional keys used to define the kind of experiment (standard, custom, public or private, distributed), the available configuration parameters (for custom experiments), or the contents of the Grafana dashboard. An important field that is included in the definition of every task is the 'Order' value, which is used during the composition process while generating the final list of actions to execute.

Listing 1 and Listing 2 show an example of the format of an UE and test case description file, respectively. More information about the composition process can be found in Section 4.2.1.

```
TestUE:
  - Order: 1
    Task: Run.Message
    Requirements: [UE1]
    Config:
      Message: This is a dummy entity initialization
      Severity: INFO
  - Order: 10
    Task: Run.Message
    Config:
      Message: This is a dummy entity closure
      Severity: INFO
```

Listing 1 Example of an UE definition file

```
Slice Creation:
  - Order: 5
    Task: Run.SingleSliceCreationTime
    Config:
      ExperimentId: "@{ExperimentId}"
      WaitForRunning: True
      Timeout: 60
      SliceId: "@{SliceId}"
Standard: True
Distributed: False
Dashboard:
  - Name: "Slice Deployment Time"
    Measurement: Slice_Creation_Time
    Field: Slice_Deployment_Time
    Unit: "s"
    Type: Singlestat
    Percentage: False
    Size: [8, 8]
    Position: [0, 0]
```

```
Gauge: True
Color: ["#299c46", "rgba(237, 129, 40, 0.89)", "#d44a3a"]
Thresholds: [0, 15, 25, 30]
```

Listing 2 Example of a test case definition file

3.4.2.2. Resources

The files contained in the Resources folder are used to describe certain physical or logical equipment that cannot be used concurrently by several experiments. For example, the UE configuration file for a specific mobile phone is usually tied to one resource file, so that this phone is only used by a single experiment at a time. Another possible example is the use of a certain probe during the execution of a particular kind of experiment.

Resource definition files contain the following keys:

- **Id**: Resource ID. This ID must be unique to the facility and will be used to identify the resource during test case execution.
- **Name**: Name of the resource (visible on the ELCM dashboard).
- **Icon**: Resource icon (visible on the ELCM dashboard).

Required resources are configured per task. When an experiment execution is received, the ELCM will generate the list of all required resources for all the tasks in the experiment (either from a test case or UE definition). When an experiment starts, all these resources will be locked and the execution of other experiments with common requirements will be blocked, until the running experiment finishes and their resources are released.

3.4.2.3. Scenarios

In the context of the ELCM a Scenario is a collection of configuration values that are used to further customize the behaviour of a deployed slice. These files contain a dictionary with a single key (that defines the Scenario name) and, as value, a dictionary that contains the collection of values that are to be customized by the Scenario. More information about the usage of scenario files can be seen in Section 4.3.

3.4.3. Experiment Registry

The Experiment Registry contains information about each independent experiment execution, including logs and additional metadata, such as the start and end times, or the final execution status of each stage, along with certain files generated by the executed tasks. The Experiment Registry is used to support some of the endpoints of the Open APIs, for example, those that allow the retrieval of experiment logs.

3.4.4. Grafana dashboard generator

The ELCM is able to request the generation of custom dashboards to a running Grafana [14] instance where the experimenter can review the results generated by an experiment. In order to generate the dashboard the ELCM will use the information contained in the Facility Registry, where the platform administrators can include the definition of several Grafana panels

following the format described in Section 4.5. Grafana will use the results generated by the experiment in order to populate the contents of the Dashboard's panels.

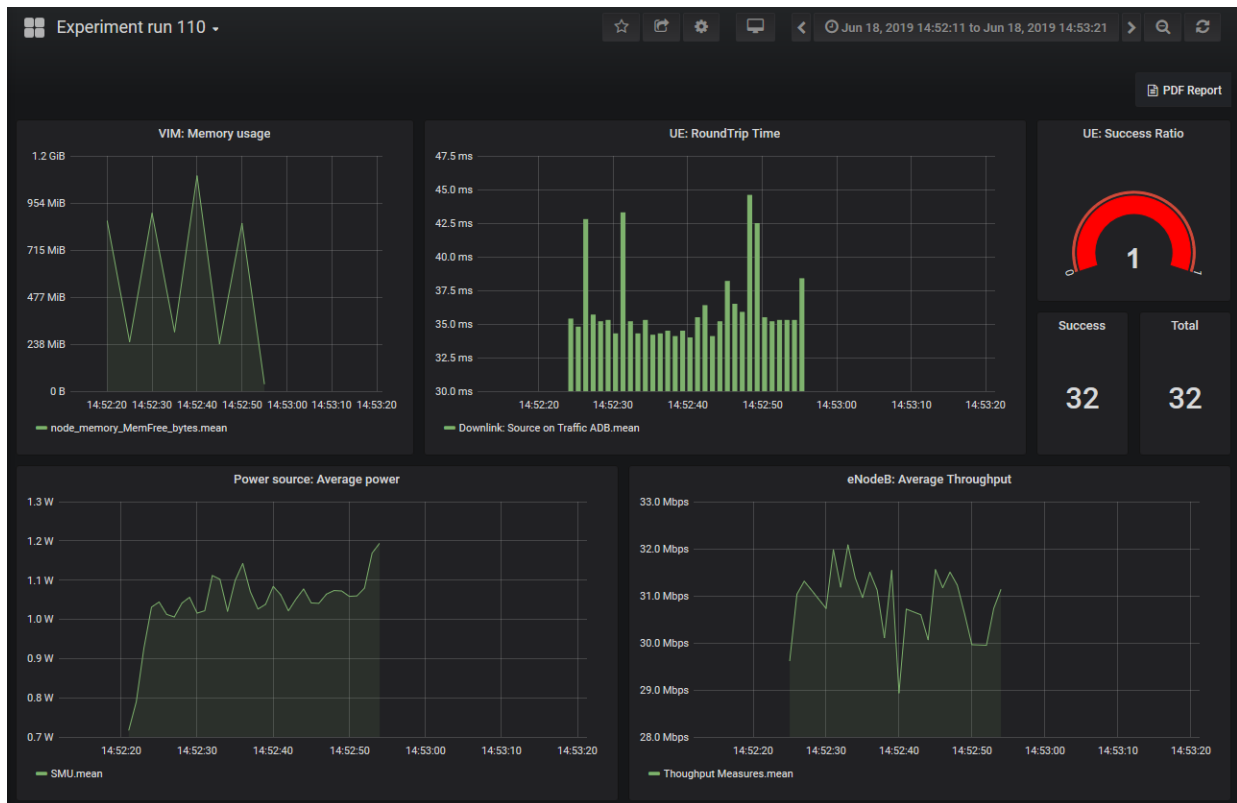


Figure 6 RTT experiment dashboard generated by the ELCM

Differences from Release A:

- Additional values for configuring the color of elements and the rendering of dots in line graphs have been added for Release B, but there are no important changes in the functionality.

4. ELCM IMPLEMENTATION

4.1. Experiment life-cycle implementation

This section provides some additional details about the logic and implementation of the experiment life-cycle, from the reception of an execution request until the end of the experiment execution.

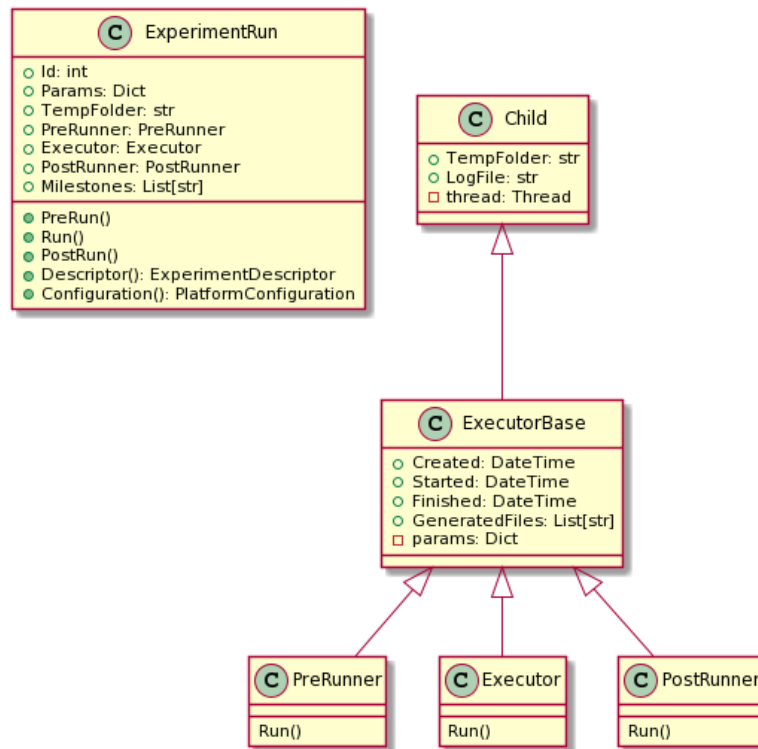


Figure 7 Main classes on the Experiment life-cycle implementation

Please note that in this section the following convention will be followed:

- Class names and their instances are presented in *Italics* style.
- Variable names and methods are presented in **bold** style.

4.1.1. The ExperimentRun class

The instances of the *ExperimentRun* class are responsible for storing the current information about an experiment execution. It also contains references to the *PreRunner*, *Executor*, and *PostRunner* instances, which are the entities that handle the execution of each independent stage of the experiment.

When an experiment execution is received, the *Scheduler* will create a new instance of the *ExperimentRun* class, identified by a unique integer *Id*. As part of the parameters required for the creation of the *ExperimentRun*, it is necessary to provide a dictionary that contains all the configuration values and variables required during the execution of the experiments. This

dictionary will be known as the **Params** of the experiment, and will be shared with the *PreRunner*, *Executor* and *PostRunner* instances for communication within the different stages.

Two important values are always contained in the **Params** dictionary:

- The **Descriptor** (the Experiment Descriptor) contains all the information about the experiment execution, including the test cases to run, the equipment to use among others.
- The **Configuration** is the Platform Specific Configuration generated by the Composer. It is created by the Composer using the Experiment Descriptor as input.

These two values are available for inspection during the experiment execution, and drive the actions performed by the different Executors. The *ExperimentRun* class provides methods for starting each stage independently, however, the logic for starting the correct stage and to transition from one to the next resides in the Experiment Queue, which is handled by the *Scheduler*. Likewise, the execution of each stage is delegated to the different Executors.

Differences from Release A:

- *ExperimentRun* now contains a list of **Milestones**, which are used to mark the execution of certain actions during distributed experiments, as well as information regarding the connection with the remote side during such experiments (not pictured in Figure 7).

4.1.2. The ExecutorBase and Child classes

The minimal logic for executing code in parallel threads is contained in the *Child* class. This class includes all the functionality for managing a separate thread where it's possible to run specific methods, as well as the logic for handling the creation and destruction of temporary folders (so that every thread has its own location on the disk where they can store intermediate results) and log files.

ExecutorBase, that extends the *Child* class, provides the extra functionality that is common to all the Executors (*PreRunner*, *Executor* and *PostRunner*). This includes information about when the Executor was created, when it started and finished its execution, and the list of messages that have been generated. These messages are separated from the full logs and provide a fast way for tracking the progress of the execution. For example, a new message will be generated when the Executor starts processing a new *Task*.

All the executors run a series of Tasks. In the case of the *PreRunner* and *PostRunner*, this list is static and common to all experiments. The Tasks performed by the *Executor* are generated by the *Composer*, depending on the test cases and UEs selected in the experiment, and are available as the **RunTasks** variable contained in the Platform Specific Configuration.

Differences from Release A:

- *ExecutorBase* now saves the location of every file generated by an executor, which can later be collected and saved along with other experiment execution information.

4.1.3. Tasks

In the context of the Experiment Life Cycle Manager, a *Task* is the minimal action that must be performed in order to run an experiment, and may involve delegating the execution to an external entity. For example, a *Task* may be used in order to execute a TAP TestPlan that will perform some measurements, running a script through the command line for configuring some equipment, or sending values to an InfluxDB database. Tasks may run for as long as needed, but only one *Task* can run on a given executor at a time.

Like in the *ExperimentRun* class, Tasks receive a dictionary of parameters that further refine their behavior, and it's possible to conditionally run a task depending on the values contained in this dictionary.

The following Task types are available for use during the definition of the implementation of a certain test case, or for the configuration of certain UEs:

- Run.CliExecute: Executes a script or command through the command line.
- Run.CompressFiles: Generates a Zip file containing all of the specified files and folders.
- Run.CsvToInflux: Parses the contents of a CSV file, and sends the values to the configured Influx database.
- Run.Delay: Performs a timed wait.
- Run.Message: Adds a message to the execution log, using the configured severity level.
- Run.Publish: Saves one or many new (key, value) pairs, making them available for variable expansion.
- Run.PublishFromFile and Run.PublishFromPreviousTaskLog: Reads the contents of a file or the previous task's log, looking for lines that match a specific regular expression pattern. The groups found on the last match are published for variable expansion.
- Run.SingleSliceCreationTime: Retrieves the Slice Creation Time metrics for the network slice deployed during the Pre-Run stage from the Slice Manager, and sends these values to the configured InfluxDB database.
- Run.SliceCreationTime: Provides a ready to use implementation of the Service Creation Time test case that can be seen in Section 4.2.1.6 of Deliverable D6.2 [10]. The task performs several cycles of slice creation and deletion, recording the Slice Creation Time values of each iteration.
- Run.TapExecute: Executes a TAP test plan, with the possibility of configuring external parameters.

Additionally, the ELCM can be extended with new *Task* types, depending on the particular needs of a platform. The process for creating new *Task* types is documented in Section 4.1.1.2 of Deliverable D5.3 [9].

Differences from Release A:

- Release A only provided the Message, CliExecute and TapExecute tasks.

4.2. Composer

The *Composer* is the entity that generates the Platform Specific Configuration (the set of configuration values and Tasks that need to be run in order to perform an experiment execution). This configuration (an instance of the *PlatformConfiguration* class) is generated by using the Experiment Descriptor received as part of the execution request as well as the contents of the Facility Registry.

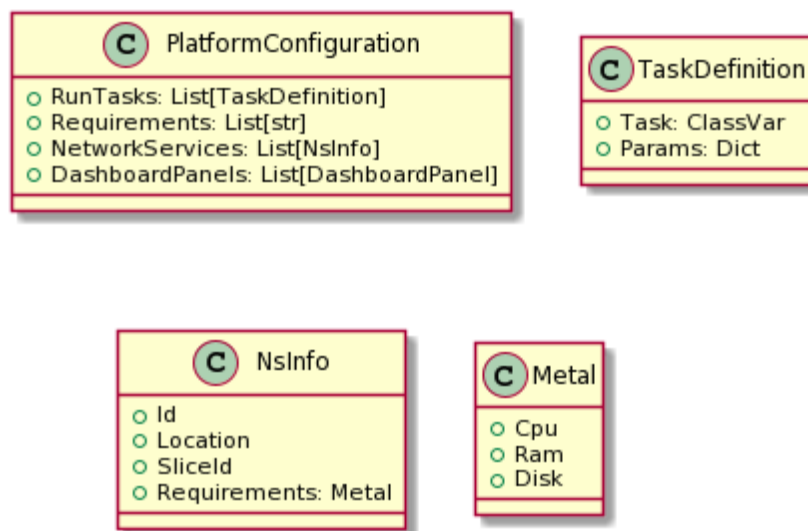


Figure 8 PlatformConfiguration and auxiliary classes

Tasks are saved in the form of a *TaskDefinition*, which is the minimal information required for creating an instance of the actual *Task* that will be run. The Task type is saved as a ClassVar Python variable, which is not instantiated until the *Task* execution is about to start. This is due to the existence of the variable expansion procedure: Since the parameters for the *Task* may include values that are not known before all the previous Tasks have finished their execution, it is better to delay the instantiation until the *Task* is really needed, and setting their parameters only once.

Along with the list of Tasks to be run, the configuration saves information about the required resources and network services. Initially, only the network service **Id** and **Location** are set, while the **SliceId** and **Requirements** values are obtained during the deployment step of the Pre-Run stage. The requirements are requested to the Slice Manager during the feasibility check, and the **SliceId** is defined once the network slice has been instantiated.

Finally, the Platform Configuration instance also includes information about the list of Grafana panels that will later be used by the Dashboard Generator.

4.2.1. The composition process

The process followed by the *Composer* in order to generate the Platform Specific Configuration is as follows:

- As part of the Experiment Descriptor, the *Composer* receives:
 - o A list of test cases and UEs.

- A list of network service ids and their deployment location.
- For each of the UEs selected, the *Composer* will add to a temporary list the information of all the tasks that belong to that particular UE, as a *TaskDefinition* instance. For example, for an Experiment Descriptor that contains the 'TestUE' presented in Listing 1, the composer adds two 'Message' tasks with orders 1 and 10.
- For each Test Case, the *Composer* will add their actions to the same list. Following the example in Listing 2, the Composer adds a 'SingleSliceCreationTime' task with order 5. Additionally, if the test case contains the definition of dashboard panels, these are used to generate *DashboardPanel* instances and added to the **DashboardPanels** list, which will be used during the Grafana dashboard generation presented in Section 4.5.
- The *Composer* generates the final list of Tasks by sorting the temporary list following the 'Order' values. This process generates the contents of the **RunTasks** list, which is used during the Run stage of the experiment execution.
If multiple tasks share the same order, they will be run with undetermined precedence. For this reason, it's important to define tasks that configure and initialize the equipment with low order values, measurement tasks in the middle and task that finalize the processes with higher ones.
- The *Composer* traverses the final list of Tasks, and adds any requirements to the **Requirements** list. In our example, this list would contain 'UE1'.
- Each network service included in the Experiment Descriptor is used to generate an *NsInfo* instance, and added to the **NetworkServices** list. This information, along with the **Slice** and **Scenario** values of the Experiment Descriptor, will be used during the Network Services deployment (Section 4.3).

Differences from Release A:

- The Platform Specific Configuration now contains information about the requirements of the experiment, both physical resources of the platform and for the deployment of network services.

4.3. Experiment execution workflow

The general workflow during the execution of experiments is common to all of the experiment types supported by the ELCM. However, there are certain differences in the followed approach, which are summarized in this section.

4.3.1. Standard and custom experiments

All experiments follow the general workflow of Standard experiments, which consist in the execution of a common Pre-Run stage, the execution of a test case(s) and UEs specific set of Tasks, and finally a common Post-Run stage.

Custom experiments also allow the definition of certain configuration parameters, which experimenters can customize, creating a test case execution that is more fine-tuned to their needs. This functionality makes use of the variable expansion capabilities provided by the

ELCM, mapping certain values to the contents of the 'Parameters' field included in the Experiment Descriptor at run time.

4.3.2. MONROE experiments

MONROE [11] experiments delegate the execution of the experiment to a physical or virtual MONROE node available in the platform, making use of the MONROE TAP plugin and TAP agent developed as part of the 5Genesis project. From the point of view of the ELCM, a special kind of experiment acts as a wrapper.

The execution of this kind of experiment consists of the execution of a single Task, which performs the execution of a pre-created TAP test plan that drives the communication with the MONROE node and defines the required meta-data for integrating with other 5Genesis components. The collection of steps contained in this TAP test plan can be seen in Figure 9.

The required information, such as the MONROE experiment to run and the experiment parameters are taken from the **Application** and **Parameters** fields on the Experiment Descriptor, and are set using external parameters when performing the execution of the TAP test plan.

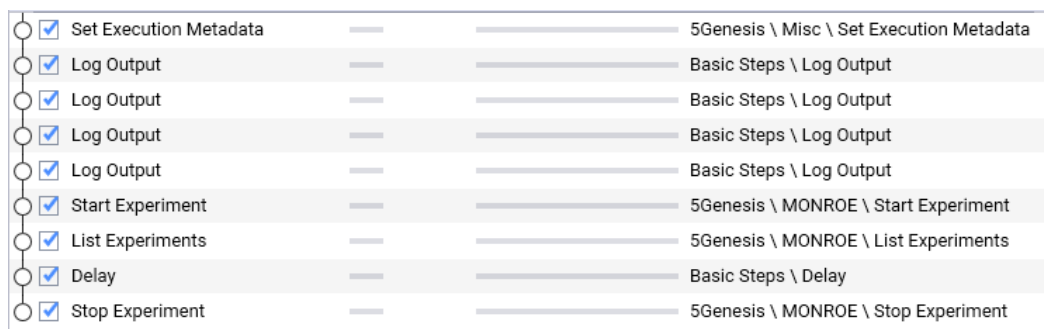


Figure 9 TAP test steps for MONROE experiments

Since MONROE nodes can execute a large variety of different test cases, it is not possible to perform the creation of customized dashboards for every experiment type. For this reason, the automatic creation of dashboards documented in Section 4.5.1 was developed.

4.3.3. Distributed experiments

Distributed experiments are handled using the same workflow as standard experiments. However, two coordination steps are performed at the start and end of the experiment execution. Additionally, new Task types are available for use during the Run stage, which facilitate the coordination and information exchange between the involved platforms. These additional procedures make use of the East/West interface defined in Section 5.2.

The following Task types are available during a distributed experiment execution:

- Remote.WaitForMilestone: Halts the execution of additional tasks until the remote side specifies that a certain milestone has been reached.
- Remote.GetValue: Halts the execution of additional tasks until a certain value can be obtained from the remote side. When received, the value will be published internally and available for variable expansion.

The creation of a distributed experiment is a collaborative activity between the two platforms involved in the execution of the experiment. Each platform is responsible for the definition of their set of actions, as only they have the required knowledge on the usage of their equipment, but must agree with the other platform's administrators about any necessary coordination and information exchange that is required in order to successfully execute the test case.

The general workflow during a distributed experiment is as follows:

- The Dispatcher of one of the platforms (the Main platform) receives a distributed experiment execution request, either from the Portal or through the Open APIs.
- The Dispatcher performs the initial coordination, contacting the ELCM of its own platform and the Dispatcher of the remote platform (the Secondary platform).
- Each side performs the execution of their tasks as normal, unless they reach a point where they must coordinate:
 - o If one of the platforms must wait until the remote side has performed some actions:
 - The waiting platform can use the *WaitForMilestone* task.
 - The other platform can indicate that the actions have been performed using the *AddMilestone* task.
 - o If one of the platforms requires certain information from the remote side:
 - The querying platform can use the *GetValue* task.
 - The other platform can set the value requested using any of the *Publish*, *PublishFromFile* and *PublishFromPreviousTaskLog* tasks.
- Once both platforms execute all their tasks, the Main platform requests all the generated files and results from the Secondary platform, so that they are saved along with the ones generated by the Main platform and available to the experimenter.

Differences from Release A:

- Only Standard experiments were supported during release A.

4.4. Network Services deployment

The deployment of network services and the configuration of the network slice to be used during the experiment is performed in the Pre-Run stage of the experiment execution. This slice remains active during the Run stage and is automatically decommissioned when the experiment reaches the Post-Run stage.

The ELCM delegates the creation and decommissioning of network slices to the Katana Slice Manager [6] by using the corresponding southbound interface. The ELCM uses the information received in Experiment Descriptor along with the definitions contained in the Platform Registry in order to create a NEST payload that is sent to the Slice Manager upon creation of the network slice. The Slice Manager will respond with a network slice id, which is saved by the ELCM and available to the experiment's Tasks. Finally, the ELCM requests the decommissioning of the network slice, using the received identifier as reference.

For the creation of the NEST payload, the ELCM combines three different pieces of information: A Base slice descriptor identifier, a Scenario identifier and a list of network service identifiers along with the location where they will be deployed.

The Base slice is a reference to a slice description available in the Katana Slice Manager. These descriptions define the default values for a certain kind of network slice. A scenario is a collection of configuration values that are used to further customize the behavior of a deployed slice, and are used to overwrite some of the values defined in the Base slice descriptor. These values are defined as YAML files contained in the `Scenarios` folder. Finally, the network service ids are references to Network Services that have been onboarded in the repository provided by the Open API.

Using this information, the ELCM creates a NEST payload, which is composed by three main parts:

- A reference to a base slice descriptor, from those available in the Katana Slice Manager.
- A collection of values that are to be overridden from the base slice descriptor, taken from the selected Scenario.
- A possibly empty list of references to Network Services that are to be included as part of the Network Slice.

The format of a generated NEST payload can be seen in Listing 3.

```
{
  "base_slice_descriptor": {
    "base_slice_des_id": "<Base Slice Descriptor reference>",
    // Values from the selected Scenario are included here
  },
  "service_descriptor": {
    "ns_list": [
      {
        "nsd-id": "<Network Service ID>",
        "placement": "<Network Service Location>",
      } //, [...]
    ]
  }
}
```

Listing 3 NEST payload format

4.5. Grafana dashboard generation

Platform administrators can define a set of Grafana dashboard panels for each of the test cases supported in the facility. The purpose of these panels is to display a subset of the most important or representative values measured by an experiment execution, and is independent to the in-depth capabilities provided by the 5Genesis Analytics module.

The parameters in Table 2 can be used to specify the contents and visualization format of each panel.

	Parameter	Type	Description
	Type	String	Panel type. Available values are 'SingleStat' (gauges, numeric values) and 'Graph' (time series graph)
	Name (Optional)	String	Name of the panel, if not set a default name will be generated from the Measurement and Field values
	Measurement	String	Measurement name

	Field	String	Field name
	Unit (Optional)	String	Results unit
	Size	List[int]	Size of the panel (height, width)
	Position	List[int]	Panel position in the dashboard (x, y)
	Color	List[int]	Graph or text colors. For Gauges this is a list of 3 colors, otherwise a single value.
Graph	Lines	Boolean	True to display as line graph, False to display as bars
	Percentage	Boolean	Whether the graph represents a percentage or not
	Interval (Optional)	String	Time interval of the graph. If not set, the default Grafana interval will be used.
	Dots	Boolean	Display dots along with the graph or bar
SingleStat	Gauge	Boolean	True to display as a gauge, false to display as a single numeric value
	MaxValue (Optional)	Float	Maximum expected value of the gauge, 100 if not set
	MinValue (Optional)	Float	Minimum expected value of the gauge, 0 if not set

Table 2 Available parameters for Grafana panel definition.

By using the information contained in the Dashboard section, the Dashboard generator will automatically create a JSON description of the complete Dashboard. This description is then sent as payload to the appropriate endpoint on the Grafana Dashboard REST API, in order to trigger the generation of the final dashboard.

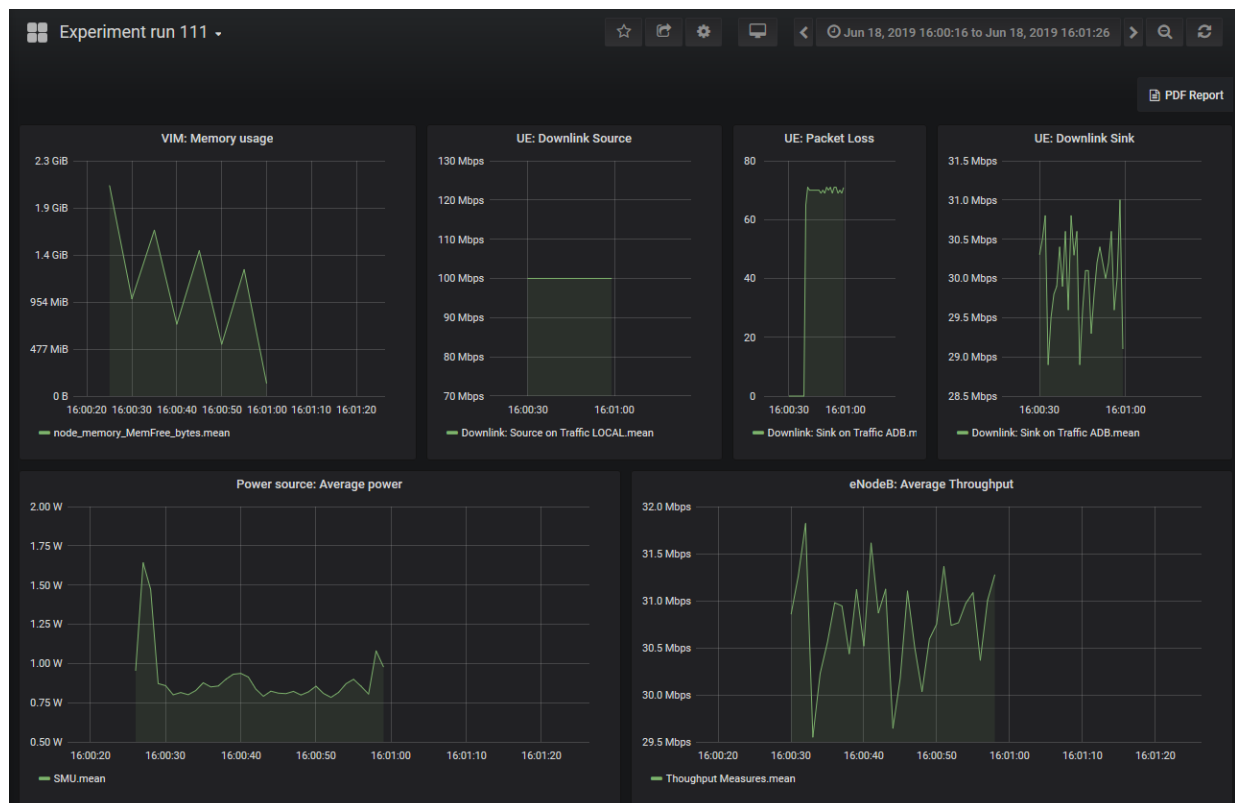


Figure 10 Throughput Grafana dashboard

4.5.1. Grafana dashboard auto-generation

The ELCM is able to generate additional panels if certain values appear on the names of the generated TAP results. For this feature to work, an additional result listener (AutoGraph) must be enabled in TAP.

This functionality is especially useful for the execution of MONROE experiments (Section 4.3.2), which already require the usage of TAP, and where the creation of customized dashboards is impossible given the variability of such experiments.

This feature works as follows:

- During the experiment execution within TAP, the AutoGraph result listener inspects the generated results for names that include information about panel generation.
- At testplan end, the result listener generates a message that contains the description of each panel to generate.
- If the test case includes a dashboard definition, the ELCM will generate the panels described in it first.
- The logs generated during the experiment execution will be parsed, looking for messages generated by the AutoGraph result listener.
- For each message detected, a new panel will be generated after the ones described in the test case.

In order to be detected by the result listener, the result name must have the following format:

```
"<Result name> [[<Panel type>]]" or "<Result name> [[<Panel type>:<Unit>]]"
```

Where:

- <Result name> is the name of the panel
- <Panel type> is one of [``Si``, ``Ga``, ``Li``, ``Ba``]:
 - o ``Si`` stands for 'Single'
 - o ``Ga`` stands for 'Gauge'
 - o ``Li`` stands for 'Lines'
 - o ``Ba`` stands for 'Bars'
- If present, 'Unit' is the unit of the results, which must be a valid Grafana value.

Differences from Release A:

- The 'Colors' and 'Dots' parameters did not exist during Release A.
- The auto-generation procedure is exclusive to Release B.

5. ELCM INTERFACES

5.1. Northbound interfaces

5.1.1. Open APIs

The ELCM provides the implementation of several endpoints of the Open APIs, in particular those related to experiment handling and metadata (Table 3) and facility description (Table 4).

It should be noted that these endpoints are not exposed directly to the experimenter, and are instead provided via the Dispatcher component, which handles the user authentication before any possible response and provides a unified interface through the Open APIs.

Endpoint	Method		
/api/v0/run	POST	Payload	Experiment descriptor in JSON format. See Figure 3.
		Response	<code>{"ExecutionId": int}</code>
		Notes	Exposed as /elcm/api/v0/run on the Open APIs. The Dispatcher verifies the payload before redirecting to the ELCM
/execution/<id>/cancel	GET	Payload	None
		Response	None
		Notes	Exposed as /elcm/execution/<id>/cancel on the Open APIs.
/execution/<id>/descriptor	GET	Payload	None
		Response	Experiment descriptor in JSON format. See Listing 4.
		Notes	Exposed as /elcm/execution/<id>/descriptor on the Open APIs.
/execution/<id>/logs	GET	Payload	None
		Response	<pre>{ "Status": str "PreRun":<LogInfo> "Executor":<LogInfo> "PostRun":<PostRun> }</pre>
		Notes	Exposed as /elcm/execution/<id>/logs on the Open APIs. Returns the logs of the three executors of the selected experiment run, along with their severity levels (see Listing 5).

Table 3 Experiment related endpoints exposed through the Open APIs

Endpoint	Method		
/facility/baseSliceDescriptors	GET	Payload	None
		Response	{ "SliceDescriptors": List[str] }
		Notes	Exposed as /elcm/facility/baseSliceDescriptors on the Open APIs. The actual list of base slice descriptors is retrieved from the Katana Slice Manager.
/facility/testcases	GET	Payload	None
		Response	{ "TestCases": [{ "Distributed": boolean, "Name": str, "Parameters": Array[str], "PrivateCustom": Array[str], "PublicCustom": boolean, "Standard": boolean }] }
		Notes	Exposed as /elcm/facility/testcases on the Open APIs, however, the Dispatcher filters the returned information based on the user credentials. Users of the Open APIs receive a list of test cases available to their account.
/facility/ues	GET	Payload	None
		Response	{ "UEs": Array[str] }
		Notes	Exposed as /elcm/facility/ues on the Open APIs.
/facility/scenarios	GET	Payload	None
		Response	{ "Scenarios": Array[str] }
		Notes	Exposed as /elcm/facility/scenarios on the Open APIs.

Table 4 Facility related endpoints exposed through the Open APIs

The following figures show the format of more complex data models received or returned by the interface.

```
{
  "Application": str, // May be null
  "Automated": bool,
  "ExclusiveExecution": bool,
  "ExperimentType": str,
  "Extra": Object[str, str] // May be empty,
  "NSs": Array[Array[str]], // (nsd id, vim location) pairs. May be empty,
  "Parameters": Object[str, str], (may be empty)
  "Remote": str, // May be null
  "RemoteDescriptor": <Descriptor>, // Same format as an Experiment
                                     // Descriptor, but without the
                                     // "RemoteDescriptor" field. May be
                                     // null
  "ReservationTime": int, // May be null
  "Scenario": str, // May be null
}
```

```

"Slice": str, // May be null
"TestCases": Array[str],
"UEs": Array[str], // May be empty
"Version": str
}

```

Listing 4 Experiment descriptor format

```

{
  "Count": {
    "Debug": int,
    "Info": int,
    "Warning": int,
    "Error": int,
    "Critical": int
  },
  "Log": Array[Array[str]] // List of (severity, message) pairs
}

```

Listing 5 LogInfo format

5.1.2. 5Genesis Portal

Though most of the communication between the Portal and the ELCM is performed via the Open APIs in Release B, as opposed to the direct communication existing during Release A, the ELCM still sends the real-time updates of a particular experiment execution directly to the Portal.

The information transmitted includes the current stage of the execution, the percentage of completion and a descriptive message about the current status. Given that this information changes at a fast rate, it was decided not to make this information available through the Open APIs and it is only used to improve the user experience in the 5Genesis Portal.

5.2. East/West interface

The East/West interface is used for communication between two different ELCM instances during a distributed experiment execution. It provides the endpoints needed for coordination and information exchange during the execution of the experiment, as well as for the retrieval of files and results from one of the platforms to the other, once the execution ends. Finally, the interface exposes an additional endpoint that is used by the Dispatcher during the initial coordination before the execution starts.

Endpoint	Method		
/distributed/<id>/peerDetails	POST	Payload	{"execution_id": int}
		Response	{ "success": boolean, "message": str}
		Notes	Used by the Dispatcher during the initial coordination phase. Sets the execution ID of the remote side of the experiment.
/distributed/<id>/status	GET	Payload	None
		Response	{ "success": boolean, "message": str, "status": str, }

			<code>"milestones": Array[str]</code> <code>}</code>
		Notes	Used to retrieve the execution status and list of milestones reached by a distributed experiment.
<code>/distributed/<id>/values</code> <code>/distributed/<id>/values/<name></code>	GET	Payload	None
		Response	<code>{</code> <code> "success": boolean,</code> <code> "message": str},</code> <code> // Either</code> <code> "values": Object[str, str],</code> <code> // or</code> <code> "value": str</code> <code>}</code>
		Notes	Returns a complete dictionary of values if no <code><name></code> was specified, or the value of that variable if it was specified.
<code>/distributed/<id>/files</code>	GET	Payload	None
		Response	The logs in txt format and any file generated by the execution in a single zip file, as a binary download.
		Notes	This file is retrieved from the Secondary platform upon execution end, and is saved along with the files from the Main platform in the Experiment Registry.
<code>/distributed/<id>/results</code>	GET	Payload	None
		Response	A JSON payload containing all the results (see Figure 16).
		Notes	These results are retrieved from the Secondary platform upon execution end, and are sent to the Results Repository of the Main platform.

Table 5 East/West interface endpoints.

The `/distributed/<id>/results` endpoint is used for transferring the contents of the Secondary platform's Results Repository to the Main platform, so that they are available at a single point for the experimenter. The chosen format tries to include all the information available in an InfluxDb, but removing a large amount of redundant information in order to reduce the size of the transferred payload.

Along with the 'success' and 'message' fields that are common to all of the East/West interface responses, the payload includes a list of all the returned measurement names, and a dictionary that contains the data from all the measurements. This data is separated in (possibly) several pieces, where each piece shares the same set of tags. This eliminates the need of sending the tag values along with the fields for every point.

Along with the tags, each piece contains a sorted list of the field's name. Finally, the 'points' field contains a list of lists, where each internal list is formed by the timestamp of the point (at the first place), followed the field values in the same order as in the header.

```
{
  "success": boolean,
  "message": str,
  "measurements": Array[str], // List of returned measurement names
  "data": Object[str, <Measurements>]
}
```

Where

```
<Measurements> = [
  {
    "tags": Object[str, str]
    "header": Array[str] // Sorted list of field names
    "points": Array[Array[timestamp, value1, value2, ...]]
  }, ...
]
```

Listing 6 East/West results payload

When the payload is received, the Main platform replaces some of the tags (such as the execution id) with the values of the Main platform and uploads the received points to the InfluxDb database. All the received measurement names are prepended with "Remote_" in order to differentiate them and avoid overwriting values generated in the Main platform.

5.3. Southbound interfaces

5.3.1. Katana Slice Manager

The ELCM is able to communicate with the Katana Slice Manager [6] by sending requests to the REST API exposed by this entity. The ELCM makes use of several endpoints, as listed in Table 6, in particular those related to the management of network slices, resource usage and available network service descriptors and base slice descriptors:

Endpoint	Method	Notes
/api/slice	POST	Requests the creation of a new slice, using a NEST descriptor as payload.
/api/slice/<id>	GET	Retrieves information about the status of an slice
	DELETE	Requests decommission of an existing slice.
/api/slice/<id>/time	GET	Retrieves information about the deployment time of an existing network slice.
/api/resources	GET	Retrieves the current computation, memory and storage usage on the VIMs in the lower layers, as well as the maximum available.
/api/nslist	GET	Retrieves information about all the available network service descriptors, in particular the computation, memory and storage requirements.
/api/base_slice_des	GET	Retrieves the list of all available base slice descriptors defined in the Katana Slice Manager.

Table 6 Slice Manager endpoints used by the ELCM

5.3.2. Network management system (NMS) and platform infrastructure

The interaction with heterogeneous components that are part of the Network Management System and other elements in the platform infrastructure is performed by making use of any of the several Tasks available in the ELCM. In particular Run.TapExecute, which allows the execution of TAP test plans, and Run.CliExecute, which can be used for executing arbitrary commands and scripts using the command line interface.

5.3.2.1. TAP (OpenTAP) as execution environment

TAP [12], or OpenTAP can be considered an execution environment, in the same way as shell scripting, but the number of additional features included in TAP (such as the result handling capabilities or the ability to abstract different components of a Platform as Instruments) make it ideal for certain automation tasks that are in line with the requirements of the 5Genesis Platforms. For this reason, helper TAP plugins have been developed in the context of the 5Genesis project, which include several test steps and result listeners that improve the integration of TAP with other components of the facility, such as the Analytics Module. These TAP plugins include:

- A set of test steps ('Set Execution Id' and 'Set Execution Metadata') that can be used for setting the value of several tags, which are required for integrating with the Analytics Module.
- An additional test step ('Mark Start of Iteration') which eases the implementation of test case iterations.
- A TAP result listener that is able to send the generated results in a format that is compatible with the Analytics Module. This result listener sends tagged results to the InfluxDb [13] database that acts as Results Repository. For details about this result listener please refer to section 5.3.3.2. of this deliverable.
- A CSV result listener which includes the same capabilities as the InfluxDb result listener.
- Instruments and steps for executing commands on remote machines using SSH, and for sending/retrieving files through SCP.

(a) Integration of TAP test plans

Figure 11 shows an example testplan that includes all current the requirements for usage on 5Genesis facilities. The first step ("Set Execution ID") will define the global execution identifier for the results. The actual value can be set using the "ExecutionID" external parameter, which means that the ELCM can modify this parameter with the correct value during the experiment execution.

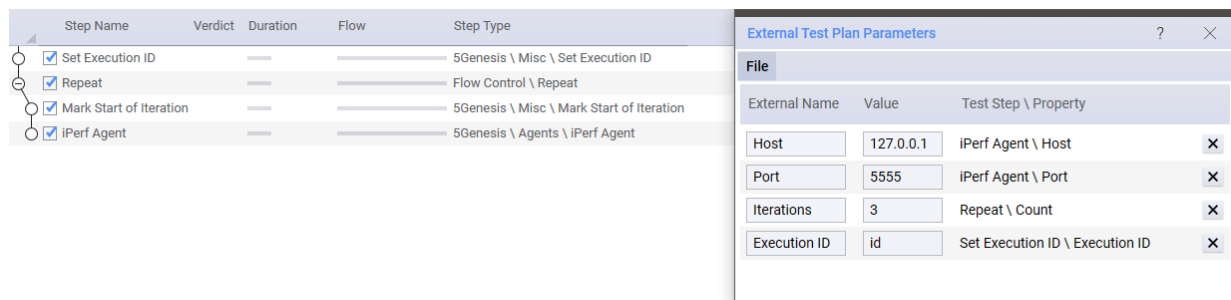


Figure 11 Example TestPlan and external parameters for ELCM

The second is a default step included in TAP, however, the number of repetitions of the loop has been externalized, so that it can be modified easily with the correct number of iterations.

“Mark Start of Iteration” is a custom step that keeps track of the current iteration number and configures any compatible result listener (in particular the InfluxDb result listener) so that every generated result will be tagged with the correct iteration number. The iteration value will increase automatically every time this step is executed inside a loop.

The fourth step (iPerf Agent) is provided as an example of the actions that can be performed during an iteration. In this case the step will use the remote iPerf agent detailed in Deliverable D3.5 [7] to start an iPerf client instance that will run for some time. Once this period finishes the step will recover every result generated and publish them so that they are sent to the InfluxDb database, correctly tagged, by the InfluxDb result listener. The IP address and port of the iPerf server have also been defined as external parameters (“Host” and “Port”) so that they can easily be customized by the ELCM or a user.

Using the Run.TapExecute task it is possible to perform the execution of this test plan as part of an experiment coordinated by the ELCM. The configuration of this Task includes the location of the test plan file to execute and the value of every external parameter to be configured. In order to integrate with the platform, every test plan executed as part of a 5Genesis experiment requires a configurable Execution ID external parameter, which will be used for identifying the experiment results.

(b) SSH TAP Plugin

SSH can be used as an easy integration path for equipment that supports this protocol, giving platform administrators the possibility of remotely controlling the equipment or transferring files from and to the device.

The SSH TAP Plugin comprises a TAP Instrument and three test steps. The instrument contains all the configuration values of the machine that will be controlled using SSH (Figure 12).

SshInstrument	
▼ Connection	
Host	192.168.1.10
Port	22
▼ Credentials	
User	user
Password/Passphrase	<input checked="" type="checkbox"/> ●●●●●●●●●●
Private Key	<input type="checkbox"/> ...

Figure 12 SSH Instrument configuration

The SSH instrument is able to connect to a remote host supporting username and password, or private key based authentication with an optional passphrase.

The following test steps are included in the plugin:

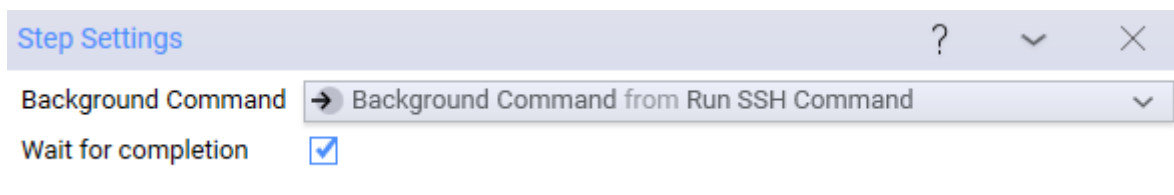
- Run SSH Command: This step is able to execute a command through SSH in the configured instrument. The step can execute this command synchronously (waiting for the command or script to end) or in the background. The step can also be configured to run the command as administrator (if the user has the required privileges in the target machine). The available settings for this step can be seen in Figure 13.

▼ Instrument	
Instrument	SSH
▼ Command	
Command	uname -a
Run in background	<input type="checkbox"/>
Timeout	<input type="checkbox"/> 60 Seconds
Run as SU	<input type="checkbox"/>
▼ Output	
Log output	<input type="checkbox"/>
Log errors	<input checked="" type="checkbox"/>
▼ Step Verdict	
Error verdict	<input checked="" type="checkbox"/> Error
Expected exit code	0

Figure 13 Run SSH Command step settings

- Retrieve Background SSH Command: This step can be used in order to synchronize the test plan execution with an SSH command that was started in the background.

This step can be configured so that the command is immediately stopped if it has not been completed, or to continue waiting. If a Timeout value has been configured in the “Run SSH Command”, it will be honored.



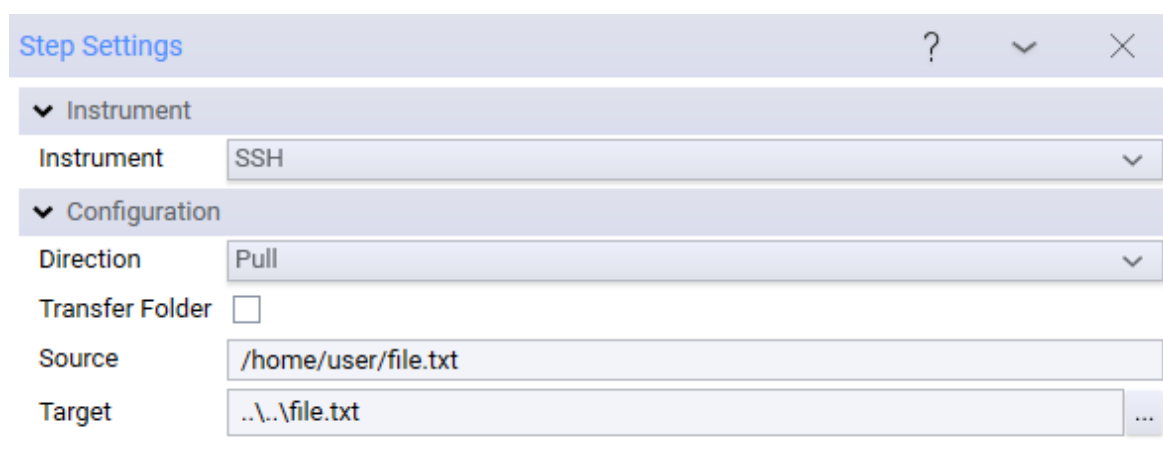
Step Settings

Background Command → Background Command from Run SSH Command

Wait for completion ☒

Figure 14 Retrieve Background SSH Command step settings

- SCP Transfer: This step can be used in order to transfer files and directories from and to the remote machine using the SCP protocol.



Step Settings

Instrument SSH

Configuration

Direction Pull

Transfer Folder

Source /home/user/file.txt

Target ..\..\file.txt

Figure 15 SCP Transfer step settings

5.3.2.2. Generic script execution and additional integration

In order to support the management of generic components, the ELCM is able to handle the execution of generic command line interface scripts and commands. Using this capability, Platform Administrators can make use of already existing programs, either developed in house or provided by the component’s manufacturer, or create new ways of interfacing with the equipment if required.

In addition to using complex scripts and programs, the ‘Run.CliExecute’ task can be used for running isolated commands. This can be useful, for example, for using existing command line utilities such as ‘curl’ or ‘Invoke-RestMethod’. If the component to control exposes a REST API, these utilities can be used to send orders to the component or retrieve information.

For platform equipment that is commonly used, it is possible to create new ELCM Tasks that are fine-tuned for handling that specific equipment or kind of interface. Information about the development process for new tasks can be seen in Deliverable D5.3 [9].

5.3.3. Analytics module and Results Registry

Due to the support of several heterogeneous methods for performing the actions required for the execution of a test case in the 5Genesis facilities, the ELCM does not provide a single way of managing the retrieval of the results and the integration with the Analytics module. Instead, several guidelines exist (regarding the format of the results and the contained information), along with a set of helper utilities that aim to automate and ease the integration in each platform.

In general, any tool can send results in a compatible way with the 5Genesis facilities, provided that:

- The results can be expressed as InfluxDb [13] points. I.e. each result has an associated timestamp and can contain as many simple values (boolean, integer, floating point or string) as needed (but no composed values). Each value is associated with a field name.
- Every value is saved with an associated 'Execution Id' tag, which univocally identifies the experiment execution that generated the result.
- If the experiment consists of multiple different iterations, each result must indicate the iteration, which generated it, using the '_iteration_' tag.

The following is a description of several helper utilities for integrating with the Analytics Module and Results Registry.

5.3.3.1. InfluxDb Helper class and CsvToInflux

The ELCM includes the implementation of a helper class, which is able to send arbitrary results to the InfluxDb instance that acts as Results Registry in the platform. This helper is used by certain ELCM Tasks that are able to send results automatically (such as the 'Run.SliceCreationTime' task), and also provides functionality to the 'Run.CsvToInflux' Task.

The CsvToInflux Task can be used for sending the contents of a CSV file to the InfluxDb database, correctly tagged with all the required metadata for integration with the Analytics Module. Using this Task, platform administrators can make use of any existing equipment that is able, either directly or by using an additional tool, to save measurements or results as CSV files.

5.3.3.2. InfluxDb Result Listener

The InfluxDb result listener for TAP is able to automatically send all the results generated by every TAP test plan to the InfluxDb instance tagged with the required metadata. The Analytics module can then retrieve these results and extract the available KPIs from them.

Additionally, the InfluxDb Result Listener can send a selection (filtered by severity) of the log messages generated during a test plan execution. This might be useful for debugging or for extracting additional information from the results, correlating these values with the events logged by TAP.

All results sent to InfluxDb must include a valid timestamp that corresponds to the moment when the measurement was obtained. By default, the result listener will look for a field called "Timestamp" (case ignored) that should contain the POSIX timestamp (the amount of time

elapsed since the midnight of January 1, 1970), however, in order to support TAP test steps that do not follow this convention, it is possible to define certain rules for obtaining the timestamp from other fields.

Figure 16 InfluxDb result listener settings

These rules can be defined by editing the “DateTime overrides” table in the result listener settings (Figure 16). This table specifies the name of the result where the rule applies, as well as the names of up to 2 fields (columns) that can be used to extract the timestamp. Additionally, two “Format” columns specify the exact format of the timestamp. Examples of some possible values for this table can be seen in Figure 17.

DateTime overrides				
Result Name	Column Name 1	DateTime Format 1	Column Name 2	DateTime Format 2
1 SMU	Time	MM/dd/yyyy HH:mm:ss		
2 Custom_BLE_Scanning	src_timestamp	dd.MM.yyyy...		
3 Example	Date	MM/dd/yyyy	Time	HH.mm.ss.fff

Figure 17 DateTime overrides

The result listener will also tag all the generated results with the correct Execution ID (as specified by the ‘Set Execution ID’ or ‘Set Execution Metadata’ steps), and the iteration number (controlled by the ‘Mark Start of Iteration’ step).

6. TESTING AND VALIDATION

Aside from the full validation process performed as part of Work Package 5, where a complete 5Genesis platform is deployed from the ground up in order to test the functionality and correct integration of all the different components, several additional tests have been performed during the development of the ELCM.

In order to iteratively test the new functionalities as they are created, the development environment includes the pre-release version of other 5Genesis components, such as the Portal, Dispatcher and Slice Manager, which provide fairly realistic conditions where the correct operation of the ELCM can be tested.

During active development the integrated debugger of the Python IDE, which allows setting breakpoints, inspecting the value of each variable and evaluate expressions in real time, is extensively used. Since at this point there is still missing functionality, either in the ELCM itself or on external components, it is common to simulate part of the logic using temporary debug code, that, for example, injects a controlled set of data to the ELCM inputs or replies with the 'correct' output regardless of the conditions. Though not entirely realistic, this practice eases the development process and allows fixing a large amount of issues at a fast rate, meaning that fewer errors are introduced or discovered as new functionality is added to the components.

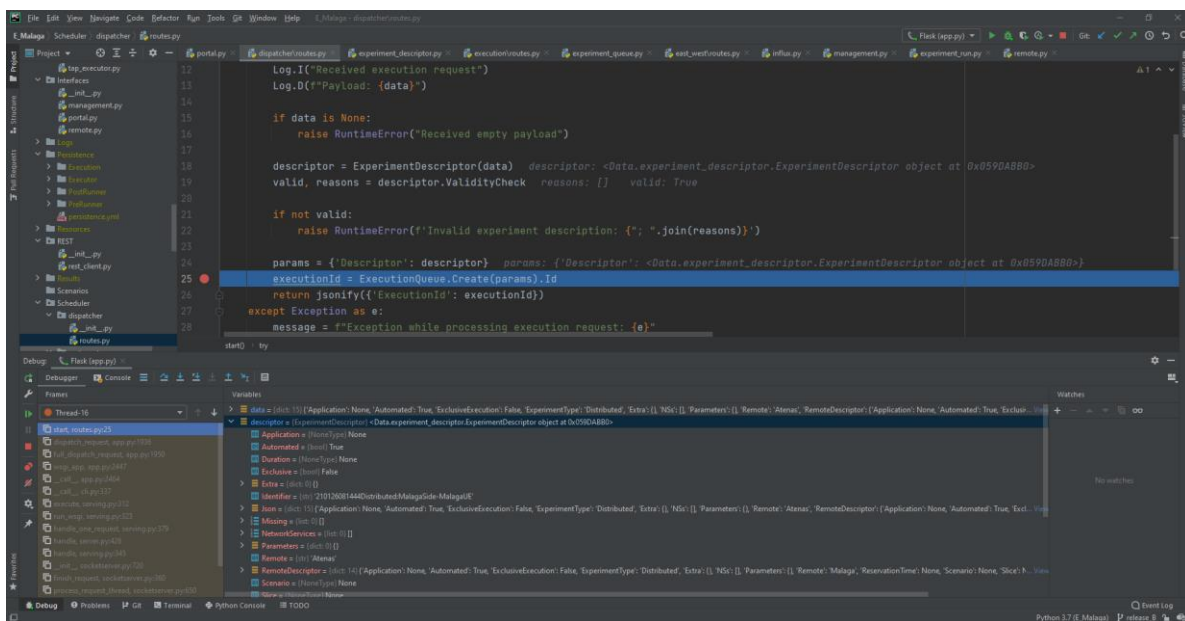


Figure 18 Python debugger

Given that the different 5Genesis components are more deeply coupled than during Release A, an additional (intermediate) integration test was performed in the Málaga platform using an initial set of Release B functionality. For this test, the pre-release versions of most of the 5Genesis components (Portal, Dispatcher, ELCM and Katana Slice Manager) were deployed in a new test environment.

As a result of this intermediate integration, a small number of issues were discovered and fixed. Moreover, several usability improvements were applied to the components, improving both

the installation process and the debugging of issues, paving the way to the final WP5 integration of the components.

In parallel with the testing based on the execution of the code, the ELCM design has also been validated using the model checking technique. Model checking is especially relevant to analyse concurrent systems with many interaction points, in order to locate potential execution sequences that leave to deadlocks or other execution errors. The basis of this method is the creation of an abstract model of the original software that can be exhaustively analysed to generate the execution graph produced by the interleaving of the processes. Such exhaustive analysis can be enriched with specific properties to be checked in all the traces in order to find more specific errors beyond deadlocks or live locks.

The language used to create the model of the ELCM is Promela, while the model checker used to analyse the system was SPIN [15]. The work done in the project, partially reported in [16], consisted in building a Promela model of the entities presented in Figure 2 the definition of a number of specific properties using Temporal Logic, and the verification work. Figure 19 shows the output of the process for a test where the model contains the parallel execution of three experiments. The most relevant results presented in the figure are the number of errors (0), the number of global states analysed (more than 171 M) and the hash factor, which indicates the quality of the analysis (149.9).

```

Bit statespace search for:
    never claim      - (not selected)
    assertion violations +
    acceptance cycles + (fairness disabled)
    invalid end states +

State-vector 484 byte, depth reached 780, errors: 0
1.1497802e+08 states, stored
1.7184312e+08 states, matched
2.8682114e+08 transitions (= stored+matched)
    8 atomic steps

hash factor: 149.419 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):
56141.612 equivalent memory usage for states (stored*(State-vector + overhead))
2048.000 memory used for hash array (-w34)
    0.076 memory used for bit stack
    0.534 memory used for DFS stack (-m10000)
2049.315 total actual memory usage
  
```

Figure 19 ELCM model checking results

The conclusion of the verification work with Promela and SPIN is that the current design of the ELCM is ready to support the execution of concurrent experiments.

7. CONCLUSIONS

The report presented the activities performed during the design and development of the experiment lifecycle manager of the 5GENESIS facility [17]. The ELCM has been developed from the ground up using Python, and uses different interfaces for communicating with specific elements of the platforms. Additionally the ELCM exposes an internal web administration interface developed in Flask. The ELCM is the entity that performs the management, orchestration and execution of Experiments in the 5GENESIS Platforms, and has been developed from the ground up as part of the 5Genesis Open Experimentation Framework.

In this document the reader can also find details about the different interfaces of this entity, namely the northbound interface (exposed by the ELCM), the southbound interfaces (used for controlling the different elements of the 5GENESIS facilities) and the East/West interface (used for communicating two different ELCM instances during a distributed experiment execution).

8. REFERENCES

- [1] Welcome to Python.org [Online], <https://www.python.org/>, Retrieved 09/2019
- [2] Flask [Online], <https://palletsprojects.com/p/flask/>, Retrieved 09/2019
- [3] 5GENESIS Consortium, D2.2 Initial overall facility design and specifications: https://5genesis.eu/wp-content/uploads/2018/12/5GENESIS_D2.2_v1.0.pdf
- [4] 5GENESIS Consortium, D2.3 Initial planning of tests and experimentation: https://5genesis.eu/wp-content/uploads/2019/02/5GENESIS_D2.3_v1.0.pdf
- [5] 5GENESIS Consortium, D2.4 Final report on facility design and experimentation planning: https://5genesis.eu/wp-content/uploads/2020/07/5GENESIS_D2.4_v1.0.pdf
- [6] 5GENESIS Consortium, D3.3 Slice management: https://5genesis.eu/wp-content/uploads/2019/10/5GENESIS_D3.3_v1.0.pdf
- [7] 5GENESIS Consortium, D3.5 Monitoring and analytics: https://5genesis.eu/wp-content/uploads/2019/10/5GENESIS_D3.5_v1.0.pdf
- [8] 5GENESIS Consortium, D3.15 Experiment and Lifecycle Manager: http://5genesis.eu/wp-content/uploads/2019/10/5GENESIS_D3.15_v1.0.pdf
- [9] 5GENESIS Consortium, D5.3 Documentation and supporting material for 5G stakeholders: https://5genesis.eu/wp-content/uploads/2020/07/5GENESIS-D5.3_v1.0.pdf
- [10] 5GENESIS Consortium, D6.2 Trials and experimentation (cycle 2): https://5genesis.eu/wp-content/uploads/2020/08/5GENESIS_D6.2_v1.0_FINAL.pdf
- [11] MONROE Project [Online], <https://github.com/MONROE-PROJECT>
- [12] Test Automation Platform (TAP) [Online], <https://www.keysight.com/en/pc-2873415/test-automation-platform-tap>, Retrieved 09/2019
- [13] InfluxDB: Purpose-Built Open Source Time Series Database [Online], <https://www.influxdata.com/>, Retrieved 09/2019
- [14] Grafana Labs [Online], <https://grafana.com/>, Retrieved 09/2019
- [15] G. Holzmann, SPIN Model Checker, The: Primer and Reference Manual, Addison-Weasley, 2011
- [16] D. Arrebola, MM Gallardo, Modelling and verification of 5GENESIS ELCM with SPIN, BsC Thesis, Univ. Malaga, 2021
- [17] H. Koumaras et al., "5GENESIS: The Genesis of a flexible 5G Facility," 2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), Barcelona, Spain, 2018, pp. 1-6, doi: 10.1109/CAMAD.2018.8514956.