D3.4 ICEI Validation Framework

Date	Author	Comments	Version	Status
06.06.2019	Cristiano Padrin (CINECA)	Initial version	V0.1	Draft
08.07.2019	Cristiano Padrin, Debora Testi (CINECA), Sadaf Alam (CSCS), Andreas Herten (JUELICH)	New version including feedback from all partners	V0.2	Draft
14.10.2019	Cristiano Padrin (CINECA), Sadaf Alam (CSCS)	New version including comments and feedback from all partners	V1.0	Final

The ICEI project has received funding from the European Union's Horizon 2020 research and innovation programme under the grant agreement No 800858.

Contents

Acronyms

AAI	Authentication and Authorization Infrastructure
ACD	Active Data Repositories
Chp.	Chapter
D	Deliverable
FURMS	Fenix User and Resource Management Service
GbE	Gigabit Ethernet
GByte/s	10^9 Byte per second
GiByte	2^30 Byte
HLST	High Level Support Team
HPC	High Performance Computing
IAC	Interactive Computing Services
ICEI	Interactive Computing E-Infrastructure (for the Human Brain Project)
IdP	Identity Provider
KPI	Key Performance Indicator
L2L	Learning To Learn
LTL	Learning To Learn
M	Month
MB/s	Megabyte per second
NETE	External Interconnect
NMH	Neuromorphic Hardware
NREN	National Research and Education Network
NRP	Neurorobotics Platform
PRACE	Partnership for Advanced Computing in Europe
SIM	CPUs where simulation is running
SP	Service Provider
Task team	Representatives of each site involved in the Task as leader or contributor
TByte	10^12 Byte

TBD	To Be Defined
v	Version
VM	Virtual Machine
WP	Work Package

1. Introduction

The **Task T3.6 E-infrastructure validation and testing** will work from **M24** to **M36** on the demonstration of ICEI e-infrastructure capabilities and perform periodic validation tests to ensure that the federated services comply with the use case requirements collected in **Task T3.1 ("Use case requirements")**.
The aim of this document is to define the validation framework that encapsulates technical requirements of use cases and associated workflows. This, in turn, enables the functional and performance requirements of ICEI infrastructure services.
This document refers to the most significant use cases identified in collaboration with the Task T3.1.

2. Validation concept in ICEI

Validation in ICEI means to establish whether the e-infrastructure and its components are suitable for the correct functioning of the use cases. A metric of acceptance has to be defined for each hardware/software component (i.e. FURMS, AAI software components, etc.) on the basis of the use cases.
The objectives of the validation framework are to verify that ICEI infrastructure and federated services allow the expected execution of use cases workflows, and measure the performance of the deployed infrastructure, through a set of tests and benchmarks, and documentation of the results.
To achieve these objectives, a set of functional, performance and regression tests will be defined, implemented and deployed to be sure that the e-infrastructure is still suitable even after any update, change or integration.
It is important to remark the difference between **validation**, **verification** and **monitoring**, in order to avoid possible misunderstandings. The **scope of validation** is to understand if the requirements can be satisfied through implementation and settings of the infrastructure and services or, in other words, if the infrastructure and services allow these actions in order to satisfy the use case requirements. The **scope of the verification** is to check infrastructure and services after that implementation and setting phases have been completed. The **scope of monitoring** is to check regularly the availability and/or uptime of infrastructure and services in order to guarantee the correctness of the use cases execution.

3. Validation approach

3.1 ICEI infrastructure

According to the Table 1 in deliverable D3.1 "Common Technical Specifications" [ICEI-D3.1], the ICEI infrastructure is made up of the following components:

- ➢ Interactive Computing Services;
- ➢ Elastic Scalable Computing Services;
- ➢ Virtual Machine Services;
- ➢ Active Data Repositories;
- ➢ Archival Data Repositories;
- ➢ Data Mover Services (intra-site);
- ➢ Data Transfer Services (inter-site);
- ➢ Data Location Service;
- ➢ Internal Interconnect;
- ➢ External Interconnect;
- ➢ Authentication/Authorization Services;
- ➢ User Support Services.

Because the aim of the validation is to demonstrate the expected functionality of the infrastructure for use cases, components relevant to functional and performance behaviour for workflows will be analysed.

3.2 Validation framework

To define the frameworks, Task T3.6 will cooperate with Task T3.1 to identify a relevant subset of use cases, in order to validate all components at least once.
The first use case will be “Learning to learn” (LTL, use case #3) described in the deliverable D3.6 “Scientific Use Case Requirements Documentation” [ICEI-D3.6]. In the section 22.4 of the cited document, the infrastructure requirements are reported, and it’s easy to see that the use case covers almost all the ICEI components (Active Data Repositories, Data Mover Services and Data Location Service are implicitly covered).
In order to validate each component, each requirement will be analysed and a test will be defined and run.
For each hardware/software component, the contribution of an expert team per site could be needed to define the proper acceptance metrics and sequence of specific commands to be executed in order to validate the components.
The first steps of validation framework applied to the “Learning To Learn” and “Neurorobotics Platform” use cases will be introduced in the Annex 1, Annex 2 and Annex 3.

Table 1: Validation framework

Infrastructure requirements documentation	Each infrastructure use case requirements will be analysed in order to identify all components and their metrics of acceptance.
Test definition	For each identified component a sequence of commands to test it will be defined.
Running test and validation component	Each test defined will be run, and the results will be analysed in order to validate the component tested.
Validation of infrastructure	The sequence of the test defined will be run following the workflow of the analysed use case.
Deploy of manual/automatic procedure	The sequence of the test run will be collected in a validation procedure and deployed.

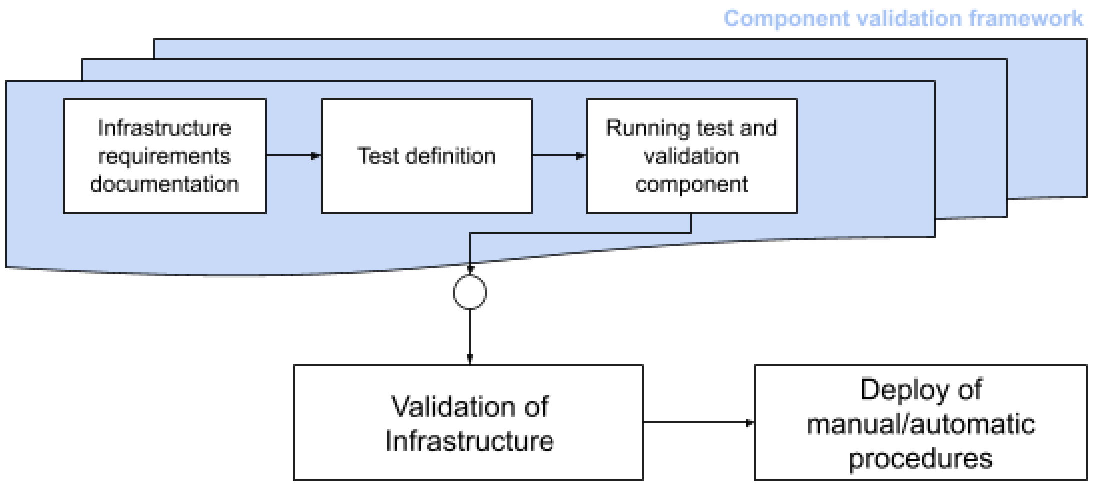
Component validation framework
Infrastructure requirements documentation
Test definition
Running test and validation component
Validation of Infrastructure
Deploy of manual/automatic procedures

Figure 1: Validation framework

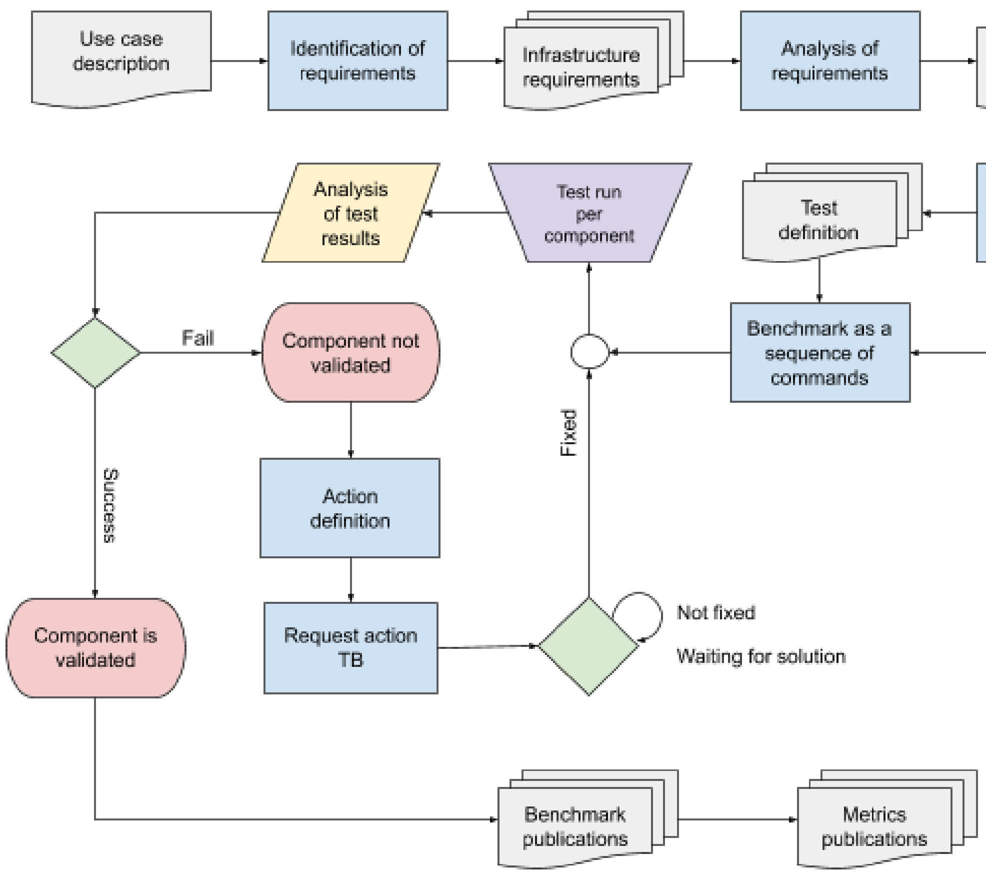

Figure 2: Component validation framework

3.2.1 Infrastructure requirements documentation

Starting from the use case description, the infrastructure requirements will be identified and analysed in order to understand which hardware/software components are required (e.g. hardware component: memory size per node; software component: OpenStack - components like: Compute, Block Storage, Identity Service, Image Service, Telemetry Service, Dashboard, etc), and in order to define the acceptance metric(s) per each component (e.g. acceptance metric for memory size per node: minimum value (in GiB) available per node; acceptance metric for OpenStack components: fixed number of successes in test results for each functionality). The acceptance metrics definition will identify the expected results of validation tests.

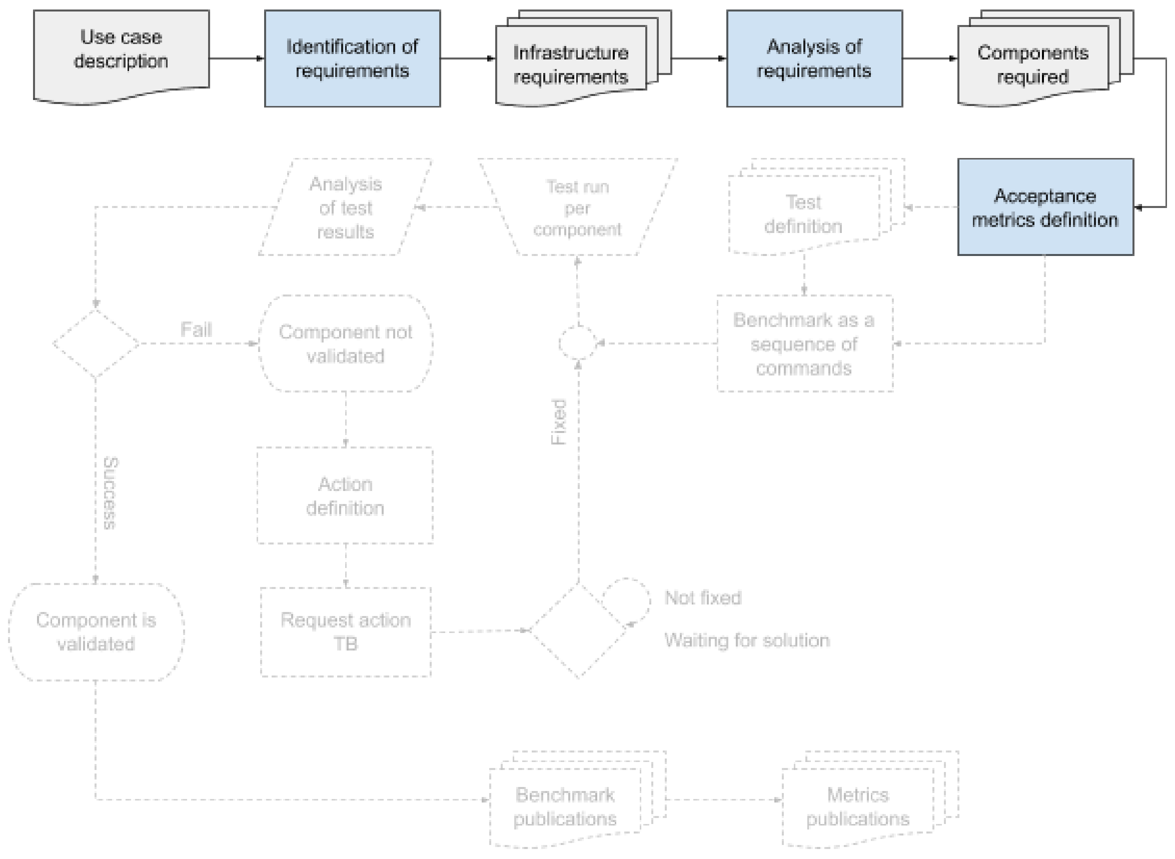

Figure 3: Component validation framework: infrastructure requirements analysis

3.2.2 Test definition

After the acceptance metrics definition step, the sequence of commands to verify each hardware/software component will be defined. Each sequence will define the test (that will be described into a card) and the benchmark that should be run to validate the component.

For example, to test the memory size per node (hardware component) on a system, the sequence of commands will:

- ➢ check the configuration file of the scheduler installed to extract memory settings for each queue;
- ➢ extrapolate the minimum value of memory size per node (if needed);
- ➢ compare the memory size values according to the acceptance metric defined.

This is because sometimes information reported in the architecture documentation doesn’t consider the scheduler configuration that can set the memory available for users to a value less than the maximum value declared in the system description.

According to the specific scheduler, these commands could be customized for each system. Every sequence of commands will be collected in specific benchmarks in order to validate the single hardware/software component.

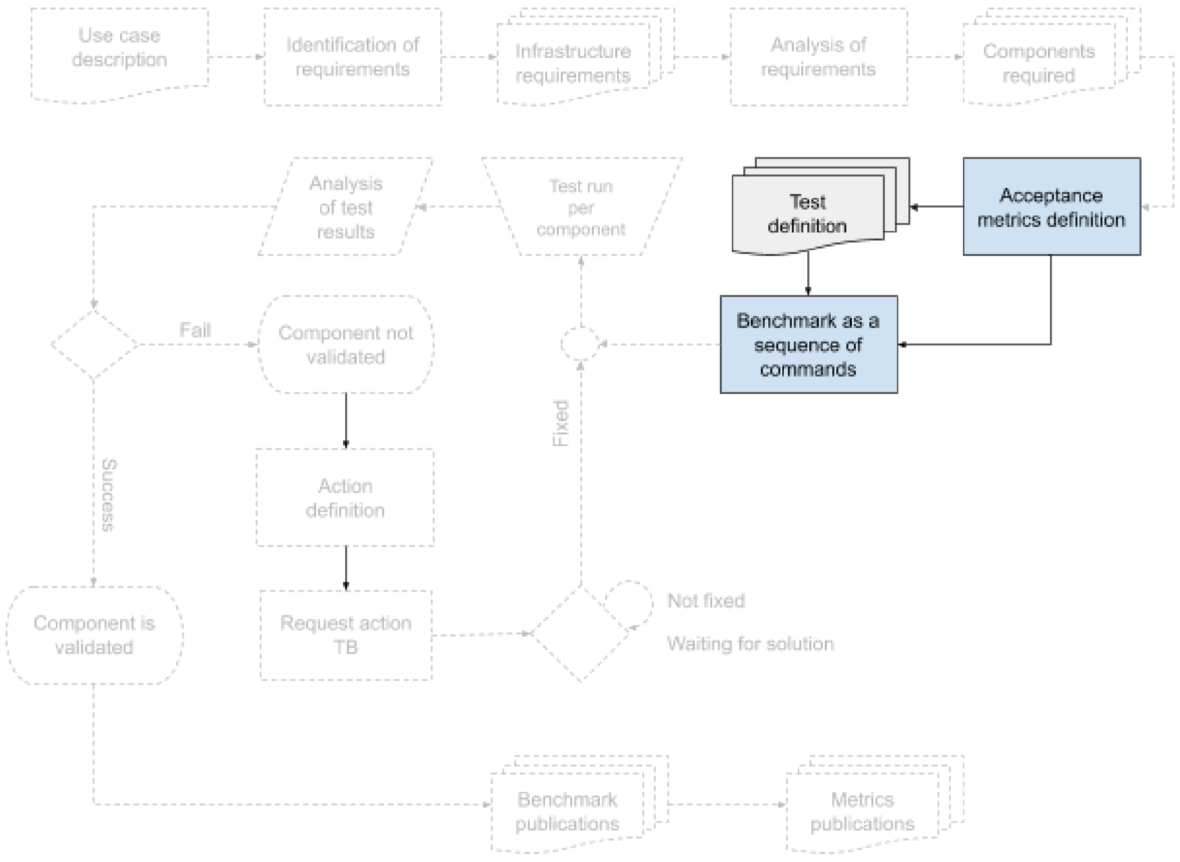

Figure 4: Component validation framework: test definition

3.2.3 Running test and validation component

Every benchmark will be run in order to evaluate the hardware/software component to be validated. The analysis of results of a benchmark is based on the comparison of the benchmark result(s) with the expected result(s) identified by the acceptance metrics definition, and it will establish if the tested component is validated or not.

If the analysis results do not validate the hardware/software tested component, an action has to be defined (e.g. open an issue to site representative in order to report the failure and asking for a solution) and submitted as proposed action to the Technical Board (or Work Package Leader). When the issue is fixed, the benchmark will be run again in order to evaluate the hardware/software component.

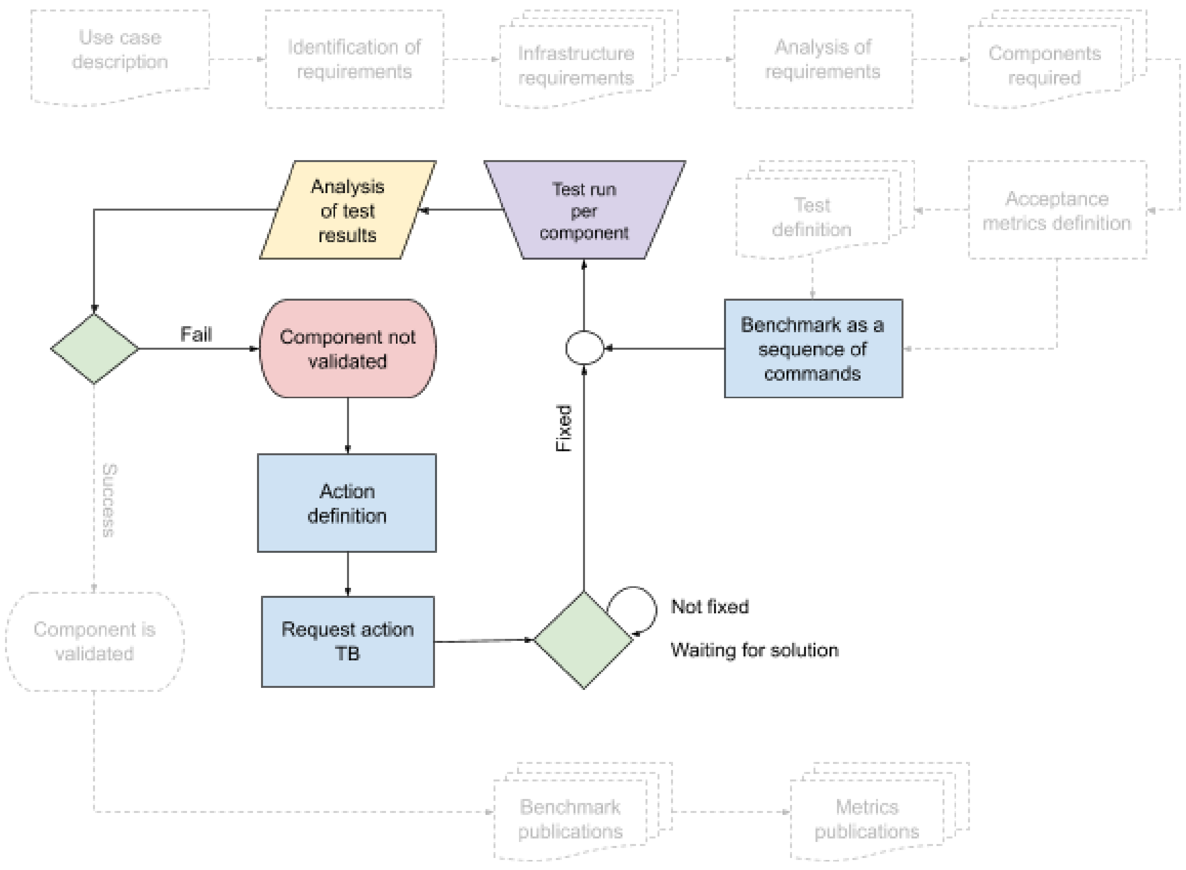

Figure 5: Component validation framework: running test and validation component (fail)

If the analysis results validate the hardware/software tested component, the benchmark and the metrics are published, and the framework will move to the next step (see below, section 3.2.4 Validation of infrastructure).

3.2.4 Validation of infrastructure

In order to validate the infrastructure or, in other words, in order to verify if the infrastructure is suitable for a specific use case, all benchmarks defined have to validate the corresponding hardware/software component required by the given use case. The collection of benchmarks identified for testing all components of the relevant use case(s) will make up the validation procedure that will be adopted to validate the use case(s) workflow(s).

3.2.5 Deploy of manual/automatic procedure

The identified procedures in previous step (3.2.4 Validation infrastructure) will be classified as manual or automatic, and reviewed. After approval, the procedures will be published and deployed.

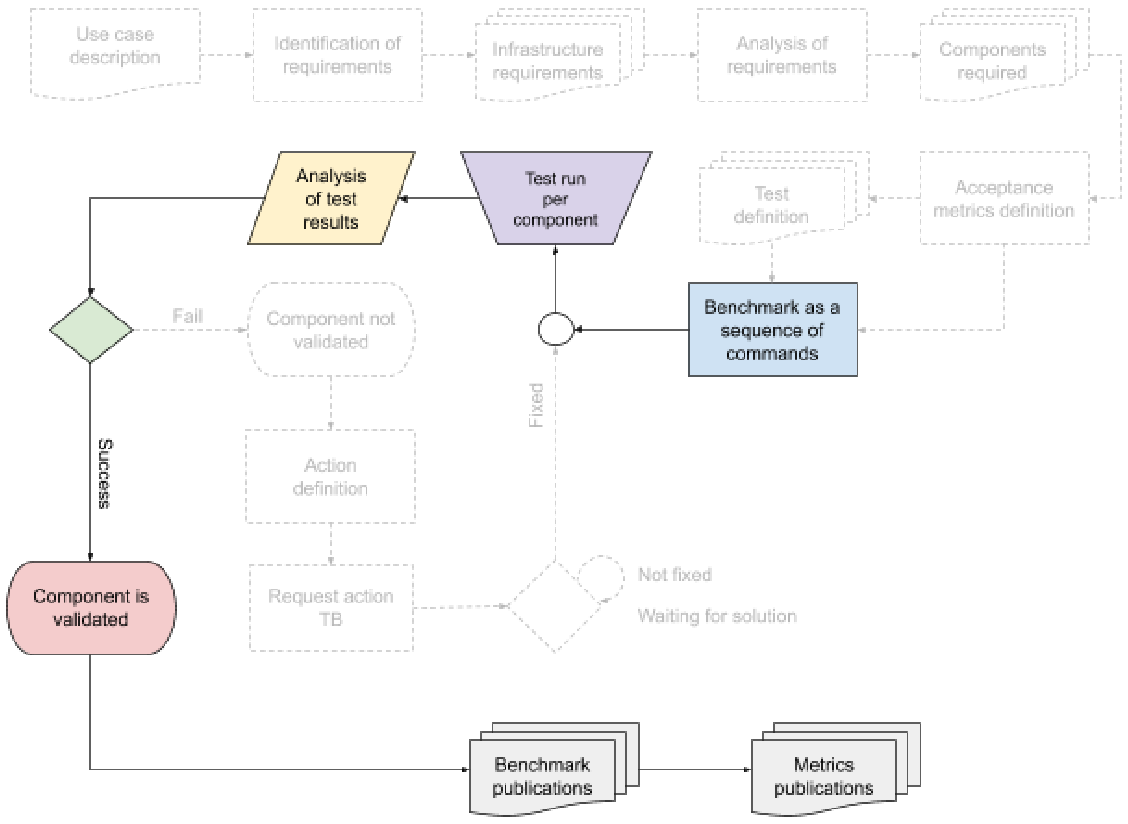

Figure 6: Component validation framework: running test and validation component (success)

3.3 Roles in validation framework

The validation process has to be based on a collaboration between the Task 3.6 team and the sites expert teams.

3.3.1 Roles in the "Infrastructure requirements documentation" step

Task leader, in collaboration with the leader of Task T3.1, has to identify the set of relevant use cases.
Task leader and contributors are in charge of the infrastructure requirements analysis of each use case in order to describe the acceptance metrics definition for each component.
The sites expert teams are in charge of identifying the values to be associated with the acceptance metrics description.

3.3.2 Roles in the "Test definition" step

The sites expert teams are in charge of identifying the sequence of operational commands in order to test the hardware/software components installed in their own site, filling the card template described below (see section 3.5 "Test card template").
The task leader and contributors are in charge of collecting all filled cards, classify the tests, and build up the benchmarks to be run.

3.3.3 Roles in the "Running test and validation component" step

The task team will schedule the run of benchmarks and collect the results. Next, the team will analyze the results. If a component will not be validated, an action definition will be discussed internally the task team and the issue will be addressed to the WP3 leader or to the site where the test failed, according to the nature of the issue. When the issue will be solved, the test will be run again.
When a component is validated, a description of the benchmark and metrics obtained will be published in the official repository, and linked to the official web portal, that will be defined for ICEI.

3.3.4 Roles in the "Validation infrastructure" step

In order to define a procedure, the task team will be in charge of collecting benchmarks related for a specific use case, and run the procedure to verify the suitability of the infrastructure for that use case. In case of success, a document for the procedure description will be produced.

3.3.5 Roles in the "Deploy of manual/automatic procedure" step

The sites expert teams are in charge of reviewing the documentation and the procedure.
The task team is in charge to deploy the procedures and publish a user guide in the official repository, and linked to the official web portal, that will be defined for ICEI, after sites expert teams approval.

3.4 Test and benchmark environment

The aim is to identify a common environment to avoid various test proliferation. However, due to heterogeneous infrastructure, some tests should be customizable to run on different systems and architectures when needed. A collaboration with the sites expert teams is needed in order to achieve the right customization.

For the same reason, even if the benchmark environment should be common to the federated systems, the heterogeneous infrastructure cannot guarantee that a benchmark can run in the same way in different sites. Collaboration with the sites expert teams is needed in order to achieve the right customization.

3.5 Test card template

Each test will be described by a standard set of information, and the cards of those tests will be stored in the official repository, that will be defined for ICEI. Below a table with all possible fields.

Table 2: Description of fields to define a test

Tag Name	Tag description.
Related Use Case(s)	Use Case(s) covered by this test (same id numbers as used in [ICEI-D3.6]).
Identifier Number TID#	Unique identifier of the test.
Release #.#	Progressive number to identify the version of the test. The first number will identify the major release, the second number will identify the minor update of the description.
Owner(s)	People in charge of the maintenance of the test description, the execution and the evaluation of the metrics.
Component	List of involved components/functionalities in the test.
Site(s)	List of sites where the test is performed.
Description	Brief description of the test.
Command/code	Default script location (or command line) to execute the test.
Input (optional)	Description of the type and format of input data
Output	Description of expected outcome/metric to consider the test executed successfully or not.
Metric(s)	The expected value of the outcome/metric identified above.

4. Key Performance Indicators (KPIs)

The number of tests will increase during the next months of activities. We expect to raise this number according to the implementation of services listed in section 3.1 ICEI infrastructure.

Table 3: KPIs for test definition and execution

	M27	**M30**	**M33**	**M36**
# of defined tests	3	6	9	12
# of executed tests	0	4	8	12
# of use cases covered	5	11	15	15

5. Road-map of activities

Five phases are identified to achieve the objectives of Task T3.6:

- ➢ phase 1: identification of a subset of relevant use cases in collaboration with Task T3.1;
- ➢ phase 2: validation framework for the subset of relevant use cases identified in phase 1, with exclusion of the last step ("Deploy of manual/automatic procedure");
- ➢ phase 3: validation framework for the residual use cases, with the exclusion of the last step ("Deploy of manual/automatic procedure");
- ➢ phase 4: a review of manual/automatic procedure and approval for each use case (first part of "Deploy of manual/automatic procedure" step);
- ➢ phase 5: deploy of procedures and guidelines (the second part of "Deploy of manual/automatic procedure" step).

The timeline of activities is described in the following Figure 7.

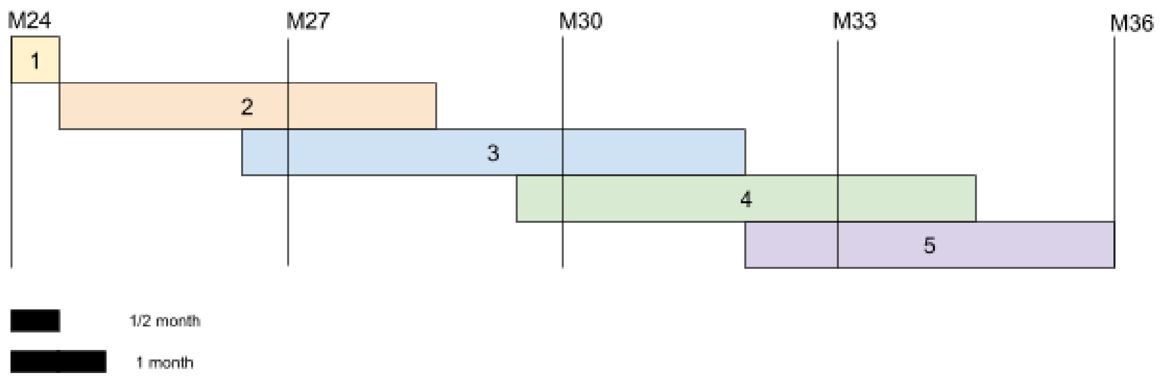

Figure 7: Timeline

6. Relation to the use case definition activities

To achieve maturity of tests, the activities of Task T3.6 will be worked in parallel with the implementation of use cases.
If some needed information to achieve the validation objectives for a specific test will not be available in the description of a use case, the owner(s) of that test will ask for a revision of the use case in order to obtain an update of use case requirements.
The validation activity in ICEI will be focused on the infrastructural aspect to avoid a repetition of work made in SGA2.

7. Summary and concluding remarks

The validation framework presented in this deliverable will allow to validate the ICEI infrastructure taking its distributed nature into account. The approach is closely linked to the use and science cases identified for ICEI. As a first step towards implementing this framework, we use the Learning-to-learn demonstrator defined in Appendix 1. Because all required services are not yet in place, only the first two steps of framework have been validated, namely “Infrastructure requirements documentation” (see Appendix 1) and “Test definition” (see Appendix 2).

In Appendix 3, an example of instantiation of the Component Validation Framework limited to “Infrastructure requirements documentation” and “Test definition” steps at a Fenix Site for the “Neurorobotics platform (NRP), large-scale brain simulations” use case will be introduced.

For validating the ICEI infrastructure we will also leverage the ICEI Application Benchmark Suite. Details for this benchmark suite are documented in Appendix 4.

8. References

[ICEI-D3.1] ICEI deliverable D3.1, “Common Technical Specifications”, version 3.1, November 2018.

[ICEI-D3.6] ICEI deliverable D3.6, “Scientific Use Case Requirements Documentation”, version 3.1, November 2018.

[ICEI-D4.15] ICEI deliverable D4.15, “Tender Documents (Part 2)”, version 2.2, August 2019.

9. Annex 1: Learning To Learn: Infrastructure requirements documentation

9.1 Introduction

In this section, the first step "Infrastructure requirements documentation" of the validation framework will be described in order to demonstrate that the activity has begun. The identified use case is the **"Learning-to-learn (LTL) in a complex spiking network on HPC and Neuromorphic hardware interacting with NRP (#3)"** ([ICEI-D3.6], section 22).
The choice for the Learning-to-learn use case has been made because this is at a relatively advanced planning stage, and its requirements cover a significant number of ICEI services.

The aim of this appendix is to document the first step of the "Component validation framework" using a concrete example.

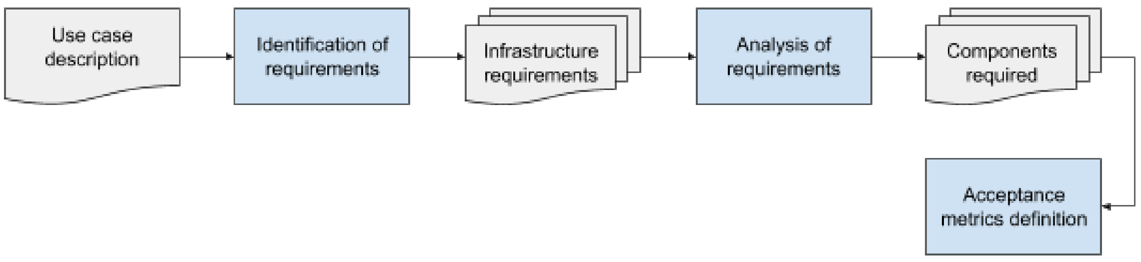
Use case description
Identification of requirements
Infrastructure requirements
Analysis of requirements
Components required
Acceptance metrics definition

Figure 8: Infrastructure requirements analysis

9.2 Use case description

The workflow was earlier documented in [ICEI-D3.6] and a graphical representation is reproduced in Figure 10. It shows the following steps:

1) Information about the configuration and deployment of the experiments stored in git or collab repository define the input coding.
2) The input coding is used to start the controller script.
3) The controller script interacts with simulations running on CPUs (SIM) or neuromorphic hardware (NMH).
4) NMH/SIM send and receive information from the virtual/real environment.
5) Results from the simulation running in the NMH/SIM are pre-processed.
6) The pre-processed results are sent to the outer loop L2L algorithm running on HPC.
7) The output of the simulation of L2L algorithm is sent to the long term storage (where it can be later retrieved/analysed/post-processed).
8) The L2L algorithm evaluates the fitness of the simulations and produces new configuration to be run in a next iteration of fitting.

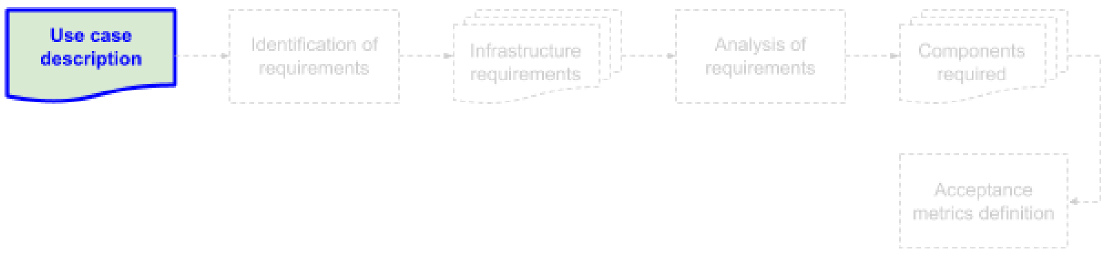
Use case description
Identification of requirements
Infrastructure requirements
Analysis of requirements
Components required
Acceptance metrics definition

Figure 9: Use case description

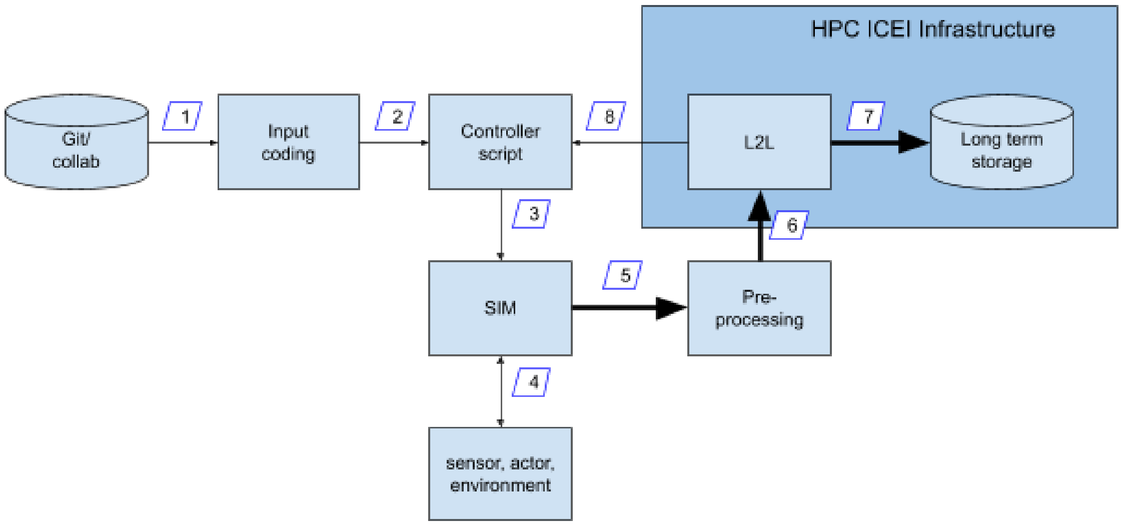

Figure 10: Graphical representation of the learning-to-learn workflow taken from [ICEI-D3.6] (Figure 47, p. 188)

The ICEI infrastructure is involved with:

- the data transport (in the ICEI infrastructure) of the Pre-processing output as input for the L2L algorithm;
- the execution of L2L;
- the data transport (out of ICEI infrastructure) of the L2L output as input for the Controller and the data transport of the same L2L output to the ICEI long term storage;
- the L2L output preservation in the ICEI long term storage.

9.3 Identification of requirements

First useful information has been provided in section “22.3.6 Components running on or communicating with ICEI Infrastructure”, [ICEI-D3.6], p. 199.

About *Data transport*: A full scale deployment of the workflow would require a bandwidth up to 8 Gbit/s for data transport link 5) and 6), 1 Gbit/s for links 7) and 8). The average required bandwidth is claimed to be 1 Gbit/s for all these steps. Data is accessed in chunks of 1 GiByte per instance and up to hundreds of parallel instances

are expected. The data transport via link 5) or 6) is via external network. Without additional investments into a dedicated network link it cannot be expected that a bandwidth of 1 Gbit/s could be sustained. For the foreseeable future therefore a bandwidth of 10 MByte/s is foreseen.
About the *Processing station* (L2L), the whole training will require: 500.000 core hours, 25 compute nodes (like JURECA compute nodes) or more, about 100 GiByte of RAM. The required software stacks are python scripts.

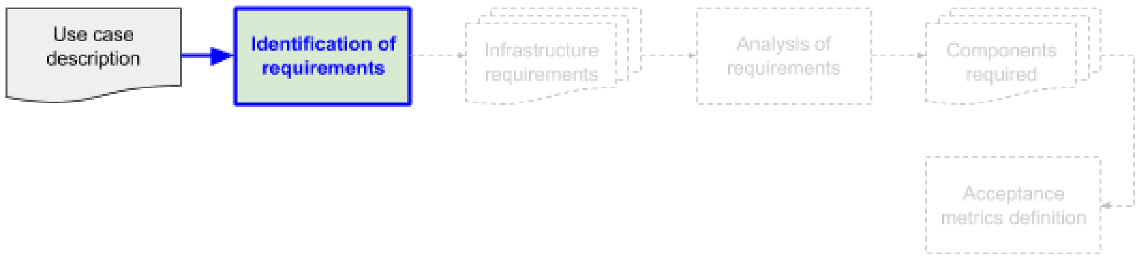
Use case description
Identification of requirements
Infrastructure requirements
Analysis of requirements
Components required
Acceptance metrics definition

Figure 11: Identification of requirements

9.4 Infrastructure requirements

Other information has been provided in section 22.4 “Infrastructure requirements” [ICEI-D3.6].

<u>*Interactive Computing Services*</u>: **4 hours of walltime** or more are required. **Required software stacks**:

- (*BrainScaleS*) System C Compiler to build gcc >=7.2;
- (*BrainScaleS*) System Python installation to execute Spack;
- (*BrainScaleS*) Spack packages: autoconf, automake, bazel, gccxml, gsl, intel-tbb, libelf, liblockfile, npm, pkg-config, py-cartopy, pylxml, py-mock, texinfo, xerces-c, binutils+gold+plugins, vim, emacs ~X, tmux, ncdu, units, ranger, py-ranger, mosh, mercurial, git, git-review, py-git-review, cmake, doxygen, doxygen+graphviz, bear, rtags, cppcheck +htmlreport, ffmpeg, gdb, llvm, genpybind, node-js, openssh, emscripten, boost@1.66.0+graph+icu+mpi+python+numpy, yamlcpp+shared, tensorflow, log4cxx, googletest, googletest +gmock, gflags, cereal, py-pybind11@2.2.0:, py-bokeh, pypygtk, gtkplus, cairo+X, py-pyside, py-slurm-pipeline, nest@2.2.2+python, py-brian, py-brian2, py-elephant, pypynn@0.7.5, python, py-cython, py-pip, py-pylint, py-ipython, py-virtualenv, py-matplotlib~tk+qt+ipython, py-numpy, pypandas@0.19.0:, py-pytables@3.3.0:, py-scipy, py-scikitimage, py-seaborn, py-sympy, py-statsmodels, py-lmfit, pysymfit, py-sqlalchemy, py-pyyaml, py-autopep8, py-flake8, py-jedi, py-sphinx, py-doxypy, py-nose, py-junit-xml, pyxmlrunner, py-pytest, py-pytest-xdist, py-line-profiler,

pyattrs, py-setuptools, py-tabulate, py-html, py-html5lib, pypillow (cf. meta Spack package)
- (*SpiNNaker*) Remote server requires Oracle Java >= 1.8 with Maven >= 3.5 installed. Other libraries are automatically provided via Maven. The VM system runs XenServer 7.
- (*SpiNNaker*) Remote client requires Oracle Java >= 1.8 installed on a VM template, along with scripts to start and execute the client, which should be loaded on to the VM and executed when it starts.
- (*sPyNNaker*) CPython 2.7 and pip >= 9. Other libraries are automatically provided via pip.
- Minimum **RAM per node** required by the software stack is **16 GiByte** (better **64 GiByte** per node).

Scalable Computing Services: are required for the analysis steps.
Virtual Machine Services: are required.
Archival Data Repositories: are required for the analysis steps (10 TByte permanent, 5 TByte temporary, see 22.3.4 “Data Repository” section in [ICEI-D3.6]).
Data Transfer Services: Inter-site transfer orchestration service.
Internal Interconnect: **10 GbE**.
External Interconnect: **10 GbE**.
Authentication/Authorization Services: service accounts, fine-granular control of access to data, HBP account/authentication system, export control rules apply to the BrainScaleS.
User Support Services: **HLST** (maybe).

Additional requirements, according to the modifications described in the [ICEI-D4.15], section 5.2, are:
Scalable Compute Services: **GPU-accelerated nodes** for running machine learning applications;
Interactive Compute Services: **data post-processing**;
Active Data Repositories: to hold transient data products with a **capacity of 10s of TByte**;
Archival Data Repositories: to hold input data as well as final data products with a **capacity of 10s of TByte**;
External network connectivity: to transfer data between an ICEI data centre and a neuromorphic systems site.

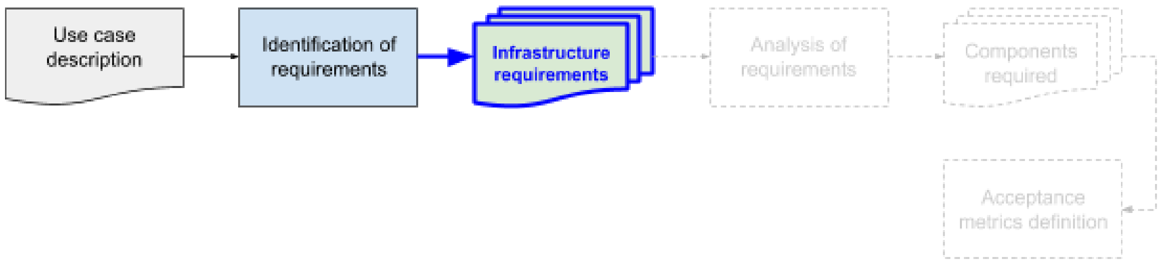
Use case description
Identification of requirements
Infrastructure requirements
Analysis of requirements
Components required
Acceptance metrics definition

Figure 12: Infrastructure requirements

9.5 Analysis of requirements

The requirements identified in the section 22.3.6 [ICEI-D3.6] are:

- a network link between NMH/SIM and ICEI infrastructure with an average bandwidth of 10 MByte/s;
- a HPC system with 25 compute nodes (like JURECA compute nodes) available and 12 GiByte RAM per node;
- a parallel file system to guarantee the parallel data access up to 100 simultaneous accesses to 1 GiByte data size per access;
- the L2L third-party software components needed for the L2L software (e.g., Python).

The JURECA compute node: 2 Intel Xeon E5-2680 v3 Haswell CPUs per node, with 2 x 12 cores at 2.5 GHz, Intel Hyperthreading Technology (Simultaneous Multithreading) and AVX 2.0 ISA extension. RAM available from 128 GiByte per node (most of compute nodes) up to 512 GiByte per node (64 compute nodes). The number of needed cores and/or compute nodes will be reduced or grown according to the compute nodes specification.
Each federated partner site will configure its own parallel filesystem; configuration are in charge of the site.
The amount of core hours is dependent by the compute nodes specification.
The temporary Archival Data Repository size required is 5TByte, the permanent Archival Data Repository size required is 10s of TByte.

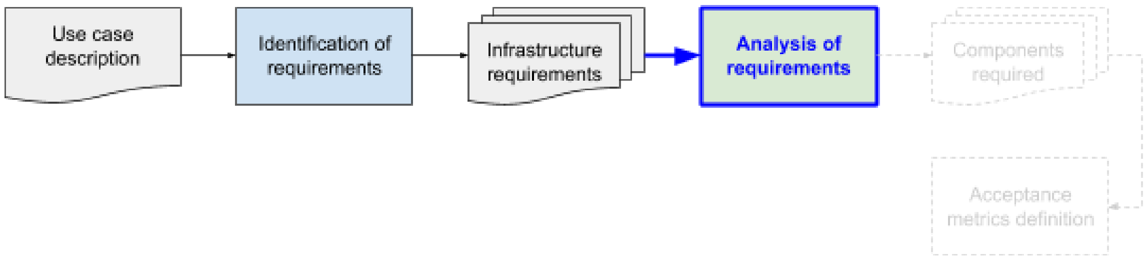
Use case description
Identification of requirements
Infrastructure requirements
Analysis of requirements
Components required
Acceptance metrics definition

Figure 13: Analysis of requirements

9.6 Components required

According to the previous steps, the hardware requirements are related to the network configuration (internal and external), the compute nodes architectures (number of cores, memory available per core), the storage size available.
The software components required are related to scheduler configuration, filesystems configuration, HPC system environment configuration, HPC tools for interactive computing (e.g. containers and OpenStack).

Other ICEI services required are related to resources provider (FURMS), resources allocation (FURMS), Identity Provider (FENIX Central IdP), authorization/authentication (FENIX AAI), data transfer service.

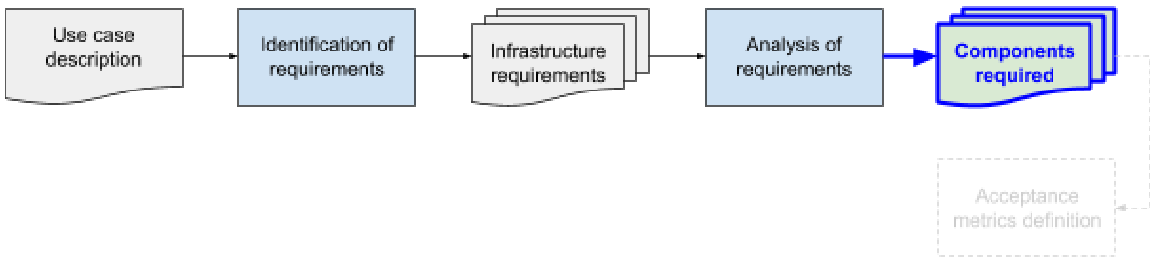
Use case description
Identification of requirements
Infrastructure requirements
Analysis of requirements
Components required
Acceptance metrics definition

Figure 14: Components required

9.7 Acceptance metrics definition

For each component, at least an acceptance metric has to be defined. Metrics can be one or more numeric or logical values.

9.7.1 Numeric metric values

In order to satisfy performance requirements of the use case, like the transfer speed, the acceptance metric will be defined as the minimum value required (e.g. for the **internal network** the **acceptance metric** will be defined as a speed **greater or equal to 10 MByte/s**: if lower, the component will not be validated).
When the use case requirements are related to size value, like storage size or memory size, again the acceptance metric will be defined as the minimum value required (e.g. for the **memory size** the **acceptance metric** will be a size **greater or equal to 96 GiByte per node**: if lower, the component will not be validated).
When the use case requirements are related to the amount of resources, like the amount of nodes/cores or the walltime limit, again the acceptance metric will be defined as the minimum value required (e.g. for the **number of cores** the **acceptance metric** will be a value **greater or equal to 600 cores**: if lower, the component will not be validated).
In order to define the acceptance metrics for the software components, it is important to identify the (minimum) release/version of each component (e.g. for **gcc** the release has to be **greater or equal to 7.2**).
Acceptance metrics related to certain components like scheduler or filesystem can be numeric. For example, because several products exist to schedule jobs on an HPC system (like PBS, SLURM, LSF, LoadLeveler) one acceptance metric will be defined as the minimum release required: if the release will be lower than the minimum, the component will not be validated.
Acceptance metrics for tools related to a service component, like data transfer, AAI or FURMS, will be defined as the minimum release required: if the release will be lower than the minimum, the component will not be validated.

9.7.2 Logical metric values

Acceptance metrics related to configuration of certain components like scheduler or filesystem, can be logical in some cases. For example, because the scheduler is in charge to allocate computational resources, good candidate as acceptance metrics are:

- availability of a partition that can allocate the hardware components required (cores and memory size per core), for the required time (walltime): if false, the component will not be validated;
- scheduler configuration allows the exclusivity of resources: if false, the component will not be validated;
- scheduler configuration allows the interactive submission of a job: if false, the component will not be validated.

Acceptance metrics for a service component, like AAI or FURMS, can be logical. For example:

- the user recorded in a user database of site or project (like PRACE or HBP) can be identified and defined on ICEI infrastructure [false/true];
- the project account can be defined on the required site [false/true];
- the core/hours budget can be associated to a project account [false/true];
- the accounting system can update regularly the resource consumption [false/true];
- the identified tool to access allows the security access to the infrastructure [false/true].

Acceptance metrics for service components, like data transfer, will be numeric and/or logical. For example:

- the identified tool to transfer data allows the data transfer according to the security policies [false/true].

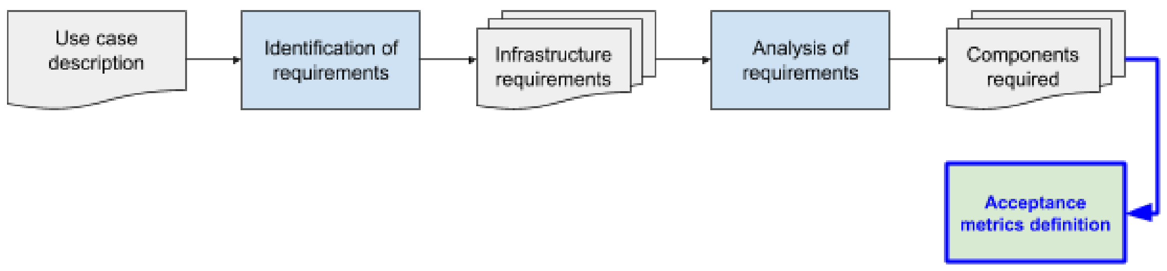
Use case description
Identification of requirements
Infrastructure requirements
Analysis of requirements
Components required
Acceptance metrics definition

Figure 15: Acceptance metrics definition

10. Annex 2: Learning To Learn: Test definition

10.1 Introduction

In this Appendix 2, the second step "Test definition" of the validation framework will be described in order to demonstrate that the activity has begun. The identified use case is the **"Learning-to-learn (LTL) in a complex spiking network on HPC and Neuromorphic hardware interacting with NRP (#3)"** ([ICEI-D3.6], section 22).
The choice for the Learning-to-learn use case has been done because this is quite mature, and requirements cover a significant number of services.
The aim of this appendix is to validate the second step of "Component validation framework".

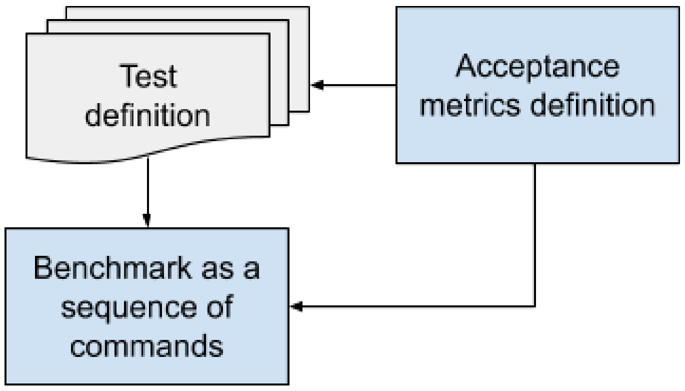
Test definition
Acceptance metrics definition
Benchmark as a sequence of commands

Figure 16: Test definition framework

10.2 Test definition step

Following the framework, from the Acceptance metrics definition step will be possible to define the set of tests needed to validate each component. In the following subsections, a subset of tests will be defined for some requirements.

10.2.1 Test definition step: External and Internal Interconnect

To validate available internal and external interconnect performance, NETI and NETE, respectively, micro-benchmarks will be used. Internal network can be validated using the iperf3 tool while for external data transfer the secure copy tool could be used.
According to these observations, the test definition for the External Interconnect requirement could be:

- obtain the access to the PRACE Monitoring tool;
- check the network performance results, limited to the ICEI sites;
- compare the network performance results with the Acceptance Metric;
- mark the test result with "Success" or "Fail" according to the outcome of previous comparison.

Alternately, after installation of the appropriate tool (e.g. iperf3) in order to measure the network performance on ICEI sites:

- ➢ run the commands to check the network speed for upload and download operations;
- ➢ check the network performance results obtained;
- ➢ compare the network performance results with the Acceptance Metric;
- ➢ mark the test result with "Success" or "Fail" according to the outcome of previous comparison.

About the Internal Interconnect requirement, it would be enough to check sites documentation:

- ➢ access to the public documentation of each ICEI site;
- ➢ check the network speed declared;
- ➢ compare the value of network speed declared with the Acceptance Metric;
- ➢ mark the test result with "Success" or "Fail" according to the outcome of previous comparison.

In case these tests would marked as "Fail", the following actions will be needed:

- ➢ contact the ICEI site representative where the test fails;
- ➢ ask for a quick update of network configuration, and wait for a reply:
    - ○ if for External Interconnect the request has to be redirected to the appropriate NREN by the ICEI site representative;
- ➢ explore other transfer tools for improving use of available hardware connectivity
- ➢ repeat the failed test.

10.2.2 Test definition step: Interactive/Scalable Computing Services - Available user's resources

When the use case requirements are related to the amount of resources, like the amount of nodes/cores, or the walltime limit, or the memory size available per node/core, again the test will be defined as the check of system description and configuration. In this case, the test is a simple check of sites documentations.
However, because each site can limit the user's usage of resources through the scheduler configuration, for a deeper validation, after the login on ICEI integrated systems, it is possible to use some commands in order to check the appropriate scheduler configuration.
For example, in order to check the available resources managed with SLURM, the command:

scontrol show partition

returns the list of defined queues on the HPC system with all needed information for each queue. For a PBS scheduler, the command:

qstat -f -Q

returns the same information. Results can be compared with the acceptance metric.
According to these observations, the test definition for available user's resources could be:

- ➢ access to the public documentation of each ICEI site;

- check the architecture details of the appropriate system;
- compare the values declared for the appropriate resource (number of nodes/cores available, walltime limit, memory size per node/core) with the appropriate Acceptance Metric;
- mark the test result with “Success” or “Fail” according to the outcome of previous comparisons.

Alternately:

- login into the ICEI integrated system;
- run the appropriate command, according to the scheduler installed, in order to check the scheduler configuration;
- extrapolate the values for the appropriate resource (number of nodes/cores available, walltime limit, memory size per node/core);
- compare the extrapolated values with the appropriate Acceptance Metric;
- mark the test result with “Success” or “Fail” according to the outcome of previous comparisons.

In case these tests would marked as “Fail”, the following actions will be needed:

- contact the ICEI site representative where the test fails;
- ask for a quick update of scheduler configuration, and wait for a reply;
- repeat the failed test.

10.2.3 Test definition step: Active Data Repositories

When the use case requirements are related to size value, like storage size or memory size, again the test will be defined as the check of component configuration.

In order to test the memory size available to store data on an ICEI system, it will be enough to check sites documentation and compare the declared value of the memory size available with the acceptance metric.

For a deeper validation, it could be possible to use some commands in order to explore the memory size available. For example, after the login on the selected system, the command *quota* will return the user’s disk usage and limits. Results can be compared with the acceptance metric.

According to these observations, the test definition for Active Data Repositories requirements could be:

- access to the public documentation of each ICEI site;
- check the values related to the sizes of memory provided to store and manage user’s data (it is usual to find more than one filesystem partition) of the appropriate system;
- compare the values declared for the appropriate memory size available (e.g. for user’s private storage area, for user’s shared storage area, for project’s shared storage area, etc.) with the appropriate Acceptance Metric;
- mark the test result with “Success” or “Fail” according to the outcome of previous comparisons.

Alternately:

- ➢ login into the ICEI integrated system;
- ➢ run the appropriate command according to the Operating System, in order to check the memory size available for each user's storage area;
- ➢ extrapolate the values for the appropriate storage area (user's private storage area, user's shared storage area, project's shared storage area, etc.);
- ➢ compare the extrapolated values with the appropriate Acceptance Metric;
- ➢ mark the test result with "Success" or "Fail" according to the outcome of previous comparisons.

In case these tests would marked as "Fail", the following actions will be needed:

- ➢ contact the ICEI site representative where the test fails;
- ➢ ask for a quick configuration change in order to make available the right value, and wait for a reply;
- ➢ repeat the failed test.

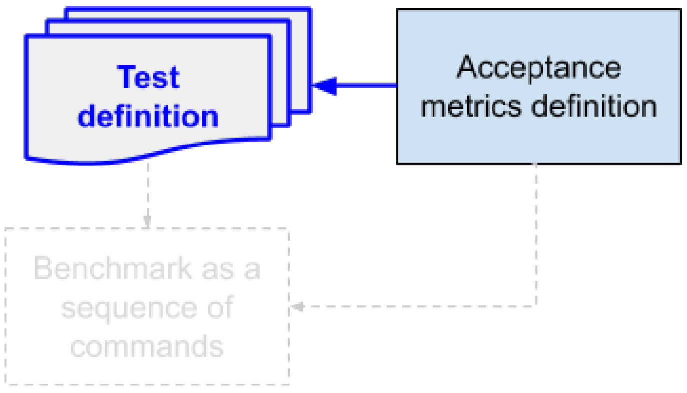
Test definition
Acceptance metrics definition
Benchmark as a sequence of commands

Figure 17: Test definition step

10.3 Benchmark definition step

Following the framework, information coming from test definition and acceptance metrics will be used to identify the set of benchmarks as sequence of commands. In the following subsections, a subset of Benchmark definitions are introduced.

10.3.1 Benchmark definition step: External Interconnect

To test the external connectivity, secure copy (scp) will be used as micro-benchmark to validate the connectivity between an ICEI site and an external facility. By performing multiple measurements with a duration of 10-20 seconds it will be possible to determine an average and maximum bandwidth and compare this to the needs of the use case representatives.

Tag Name	External Connectivity L2L
Related Use Case(s)	Learning-to-learn (LTL) in a complex spiking network on HPC and Neuromorphic hardware interacting with NRP
Identifier Number TID#	(TBD)
Release #.#	(TBD)
Owner(s)	JUELICH team
Component	NETE, IAC, ACD
Site(s)	JUELICH
Description	This test uses secure copy executed on an interactive compute node (IAC) to copy a file of size 1 GiByte from an external site, i.e. a site not connected to the PRACE network, to a local Active Data Repository (ACD) via the external network link (NETE).
Command/code	for i in `seq 1 10`; do scp <remote> /tmp/tst.$$; done; rm /tmp/tst.$$
Input (optional)	File of size 1 GiByte, e.g. created using the following command: dd if=/dev/zero of=tst.$$ bs=$((1024 * 1024)) count=1024
Output	Bandwidth as reported by secure copy
Metric(s)	Maximum, minimum and average transfer speed in units of MByte/s

10.3.2 Benchmark definition step: Interactive/Scalable Computing Services - Available user's resources

To test the amount of resources, for those systems managed by SLURM scheduler, the *scontrol* command of SLURM will be used as micro-benchmark to validate the availability of number of nodes, cores and memory size per node/core. This test will be run first on Marconi system (CINECA).

Tag Name	Availability of resources L2L
Related Use Case(s)	Learning-to-learn (LTL) in a complex spiking network on HPC and Neuromorphic hardware interacting with NRP

Identifier Number TID#	(TBD)
Release #.#	(TBD)
Owner(s)	CINECA team
Component	IAC
Site(s)	CINECA
Description	This test uses *scontrol* command of SLURM in order to identify all possible queues configuration and extract the information related to the user's resources availability.
Command/code	(Sequence of manual commands) scontrol show partition | grep PartitionName scontrol show partition | grep MaxNodes scontrol show partition | grep DefMem
Input (optional)	None
Output	Queues name Maximum value of user's node available Maximum value of user's memory size per node/core available
Metric(s)	Minimum number of nodes required (measure unit not available) Minimum memory size per node/core available in units of MByte

10.3.3 Benchmark definition step: Active Data Repositories

To test the amount of memory available for the user's Active Data Repositories a micro-benchmark will be first performed on Marconi system (CINECA) with the command *cindata* (a custom script of CINECA) to validate the availability of memory size.

Tag Name	Active Data Repositories L2L
Related Use Case(s)	Learning-to-learn (LTL) in a complex spiking network on HPC and Neuromorphic hardware interacting with NRP
Identifier Number TID#	(TBD)
Release #.#	(TBD)

Owner(s)	CINECA team
Component	ACD
Site(s)	CINECA
Description	This test uses *cindata* command of CINECA in order to identify all possible areas available to each user in order to store data on directly on the system.
Command/code	cindata -u $USER
Input (optional)	None
Output	Maximum value of available memory to store data
Metric(s)	Minimum value of memory size available in units of GiByte

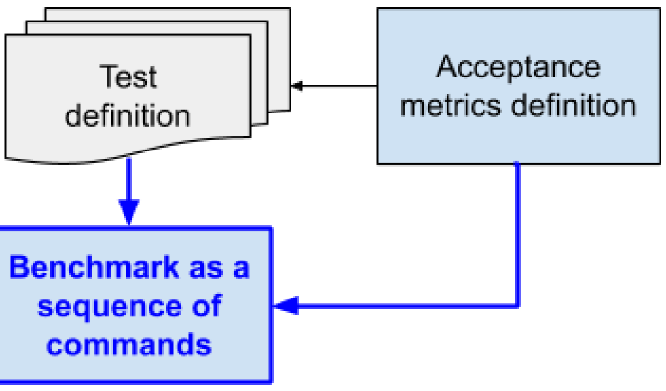
Test definition
Acceptance metrics definition
Benchmark as a sequence of commands

Figure 18: Benchmark definition step

10.4 Following steps

“Running test and validation component”, “Validation of infrastructure” and “Deploy of manual/automatic procedure” steps of the framework will be validated for the LTL use case as soon as possible.
It is possible to imagine that each benchmark implemented will be run/executed and the outputs will be compared with defined acceptance metrics in order to validate each service component. Sometimes a privileged user (e.g. system administrator) will be in charge to run the benchmark (“Running test and validation component” step).
When all components will be validated, the service will be considered validated, and if all services will be validated, than the whole infrastructure will be validated (“Validation of infrastructure” step).
When a component is validated, the benchmark and acceptance metrics will be published. Accordingly, when a service/infrastructure will be validated, the collection of benchmarks and acceptance metrics will be published in order to identify the first steps for verification and monitoring phases (“Deploy of manual/automatic procedure” step).

11. Annex 3: Neuro-robotics Platform (NRP)

11.1 Introduction

Deliverable D3.6 [ICEI-D3.6] and D3.1 [ICEI-D3.1] provide mappings of the ICEI use cases to a Fenix infrastructure site. Currently, ICEI resources are available at CSCS. The CSCS was selected as an ICEI site that specialises on the use/science case #11 "Neurorobotics platform (NRP), large-scale brain simulations". The steps "*infrastructure requirements analysis"* and *"test definition"* presented in the ICEI component validation framework is explained in the following as an example.

11.2 Component Validation Framework: Use Case Description

The Neurorobotics Platform (NRP) is a tool for studying models for brain, body, and environment in closed perception-action loops through interactive in-silico experiments. It effectively allows scientists to virtualize brain and robotics research. Applications used are NEST and Gazebo.
Two main components of the workflow that work in a coupled manner is to start a web service frontend and to run large-scale simulations. Workflow for starting the web service frontend is shown in the following figure.

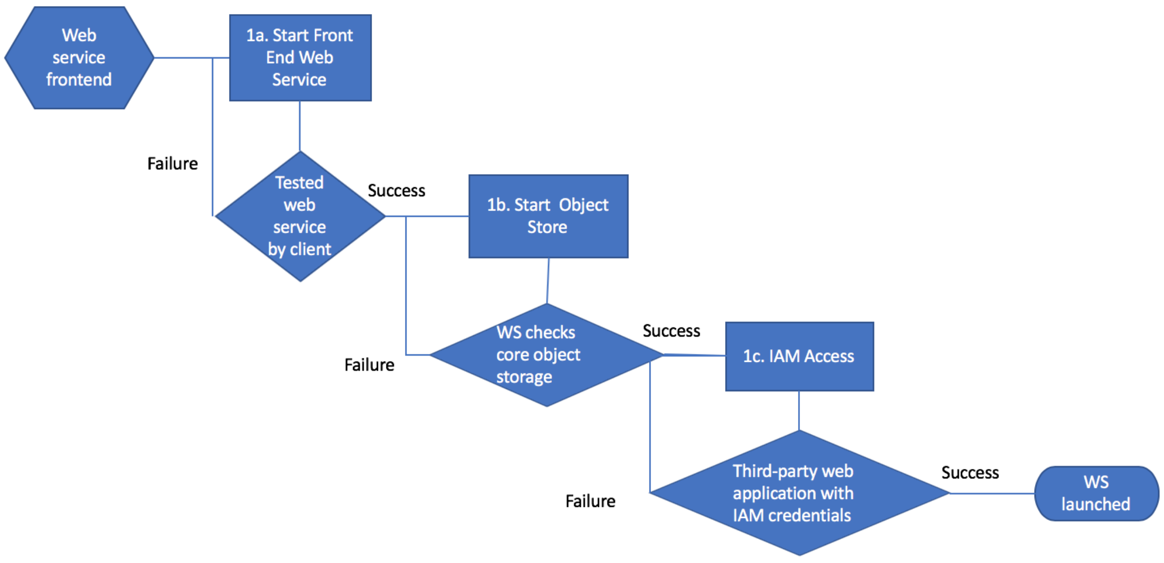
Web service frontend
1a. Start Front End Web Service
Failure
Tested web service by client
Success
1b. Start Object Store
Failure
WS checks core object storage
Success
1c. IAM Access
Failure
Third-party web application with IAM credentials
Success
WS launched

Figure 19: Neurorobotics Platform, workflow for web service frontend

11.3 Component validation framework: Infrastructure Requirements Analysis

Table A.1: Infrastructure Requirements Analysis

Component Name	Action	Dependencies
Web Service Frontend	1a. **Start** Frontend web service	VM (Pollux)
	1b. **Start** object store	ARD (Object store)
	1c. AAI access	AAI
Start Simulation	2a. **Start** 3rd party web app (**delegation**)	VM (Pollux), AAI
	2b. IAM user (**access delegation)**	AAI
	2c. Interactive **allocation** on Piz Daint	SCC, ACD (Piz Daint and its scratch)
	2d. **Move** data	ACD, ARD (future data mover)
	2e. **Submit** job (delegation)	SCC (Piz Daint)

11.4 Component validation framework: Test Definition

Example of one of the defined validation tests for NRP, starting the web service front-end using the VM service is shown in the figure. In general a test has attributes such as input, output, action and validation metric.

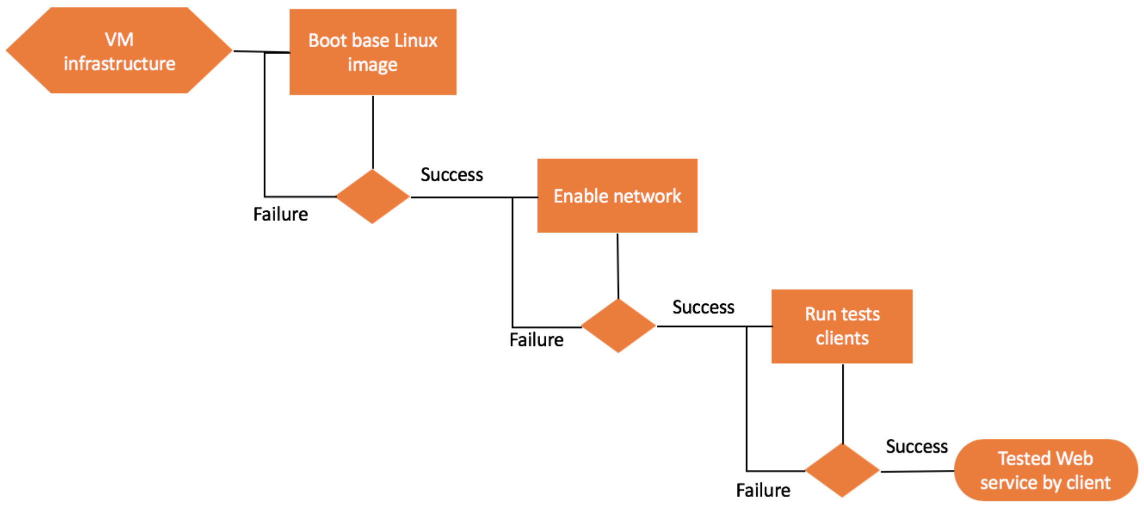

Figure 20: Example of one of validation tests for NRP

Validation of the virtualized infrastructure for NRP comprise the following steps (CSCS OpenStack cluster):

- ➢ boots base linux image
- ➢ enabled network
  - ○ receiving web request over internet (listen on web ports, e.g. 5000 for http and 8080 for https)
  - ○ connections to other infrastructure
    - ■ reach IAM infrastructure
    - ■ reach object storage
    - ■ reach FirecREST
  - ○ starts simple web server listening for requests with SSL (production environment)
  - ○ starts simple web server listening for requests without SSL (dev environment)

A test client will have the following properties:

- Simple client, e.g., curl, requests application, or browser
- Test client runs on independent infrastructure
- Client can issue request to web service

Output will be validated by the test client:

- Tested Web service by client

12. Annex 4: ICEI Application Benchmark Suite

12.1 Introduction

In this Appendix 4 will be presented again the Application benchmark suite for ICEI, in order to give a complete overview about the activities. This appendix is an extract of the "ICEI Benchmarks Overview & Instructions" document.

12.2 The benchmark suite

Following the recommendation of the experts attending the special review of the ICEI project on July 23, 2018, the project is in the process of implementing an "ICEI Application Benchmark Suite". This benchmark suite will be used for all procurement of equipment. Taking site specialisation into account, for different procurements different parts of the benchmark suite will be used and different weights will be foreseen for determining a score.

The components of the benchmark suite have been chosen such that it represents the breadth of the HBP science and use cases. The benchmarks are either directly based on applications or are based on micro-benchmarks with parameters chosen such that they reflect the anticipated use of the ICEI infrastructure. The selection of applications was performed based on the following considerations:

- How often is the application used today and can it be expected to be used in the future?
- Does the application help to cover the different ICEI science and use cases?
- Does the application present a wider class of applications?
- What is the level of maturity of the application implementation? How well is the performance behaviour understood?
- How easy can the application be built by a supplier? Is the application benchmark easy to port to different architectures?

For providing the application benchmarks to the suppliers, the JUBE benchmarking environment is used. This is a Python based framework to create benchmark sets, run those sets on different computer systems and evaluate the results. For each benchmark execution, the benchmark data is written out in a certain format that enables the benchmark to deduct the desired information. JUBE allows the integration of application-specific workflows to create a reproducible, automatic benchmark execution environment. Benchmark results are gathered and structured within JUBE. Parameterization, either for the compilation or execution phases, can be combined in a single point inside the JUBE configuration files.

In the remaining part of this section we provide an overview over the different application benchmarks.

12.3 Arbor

Arbor is a simulation library for networks of morphologically detailed neurons. Such simulations have two main computational "parts":

1. The time stepping of cell state: Cells are represented as trees of line segments, on which partial differential equations (PDEs) for potentials are solved using the finite-volume method. Furthermore, on each discretized location on the tree, state equations for ion channels and synapses are solved. This part of the simulation is computationally intensive, taking advantage of SIMD vectorization on CPU cores, and SIMT execution on GPUs.
2. Event driven simulation: Cells generate events called spikes, which are communicated globally using MPI. Each MPI rank then generates local events to be delivered to cells on that rank. Typically each spike will generate in the order of 1000 to 10000 deliverable events, which have to be queued. In simulations with simple cells (i.e. cells with low computational overheads), the memory and network overheads of event delivery can dominate simulations.

The benchmark is designed to test both parts of the workflow.

12.4 Bcfind (Brain cell finder)

Bcfind (Brain Cell Finder) is a tool for analysing mouse brain images. It uses a two-step machine learning approach to effectively localize single neurons in TByte-scale images of whole mouse brains. The algorithm first uses a deep 3D convolutional network to perform a semantic deconvolution, i.e. to render the image close to an ideal one where only cell bodies are visible and other elements are no more present. On this 'cleaned' image, a mean shift clustering is performed to localize the centroid of cell bodies.

12.5 Cosbench

Cosbench is an established benchmark to measure the performance of object storage. In the context of ICEI it is used to measure the performance of OpenStack Swift, which is the technology chosen for interfacing with the federated ICEI Archival Data Repositories.

12.6 Elephant ASSET

Elephant (Electrophysiology Analysis Toolkit) is a library including a set of tools for analysing spike train data and and other time series recordings obtained from experiments or simulations. Elephant is written in Python using mpi4py for multi-tasking and NumPy and SciPy for different computational tasks. For the ICEI Application Benchmark Suite the tool ASSET (Analysis of Sequences of Synchronous EvenTs) was chosen, which is believed to be a typical tool. It implements a fully automated method that determines diagonal structures in the intersection matrix, which contains in each entry the degree of overlap of active neurons in two corresponding time bins. When executing the tool a large number of accesses to complex data structures are performed.

12.7 IOR

IOR is a synthetic I/O benchmark that allows to generate different I/O patterns and use different I/O interfaces, e.g. POSIX, MPI-IO and HDF5. It is thus a useful tool to mimic the I/O behaviour of different applications without the need of compiling and executing potentially complex applications. The benchmark will be used to test the performance of implementations of Active Data Repositories (ACD).
The benchmark will comprise different parts. Some parts are based on typical configurations as they are, e.g., use for the IO500 benchmark. For other parts the parameters have been chosen such that they mimic the I/O patterns of brain image analysis applications. For these applications random access with small transfer sizes dominate.

12.8 NEST

NEST is a simulator for spiking neural network models that focuses on the dynamics, size and structure of neural systems rather than on the exact morphology of individual neurons. Networks of various sizes can be simulated, for example models of information processing or models of learning. This application starts to become a widely use simulator within the HBP.
NEST is a community code with an active user base. It runs on many supercomputers in the world with a proven high level of scalability. The program is written in C and uses MPI and OpenMP for parallelisation.

12.9 Neuron/CoreNeuron

NEURON is a simulation environment for modelling networks of neurons with complex branched anatomy and biophysical membrane properties. This includes extracellular potential near membranes, multiple channel types, inhomogeneous channel distribution and ionic accumulation. It can handle diffusion-reaction models and integrating diffusion functions into models of synapses and cellular networks. It is a very widely used simulator.
The compute engine of the NEURON simulator has been extracted and is being optimised as a library called CoreNEURON, which has partially been developed within HBP. NEURON simulator can be configured to use CoreNEURON library for efficient execution.

12.10 Neuroimaging Deep Learning

By moving towards very high resolution brain images, the so far largely manual or semi-automatic analysis of the images stops being feasible. Deep learning methods start to be successfully used for automatized processing of brain images.
The benchmark is based on TensorFlow (using Horovod for parallelisation) and implements a production use case using human brain images.

12.11 TVB-HPC

The Virtual Brain (TVB) is a software which has become a validated and popular choice for the simulation of whole brain activity. With TVB, the user can create simulations using neural mass models which can have outputs on different experimental modalities (EEG, MEG, fMRI, etc). TVB allows scientists to explore and analyse simulated and experimental signals and contains tools to evaluate relevant scientific parameters over both types of data. Internally, the TVB simulator contains several models for the generation of neural activity at region scale. Currently, the models simulated in TVB are written in Python and have not been optimized for parallel execution or deployment on High Performance Computing architectures.
The TVB-HPC project aims for a code that can exploit different supercomputers, including GPU-accelerated systems. TVB-HPC is written in Python and leverages the capabilities of Numba, which is a tool that translates Python functions to optimized machine code. It supports different targets, including processor architectures with vector/SIMD units or GPU accelerators.

12.12 Uptake of benchmarks per site

Application benchmarks in "**bold**" are placed in a single site.

Site	Application Benchmarks
JÜLICH	**Elephant ASSET**, NEST, **TVB-HPC**, **Arbor**, Neuroimaging Deep Learning, IOR
CEA	Neuron, NEST, IOR, **Cosbench**
CINECA	Neuron, NEST, IOR, Neuroimaging Deep Learning, **TensorFlow**, **Bcfind**
BSC	IOR, Neuron, NEST, Neuroimaging Deep Learning