International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Q.4068

(08/2021)

SERIES Q: SWITCHING AND SIGNALLING, AND ASSOCIATED MEASUREMENTS AND TESTS

Testing specifications – Testing specifications for IMT-2020 and IoT

Open application program interfaces (APIs) for interoperable testbed federations

Recommendation ITU-T Q.4068

ITU-T Q-SERIES RECOMMENDATIONS

SWITCHING AND SIGNALLING, AND ASSOCIATED MEASUREMENTS AND TESTS

SIGNALLING IN THE INTERNATIONAL MANUAL SERVICE	Q.1–Q.3
INTERNATIONAL AUTOMATIC AND SEMI-AUTOMATIC WORKING	Q.4–Q.59
FUNCTIONS AND INFORMATION FLOWS FOR SERVICES IN THE ISDN	Q.60–Q.99
CLAUSES APPLICABLE TO ITU-T STANDARD SYSTEMS	Q.100–Q.119
SPECIFICATIONS OF SIGNALLING SYSTEMS No. 4, 5, 6, R1 AND R2	Q.120–Q.499
DIGITAL EXCHANGES	Q.500–Q.599
INTERWORKING OF SIGNALLING SYSTEMS	Q.600–Q.699
SPECIFICATIONS OF SIGNALLING SYSTEM No. 7	Q.700–Q.799
Q3 INTERFACE	Q.800–Q.849
DIGITAL SUBSCRIBER SIGNALLING SYSTEM No. 1	Q.850–Q.999
PUBLIC LAND MOBILE NETWORK	Q.1000–Q.1099
INTERWORKING WITH SATELLITE MOBILE SYSTEMS	Q.1100–Q.1199
INTELLIGENT NETWORK	Q.1200–Q.1699
SIGNALLING REQUIREMENTS AND PROTOCOLS FOR IMT-2000	Q.1700–Q.1799
SPECIFICATIONS OF SIGNALLING RELATED TO BEARER INDEPENDENT CALL CONTROL (BICC)	Q.1900–Q.1999
BROADBAND ISDN	Q.2000–Q.2999
SIGNALLING REQUIREMENTS AND PROTOCOLS FOR THE NGN	Q.3000–Q.3709
SIGNALLING REQUIREMENTS AND PROTOCOLS FOR SDN	Q.3710–Q.3899
TESTING SPECIFICATIONS	Q.3900–Q.4099
Testing specifications for next generation networks	Q.3900–Q.3999
Testing specifications for SIP-IMS	Q.4000–Q.4039
Testing specifications for Cloud computing	Q.4040–Q.4059
Testing specifications for IMT-2020 and IoT	**Q.4060–Q.4099**
PROTOCOLS AND SIGNALLING FOR PEER-TO-PEER COMMUNICATIONS	Q.4100–Q.4139
SIGNALLING REQUIREMENTS AND PROTOCOLS FOR IMT-2020	Q.5000–Q.5049
COMBATING COUNTERFEITING AND STOLEN ICT DEVICES	Q.5050–Q.5069

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T Q.4068

Open application program interfaces (APIs) for interoperable testbed federations

Summary

Recent technological developments concerning Internet have become more complex to test and to use in the real world. A larger diversity of conditions must be addressed, and scalability needs to be assessed. The need to do experimentation within testbeds has become more important for testing new use cases in real conditions. This evolution increases the need for and practice of federating and interconnecting different testbeds. However, this powerful approach lacks the availability of clearly standardized application program interfaces (APIs) to support such federation of existing testbeds and resources to support experimentation, testing and validation of new technologies, services and solutions in order to enhance the interoperability of testbeds.

Recommendation ITU-T Q.4068 presents a set of open APIs for interoperable testbed federation able to manage not only the interconnection and the interoperability of testbeds in a federation, but also to handle resources advertisement, allocation and provision. The APIs are designed to manage the users involved in the federation such as the experimenters and to assign roles to the users. In the same way, the usage of a resource is attributed to an experimenter through the open APIs for interoperable testbed federation.

History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T Q.4068	2021-08-29	11	11.1002/1000/14765

Keywords

Conformance and interoperability, open API, testbed, testing.

* To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents/software copyrights, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the appropriate ITU-T databases available via the ITU-T website at http://www.itu.int/ITU-T/ipr/.

Table of Contents

Recommendation ITU-T Q.4068

Open application program interfaces (APIs) for interoperable testbed federations

1 Scope

This Recommendation provides a generic reference model for testbeds federation and describes the elements of this reference model.

This Recommendation contains a technical framework consisting of guidelines, which provides a common reference for developers in order to facilitate the implementation and promotion of interoperability of testbeds. It more specifically:

– defines the potential of improvements of testbed interoperability and federation to enhance the interoperability of testbeds;

– describes a reference model and technical framework for interoperable testbeds federation;

– defines the requirements of open application program interfaces (APIs) for testbed federation;

– illustrates the instantiation of the generic model through open APIs;

– specifies open APIs enabling interconnection and interoperability among different testbeds;

– provides reference metrics to be standardized in terms of data format, in order to ease the integration and interoperability of testbeds.

NOTE – The security of the testbed federations is out of scope of this Recommendation. It will be defined in a standalone document.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

None.

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

3.1.1 cipher key [b-ITU-T Q.1743]: A code used in conjunction with a security algorithm to encode and decode user and/or signalling data.

3.1.2 credential [b-ITU-T Y.2720]: An identifiable object that can be used to authenticate the claimant is what it claims to be and to authorize the claimant's access rights.

3.1.3 member [b-ISO/IEC 17000]: Body that operates under the applicable rules and has the opportunity to take part in the management of the system or scheme.

3.1.4 resource [b-ITU-R BT.1699]: A network data object or a service which is uniquely identified in a network. – A well-defined capability or asset of a system entity, which can be used to contribute to the realization of a service. Examples: MPEG decoder, graphics system.

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

3.2.1 project: Group of experimenters, experiments and resources used for a defined purpose.

3.2.2 sliver: A container for one or several physical resources or for one or several virtual resources provided by a single testbed.

NOTE – An active sliver is a sliver running a program provided by the experimenter.

3.2.3 testbed: Platform to realise scientific tests within new technologies on an environment fully controlled by experimenters.

3.2.4 testbed slice: Collection of slivers available in different testbeds.

NOTE – A testbed slice can be simply called a slice. Indeed, the term "testbed slice" is interchangeable with "slice" in the whole Recommendation. The notion of "testbed slice" is defined in this Recommendation to avoid confusion with network slice defined in the context of 5G core. Indeed, the definition of slice for testbed is anterior to the definition of network slice[1].

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

AMC	Autonomic Management and Control
API	Application Programming Interface
BSS	Business Support System
CSP	Communications Service Provider
E2E	End to End
EMS	Element Management System
GANA	Generic Autonomic Networking Architecture
ICT	Information and Communications Technologies
IoT	Internet of Things
IP	Internet Protocol
ISV	Independent Software Vendor
KP	Knowledge Plane
MANO	Management and Orchestration
MEC	Multi-Access Edge Computing
NE	Network Element
NF	Network Function
NFV	Network Functions Virtualization
NMS	Network Management System
NUT	Network Under Test

1 Key GENI Concepts: https://groups.geni.net/geni/wiki/GENIConcepts#Slice, 23 January 2014

ONIX Overly Network for Information exchange
OSS Operations Support System
PEM Private-Enhanced Mail
PKI Public Key Infrastructure
PNF Physical Network Function
QoS Quality of Service
RAN Radio Access Network
RFC Request for Comments
RPC Remote Procedure Call
RSA Rivest-Shamir-Adleman
RSpec Resource Specification
SDN Software Defined Networking
SDO Standards Development Organization
SLA Service Level Agreement
SSL Secure Sockets Layer
SUT Systems Under Test
TaaS Testbed-as-a-Service
URL Uniform Resource Locator
URN Uniform Resource Name
UTC Coordinated Universal Time
VNF Virtualized Network Function
XML extensible Markup Language
WSN Wireless Sensor Network

5 Conventions

None.

6 Generic reference model for testbeds federations and key players

Rationale on the need for federated testbeds (stakeholders' perspectives)

Today there are many testbeds available for research purposes, and many testbeds continue to be built by both research communities and by the Industry. The Industry continues to build its own testbeds that are used internally within organizations such as network operators (or communications service providers) or vendors organizations, and in some cases some industrial testbeds can be used in multiple organizations based on certain collaboration agreements that are closed only to the partners.

Over the years it has been increasingly the experience that single isolated standalone testbeds are not sufficient to test and trial out certain technology use cases because the use cases require the use of components and resources located in different testbeds (due to the varying capabilities of the different testbeds that need to be used). New information and communications technologies (ICTs), networks and industry-oriented applications are becoming increasingly complex to test using

standalone testbeds. Hence, federated testbeds bring sustainability in fostering environments for quick innovations and the testing of complex technologies and use cases, and for enabling quicker innovations and quicker time to market for products and services.

In this regard, federated testbeds may bring a lot of value to 'research use-cases' and "industry real technology deployment use cases". In general, there is an urgent need to build an ecosystem for enabling sustainable testbeds development, evolutions, and federations. Federated testbeds then bring a lot of value to research use cases and industrial use cases. However, standards have been lacking in this increasingly very important area of testbeds federations and interoperability. To enable the development of standards for testbeds federations, a generic reference model for testbeds federations first needs to be developed. This should be shared among various SDOs/Fora that are working on standards for testbeds, so as to help achieve harmonized and interoperable testbeds specifications and instantiations of the reference model in the development of various types of testbeds across the multi SDOs/Fora based on a commonly shared reference model for testbeds federations as the blueprint.

This implies the need for an ecosystem to be established around the testbeds federations reference model, its associated APIs, and its instantiations and use cases, in order to facilitate the engagements of various stakeholders required to be part of the ecosystem. Examples of such stakeholders are: SDOs/Fora, research communities, researchers on new technologies such as 5G and beyond, industry users of testbeds, testbeds suppliers for emerging technologies, communications service providers (CSPs), network operators, infrastructure vendors/suppliers for ICT and verticals, open source and open hardware projects, and regulators.

Other benefits of a reference model for testbeds federations include:

- The reference model and associated APIs serve as an enabler for automation for testbeds discovery and testbeds services offerings, and testbeds interoperability.
- New business models for testbeds suppliers that derive from the testbeds federations reference model and associated APIs, such as testbed-as-a service (TaaS).
- Usage of context of a testbed and federated testbeds could be considered for certification purposes or other purposes.
- The standardized testbeds federations reference model is useful for aligning and streamlining various stakeholder architectures to strengthen the ecosystem and value-chain for increased collaboration and leveraging of testbeds for 5G and beyond technologies among stakeholders (CSPs, vendors, testbed suppliers, open-source and open hardware projects and 'disruptive players', and other players).

For communication services providers (CSPs) such as network operators for 5G and beyond technologies, federated testbeds built on the basis of the reference model for testbeds federations can play a very important role in testing emerging autonomic/autonomous networks (ANs) technologies such as:

- Intra-CSP federations of autonomic/autonomous networks autonomic management and control (AMC) platforms such the ETSI GANA knowledge plane (KP) platforms [b-ETSI TS 103 195-2] across network segments of an AMC platform's responsibility, and inter-CSPs AMC platforms federations as they inherently and correspondingly require federated testbeds to test the cross-domain AMC operations.
- CSPs sometimes need to leverage and use testbeds/testbed assets from other stakeholders to build an E2E network (e.g., a 5G E2E network) and its corresponding AMC platforms (e.g., KPs) in order to test the federated AMC operations across the various network domains (e.g., AMC operations such as E2E self-optimization of AN resources, self-protection and self-defence against detected security attacks, threats and risks to address security challenges that have impact on various network domains).

- Intra-CSP federations of ANs' AMC platforms across network segments and inter-CSPs ANs' AMC platforms federations as the two inherently and correspondingly require federated testbeds to test cross-domain AMC operations.

More detail on the use case for federated testbeds for testing ANs is presented in Appendix I.

The generic reference model for testbeds federations is presented in Figure 1. The reference model is described in this clause.

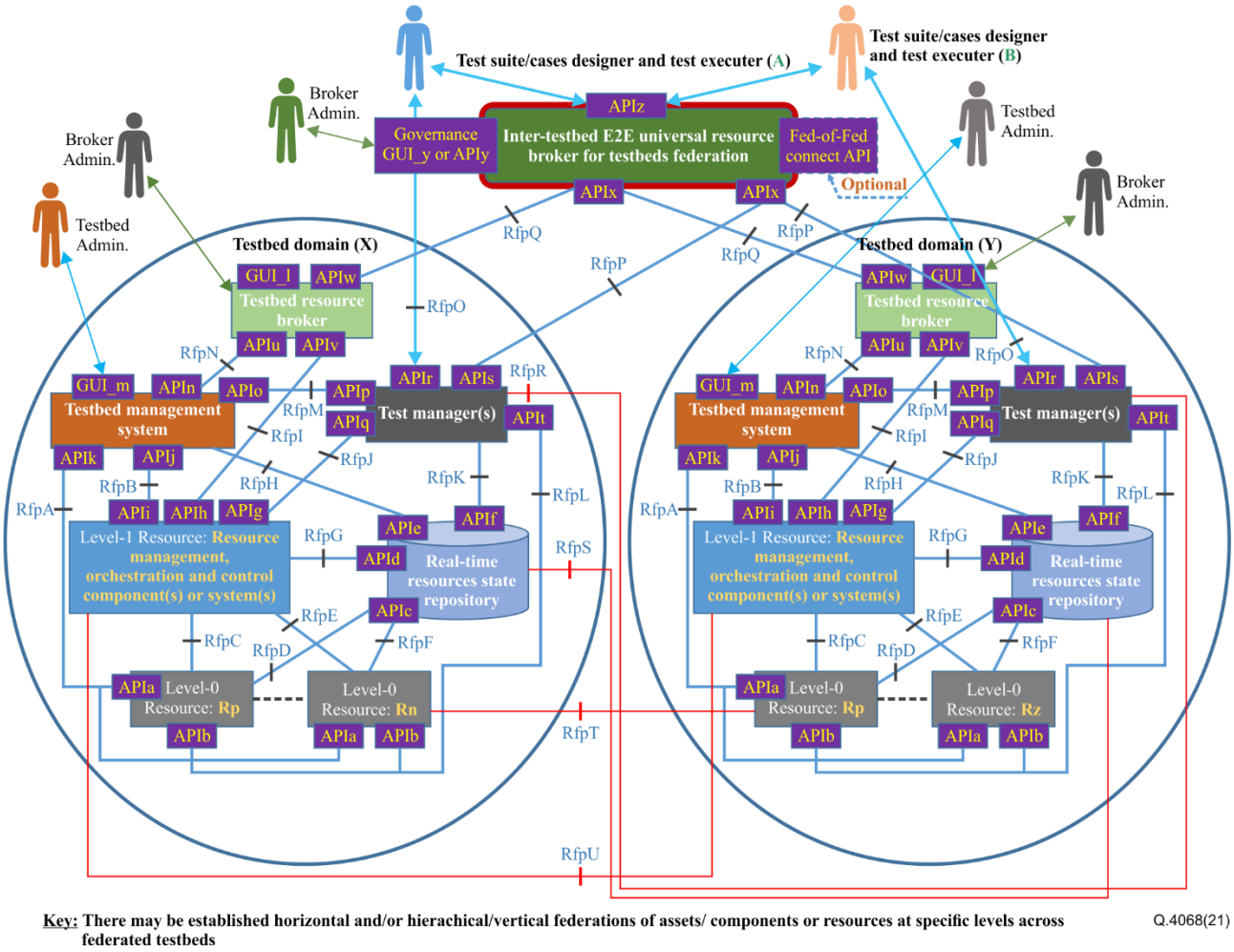

Figure 1 – Generic federated testbed model

NOTE 1 – The generic model (and its reference points (Rfps) and APIs), as shown in Figure 1, is agnostic to communication technologies and media types that may be used in communications between functional blocks (FBs) and to implement the FBs, as is expected of a reference model.

The reference model consists of the following functional blocks (i.e., building blocks), associated reference points (Rfps) and APIs that may be used to implement the reference points between functional blocks (FBs), and it also indicates the players/actors that may be involved:

1) **Testbed domain**: A testbed domain models an abstraction (i.e., concept) of a testbed, independently of type of a testbed, and with capability for getting federated with another testbed or testbeds to form a federation of testbeds. This means the model of a 'federaterable' testbed is represented generically by the concept of a testbed domain that can be instantiated to create a specific type of a testbed such that the functional blocks and reference points (Rfps) get to be further detailed (during instantiation) by implementation specific aspects and details that need to be considered when creating a testbed of a specific type (kind) such as a radio access network (RAN) testbed, a multi-access Edge computing (MEC) testbed, an Internet of things (IoT) testbed, etc. A testbed domain consists of the

following FBs (that can be treated as components) and can communicate with other FBs through the reference points (Rfps) defined 1n the diagram. The reference points need not necessary be implemented by an application programming interface (API) but may be implemented using protocols. The Rfps are 'named' by indexing, as are the APIs. The APIs indicated on each FB are means by which the FB in question provides 'services' to the FB on the other end of the reference point (Rfp). The FBs of the testbed domain concept are described below:

- **Testbed resource broker**: This represents an FB (e.g., component) that dynamically builds a picture of the capabilities and associated resources of the testbed that can be requested for by testbed users to be utilized by the users in test scenarios in standalone manner (within the confines of the testbed domain only) or in federations with other capabilities and resources exhibited or hosted in other testbed(s) such that the broker is the entry point for testbed users to request for the capabilities and resources through the umbrella **inter-testbed E2E universal resource broker** for testbeds federation that collectively exposes to prospective users the capabilities and resources available in different testbeds under its responsibility through a unified interface presented to the users. Requests by users can be accepted or denied by the testbed resource broker.
- **Testbed management system**: This represents an FB (e.g., system) that provides an interface such as a graphical user interface (GUI) to a testbed administrator/admin (Human) so that the testbed admin can manage the testbed to prepare the testbed in such a way that the testbed can provide testbed services to prospective users and can also participate in testbed federations with other testbeds and the **inter-testbed E2E universal resource broker** for testbeds federation. The testbed management system is used by the testbed administrator in establishing connectivity of the testbed with other testbeds that should be interconnected with this testbed in order to provide federated capabilities and resources to prospective users of federated testbeds. All assets of the testbed are managed by the testbed administrator via the testbed management system. Through the testbed management system, mainly, the testbed administrator decides and makes those FBs (components of the testbed) that can be connected with FBs in another testbed be connected according to some policies and collaborations SLAs that govern the connectivity and federations that may be allowed between the testbed and another testbed(s). All assets (FBs/Components) defined and shown for the testbed domain concept register themselves into the testbed management system and supply monitoring data to the testbed management system that help the testbed management system to keep track of the status and performance of the assets in real time. Assets can be reconfigured or switched off using the testbed management system, apart from providing the testbed administrator to directly access and manage the assets without the involvement of the testbed management system.
- **Test manager(s)**: This represents an FB (in the form of a system) that is used by testbed users to design, compile and execute test cases. A testbed user gets access to a test manager instance granted by the **testbed resource broker** of the targeted testbed upon the acceptance of the request the user sent earlier to the testbed resource broker (via the **inter-testbed E2E universal resource broker for testbeds federation**). The testbed user is then able to upload some test cases or test suites if the testbed domain allows that and then compile and/or execute the test cases, or the user is only allowed to design, compile and execute test cases directly on the test manager without uploading test cases/suites from outside. This all depends on the capabilities and policies of the testbed domain.
- **Level-1 resource, which can be considered as a resource management, orchestration and control component(s) or system(s)**: This pertains to an FB that represents a resource that can either be considered in a certain test scenario as a

component under test (CUT) or system under test (SUT), or in some test scenarios can be considered as a resource required to participate in the test scenario or test case without itself being a CUT or SUT. A Level-1 resource of this nature is positioned at a level outside of the underlying network infrastructure level of a network architecture and its management and control architecture, and so is positioned at the level of the management and control architecture of the underlying network infrastructure (i.e., the level where network elements/functions (NEs/NFs) are positioned). Examples of such a Level-1 resource are OSS/BSS, ETSI GANA knowledge plane (KP) platform, E2E service universal orchestrator, domain controller, SDN controller, and more examples provided later in this present document. In order to serve multiple test requirements and test scenarios that may be required to be executed in the testbed domain by independent testbed users, the concept of testbed slices or tenants and resource slices or tenants should be supported by the testbed domain, and the slicing of a Level-1 resource may be required in some scenarios. The concept of multi-level slicing, that starts from the Level-0 resource (see description of Level-0 resource below) up to the testbed domain level itself, is applicable to the testbed domain concept. The same applies to multi-tenancy to support multiple independent testbed users that can be offered a slice of certain granularity and composition by the testbed domain and run their tests independently of each other.

- **Level-0 resource**: This pertains to an FB that represents a resource that can either be considered in a certain test scenario as a component under test (CUT) or system under test (SUT), or in some test scenarios can be considered as a resource required to participate in the test scenario or test case without itself being a CUT or SUT. A Level-0 resource of this nature is positioned at a level of the underlying network infrastructure level of a network architecture and its management and control architecture. That is to say such a level-0 resource is within the level where network elements/functions (NEs/NFs) are positioned. Examples of such a Level-0 resource are a protocol, network element/function (NE/NF) i.e., a physical network function (PNF) or virtualized network function (VNF) in general, protocol stack or stacks as a whole, network switch, router, base station, and more examples provided later in this present document. In order to serve multiple test requirements and test scenarios that may be required to be executed in the testbed domain by independent testbed users, the concept of testbed slices or tenants and resource slices or tenants should be supported by the testbed domain, and the slicing of a Level-0 resource may be required in some scenarios. The concept of multi-level slicing, starting from the Level-0 resource even up to the testbed domain level itself, is applicable to the testbed domain concept, and the same applies to multi-tenancy to support multiple independent testbed users to be offered a slice of certain granularity and composition by the testbed domain and run their tests independently of each other.
- **Real-time resources state repository**: This pertains to an FB that represents real-time repository (inventory) of the resources of the testbeds that can either be considered in a certain test scenario as a component under test (CUT) or as a system under test (SUT), or in some test scenarios can be considered as a resource required to participate in the test scenario or test case without itself being a CUT or SUT. Such resources (a collection of Level-1 and Level-0 resources) of the testbed domain constitute the functional capabilities and resources of the testbed domain that can be required in a standalone manner or in federation with capabilities and resources in other testbed domains to fulfil some testing requirements of prospective testbed users. The state of Level-1 and Level-0 resources need to be continuously tracked in real time so that resources available to serve test requirements conveyed by incoming requests from testbed users can be known by the testbed resource broker in real-time. Other FBs of the testbed domain may query the repository for the state of various resources. For

example, by querying the state of resources currently serving the test cases running in the testbed, the testbed resource broker can create and maintain knowledge about the capacity to accommodate new test requirements or requests from testbed users and interact with orchestrators to create new slices of resources to serve new coming demands.

2) **Inter-testbed E2E universal resource broker for testbeds federation**: This pertains to an FB that provides the primary entry point into the system of federated testbeds to prospective users (testbed users) of the system of federated testbeds. It provides 'search and query and find services' that enable the prospective user of testbed service(s), i.e., the **test suite/cases designer and test executer** to find/discover testbeds that are available, to accept new requests within the time of interest to the prospective testbed user as well as their capabilities topology information pertaining to their interconnection and federations with other testbeds. The **inter-testbed E2E universal resource broker for testbeds federation** maintains an information inventory about testbeds under its testbeds federation responsibilities. A prospective testbed user (**test suite/cases designer and test executer**) can query the broker for testbeds that fulfil certain capabilities and requirements such as end-to-end latency within the scope of the single testbed or across multiple testbeds, before the prospective user can then select testbeds and launch requests for testbed services. The E2E resource broker provides a governance GUI or API to a **broker admin** to manage the resource broker, e.g., by configuring policies that govern its operations and processing and admitting or rejecting requests for testbed services coming from prospective testbed users. Testbed domains register themselves with the **inter-testbed E2E universal resource broker for testbeds federation** through their testbed resource brokers. Capabilities description information about test managers available within the testbed domain are advertised to the E2E resource broker by the test managers themselves so that a prospective testbed user can also query the E2E resource broker about the types and capabilities of test managers available in the testbed domain. However, it is only upon an acceptance of a testbed service request from a prospective testbed user (through the E2E resource broker) by the testbed resource broker of the testbed domain that the testbed resource broker provides to the E2E resource broker information about actual instances of test manager(s) in the testbed domain. The testbed user can then connect to the specific test manager(s). The E2E resource broker may also connect to the test managers through an API dedicated for such a need, e.g., to enable the E2E resource broker to access state information about a specific test manager, or for cases whereby some test results could be shared with the testbed user via the E2E resource broker. If it is not possible that the test manager provides direct access to those kinds of test results directly to the user (test executor), though Test Executor should primarily be able to access test results directly from the test manager(s). The **inter-testbed E2E universal resource broker for testbeds federation** as a hierarchical E2E resource broker that covers a certain scope of testbed domains may be connected (optionally) to a higher level **inter E2E brokers global universal resource broker for testbeds federation** through its **fed-of-fed connect API** if it implements such an API, in order for the **inter-testbed E2E universal resource broker for testbeds federation** to participate in a larger global system of 'federations of federations' as described later in this clause.

3) **Roles or players/actors**: These represent the human roles or players/actors that interact with some FBs of a testbed domain. Such roles or players/actors are described below:

- **Testbed admin**: This represents a role or actor/player that uses an interface, e.g., a GUI, provided by the **testbed management system** to manage the testbed to prepare the testbed in such a way that the testbed can provide testbed services to prospective users and can also participate in testbed federations with other testbeds and with the **inter-testbed E2E universal resource broker for testbeds federation**. The testbed

admin uses the testbed management system in establishing connectivity of the testbed with other testbeds that should be interconnected with this testbed in order to provide federated capabilities and resources to prospective users of federated testbeds. Through the testbed management system, mainly, the testbed administrator decides and makes FBs (e.g., components) of the testbed that can be connected with FBs (e.g., components) in another testbed be connected according to some SLA policies and collaborations that govern the connectivity and federations that may be allowed between the testbed and other testbed(s).

- **Broker admin**: This represents a role or actor/player that uses an interface e.g., a GUI, provided by the **testbed resource broker** to manage the testbed resource broker, e.g., by configuring policies that govern its operations and processing and admitting or rejecting requests for testbed services coming from prospective testbed users.
- **Test suite/cases designer and test executer**: This represents a role or actor/player that uses an API and/or associated GUI provided by the **inter-testbed E2E universal resource broker for testbeds federation** to request for testbed services to be delivered to the actor/player (regardless of whether the requested services are to be delivered by one testbed or multiple federated testbeds under the responsibility of the **inter-testbed E2E universal resource broker for testbeds federation.** The role is presented as a combined role for a test designer and test executor but may be split accordingly. The test suite/cases designer and test executer also uses other services provided by the **inter-testbed E2E universal resource broker for testbeds federation**, e.g., 'search and query and find services' that enable the prospective user of testbed service(s), i.e., the **test suite/cases designer and test executer** to find/discover testbeds that are available to accept new requests within the time of interest to the prospective testbed user as well as their capabilities topology information pertaining to their interconnection and federations with other testbeds.

The FBs have reference points (Rfps) defined for their interaction with other FBs as shown in the figure. The Rfps may be solely implemented by way of the APIs defined at the endpoints of the specific Rfps.

NOTE 2 – The Reference Points (Rfps) between FBs that reside in different testbed domains are a subject for further study and description.

The APIs indicated on each FB are a means by which the FB in question provides 'services' to the FB on the other end of the reference point (Rfp). Table 1 provides a high-level description of the APIs.

NOTE 3 – The detailed specification of the APIs is work that is not in the scope of this document, and the work on specification of the APIs is yet to be carried out.

Table 1 – High-level description of the APIs

Count	API name (identification tags)	Short description
1	APIa	The API is meant for providing descriptive information about the resource, its state and usage in real-time upon the invocation of the API by the testbed management system.
2	APIb	The API is meant for enabling a test manager to interact with the resource, in cases where the resource is either a component under test (CUT) or system under test (SUT), or provides an interface that can be used by a Test System to configure it, such that the test manager may configure the resource as may be required for some test scenario, and/or pull test results from the resource after a completion of a test.

Table 1 – High-level description of the APIs

Count	API name (identification tags)	Short description
3	APIc	The API is meant for use by Level-0 resources to dynamically provide information in real-time about their state in terms of usage and other information such as performance data and workload being sustained on the resource.
4	APId	The API is meant for use by Level-1 resources to dynamically provide information in real-time about their state in terms of usage and other information such as performance data and workload being sustained on the resource.
5	APIe	The API is meant for use by testbed management system to enable testbed admin to pull and view (visualize) information about the State of resources (Level-0 and Level-1) from the real-time state repository, especially when the testbed admin intends to view the state of certain resources before deciding to cause connectivity to be established among resources in the testbed and/or establishing connectivity to resources in other testbeds.
6	APIf	The API is meant for use by test manager to pull information about Level-0 and Level-1 Resources that may be required to participate in a certain test scenario a testbed user may want to execute. The APIs is also meant for use by a Test Manager to pull Information about the state of resources (Level-0 and Level-1) from the real-time state repository, especially when state of a resource plays a role during the execution of certain test cases or when the test manager may require to use the state of certain resources in deciding to cause connectivity to be established among resources in the testbed and/or establishing connectivity to resources in other testbeds.
7	APIg	The API is meant to be used by a test manager to execute certain test cases that are meant to test the resource during a scenario in which the resource is a system under test (SUT) or a component under test (CUT). The API is also meant for use by a test manager to request the resource to execute a certain behavior that is required by a test case(s) being executed by the test manager, including configuring some Level-0 resource(s) that can only be configured via the Level-1 resource.
8	APIh	The API is meant for use by the testbed resource broker to gather information about the capabilities of the resource, its state and its availability to serve testbed services request(s) received from prospective testbed user(s). In case the resource is an orchestrator of resources (Level-1 and/or Level-0), then the same API is also used by the testbed resource broker to request the resource for orchestration of instance(s) of certain Level-1 resource(s) and/or Level-0 resource(s) as Slices that are required to fulfil requirements of a testbed service request received from a prospective testbed user, when there is no existing instance (slice) of the required type of resource(s) that is already available to fulfil the requirements of the newly received request for a prospective testbed user. The same API is also used by testbed broker to obtain information about capabilities and state of resources directly under the management and control responsibility of the Level-1 resource.

Table 1 – High-level description of the APIs

Count	API name (identification tags)	Short description
9	APIi	The API is meant for providing descriptive information about the resource, its state and usage in real-time upon the invocation of the API by the testbed management system.
10	APIj	The API is meant for providing updates to descriptive information about the resource, its state and usage in real-time upon the invocation of the API the Level-1 resource, whenever there are changes that have occurred on the resource.
11	APIk	The API is meant for providing updates to descriptive information about the resource, its state and usage in real-time upon the invocation of the API the Level-1 resource, whenever there are changes that have occurred on the resource.
12	GUI_l)	The GUI is meant for use by the broker administrator (broker admin) for performing all necessary broker governance and management activities and operations on the broker. For example, the broker admin installs the policies that govern the operation of the broker in terms of how it registers the testbed domain to the inter-testbed E2E universal resource broker for testbeds federation. It also installs policies that govern the services offered to prospective users of testbeds (including the policies for testbed usage by prospective users). Via the GUI (API), the broker admin can manage the broker to prepare the broker in such a way that the broker can register with the inter-testbed E2E universal resource broker for testbeds federation, such that the broker is ready to provide its services to prospective testbed users.
13	APIm (or GUI_m)	The API is meant for use by the testbed administrator (testbed admin) for performing all necessary Testbed management activities and operations. Via the GUI (API), the testbed admin can manage the testbed to prepare the testbed in such a way that the testbed can provide testbed services to prospective Users and can also participate in testbed federations with other testbeds and the inter-testbed E2E universal resource broker for testbeds federation. The GUI of the testbed management system is used by the testbed administrator in establishing connectivity of the testbed with other testbeds that should be interconnected with this testbed in order to provide federated capabilities and resources to prospective users of federated testbeds.
14	APIn	The API is meant for use by the testbed resource broker to register itself into the testbed management system, provide descriptive state and change of state information in real-time to the testbed Management System. The API is also used by the testbed resource broker to obtain descriptive information about all resources and other entities of the testbed domain and their capabilities descriptions (as the various resources and entities not only update the real-time state repository but the testbed management system as well, and the information is kept in sync and consistent between the testbed management system and the real-time state repository). NOTE – The testbed resource broker may use an API provided by the real-time state repository for directly pulling out information about Resources available in the testbed domain and their capabilities descriptions.

Table 1 – High-level description of the APIs

Count	API name (identification tags)	Short description
15	APIo	The API is meant for use by a test manager to register itself into the testbed management system, provide descriptive state and change of state information in real-time to the testbed management system, because a test manager could be a considered as a resource itself.
16	APIp	The API is meant for providing descriptive information about a test manager, its state and usage in real-time upon the invocation of the API by the testbed management system.
17	APIq	The API is meant for providing Test Results to a test manager(s) that involved the resource in a test case as a component under test (CUT) or system under test (SUT). The same API is also used for communicating to a test manager some feedback (e.g., errors or failures during the execution) to some invocations triggered earlier on the resource by the test manager.
18	APIr	The API is meant for use by a test suite/cases designer and test executer (upon the acceptance of its request for testbed service by the testbed resource broker) to connect to the test manager instance assigned to the testbed user to use the test manager to design, compile and run test cases, or to upload and compile some Test Cases designed offline and execute them. Through the API, the testbed user is able to upload some test cases or test suites if the testbed domain allows that and then compile and/or execute the test cases, or the user is only allowed to design, compile and execute test cases directly on the test manager without uploading test cases/suites from outside.
19	APIs	The API is meant for use by the inter-testbed E2E universal resource broker for testbeds federation to connect to the test manager, e.g., to enable the E2E resource broker to access state information about the specific test manager, or for cases whereby some test results could be shared to the testbed user via the E2E resource broker if not possible that the test manager provides direct access to those kinds of test results directly to the user (test executor), though primarily the test executor should be able to access test results directly from the test manager(s).
20	APIt	The API is meant for providing test results to a test manager(s) that involved the resource in a test case as a component under test (CUT). The same API is also used for communicating to a test manager some feedback (e.g., errors or failures during the execution) to some invocations triggered earlier on the resource by the test manager.
21	APIu	The API is used by the testbed management system to keep synchronizing with the testbed resource broker on the state of the broker. The same API is also used by the testbed management system to provide updates to any changes in the descriptive information about all resources and other entities of the testbed domain and their capabilities descriptions (Information that is kept in sync and consistent between the testbed management system and the real-time state repository).
22	APIv	The API is used by a Level-1 resource to push updated Information (updates) about state of resources under the management and control responsibility of the Level-1 resource and their capabilities descriptions, and any changes that may have occurred to the resources and capabilities. The same API is also used for synchronizations between the Level-1 resource and the testbed Resource Broker.

Table 1 – High-level description of the APIs

Count	API name (identification tags)	Short description
23	APIw	The API is meant for use by the inter-testbed E2E universal resource broker for testbeds federation, after the testbed resource broker has registered itself with it via APIx, to then obtain (pull) descriptive information about all testbed resources available in the Testbed domain to serve testbed services requests that may come from the E2E Resource Broker and their capabilities descriptions. The same API is also used by the E2E resource broker to provide synchronization related descriptive state and change of state information in real-time to the test broker.
24	APIx	The API is used by the testbed resource broker to push updated information (updates) about state of resources of the testbed domain and their capabilities descriptions, and any changes that may have occurred to the resources and capabilities. The same API is also used, complementarily to APIw, for synchronizations between the testbed resource broker and the inter-testbed E2E universal resource broker for testbeds federation.
25	APIy (or GUI_y)	The API is meant for use by the broker administrator (broker admin) for performing all necessary broker governance and management activities and operations on the Broker. For example, the broker admin installs the policies that govern the operation of the Broker in terms of admitting (or not admitting) testbed domains in their attempts to discover and register with the broker, as well as policies that govern the services offered to prospective users of testbeds registered with Broker (including the policies for testbeds user registrations). Via the GUI (API), the broker admin can manage the broker to prepare the broker in such a way that the broker can expose the APIz and any GUIs of the broker that can be made available to prospective testbeds users, such that the broker is ready to provide its services to prospective testbed users and to testbeds intending to register with it.
26	APIz	This API provides the entry point into the system of federated testbeds to prospective users (testbed users) of the system of federated testbeds. It provides 'search and query and find services' that enable the prospective user of testbed service(s), i.e., the test suite/cases designer and test executer to find/discover Testbeds that are available to accept new requests within the time of interest to the prospective testbed user as well as their capabilities topology information pertaining to their interconnection and federations with other testbeds. A prospective testbed user (test suite/cases designer and test executer) can query the broker for testbeds that fulfil certain capabilities and requirements such as end-to-end latency within the scope of the single testbed or across multiple testbeds, before the prospective user can then select testbeds and launch requests for testbed services.

NOTE 4 – The APIs have been named (to have a way to distinguish them).

NOTE 5 – Elements of the APIs for security perspectives may be provided by groups such as ITU-T SG17. The security of the testbed federations is out of scope of this Recommendation.

Level-1 Types of resources are illustrated as follows:

Examples of Level-1 resource that could be used within a test scenario are listed below:

A resource could embed a test agent or not. A test agent could participate for a test case itself which could send results to the test manager. Figure 2 shows a Level-1 resource.

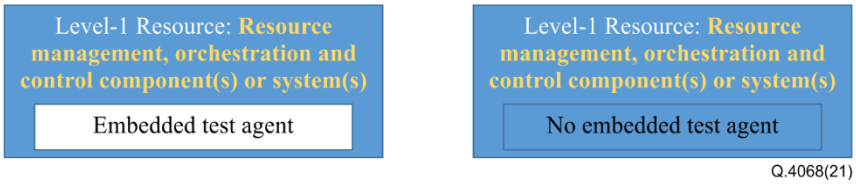

Figure 2 – Level-1 resource

- ETSI GANA knowledge plane (KP) platform as whole with KP decision elements (DEs), MBTS, an overlay network for information exchange (ONIX) system of federated information servers, see [b-ETSI TS 103 195-2]
- OSS/BSS
- E2E service universal orchestrator
- Domain controller
- SDN controller
- MANO stack
- Big data analytics platform
- GANA ONIX as standalone system
- Performance monitoring component or system
- EMS/NMS
- Fault management system
- Configuration manager component
- Security management system

NOTE 6 – In some test scenarios a resource may itself be considered as a CUT or SUT or may simply be required to participate in a test scenario.

Level 0 types of resources are illustrated as shown in Figure 3.

Figure 3 – Level-0 resource

Examples of Level-0 resource are listed below:

NOTE 7 – The generic model and its reference points (Rfps) and APIs, as shown in Figure 1, is agnostic to communication technologies and media types that can be used at Level-0 resources and their implementations, as is expected of a reference model.

- Protocol
- Network element/Function (NE/NF) – PNF or VNF in general
- Protocol stack or stacks as a whole
- Network switch
- Router
- Base station

- RAN component
- Bridge
- Signalling gateway
- Database
- IP Host with a client application part
- Network functions virtualization infrastructure (NFVI) host platform
- Network (presenting itself as single logical entity)
- Application in general
- Application server
- Load balancer
- Firewall or security gateway, intrusion prevention system (IPS), intrusion detection system (IDS), etc.
- Passive monitoring probe
- Active monitoring probe

NOTE 8 – In some Test Scenario a Resource may itself be considered as a Component Under Test (CUT) or System Under Test (SUT) or may simply be required to participate in a Test Scenario.

Specialization of the testbed domain concept: A specialization of the testbed domain concept is described in Figure 4.

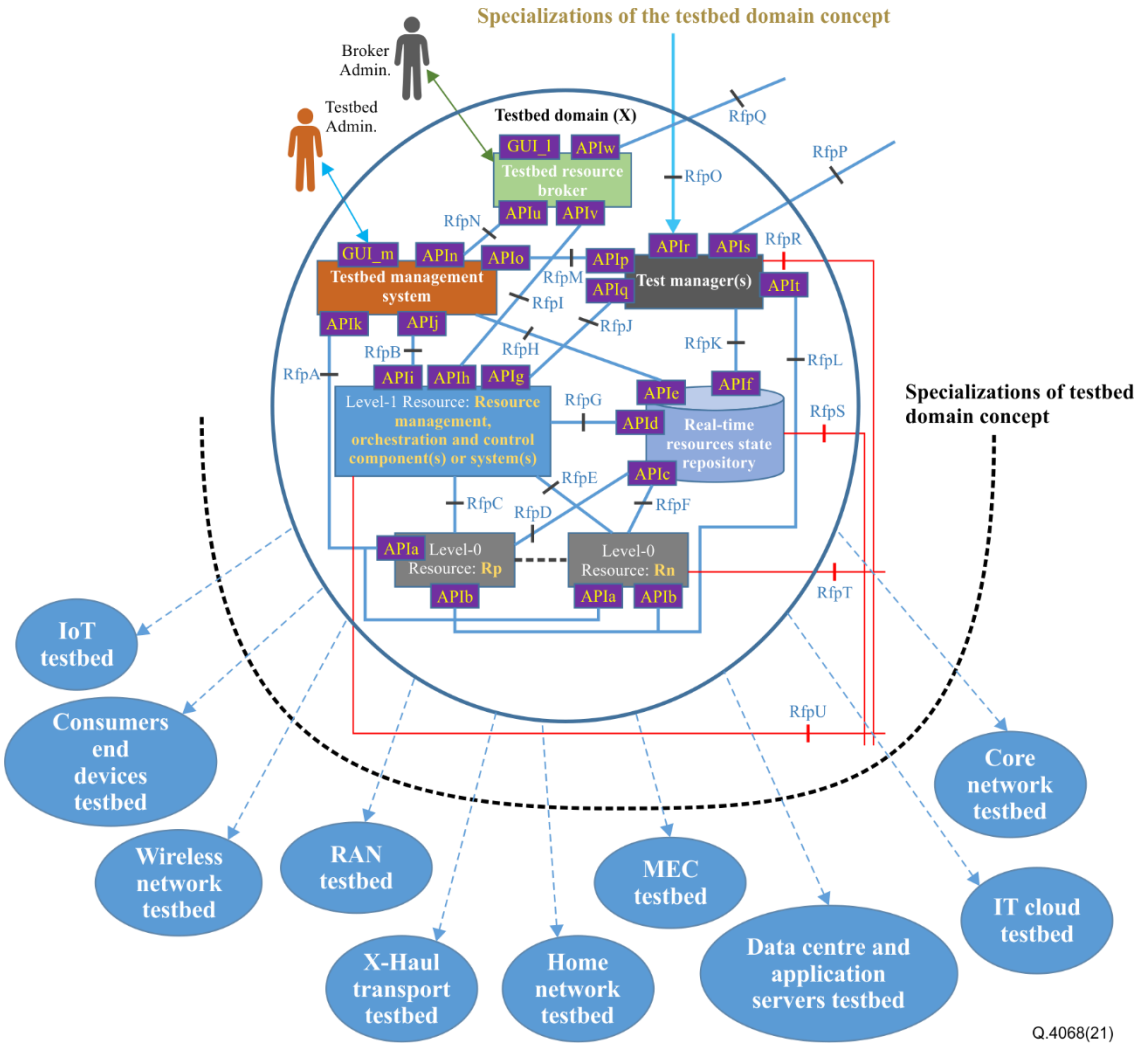

Figure 4 – Specialization of the testbed domain concept

Figure 4 shows that the testbed domain concept should be used in the creation, designing and building of a testbed that is meant to be federated with other testbeds, such that the resultant testbed should be seen as a specialization of the testbed domain concept when it implements the functional blocks (FBs), reference points (Rfps) and APIs prescribed by the testbed domain concept and generic federated testbed model presented in Figure 1.

Figure 5 describes the case of 'federations of federations' in testbeds, a level above the level of the universal E2E resource broker. In the case of federations of federations in testbeds the **Inter-Testbed E2E universal resource broker for testbeds federation** as a hierarchical E2E resource broker that covers a certain scope of testbed domains gets connected to a higher level **inter E2E brokers global universal resource broker for testbeds federation** through its **Fed-of-Fed connect API** if it implements such an API, in order for the **inter-testbed E2E universal resource broker for testbeds federation** to participate in a larger global system of 'federations of federations'. Federations of federations in testbeds are a way to expose testbeds services to global testbeds users within geographical scope for which the federated testbeds are meant to be used in adherence to certain constraints on the testbeds within the scope of the system, e.g., end-to-end QoS requirements (e.g., end-to-end latency, throughput, etc.) between testbeds within the scope of the system of federated testbeds, access constrains and governance and or regulatory policies that restrict the usage of the testbeds by prospective testbeds users from certain geographical locations. Some testbeds may be organized by way of factors such as proximity among testbeds, E2E QoS requirements across the federated testbeds, and governance and regulatory policies, into forming a federation system of testbeds that then exposes itself via a hierarchical **inter E2E brokers global**

universal resource broker for testbeds federation, in order to provide testbeds services to certain prospective testbeds users that may be required to access the testbeds via the upper hierarchical **inter E2E brokers global universal resource broker for testbeds federation.** This means that an **inter E2E brokers global universal resource broker for testbeds federation** may be associated with a geographical or administrative domain of scope of federated testbeds, e.g., one could be scoping a whole country or region within a country or scoping multiple countries or administrative and regulatory boundaries.

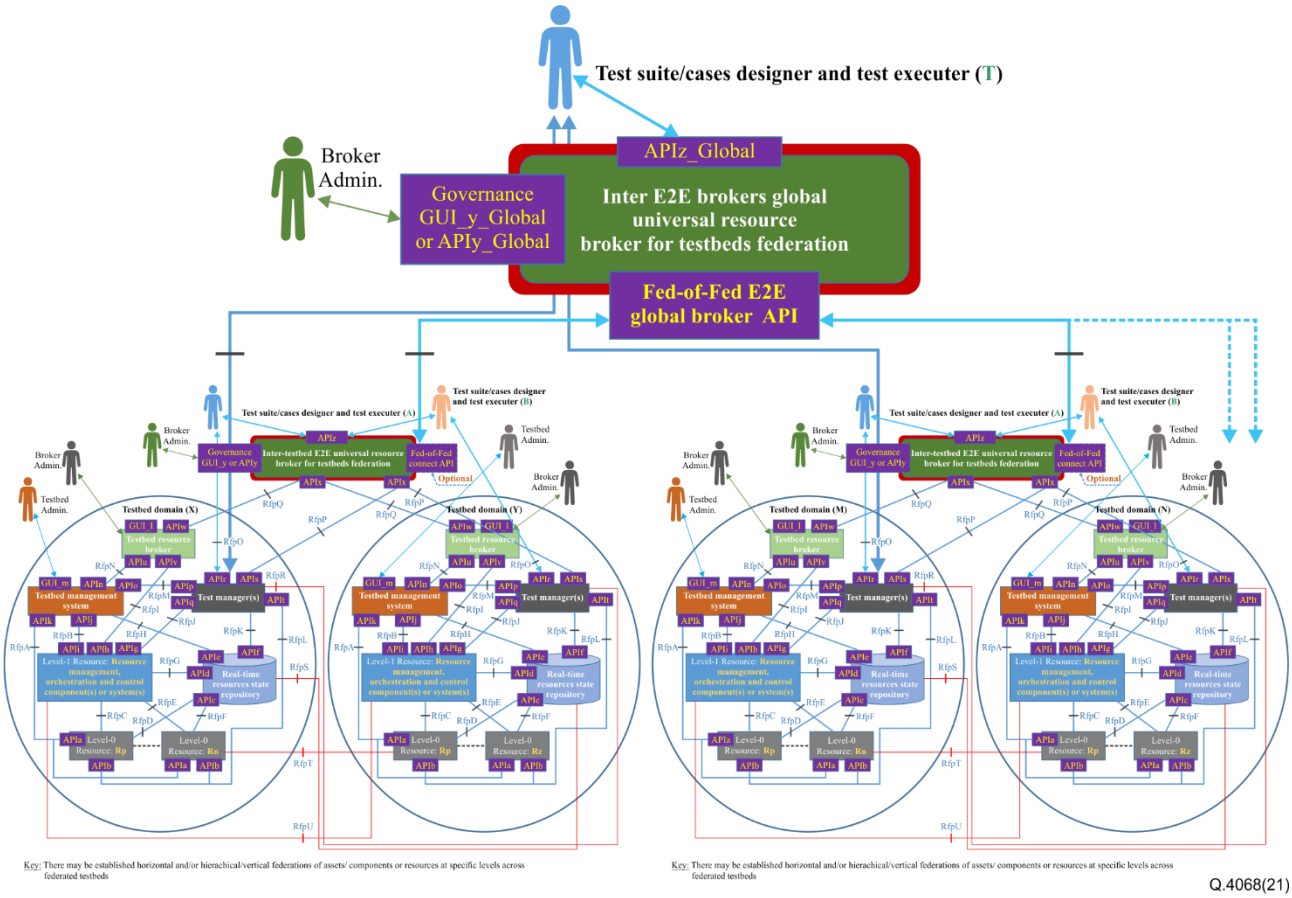

Figure 5 – The case of 'federations of federations', a level above the level of the universal E2E resource broker

The types of APIs required in the overall testbeds federation reference model are described below:

a) **Federation generic APIs**: These capture common methods (procedures) and attributes that would be seen as being extended by some APIs at 'concrete Testbed and resource level' (e.g., for RAN testbed, core network testbed). Common methods and attributes of the federation generic APIs are agnostic to the type of testbed. The work on specifying the generic APIs should also be inspired by any already existing APIs and implementation experiences from the various testbeds R&D projects (past, present and future) to help capture 'common methods and attributes' that should be agnostic to testbed type while leaving out the methods and attributes that are testbed-specific for specialized APIs in specific testbeds types that are then expected to get invoked by the generic APIs.

b) **Testbed-specific specialized APIs in specific testbeds types**: These extend the federation generic APIs by inheriting the generic APIs and their common methods (procedures) and adding additional methods and attributes that are specific to the type of testbed, or they simply get invoked by the generic APIs.

NOTE 9 – The detailed specifications of APIs of the reference model (as presented in Figures 1 to 5 and the associated descriptions) are outside the scope of this present document, and it is expected that separate documents on this subject would be developed.

- **APIs invocations approaches**: The generic APIs should invoke testbed-specific specialized APIs in specific testbeds types during execution of use cases for testbeds and testbeds federations.

Instantiations of the reference model for testbeds federations:

- The generic model (the reference model) for testbeds federations, as presented in Figures 1 to 5 and the associated descriptions, should be instantiated in creating, designing and building specific testbeds types as discussed earlier in the context of specializations of the testbed domain concept.

NOTE 10 – The subject of instantiations of the reference model is outside the scope of this present document, and it is expected that separate documents on this subject would be developed.

Roles of various stakeholders around the testbeds federation reference model

Stakeholders that need to be engaged to play specific role around the testbeds federation reference model, its APIs and its instantiations using various testbeds that get federated are SDOs/Fora, research communities/researchers on 5G and beyond, industry users of testbeds, testbeds suppliers for 5G testbeds and other testbeds, CSPs and enterprises, infrastructure vendors/suppliers for ICT and verticals, ISVs, Open Source and Open Hardware projects, regulators, owners of existing testbeds and platforms for 5G and beyond, and any other interested parties.

Examples of potential roles of certain stakeholders in the ecosystem of federated testbeds that is to be centred around the reference model for testbeds federations presented in the present document:

- SDOs/Fora: Can potentially share the burden on APIs standardization and on roadmaps in a harmonized and collaborative way.
- All stakeholders: Help contribute to capturing Requirements to be fulfilled by the testbeds federations APIs.
- Testbeds suppliers, CSPs, enterprises, infrastructure vendors, ISVs: Help contribute to derivation of potential new business models for testbeds suppliers that derive from the testbeds federations reference model.
- 5G and beyond research/R&D communities: Bring experiences to the discussions of the reference model and APIs Specifications and contribute what may have been achieved already in this area of testbeds federations – with respect to existing API implementations by Research communities and by the Industry as well. For example, the work on defining 'common methods and attributes of the testbeds federation APIs' that are agnostic to the type of testbed can be inspired by APIs and experiences from various R&D projects on testbeds to help capture 'common methods and attributes' while leaving the methods and attributes that are testbed-specific to the specialized APIs in specific testbeds types that get invoked by the generic APIs inherited from the generic model of a testbed domain concept.
- Owners of existing testbeds and platforms for 5G and beyond: Embark on transformation or evolution of existing testbeds (industry and potential research testbeds as well) and federation APIs to meet the requirements of the testbeds federation model being standardized. Make efforts on the instantiations of the reference model in building industry-grade new testbeds and/or transformations of existing testbeds.
- Open Source and Open Hardware projects: Help contribute to the building of open networking platforms (ONPs) testbeds that conform to the testbeds federation reference model and its associated APIs. Also make efforts on the instantiations of the reference model in building industry-grade new testbeds and/or transformations of existing testbeds.

7 Potential of testbed interoperability and federation

The global research community is currently triggering a quick development of ICT, such as 5G and the core network. All the new systems become more complex and the interactions between their components are growing. To detect the errors and validate new concepts, researchers need to test their solutions in an environment which should be the closest to reality and therefore, to the constraints of the production in the industry.

The purpose of a testbed federation is exactly fulfilling the needs of the research community and solving the problems met by the industry pre-deployments activities for large scale tests in real conditions. A federation of testbeds offers more possibilities to tackle the research challenges through more broadly available and heterogeneous resources. Of course, each experiment made inside a testbed federation should be replicable to allow the comparison during the different iterations of the experiment.

The interconnection of the testbeds allows researchers and the industry to test and validate new technical solutions in different configurations. Indeed, each testbed has its own unique features to implement experiments linked to different ICT research domains. Federating testbeds permits the allocation of a large number of heterogeneous resources to a particular experiment during a determined duration. Particularly, experiments focused on scalability can be put in place without high cost because the resources are shared across the testbed federation. The benefits of a testbed federation are numerous and include a reduced time to market, a quick validation of new ideas, a large variety of possible experiments and finally, no investment to create a testbed for a particular experimentation.

The general architecture of a testbed federation is illustrated in Figure 6.

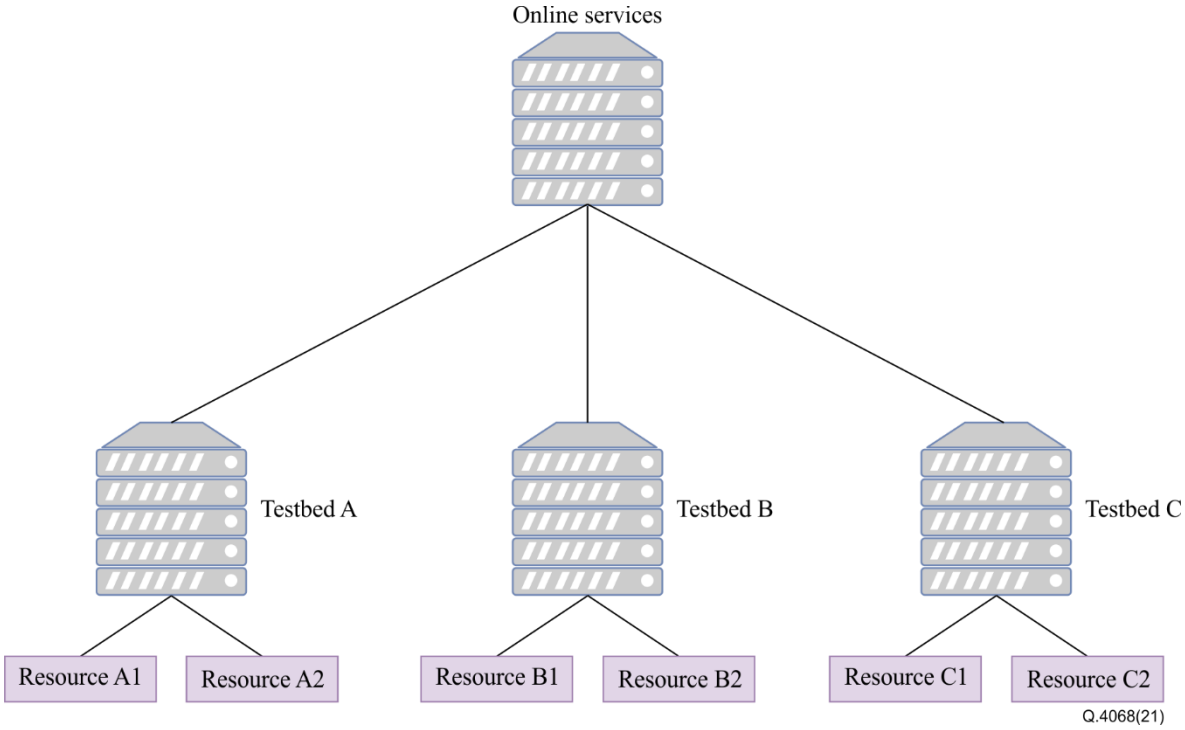

Figure 6 – General architecture

The federation of testbeds implies by nature a certain level of interoperability among the testbeds. The interoperability of testbeds is achieved through common APIs which manage the different technical aspects of the testbed federation.

First of all, the user, who is typically a researcher, needs to take part in a project inside the testbed federation. So, the user must be authenticated and authorized in the testbed federation to get access

to the resources provided by the testbeds. Of course, the best practices and the latest developments made in the context of security and data protection should be applied to ensure that the researcher's experiment is done in a safe environment.

An important step to realise when preparing an experiment is the reservation of the resources provided by the different testbeds. Indeed, a resource cannot be used at the same time in different experiments because the results of an experiment will be influenced by the actions of another experiment. So, this means that a clear isolation of the resources should be provided for each testbed during each experiment and as a consequence, each testbed should be aware of the current resource reservations. In the same way, an authorization mechanism should be put in place for each testbed to guarantee the correct usage of a resource during an experimentation.

Finally, each testbed should exchange information for the centralised management of the resources. Indeed, a global view of the resource utilisation is needed for researchers to avoid reserving a resource already involved in an experiment. Consequently, a set of APIs is required for data exchange between the testbeds and for the management of the resources.

8 Elements of a reference model of testbed federation

This clause presents the reference model of testbed federation and its related elements. The final objective of the reference model is to represent the mandatory elements allowing the federation of different testbeds through a set of common APIs. The reference model is applicable to all kinds of testbeds and so, is fully generic. The APIs defined in the context of this reference model are common and universal. They can be applied for all the testbeds, independently of the domain of research intended by each testbed. The design of the common APIs described in this Recommendation avoids the creation of complementary and specific APIs for testbed federation.

This reference model of testbed federation is implemented in the APIs presented in the following clauses, namely aggregate manager API, slice authority API and member authority API.

The reference model to federate the testbeds is described in detail in Figure 7.

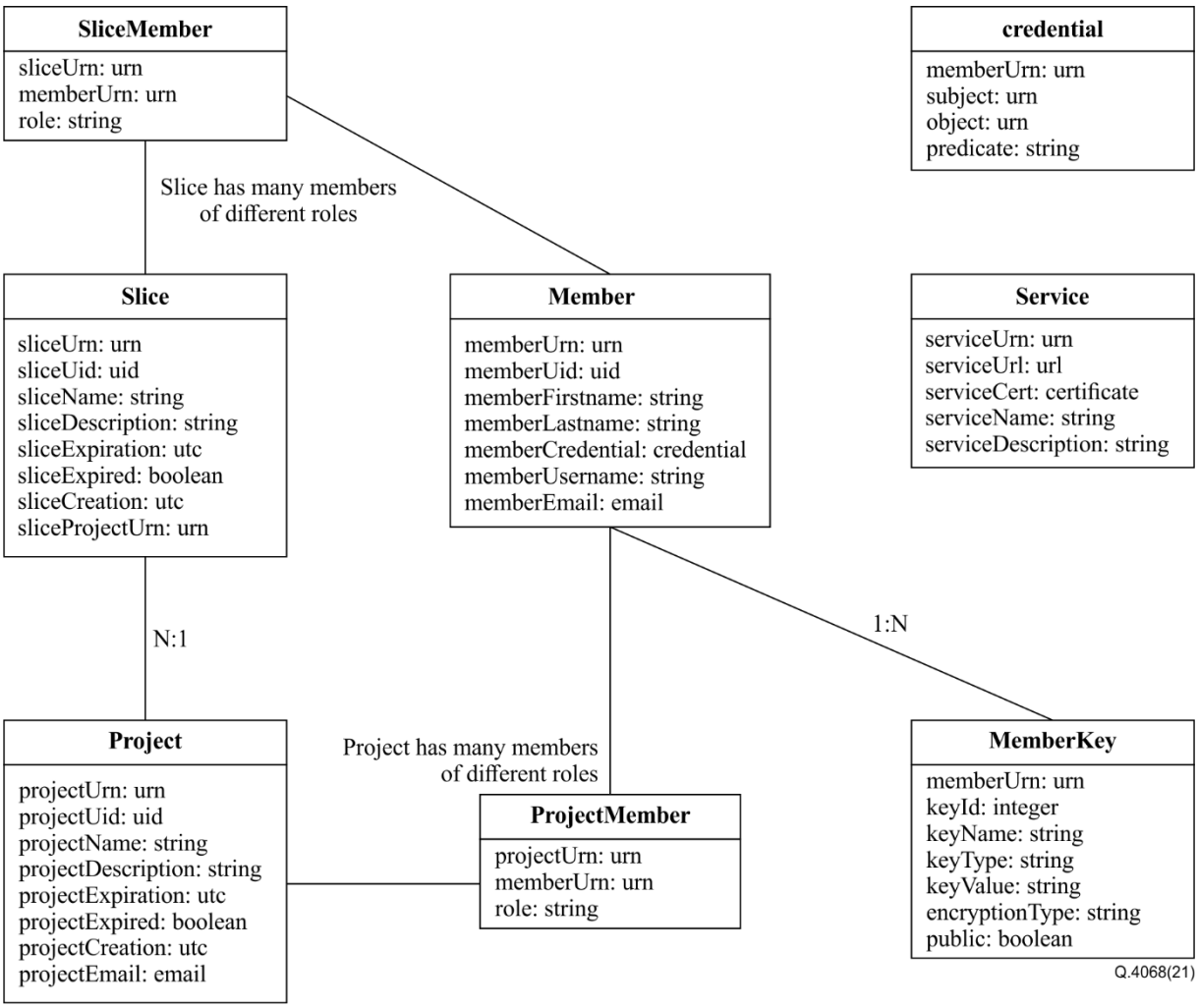

Figure 7 – Reference model

This section describes the reference model and its components.

Credential contains the link to the member that is the end experimenter (*memberUrn*) and the credential of this member. In detail, *memberUrn* is a unique identifier for the member who has received his credential. The credential is composed of several elements:

- *subject:* the owner of the credential and more precisely, his unique identifier.
- *object:* the unique identifier of the object to protect. This ensures that the owner of the credential gains privilege for this object.
- *predicate:* this string is used to check if the credential of the member exists and was issued by a recognized authority of the testbed federation.

Member represents the user of the tested federation, i.e., the experimenter. This part contains the personal data of the user. The member has two unique identifiers, the *memberUrn* and the *memberUid*. The *memberUrn* is a uniform resource name (URN) used across all the testbed federation to adequately identify the user: it can be considered as a link or a reference to the user, generally an experimenter. The *memberUid* is also a unique identifier but is employed internally in a particular authority of the testbed federation. Indeed, *memberUrn* is exposed in the frames exchanged by the different components of the testbed federation architecture and permits clear identification of the user member among the testbeds. The *memberFirstname* and *memberLastname* are the first name and the last name of the member. The *MemberCredential* attribute is the credential owned by the member. The *MemberUsername* is, of course, the username used for logging into the testbed federation. Finally, the *MemberEmail* is the email address of the member;

the email address is used for the creation of the user account and can serve as a username. A member is the owner of cipher key(s), project(s) and slice(s) and so, has sufficient rights to handle these objects for the building of an experiment.

MemberKey is composed of the following attributes:

- *memberUrn:* the link to the member owning the key.
- *keyId:* the unique identifier of the key. Typically, it can be a fingerprint or a hash coming from a public key.
- *keyName:* the name of the cipher key.
- *keyType:* the type of the cipher key. Several sorts of cipher keys are supported among the testbeds and so, this attribute directly contributes to the interoperability of testbeds at the security level. An example could be private-enhanced mail (PEM).
- *keyValue:* the value of the cipher key.
- *encryptionType:* the kind of encryption used by the cipher key, for example Rivest-Shamir-Adleman (RSA).
- *public:* this Boolean value indicates if the cipher key is public or private.

A member can have several cipher keys.

Project represents a project made in the testbed federation. An experiment is made inside a project. A project can have several members with different roles for the members. In the same manner, a project can contain an unlimited number of existing slices required for an experiment. Indeed, the role of a project is to group slices for experimentation. The following attributes are employed to describe a project in the context of testbed federation:

- *projectUrn:* the link to the project used inside the testbed federation. Indeed, it allows a project to be uniquely identified.
- *projectUid:* the unique identifier of a project within an authority. This means that *projectUid* is only valid inside a given authority.
- *projectName:* the short name of the project.
- *projectDescription:* the description of the project.
- *projectExpiration:* the data and time when the project will be finished.
- *projectExpired:* a Boolean value indicating if the project is already finished or not.
- *projectCreation:* the date and time when the project was created.
- *projectEmail:* email address associated to the project.

ProjectMember this allows linking of a member to a project. Basically, two URNs (*projectUrn* and *memberUrn*) are used to define the relationship between a particular project and a particular member. In this context, an experimenter can create his own experiment in a given project, if and only if the experimenter is a member of this project. The role of the experimenter is described with the attribute *role.*

Service describes a service given by a component of the testbed federation. Generally, a service is instantiated on a server hosted in the federation. A service has the following attributes:

- *serviceUrn:* the URN for the service. It permits referencing the service across all the components of the testbeds.
- *serviceUrl:* the URL to reach the service.
- *serviceCert:* the certificate attributed to the service, and by extension, to the server offering this service. Typically, it can be a X.509 certificate.
- *serviceName:* the name of the service.

- *serviceDescription:* the description of the service.

Slice represents a set of resources distributed among the testbeds. As mentioned by the definition in clause 3.2.2 of this Recommendation, a slice is composed of slivers available in different testbeds. A sliver is a resource provided by a testbed and can have different natures such as a physical machine, a virtual machine, an IoT sensor or a network slice. So, a slice provided by the testbeds contains several slivers. A sliver can be a network slice defined in the context of 5G core.

The attributes for a slice are given below:

- *sliceUrn:* the URN identifying a slice across the testbeds.
- *sliceUid:* the unique identifier of a slice. The unicity of a slice is relative to the authority managing this slice.
- *sliceName:* the short name of the slice.
- *sliceDescription:* a text describing the slice.
- *sliceExpiration:* the date and time when the slice will be expired.
- *sliceExpired:* a Boolean value showing if the slice has already expired.
- *sliceCreation:* the date and time when the slice has been created.
- *sliceProjectUrn:* the URN of the project to which the slice is linked.

A given slice belongs to a project which has different members with different roles as already mentioned. By consequence, several members can access a slice in a project. So slices have a multi-tenant nature as several members of the same project can access the underlying resources of a slice.

SliceMember allows associating members to a slice. Indeed, *sliceUrn* and *memberUrn* link a particular slice to a given experimenter. Each associated member has a role defined through the *role* attribute.

9 Testbeds federation APIs requirements

Multiple APIs should be specified to cover all the expected functionalities concerning the federation of testbeds. All the requirements mentioned in this clause are common all the open APIs.

The requirements for all the testbeds federation APIs are described below:

- Management of the testbeds and their resources towards the testbed federation: the APIs should be able to manage each testbed and the associated resources in a well-defined manner. Each component of the API architecture should have a clear role and expose its capabilities through the APIs. This permits the global management of all the resources distributed among the testbeds.
- Data encryption: the data exchanged between all the entities must be encrypted to ensure security and data protection.
- Authentication and authorization: the users must be authenticated and authorized to access the resources of the testbeds and to manipulate them through the methods exposed by the open APIs for interoperable testbed federations.
- Mandatory and optional arguments passed in the methods: the open APIs provide operations with mandatory and optional arguments. This permits use of the methods in different configurations and independently in the different federated testbeds. Eventually, it will facilitate the integration of new testbeds in the federation.
- Versioning of the APIs: it will guarantee a part of the testbed interoperability. Indeed, in the initial phase of transactions between the federation entities, each component of the federation architecture will be able to check its possibilities of communication and adapt them if it is necessary.

- Extensibility of the APIs: the APIs should be able to be extended to cover future needs which are not yet defined.
- Portability: the APIs should be portable in two directions. The first direction concerns the source code and the API source code should not be dependent on any programming language. Secondly, the APIs can be used in different environments without depending on the system interfaces. The application logic should not be linked to any system interface.
- Unique identifiers used across the APIs: this allows identification of each resource of the testbed federation in a unique way without any confusion. Clear identification of all the resources permits use of the resources in all the entities registered in the testbed federation.
- Standardized communication protocol: the protocol is used for communication between the server and the client, and eventually, between the different components of the API architecture. A common communication protocol is requested to correctly manage the testbed federation and should be understandable for each component involved in the testbed and resources control. This protocol should be available on both sides and ideally, the implementation of the protocol should not be dependent on a specific programming language.
- Interoperability: the testbed federation APIs should ensure interoperability between the components of its architecture. A client component should be able to communicate and exchange data with the server components in a well-defined manner and without any ambiguity. Clients and servers should understand the data exchanged between them.
- Abstraction of the heterogeneous technologies and communication protocols under test: the APIs defined in this document should abstract the technologies and protocols under test. Indeed, the APIs are completely independent of each technology and protocol under evaluation and there is no interference between the APIs managing the testbed federation and the objects under evaluation.
- Openness of the APIs: the APIs should be compliant within the criteria defining the openness of an API. This means that the APIs are interoperable, portable and extensible.
- Scalability: the APIs should be scalable. By definition, this means that the testbed federation APIs are extensible and apply demand balancing for requests made by the callers. These APIs should be designed for scalability from the beginning of inception and in all its future releases.

10 Some APIs for illustration of an instantiation of the generic model

The API architecture is based on a software architecture where different kinds of entities interact between them. The entities involved in the testbed federation are listed below:

- Aggregate: the aggregate entity is a collection of resources which are managed through the aggregate manager API presented below. An aggregate is, for instance, a testbed running the aggregate manager API in its premises.
- Member: a member can be an experimenter or another user consuming the resources exposed through the open APIs for interoperable testbed federations.
- Authority: an authority is a service managing the members and their permissions to access the resources of the testbeds. The authorities are free to implement their authorization scheme, but the credentials should be passed to the methods of the APIs as parameters. In the same manner, each authority is independent and can trust or not the other authorities of the testbed federation. There are two types of authorities:
  - Member authority (MA): this entity manages the attributes of the members.
  - Slice authority (SA): this entity manages the slices and generates the credentials for the members to access the slices.

In this context, a federation can be considered as a collection of authorities and aggregates. These entities have established common policies to ease the sharing of resources for the members. A software service named federator is built for a federation and its main role is to list all the member authorities, Slice authorities and aggregates instantiated in the federation of testbeds. The federation registry is run by the federator and encompasses in fact two directories registering the aggregate managers and the authorities. The aggregate manager directory contains the list of all the aggregate managers of the testbed federation. So, if a testbed should be added in the testbed federation, the testbed must run an aggregate manager in its premises and then, register its aggregate manager in the aggregate manager directory. As result, the aggregate manager API can be used between this new testbed and the federator. The second part of the federation registry is the authority directory listing all the member and slice authorities existing in the testbed federation. Figure 8 shows the architecture of a testbed federation.

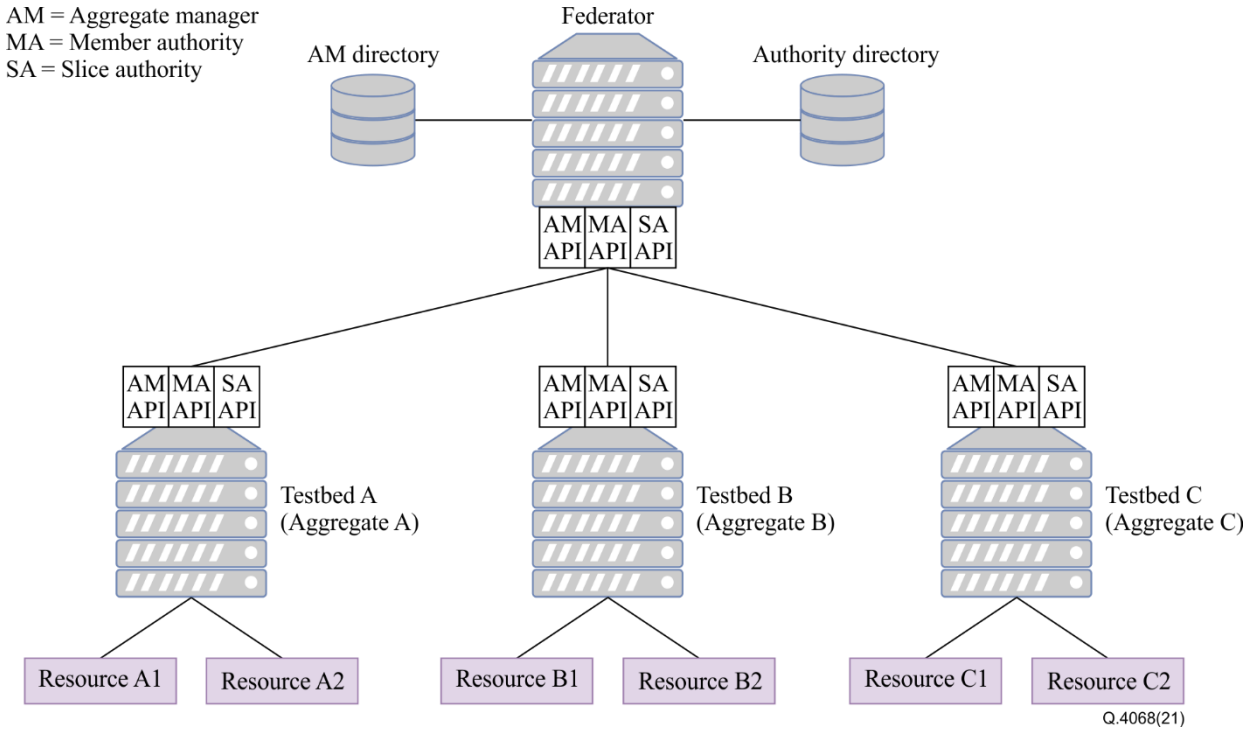

Figure 8 – Architecture overview

In Figure 8, each resource can be considered as a sliver. All the resources of one testbed can be grouped to form also a sliver. For example, grouping resource A1 and resource A2 will create a sliver. On the other hand, if an experimenter creates a group of resources containing resource A1 and resource B2, it will be a slice, because these two resources are not located in the same testbed.

Mutual trust is necessary between the different entities of a testbed federation. This is achieved through the federation registry which provides the PKI certificates to each aggregate entity. This means that each testbed joining the federation must accept the certificates as trust roots.

The mechanisms put in place for the authorities allow distributed resources allocation among the different components of the federation architecture in a safe manner. Indeed, each testbed can decide which of its resources are available in the federation and when they can be reserved for experimentation. The experimenters are aware at any time if the distributed resources can be allocated or not for their experiments.

11 Aggregate manager API

The aggregate manager (AM) API enables the federation of testbeds through the instantiation of an aggregate manager into each individual testbed. The main goals of the aggregate manager API are the advertisement of resources from a testbed and their allocation to slices. A slice is composed of different slivers. Each sliver is, in fact, a virtual or physical resource available in the network and uniquely identified. This API is the control plane interface where the resources can be discovered and reserved for experiments. The aggregate manager API is generic by definition and so, this API can be used independently of the type of testbed. Indeed, a resource is not linked to a particular kind of physical or virtual object; each sort of object can be a resource used through the aggregate manager API. The aggregate manager API is in charge of the orchestration of the testbed federation.

The communication between the server and a client is realised through XML-RPC over an SSL connection. A connection is only possible if the client is providing the right certificates which are used to authenticate the users. For the methods of the aggregate manager API requiring authorization, the user must provide his credentials via the parameter *credentials* of a given method. In summary, certificates ensure the user authentication and the credentials, as well as user authorization.

The data structure for the data exchanges between a server and a client is resource specification (RSpec). An RSpec file is, in fact, an XML document listing and describing the resources of a testbed.

This API has different methods offering all the features needed for the utilisation of the resources provided by the different testbeds. Table 2 presents all the methods designed in the context of the aggregate manager API:

Table 2 – Aggregate manager API

Method	Operation description	Parameters
GetVersion	*obtain the version of this API*	*options*
ListResources	*return the list of available resources in a testbed*	*credentials, options*
Describe	*provide the description of resources*	*urns, credentials, options*
Allocate	*reserves the resources*	*sliceUrn, credentials, rspec, options*
Renew	*extend the duration of a reservation*	*urns, credentials, expirationTime, options*
Provision	*provision resources*	*urns, credentials, options*
Status	*get the status of slivers*	*urns, credentials, options*
PerformOperationalAction	*perform an action on the slivers*	*urns, credentials, action, options*
Delete	*make the slivers unallocated*	*urns, credentials, options*
Shutdown	*shut down slivers*	*sliceUrn, credentials, options*

From the point of view of the experimenter, the following steps must be done to build an experiment through the methods exposed by the aggregate manager API:

1) Discover the resources (method *ListResources*).
2) Request the resources through a reservation (method *Allocate*).
3) Provision the reserved resources (method *Provision*).
4) Start the resources (method *PerformOperationalAction*).
5) Check the status of the resources started (method *Status*).
6) Extend the reservation of the resources (method *Renew*).

7) Free the resources after an experiment (method *Delete*).

More details about each method of the aggregate manager API are given below.

GetVersion: This method permits to check the version of the API and the formats used for the RSpec documents in a particular testbed. The versions of the aggregate manager API and RSpec are defined statically. The argument *options* are optional and can contain data specific to the implementation of an aggregate manager. The information returned by the method is an XML-RPC structure composed of the following elements:

- *code:* indicates if the call of this method is successful or not.
- *value:* the list of versions for the API, the different RSpec formats, the supported credential types and eventual options for the sliver allocation.
- *output:* the result of the call which is, in fact, a human-readable message (XML-RPC string). This message can help the experimenter to solve a problem encountered by the Aggregate Manager in the testbed.

ListResources: The method returns the detailed list of resources available in a testbed. The first parameter is *credentials* and corresponds to a valid user credential which will grant the rights to the user calling this method. The second parameter is *options* built with three sub-options:

- *available:* is an optional Boolean value indicating if the experimenter wants only available resources (true) or all the resources (false).
- *compressed:* is an optional Boolean value specifying the compression of the data returned by the method. The compression is done if the value is true.
- *rspecVersion:* is a required XML-RPC structure indicating the type and the version of the RSpec used for the advertisement of the testbed resources.

The method returns an RSpec document listing and describing all the resources of the testbed accordingly to the parameters given by the experimenter. This RSpec is incorporated in the *value* field of the returned structure beside the fields *code* and *output* indicating respectively the eventual error code and the debugging message.

Through this operation, the discovery of all the resources existing in a testbed permits the onboarding of these resources in the testbed federation and so, can possibly be used in all the components of the federation.

Describe: This method retrieves an RSpec document describing the resources of a testbed. The first parameter *urns* is the list of URNs which can be a slice URN of a set of sliver URNs. The argument *credentials* correspond to the user credentials. Finally, the parameter *options* is composed of these different fields:

- *compressed:* Boolean value indicating the compression of the returned data.
- *rspecVersion:* the type and the version of the expected RSpec.

Apart from the error code and message, the method provides the experimenter a *value* field containing all the information for the requested URNs under the format of an RSpec document.

Allocate: This method reserves the resources in a testbed and returns an RSpec file describing the reserved resources. In greater detail, several parameters should be provided to the method:

- *sliceUrn:* is the URN of the slice to be reserved.
- *credentials:* the credentials of the user. Only authorized experimenters can reserve resources in the testbeds.
- *rspec:* the RSpec file containing the resources pointed out by the *sliceUrn* argument.
- *options*: is not mandatory and consists of the expiration time of the slivers.

The method returns, as usual, an XML-RPC structure composed by a *value* field for all the newly allocated slivers. An error code and debugging message are also returned in case of an error in the testbed.

Renew: This method allows the extension of a reservation. It means that the slivers given in parameters will be reserved for a longer duration in an experiment. The arguments of this method are described below:

- *urns:* the URNs of the resources for which the expiration time of the reservation should be extended.
- *credentials:* the credentials of the experimenter.
- *expirationTime:* the date and time for the expiration of a reservation, expressed in the RFC 3339 format. The time zone is UTC.
- *options:* this parameter contains a field named *bestEffort* to indicate if all the slivers or only a part of them should be renewed. This field is a Boolean value: true will try to renew all the slivers. If it is not possible, the slivers keep the original expiration time. If *bestEffort* is false, an error code will be returned by the method if the expiration time of a sliver was not extended as expected.

The method provides the experimenter with an XML-RPC structure containing a *value* field composed by all the information for each sliver renewed. Basically, the URN, the allocation status, the operational status, the expiration time and an optional error message are returned by the method for each sliver renewed.

Provision: This method sets up the reserved resources, leading to the instantiation of these resources in the testbed. As a result, the method returns a list of reserved and configured resources. The method takes three parameters:

- *urns:* the URNs of the resources to be provisioned.
- *credentials:* the credentials of the user.
- *options:* in this parameter, a mandatory field *rspecVersion* is provided and contains the type and the version of the RSpec returned by the operation. Other options can be included, typically the *bestEffort* and *endTime* fields based on the values provided already through the methods *Allocate* and *Renew*. The *users* field can be also entered by the experimenter in case of a resource requiring a user login.

The method returns an XML-RPC structure composed by a *value* field listing all the provisioned slivers. For each sliver, the URN, the allocation and operational statuses, the expiration time and a possible error message are returned in the list of slivers.

Status: The method allows the experimenter to check the current status of resources. Indeed, the experimenter can know if the resources have been started correctly before using them in his experiment. In this context, the *status* operation can take three parameters:

- *urns:* the URNs of the resources.
- *credentials:* the credentials of the experimenter.
- *options:* this parameter is completely optional.

The method returns a *value* field in the XML-RPC structure with all the information for each sliver. Indeed, the sliver URN, the allocation and operational statuses, the expiration time and an error message are sent back to the user through this method.

PerformOperationalAction: This method starts the resources. For instance, this method can boot a virtual machine. This method has four parameters:

- *urns:* the URNs for the resources.
- *credentials:* the credentials of the experimenter.

- *action:* the action to be performed. There are three possibilities: *start*, *stop* and *restart*.
- *options:* the supported option is *bestEffort*. The default value of *bestEffort* is false and an error code is returned if a sliver fails.

An XML-RPC structure is returned by this operation with a *value* field containing the information for each concerned sliver. The sliver URN, the allocation and operational statuses, the expiration time are sent back by the method. Optionally, the resource status and an error message could be also returned.

Delete: This method terminates the reservation of resources. This means that the testbed frees the resources which become available for the other experimenters. The parameters are:

- *urns:* the URNs of the slivers.
- *credentials:* the credentials of the users.
- *options:* the *bestEffort* option can be included in this argument.

The operation returns a structure containing a *value* field composed by the information related to the slivers unallocated. For each sliver, the URN, the allocation status and the expiration time are returned to the experimenter. An error message could be also returned at the same time.

Shutdown: This method is generally used by the administrators to stop the resources of a slice, typically when the slice is not running correctly. The parameters of this operation are:

- *sliceUrn:* the URN of the slice to stop.
- *credentials:* the credentials of the experimenter.
- *options:* no options are required.

The method returns a *value* field containing an XML-RPC Boolean. True indicates that the resources were shut down properly.

12 Slice authority (SA) API

The slice authority (SA) API is based on an abstraction of an account. Through it, the resources of several testbeds can be reserved from an account. In fact, it is an authorization associating the account to the testbed resources. This reservation of resources is a step needed to prepare an experiment using the resources provided by the different testbeds. The slice authority API is applied on all the kinds of testbeds. Indeed, the API is generic and completely independent of the testbed type. All the attributes linked to the slice authority API are valid for all the sorts of encountered testbeds.

The slice authority API manages the slices and therefore, their permissions. This API provides different services which target different parts of the reference model.

This API was defined within the methods described in Table 3.

Table 3 – Slice authority API

Method	Operation description	Parameters
Create	*create a new slice or sliver or project*	*type, credentials, options*
Update	*update a slice or sliver or project*	*type, urn, credentials, options*
Delete	*delete a sliver*	*type, urn, credentials, options*
Lookup	*find slices or slivers or projects matching criteria*	*type, credentials, options*
GetCredentials	*provide the list of credentials*	*sliceUrn, credentials, options*
ModifyMembership	*modify the membership for a slice or project*	*type, urn, credentials, options*
LookupMembers	*find the members of a slice or project*	*type, urn, credentials, options*
LookupForMember	*get the slices or projects linked to a member*	*type, memberUrn, credentials, options*

The details of each method are:

Create: This method creates a new object which can be a slice, a sliver or a project. The parameters of this operation are:

- *type:* the type of the object to be created. There are several possibilities: slice, sliver or project.
- *credentials:* the credentials of the user calling this method.
- *options:* a dictionary of fields which define the properties and attributes of the new object. Some fields are mandatory, otherwise optional: this depends on the type of the object.

As a result, the method returns the dictionary composed by the fields given in parameters to the method.

Update: This method updates the object according to the parameters given by the user. These parameters are:

- *type:* the type of object to be updated.
- *urn:* the URN of the object. Of course, the URN serves as an identifier for the object.
- *credentials:* the credentials of the user.
- *options:* this parameter contains the fields to be updated on the given object.

This method updates only one object per call and returns nothing.

Delete: The method erases an instance of the given object specified by the URN. A single object is removed per method call. Some objects cannot be deleted accordingly to the defined authority policy. The parameters of this operation are listed here:

- *type:* the type of the object to be removed.
- *urn:* the identifier of the object.
- *credentials:* the credentials of the user.
- *options:* this parameter is not necessary.

The method returns nothing as an answer.

Lookup: This operation finds objects corresponding to the options given in parameters. The parameter *options* has the role of filter for the research of objects. The method arguments are explained below:

- *type:* the type of the objects.
- *credentials:* the credentials of the user.
- *options:* this parameter defines the filter used to find the right objects.

The method sends back a list of dictionaries with the field/value pairs for each object found. The indexation of the list of dictionaries is made with the URN of each object.

GetCredentials: This method provides the list of credentials of the user for the given slice. First of all, the parameter *sliceUrn* is the unique identifier and reference for the slice. The second parameter *credentials* is the credentials of the user calling the method. This means that the user should know at least one of its credentials to get access to the list of all its credentials. The parameter named *options* is optional and can contain a cipher key. This parameter is kept open for further implementations with specific certificates.

ModifyMembership: This method is used to change the membership, to give or remove roles to members for the object passed in the parameters. The list of parameters for this operation is given below:

- *type:* the type of object for which the membership will be modified. Two types are defined: SLICE and PROJECT.
- *urn:* the URN of the slice or the project. This depends on the value of the previous parameter.
- *credentials:* the credentials of the member.
- *options:* there are several fields to be set in the function of the needs of the member:
  - *membersToAdd:* the list of members to add to the slice or project. This list contains the *memberUrn* and the role of each member to be added.
  - *membersToRemove:* the list of members to be removed from a slice or project. The *memberUrn* of each member is sufficient in this case.
  - *membersToChange:* the list of members comprising the *memberUrn* and the new role for each member of the given object.

LookupMembers: This method returns the list of members and their roles for the given object. There are four parameters:

- *type:* the type of object. Two values are possible: SLICE or PROJECT.
- *urn:* the URN of the object. The members and roles for this particular object will be returned by this operation.
- *credentials:* the credentials of the user calling the method.
- *options:* optional parameter which is not currently used.

The method sends back a list of dictionaries. For each of them, a *memberUrn* and the associated role are returned as pairs.

LookupForMember: This method is similar to the previous method, but only one member is given in the parameter *memberUrn*. The first parameter named *type* contains the sort of object: only SLICE and PROJECT are tolerated. Of course, the parameter *credentials* is the credentials of the user making the method call. The parameter *options* is completely optional and has currently no specific purpose.

The method returns a list of dictionaries. A pair of *memberUrn/role* is encompassed in each dictionary for the different objects for which the member has a role.

13 Member authority (MA) API

The member authority (MA) API models a user registered into an accredited project. In the context of a project, the user can do experiments when creating slices. A slice contains the resources of testbeds. The API covers all the kinds of testbeds and so, not limited to a specific testbed type. The methods and related parameters defined in the member authority API are common and generic for all the sorts of testbeds and so, completely independent of a specific testbed type.

The member authority API was designed to take into account all the best practices for the security and data protection. The API offers the following methods listed in Table 4.

Table 4 – Member authority API

Method	Operation description	Parameters
Create	*create a record for a key associated with a member*	*type, credentials, options*
Update	*update the information of a member and his key*	*type, urn, credentials, options*
Delete	*delete the record for a key*	*type, urn, credentials, options*
Lookup	*find information of a given member and his key*	*type, credentials, options*
GetCredentials	*get the list of credentials for a member*	*memberUrn, credentials, options*

The details of each method are given below:

Create: This method creates a new object which can be a slice, a sliver or a project. The parameters of this operation are:

- *type:* the type of the object to be created.
- *credentials:* the credentials of the user calling this method.
- *options:* a dictionary of fields which define the properties and attributes of the new object. Some fields are mandatory, otherwise optional: this depends on the type of the object.

As a result, the method returns the dictionary composed by the fields given in parameters to the method.

Update: This method updates the object accordingly to the parameters given by the user. These parameters are the following:

- *type:* the type of object to be updated.
- *urn:* the URN of the object. Of course, the URN serves as an identifier for the object.
- *credentials:* the credentials of the user.
- *options:* this parameter contains the fields to be updated on the given object.

This method updates only one object per call and returns nothing.

Delete: The method erases an instance of the given object specified by the URN. A single object is removed per method call. Some objects cannot be deleted accordingly to the defined authority policy. The parameters of this operation are listed here:

- *type:* the type of the object to be removed.
- *urn:* the identifier of the object.
- *credentials:* the credentials of the user.
- *options:* this parameter is not necessary.

The method returns nothing as an answer.

Lookup: This operation finds objects corresponding to the options given in parameters. The parameter *options* has the role of filter for the research of objects. The method arguments are explained below:

- *type:* the type of the objects.
- *credentials:* the credentials of the user.
- *options:* this parameter defines the filter used to find the right objects.

The method sends back a list of dictionaries with the field/value pairs for each object found. The indexation of the list of dictionaries is made with the URN of each object.

Alternatively, this method can be used to retrieve the information of a member; this is done by setting the parameter *type* to MEMBER.

GetCredentials: Get the list of credentials owned by a member. The parameter *memberUrn* identifies the member. Then, the parameter *credentials* is a credential of this member; this means that the member should know at least one of its credentials to access this method and so, its other credentials. Finally, the parameter *options* is a dictionary containing eventual key/value pairs which are normally optional.

14 Reference metrics

Several reference metrics can be used in the testbed federation to manage the federation and the testbeds to ensure a good behaviour and high performance of all the components communicating through the APIs defined in this document and also the different kinds of resources. The reference metrics described below are generic and independent of the testbed type. The monitoring of the testbeds can encompass the following metrics:

- State of the power source: The experimenter can know if a component or a resource is on or off. This is particularly important for wireless and mobile resources powered by batteries.
- Energy consumption: An objective of the research done in the testbed federation is to reduce the energy consumption of the hardware under test and the software running on the hardware. So, determining the energy consumption is a very good point to observe if the hardware and the associated software respect the energy efficiency.
- Detection of Internet disconnections: As each testbed is connected through Internet to the other components of the federation, it is important to detect the disconnections. This can be done easily by pinging regularly the other components.
- Packet loss and transmission errors: Linked to the detection of Internet disconnections, the determination of packet loss and transmission errors between the components permits monitoring of the quality of service (QoS) and to take corrective actions in case of problems.
- SSH or Telnet availability: Some resources are accessible by SSH and Telnet when creating an experiment. Typically, the virtual machines or the gateways, like Raspberry PI, can be used by the experimenters and a SSH access is required to prepare the experiments. A constant monitoring of the SSH or Telnet availability can indicate if a such resource is down and then, the testbed provider can solve this issue as soon as possible.
- Number of running virtual machines: For the servers offering virtual machines to the experimenters, this metric permits determination of the usage of the server inside the testbed federation.
- CPU load: The measurement of the CPU load on a physical or virtual machine allows the experimenter to evaluate the software under test. Indeed, bugs can lead to the over-consumption of resources, notably the CPU load.

- Free RAM: The monitoring of this metric can help the experimenter to improve the performance of the software under test. If a process is using the memory extensively, it could indicate a bug or a misconception in the software under test.
- Number of simultaneous sessions: In the testbed federation, the metric indicates the utilisation of the online services offered by the different testbed providers and the federator.
- Number of refused service requests: The reliability and the detection of overload can be estimated through this metric.

The exchange of data among the components is essential to interconnect and federate all the testbeds and the data format is a crucial point. At the application level, the interoperability of the testbeds is ensured by the data format used between all the components of the architecture. RSpec is the recommended format and permits the resources hosted in each testbed to be described. The RSpec data structure is in fact an XML document following specific schemas. The purpose of RSpec is to list all the resources available in the testbeds, the resources requested by the experimenters and the reserved resources.

Some constraints exist for the testbeds using the aggregate manager API. First of all, each testbed (Aggregate) must advertise which type and which version of RSpec is currently used. In the same time, the testbed should provide the schema, the namespace and the extensions if they are available.

Of course, it is expected that the testbed returns the list of resources using the advertised RSpec format. This corresponds to the operation *ListResources*. In the same way, the calls to the methods *Allocate*, *Describe* and *Provision* return an XML-RPC structure containing a RSpec document conforming to the type and version announced previously.

The credentials structure contained in the data exchanged between the components through the open APIs for interoperable testbed federation must also indicate the type and the version. Eventually, a federation of testbeds can support different types of credentials if they are correctly advertised in the credentials structure.

The *options* parameter of the different methods described in this Recommendation permits passing of more information to the methods. This parameter is mandatory for the operations *ListResources*, *Describe* and *Provision*. The parameter is completely optional for the other methods and can serve to extend some documented features for testbeds. In all cases, the parameter named *options* should be an XML-RPC structure.

Appendix I

Example use case for federated testbeds as required by CSPs, based on the testbeds reference model: Testing federated autonomic management and control (AMC) by federated ETSI GANA knowledge planes (KPs) platforms for autonomic/autonomous 5G and beyond networks

(This appendix does not form an integral part of this Recommendation.)

Federation in general and federated testbeds in particular form a key part of the success of CSPs and other stakeholders to leverage their assets, monetize on their investments, and position themselves in their ecosystem of 5G and beyond where everything is evolving very dynamically.

The industry continues to build its own testbeds that are used internally within organizations such as CSPs or solution vendors, and in some cases some industrial testbeds are useable to multiple organizations based on certain collaboration agreements that are reserved only for the partners. Over the years it has increasingly been experienced that isolated standalone testbeds are not sufficient to test and trial out certain technology uses cases because the use cases rather require the use of components and resources located in various testbeds with each testbed bringing in the missing/required features and assets to complete the use-case. New ICT technologies, networks and vertical applications are becoming increasingly complex to test using standalone testbeds.

Federated testbeds bring sustainability in fostering environments for quick innovations and testing of complex technologies and use cases, and for enabling quicker time to market for products and services. Federated testbeds, enabled as a turnkey service such as testbed-as-a-service (TaaS), bring a lot of value to research use cases and industrial use cases. Yet, Standards have been lacking in this increasingly very important area of testbeds federations and interoperability. Standards are the enablers for interoperability and further benefits.

Therefore, it is important to note that research communities and the industry (solutions vendors/suppliers, CSPs, enterprises, and standards development organizations (SDOs)/Fora) all have roles to play in this desired ecosystem that should be built around the testbeds federations reference model presented in this present document both now and into the future in the era of *disaggregation* of ICT networks, 5G and beyond, as well the shift towards software in services, assets, etc.

A very strong synergy and a great fit has been discovered between the concepts/paradigms of autonomic/autonomous networks (ANs) and FT (federated testbeds).

Autonomous networks (ANs), powered by the autonomic management and control (AMC) paradigm [b-NGMN 5G] and employing multi-layer autonomics and multi-layer AI, are expected to drive so called knowledge planes (KPs) platforms as discussed in [b-NGMN 5G]. Standards for knowledge plane (KP) platforms for ANs now exist, with the main standard being the ETSI GANA (Generic Autonomic Network Architecture) knowledge plane (KP) concept specified in ETSI TS 103 195-2. When ANs, as represented by either individual network segments of a CSP (e.g., RAN, Edge cloud, Transport, Core network) or the whole of an end-to-end CSP network, need to be collaboratively interworked in any beneficial form, this should be achieved by having the KP platforms that are responsible for AMC of specific network segments as individual ANs that communicate with each other in the form of KP federations (more details can be found on this in [b-ETSI TS 103 195-2] and in [b-White Papers]. Through federation, knowledge exchange and transfer, metadata, events, triggers, synchronization and coordination messages, and many other forms of information are communicated in a collaborative fashion by the KP platforms to achieve E2E AMC operations such as E2E self-optimization of AN resources, self-protection and self-defence against detected security attacks, threats and risks to address security challenges that have

impact on various network domains. Testing ANs as represented by individual network segments and their associated KP level autonomics (slow control-loops) and autonomics (fast control-loops) introduced in the underlying infrastructure that constitute a specific network segment require federated testbeds that emulate the ANs' composition and target interworking of their two levels of autonomics.

When ANs (e.g., owned by CSPs or other stakeholders), using their KPs, collaborate and leverage upon the federated testbeds for the testing of the ANs' autonomics software and algorithms, a lot of value is created for all parties that may be involved. CSPs can access all the required resources and assets without needing to build, buy, or own them. They can then run important tests in 5G and beyond, fostering technological progress and innovation thanks to the federation of testbeds owned by different players. In the growing requirement of CSPs for disaggregated networks, SDOs and Fora such as ETSI, TC, INT, AFI, WG and NGMN advocate for disaggregated ETSI GANA KP platforms for specific network segments/domains, due to the call by CSPs for disaggregated networks, control planes and data planes and software platforms for AMC. Therefore, for CSPs such as network operators for 5G and beyond technologies, federated testbeds built on the basis of the reference model for testbeds federations can play a very important role in testing the emerging autonomic/autonomous networks (ANs) technologies as follows:

- Intra-CSP federations of ANs' AMC platforms such the ETSI GANA knowledge plane (KP) platforms [b-ETSI TS 103 195-2] across network segments of an AMC platform's responsibility, and inter-CSPs AMC platforms federations as they inherently and correspondingly require federated testbeds to test the cross-domain AMC operations.
- CSPs sometimes need to leverage and use testbeds/testbed assets from other stakeholders to build an E2E network (e.g., a 5G E2E network) and its corresponding AMC platforms (e.g., KPs) in order to test the federated AMC operations across the various network domains (e.g., AMC operations such as E2E self-optimization of AN resources, self-protection and self-defence against detected security attacks, threats and risks to address security challenges that have impact on various network domains).
- Intra-CSP federations of ANs' AMC platforms across network segments and inter-CSPs ANs' AMC platforms federations as the two inherently and correspondingly require federated testbeds to test cross-domain AMC operations.

Various CSPs have expressed the need for federated testbeds as summarized in the presentations from a Joint SDOs Workshop on federated testbeds for 5G and beyond [b-Workshop].

NOTE – ETSI is in charge of studying the reference model of federated testbeds and the instantiation of the reference model with respect to testing GANA knowledge plane (KP) platforms for E2E AMC across multiple domains (network segments and administrative inter CSP domains) [b-ETSI portal].

Appendix II

GENI testbed federation

(This appendix does not form an integral part of this Recommendation.)

Global environment for network innovations (GENI) [b-GENI] is a testbed federation considered as a virtual laboratory. The main activities of GENI testbed federation are research and education. The testbed federation is composed by networking and distributed systems. The experimentations done in the GENI testbed federation concern the network, security, services and applications.

The GENI testbed federation offers the following elements to experimenters:

- Computing resources located in the United States.
- Connections between the computing resources in the data link layer of the OSI model.
- Installation of operating systems or/and software on the computing resources.
- Configuration of network switches.
- Installation of communication protocols from the network layer to the upper layers on the computing resources.

Experiments with the following requirements fit the features of the GENI testbed federation very well:

- Large scalability: distributed resources are available such as virtual machines, bare machines, switches and base stations.
- Connectivity between resources without Internet protocol (IP): basically, the connection between the resources is made at the data link layer and the experimenter can program the upper layers to test his software from the network layer to the application layer.
- Programmability: all the components of the testbeds can be programmed by the experimenter, including the network elements.
- Reproducibility: the experiments can be redone in similar conditions to their first iteration.
- Availability of measurement and instrumentation tools: active and passive measurements can be visualized and analysed.

The starting point of GENI testbed federation is the United States, but the federation is extended to other continents such as South America, Europe, Asia and Oceania.

To access the GENI testbed federation, an experimenter needs to go to the GENI online portal and log in with his username and password. The experimenter must be part of an institution recognized by the GENI testbed federation. Basically, the institutions registered in the National Center for Supercomputing Applications (NCSA) can directly access the GENI testbed federation. To start an experiment, the user must join a GENI project managed by a project leader. GENI has published a repository of tutorials to help the experimenters to begin their first experiments in the GENI federation. The GENI testbed federation uses the same concepts defined in this Recommendation such as project, slice, aggregate, etc.

The workflow to create and run an experiment is described below:

1) Log in and join a project.
2) Create a slice.
3) Create a RSpec file describing the requested resources.
4) Use API calls to the aggregates to reserve the needed resources.
5) Utilize the RSpec files returned by the aggregates to prepare the resources for the current experiment.

6) Launch the measurement and instrumentation tools to observe the results of the running experiment.

7) Free the reserved resources for the next experimenter.

A testbed can join the GENI testbed federation through the following steps:

1) The testbed implements the GENI aggregate manager API.

2) The testbed signs the aggregate providers agreement.

3) A GENI slice authority is issuing credentials for the experimenters. The new testbed should recognize this authority to let the experimenters' access to the testbed resources.

4) The testbed should complete the template for the documentation related to its features and resources.

5) An integration test is made by the testbed and the other partners of the GENI federation.

A testbed can host a GENI rack. There are several sorts of GENI racks depending on the costs and the needs:

- OpenGENI: cloud applications, OpenFlow and VLAN networking for low budget.
- InstaGENI: cloud applications, OpenFlow and VLAN networking for average budget.
- ExoGENI: private and public cloud applications, big data applications, OpenFlow, flexible virtual network topologies, extensive storage for high budget.
- Cisco GENI: basically, the software of an ExoGENI rack running on Cisco servers. Other network components delivered by Cisco are used in the Cisco GENI rack.

GENI uses different APIs listed below:

- GENI AM API v3: This API corresponds to the aggregate manager API described in this Recommendation and is implemented in each testbed which has joined the GENI federation.
- Common Federation API v2: The API regroups the member authority API and the slice authority API described earlier in this document. The role of the API is to manage the federation composed by authorities and aggregates through common polices defining the sharing of resources.

To exchange information between the federation components, GENI has defined a RSpec version 3 specification. It is also based on XML-RPC structure. For the credentials, GENI uses attribute-based access control (ABAC) credentials which are in fact signed XML documents.

Appendix III

Fed4FIRE+ testbed federation

(This appendix does not form an integral part of this Recommendation.)

Fed4FIRE+ [b-Fed4FIRE] is the largest federation worldwide of next generation Internet (NGI) testbeds. The funding of Fed4FIRE+ comes from the European Union Horizon 2020 programme. Fed4FIRE+ is following the Fed4FIRE European project. The testbeds grouped into the Fed4FIRE+ federation offer services and facilities which are open, accessible and reliable. The federation of testbeds supports a large panel of research and innovation communities and initiatives, mainly in Europe. For instance, the testbeds are used for the 5G PPP (Public Private Partnership) projects and initiatives.

In Europe, the experimentally driven research is considered as important for the industry working in the domain of the Internet. Indeed, Future Internet Research and Experimentation (FIRE) infrastructures were launched in Europe since 2008 and each of them originally targeted a specific research domain. The federation of these infrastructures, like Fed4FIRE+, permits to create more innovative experiments involving several domains at the same time. In parallel, the testbed providers have developed common tools to manage the testbeds in an efficient way. The majority of testbeds are located in Europe.

The partners involved in Fed4FIRE+ continue to improve the federation services and facilities dedicated to the Internet research. This implies making the facilities more open, accessible and reliable. At the same time, the experimental processes have improved to become simple, efficient and cost effective. The trustworthiness of facilities has also increased, as has the sustainability of the experimental infrastructures.

The Fed4FIRE+ testbed federation is involved in a lot of Internet research domains such as optical networking, wireless networking, software-defined networking (SDN), tcloud computing, fog computing, data science applications, smart cities and many more. So, each testbed of Fed4FIRE+ has its own specific characteristics and is focused on a particular area such as wired network, wireless network, 5G, IoT, OpenFlow, cloud and big data. The testbeds joining Fed4FIRE+ have two options for their integration:

- Light: The testbed exposes a Web-based API to access the testbed resources in a unified manner. The disadvantage of this option is the limited control over the individual testbed resources.
- Advanced: The integration of the testbed is complete and all the experimentation steps are possible for the experimenter such as resource reservation and instantiation. To achieve full integration, the testbed must implement the federation aggregate manager API in its premises.

The choice of the type of integration is made between the new testbed provider and the technical staff of Fed4FIRE+ testbed federation. If the advanced integration is undertaken, the testbed implements the federation aggregate manager API. In all the cases, the testbed joining the federation must support the Fed4FIRE+ credentials through a client-based SSL API. Of course, the testbed should provide all the required information on how to use the resources in the context of Fed4FIRE+. Basically, the Fed4FIRE+ testbed federation uses the same set of APIs described in this Recommendation.

When the integration of the testbed is done, the resources exposed by the testbeds are displayed in the jFed tool which the main software to create an experiment inside the Fed4FIRE+ federation. Various tutorials and the related documentation are available not only for experimenters, but also for the testbed providers wanting to join Fed4FIRE+. In parallel, the connection between the testbed

and the federation core is regularly checked by an automated tool and reported on the federation monitoring website.

The process to create an experiment is straightforward:

1) Identify the technical requirements of the experiment: they depend on the item to be tested.
2) Select the Fed4FIRE+ testbeds corresponding to the requirements of the planned experiment.
3) To start the experiment preparation in the testbeds, the experimenter should create an account.
4) From his account, the experimenter obtains a certificate.
5) Afterwards, the experimenter can run the jFed tool and access all the required resources.
6) In the jFed software, the experimenter creates his own experiment and configures it.
7) Then, he runs the experiment from jFed.
8) After the end of the experiment, the experimenter collects the results and releases all the resources previously reserved.

Appendix IV

Use case for a testbed federation

(This appendix does not form an integral part of this Recommendation.)

This appendix presents a typical use case of an experiment done in a testbed federation and its related requirements. Similar use cases can be implemented in a testbed federation like Fed4FIRE+.

The use case is the experimentation of a wireless sensor network (WSN) composed by IoT devices communicating together by a wireless protocol. The goal of this experimentation is to evaluate the performance of these IoT devices in real conditions and in different environments. This means that several testbeds should be used to represent for each of them a particular environment where the wireless IoT devices are installed. Basically, the experimenter remotely accesses the testbed infrastructures to prepare and control the experiment involving the wireless IoT devices. The typical kinds of parameters to measure in such wireless network are the reliability, the latency, the radio duty cycle, the number of hops, the synchronization and the bandwidth. This experiment leads to the following requirements:

- Relevance: The tests and the parameters to measure should be relevant to the real conditions met in an industrial deployment.
- Reproducibility: The experiment should be redone in different testbeds and in different conditions.
- Repeatability: The experiment should be repeated in the same conditions.
- Automated experiment: An experiment should be run automatically.

A testbed federation like Fed4FIRE+ provides all the tools and services responding to the requirements and so, permits to the experimenter to execute and repeat the experiment in the predefined conditions.

Appendix V

Use case for a federation of testbeds

(This appendix does not form an integral part of this Recommendation.)

This appendix describes as an example a use case of an experiment realised in the context of a testbed federation like Fed4FIRE+.

The proposed use case involves two testbeds which are members of the same federation of testbeds. The first testbed is offering real sensors installed in large-scale sensor networks. These sensors are available on the market and are extensively used by the industry. The second testbed is focused on data analytics, in particular for big data, and so, provides several types of databases (MySQL, Apache Cassandra and other NoSQL databases) to store the large amount of data and several tools to monitor and analyse the received data, such as Apache Hadoop, Apache Spark and Elasticsearch.

The experiment using the two different testbeds consists in realising predictive maintenance in an industrial environment based on the data sent by the sensors. In fact, the sensors are continuously measuring different parameters in a production line composed by several industrial systems. The sensors of the first testbed are regularly sending the measurements to the second testbed. The received data are stored in the databases of the second testbed, and the monitoring and analysis tools periodically examine the large amount of data. Different methods can be used by the monitoring and analysis tools, such as event correlation techniques and root cause analysis. The abnormalities detected in the data by these tools can indicate current faults and future failures. Predictive maintenance avoids such future failures and so, reduces the costs of unscheduled maintenance.

Some requirements can be elaborated from the needs of this proposed use case. Indeed, the relevance of the testbeds towards the industry is evident and so, the first testbed should provide a similar environment to the real industrial world. This can be achieved notably by the sensors already deployed in the industry. The experiment should be automated because it can last a long period of time if a complex industrial process is to be simulated in the frame of the experiment. The testbeds used in the proposed experiment should be configurable; this means that the experimenters can change the settings of the industrial environment to get different datasets of measurements generated by the sensors. The same principles apply in the monitoring and analysis tools as a function of the data received.

Appendix VI

Use case for federations of testbeds

(This appendix does not form an integral part of this Recommendation.)

This appendix describes as an example a use case of an experiment realised in the context of two federations of testbeds using both the APIs described in this Recommendation.

A concrete use case involving two federations of testbeds is an experiment involving testbeds from the GENI testbed federation and the Fed4FIRE+ testbed federation. Currently, it is possible to use resources exposed by a testbed of the GENI federation of testbeds and to exploit these resources from an experiment done in the context of the Fed4FIRE+ federation of testbeds. Basically, GENI and Fed4FIRE+ are using almost the same APIs concerning the testbed federation, namely the aggregate manager (AM) API, the slice authority (SA) API and the member authority (MA) API. This means that the common use of APIs facilitates the exchange of information between the two well-known federations of testbeds.

The locations of the two testbed federations are different: GENI is installed in the United States of America and Fed4FIRE+ in Europe. So, a meaningful experiment involving the two continents could be for instance the benchmark of transcontinental communications. In this use case, the measurements of the performance of the transcontinental networks can be achieved through reference metrics shared by the two federations. The experiment involves several points of measurements in the two testbed federations on different types of transcontinental networks such as satellite networks or submarine communications networks. This experiment requires a tight synchronization between the two testbed federations, as well as in the management of the experiment. This will ensure correct measurements on both sides of the Atlantic ocean. The repeatability and the automation of this experiment are two important features which should be taken into account when federating two testbed federations through the APIs specified in this Recommendation.

Bibliography

[b-ITU-T Q.1743] Recommendation ITU-T Q.1743 (2016), *IMT-Advanced references to Release 11 of LTE-Advanced evolved packet core network*. https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=12982.

[b-ITU-T Y.2720] Recommendation ITU-T Y.2720 (2009), *NGN identity management framework*. https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=Y.2720.

[b-ITU-R BT.1699] Recommendation ITU-R BT.1699 (2005), *Harmonization of declarative application formats for interactive TV*. https://www.itu.int/rec/R-REC-BT.1699/en.

[b-ETSI TS 103 195-2] ETSI TS 103 195-2 V1.1.1 (2018), *Autonomic network engineering for the self-managing Future Internet (AFI); Generic Autonomic Network Architecture; Part 2: An Architectural Reference Model for Autonomic Networking, Cognitive Networking and Self-Management*. https://www.etsi.org/deliver/etsi_ts/103100_103199/10319502/01.01.01_60/ts_10319502v010101p.pdf.

[b-Fed4FIRE] Fed4FIRE+ website: https://www.fed4fire.eu/.

[b-GENI] GENI website: https://www.geni.net/.

[b-ISO/IEC 17000] ISO/IEC 17000:2004, *Conformity assessment – Vocabulary and general principles*. https://www.iso.org/standard/29316.html.

[b-NGMN 5G] ETSI TS 103 195-2, ITU-T Y.3324, NGMN 5G End-to-End Architecture Framework v3.0.8. https://www.ngmn.org/publications/5g-end-to-end-architecture-framework-v3-0-8.html.

[b-White Papers] *White papers on 5G Network Slices Creation, Autonomic Management & E2E Orchestration, with Closed-Loop (Autonomic) Service Assurance for the Slices: IoT (Smart Insurance) Use Case*. https://intwiki.etsi.org/index.php?title=Accepted_PoC_proposals.

[b-Workshop] Joint SDOs Workshop on Federated Testbeds for 5G and Beyond. www.itu.int/go/BTF4-5G.

[b-ETSI portal] ETSI portal: https://portal.etsi.org/.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	Tariff and accounting principles and international telecommunication/ICT economic and policy issues
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Environment and ICTs, climate change, e-waste, energy efficiency; construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	**Switching and signalling, and associated measurements and tests**
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2021